Справочник по Yocto Project

Scott Rifenbark

Scotty's Documentation Services, INC

<srifenbark@gmail.com>

Этот документ основан на переводе *Yocto Project Reference Manual* для выпуска 2.7.1. Свежие версии оригинальных документов можно найти на странице документации Yocto Project. Размещённые там материалы более актуальны, нежели включённые в архивы пакета Yocto Project.

Оглавление

Глава 1. Системные требования

В этом документе приведена справочная информация для текущего выпуска Yocto Project (YP). Документ лучше изучать после знакомства с основами YP. Здесь приведены определения переменных, классов и других элементов, применяемых в YP.

Вводную информацию о YP можно найти на сайте Yocto Project Website и в разделе Yocto Project Development Environment [1].

Если вы хотите использовать YP для быстрой сборки образа, не разбираясь в деталях и концепции, работайте с документом [5]. Документация по отдельным операциям представлена в [2], а обзор и концепции YP - в [1].

Дополнительные сведения о документации YP приведены в разделе Литература.

1.1. Поддерживаемые дистрибутивы Linux

Выпуски YP тестируются со стабильными дистрибутивами Linux из приведённого ниже списка. YP может работать и с другими дистрибутивами, но проверка для них не проводилась. YP не поддерживает и не включает планов по поддержке находящихся в разработке дистрибутивов по причине их частого изменения. Приветствуются исправления (patch) и сообщения об ошибках при работе с такими дистрибутивами, но следует принимать во внимание, что основные усилия разработчиков направлены на перечисленные ниже платформы

- Ubuntu 16.04 (LTS);
- Ubuntu 18.04;
- Fedora 28;
- Fedora 29;
- CentOS 7.x;
- Debian GNU/Linux 8.x (Jessie);
- Debian GNU/Linux 9.x (Stretch);
- OpenSUSE 42.3.

Yocto Project не совместим с WSL[1] и сборочный хост не будет работать на системе WSL.

При возникновении проблем воспользуйтесь ссылкой Yocto Project Bugzilla для представления ошибки. Информация о работе с системой сбора ошибок доступна на странице YP Bugzilla wiki и в разделе Submitting a Defect Against the Yocto Project [2].

Команда YP старается сделать выпуски YP полностью совместимыми с официально поддерживаемыми дистрибутивами Linux, однако могут возникать проблемы с некоторыми дистрибутивами.

1.2. Пакеты, требуемые на сборочном хосте

Список пакетов, требуемых на сборочном хосте, может быть обширным для охвата всех сценариев сборки, используемых в YP. В этом разделе рассмотрены зависимости в соответствии с дистрибутивами Linux и функциями.

1.2.1. Ubuntu u Debian

Ниже приведен список пакетов, требуемых на сборочном хосте Ubuntu или Debian.

- *Пакеты для сборки образов.*

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev \
xterm
```

- *Пакеты для сборки руководств YP.*

```
$ sudo apt-get install make xsltproc docbook-utils fop dblatex xmlto
```

Если в вашей системе сборки установлен пакет oss4-dev, могут возникнуть проблемы при сборке для QEMU по причине установки в Debian своего файла /usr/include/linux/soundcard.h. В таких случаях можно воспользоваться одним из 2 показанных ниже решений.

1.2.2. Fedora

Ниже приведен список пакетов, требуемых на сборочном хосте Fedora Linux.

- *Пакеты для сборки образов.*

```
$ sudo dnf install gawk make wget tar bzip2 gzip python3 unzip perl patch \
diffutils diffstat git cpp gcc gcc-c++ glibc-devel texinfo chrpath \
ccache perl-Data-Dumper perl-Text-ParseWords perl-Thread-Queue perl-bignum socat \
python3-pexpect findutils which file cpio python python3-pip xz python3-GitPython \
python3-jinja2 SDL-devel xterm
```

- *Пакеты для сборки руководств YP.*

```
$ sudo dnf install docbook-style-dsssl docbook-style-xsl \
docbook-dtds docbook-utils fop libxslt dblatex xmlto
```

1.2.3. OpenSUSE

Ниже приведен список пакетов, требуемых на сборочном хосте openSUSE.

- *Пакеты для сборки образов.*

```
$ sudo zypper install python gcc gcc-c++ git chrpath make wget python-xml \
diffstat makeinfo python-curses patch socat python3 python3-curses tar python3-pip \
python3-pexpect xz which python3-Jinja2 Mesa-libEGL1
$ sudo pip3 install GitPython libSDL-devel xterm
```

- *Пакеты для сборки руководств YP.*

```
$ sudo zypper install dblatex xmlto
```

1.2.4. CentOS

Ниже приведен список пакетов, требуемых на сборочном хосте CentOS Linux.

- *Пакеты для сборки образов.*

[1] Windows Subsystem for Linux.

```
$ sudo yum install -y epel-release
$ sudo yum makecache
$ sudo yum install gawk make wget tar bzip2 gzip python unzip perl patch \
diffutils diffstat git cpp gcc gcc-c++ glibc-devel texinfo chrpath socat \
perl-Data-Dumper perl-Text-ParseWords perl-Thread-Queue python34-pip xz \
which SDL-devel xterm
$ sudo pip3 install GitPython jinja2
```

- Дополнительные пакеты для Enterprise Linux (т. е. epel-release) представляют собой набор программ из Fedora на RHEL/CentOS для простой установки пакетов, не включаемых в корпоративный дистрибутив Linux по умолчанию. Эти пакеты нужно устанавливать отдельно.
- Команда makecache получает дополнительные метаданные от epel-release.

- *Пакеты для сборки руководств YP.*

```
$ sudo yum install docbook-style-dsssl docbook-style-xsl \
docbook-dtds docbook-utils fop libxslt dblatex xmlto
```

1.3. Требуемые версии Git, tar и Python

Для работы с системой сборки на сборочном хосте должны быть установлены пакеты:

- Git не ниже 1.8.3.1;
- tar не ниже 1.27;
- Python не ниже 3.4.0.

Если эти требования не выполняются, можно установить пакет buildtools, содержащий эти программы. Пакет можно загрузить в виде архива (tarball) или собрать самостоятельно с помощью BitBake.

1.3.1. Загрузка собранного архива buildtools

Загрузка и запуск подготовленного установщика buildtools обеспечивает простейший способ получить инструменты.

1. По ссылке http://downloads.yoctoproject.org/releases/yocto/yocto-2.7.1/buildtools/ найдите и загрузите файл *.sh.
2. Запустите сценарий установки, как показано ниже.

   ```
   $ sh ~/Downloads/x86_64-buildtools-nativesdk-standalone-2.7.1.sh
   ```

   В процессе работы сценария будут выводиться приглашения для выбора каталога установки. Можно выбрать, например, каталог /home/your-username/buildtools.
3. Запустите сценарий настройки среды с помощью команды

   ```
   $ source /home/your_username/buildtools/environment-setup-i586-poky-linux
   ```

   Здесь вам нужно будет указать реальный каталог установки и убедиться в использовании подходящего файла (например, i586 или x86-64).

   После завершения сценария установки будет добавлен каталог инструментов в переменную PATH и инициализированы другие переменные окружения, нужные для работы с инструментами. В результате вы получите пригодные для работы версии Git, tar, Python и chrpath.

1.3.2. Сборка buildtools

Сборка и запуск своего установщика buildtools применяется лишь на сборочных хостах, где уже имеется BitBake. В этом случае сборочный хост служит для создания файла .sh и последующего выполнения этапов для его переноса и запуска на машине, не соответствующей требованиям к Git, tar и Python.

1. На машине с BitBake нужно организовать среду сборки с помощью установочного сценария (oe-init-build-env).
2. Для сборки инструментов используется команда BitBake

   ```
   $ bitbake buildtools-tarball
   ```

   Переменная SDKMACHINE в файле local.conf определяет сборку для 32- или 64-битовой системы.

   По завершении сборки файл .sh для установки инструментов размещается в каталоге tmp/deploy/sdk каталога сборки. В имени файла присутствует строка buildtools.
3. Файл .sh file следует перенести на машину, где не выполняются требования к Git, tar или Python.
4. На этой машине запускается сценарий .sh для установки программ. Например,

   ```
   $ sh ~/Downloads/x86_64-buildtools-nativesdk-standalone-2.7.1.sh
   ```

   В процессе работы сценария выводится приглашение указать каталог для установки. Например, /home/your_username/buildtools.
5. Запускается сценарий установки параметров окружения

   ```
   $ source /home/your_username/buildtools/environment-setup-i586-poky-linux
   ```

   Здесь вам нужно будет указать реальный каталог установки и убедиться в использовании подходящего файла (например, i586 или x86-64).

   После завершения сценария установки будет добавлен каталог инструментов в переменную PATH и инициализированы другие переменные окружения, нужные для работы с инструментами. В результате вы получите пригодные для работы версии Git, tar, Python и chrpath.

Глава 2. Терминология YP

В этой главе приведены определения терминов, используемых в среде разработки YP.

Append Files - файлы дополнения

Файлы, добавляющие данные сборки в конец файла задания. Эти файлы называют файлами дополнения BitBake (.bbappend). Система сборки OE предполагает, что для каждого файла дополнения имеется соответствующий файл задания (.bb) - эти файлы должны иметь имена, которые отличаются только расширением (например, formfactor_0.0.bb и formfactor_0.0.bbappend).

Информация в файлах дополнения расширяет или переопределяет данные из файла задания с идентичным именем. Пример использования файлов дополнения приведен в разделе Using .bbappend Files in Your Layer [2].

В именах файлов добавления можно использовать шаблон % для сопоставления с именами заданий. Предположим, что файл добавления назван busybox_1.21.%.bbappend. Этот файл будет соответствовать любой версии файла задания busybox_1.21.x.bb, например

```
busybox_1.21.1.bb
busybox_1.21.2.bb
busybox_1.21.3.bb
```

Использование символа % допускается лишь непосредственно перед расширением .bbappend.

BitBake

Процессор задач и планировщик, используемый системой сборки OE для создания образов (см. [4]).

Board Support Package (BSP) - пакет поддержки платы

Набор драйверов, определений и других компонент для поддержки конкретной аппаратной конфигурации [6].

Build Directory - каталог сборки

Область, используемая системой OE для сборки кода. Область создаётся при запуске сценария организации рабочего окружения из каталога исходных кодов (oe-init-build-env). Переменная TOPDIR указывает каталог сборки.

При создании Build Directory обеспечивается достаточная гибкость. Ниже приведены несколько примеров создания каталога сборки, в которых предполагается каталог исходных кодов poky.

- Build Directory внутри Source Directory с принятым по умолчанию именем build

```
$ cd $HOME/poky
$ source oe-init-build-env
```

- Создание Build Directory в домашнем каталоге с именем test-builds

```
$ cd $HOME
$ source poky/oe-init-build-env test-builds
```

- Указание пути к каталогу и имени Build Directory. При этом должны существовать все промежуточные каталоги пути к Build Directory. Ниже показано создание каталога сборки YP-21.0.0 в имеющемся каталоге mybuilds внутри домашнего каталога

```
$cd $HOME
$ source $HOME/poky/oe-init-build-env $HOME/mybuilds/YP-21.0.0
```

По умолчанию каталог сборки включает подкаталог TMPDIR, используемый для временных файлов в процессе работы. Каталог TMPDIR не может размещаться в файловой системе NFS[1], поэтому каталог сборки по умолчанию не может размещаться на NFS. Однако если требуется разметить его в сети, это можно сделать, установив для переменной TMPDIR каталог локального диска в конфигурационном файле local.conf. В результате это разделяет каталоги TMPDIR и TOPDIR, которым является каталог сборки.

Build Host - сборочный хост

Система, используемая для сборки образов в среде разработки YP. Иногда называется хостом разработки.

Classes - классы

Файлы, обеспечивающие логическую инкапсуляцию и наследование, для упрощения многократного использования шаблонов общего назначения (Глава 6. Классы). Файлы классов используют расширение имён .bbclass.

Configuration File - файл конфигурации

Файлы с глобальными определениями переменных, пользовательскими переменными и данными аппаратной конфигурации. Эти файлы указывают системе OE, что нужно собрать и что включить в образ для поддержки конкретной платформы. Конфигурационный файлы используют расширение .conf. Файл conf/local.conf в каталоге сборки содержит заданные пользователем переменные, которые влияют на каждую сборку. Файл meta-poky/conf/distro/poky.conf определяет переменные конфигурации Yocto distro, используемые только при сборке с этим правилом. Файлы конфигурации машин в дереве исходных кодов определяют переменные для конкретного оборудования и используются лишь при сборке для соответствующей цели (например, файл machine/beaglebone.conf используется при сборке для плат Texas Instruments ARM Cortex-A8).

Container Layer - контейнерный уровень

Уровень, содержащий другие уровни. Примером контейнерного уровня является OE meta-openembedded, содержащий множество уровней meta-*.

Cross-Development Toolchain - инструменты кросс-разработки

В общем случае набор инструментов кросс-разработки представляет собой набор программ и утилит, работающих на одной архитектуре и позволяющих собирать программы для другой целевой архитектуры. Эти инструменты включают кросс-компиляторы, компоновщики и отладчики для конкретной целевой архитектуры.

- YP поддерживает два набора инструментов для кросс-разработки:
  - инструменты, применяемые BitBake при сборке образов для целевой архитектуры;
  - перемещаемые инструменты, применяемые вне BitBake при создании приложений для работы на целевой платформе.
- Создание этого инструментария достаточно просто и автоматизировано. Информация о концепциях работы инструментария приведена в разделе Cross-Development Toolchain Generation [1], а также в документе [7].

Extensible Software Development Kit (eSDK) - расширяемый пакет для разработки программ

Специализированный пакет SDK[2] для разработки приложений, позволяющий разработчикам встраивать свои библиотеки и изменения программ в образ, а также делать свой код доступным для других разработчиков [7].

Image - образ

Образ является конечным результатом процесса сборки BitBake для заданного набора заданий и связанных с ними метаданных. Образы являются двоичным выводом, предназначенным для работы на конкретном оборудовании или QEMU и применяемым для определённых задач. Список поддерживаемых YP типов образов представлен ниже (Глава 11. Образы).

[1]Network File System - сетевая файловая система.

[2]Software Development Kit - комплект для разработки программ.

Layer - уровень

Набор связанных между собой заданий. Уровни позволяют консолидировать метаданные на настройки сборки, а также изолируют информацию для разных вариантов архитектуры. Уровни являются иерархическими в части переопределения предшествующих спецификаций. Вы может включить любое число доступных уровней из YP и добавить к ним свои уровни. Внутри YP поддерживается возможность поиска нужного уровня.

Базовые сведения об уровнях приведены в разделе The Yocto Project Layer Model [1], а более подробная информация - в разделе Understanding and Creating Layers [2]. Уровни BSP рассмотрены в разделе BSP Layers [6].

Metadata - метаданные

Используются для создания дистрибутивов Linux и содержатся в файлах, которые система сборки OE анализирует при сборке образа. В общем случае метаданные включают задания, конфигурационные файлы и другую информацию, относящуюся к инструкциям по сборке, а также данные, управляющие тем, что создаётся, и влияющие на сборку. Метаданные включают команды и данные, используемые для указания используемых версий программ, места, откуда их следует загружать, и изменениях или дополнениях к этим программам (исправления или дополнительные файлы) для исправления ошибок или настройки под конкретную задачу. OpenEmbedded-Core содержит важный для работы набор проверенных метаданных.

В контексте ядра (kernel Metadata) термин относится к фрагментам конфигурации ядра и функциям, включённым в репозиторий yocto-kernel-cache Git.

OpenEmbedded-Core (OE-Core)

OE-Core - это метаданные, включающие основные задания, классы и связанные с ними файлы, которые должны быть общими для множества разных систем, производных от OE, включая YP. OE-Core представляет собой подмножество репозитория, созданного сообществом OE, которое было собрано в меньших набор проверенных заданий. Результатом стал строго контролируемый и качественный набор базовых заданий.

Эти метаданные можно просмотреть в каталоге meta Yocto Project Source Repositories.

OpenEmbedded Build System - система сборки OpenEmbedded

Система сборки в составе YP, основанная на проекте Poky, использующем BitBake для выполнения задач. В документации YP система сборки OE иногда называется просто системой сборки. При указании других систем сборки (например, на хосте или целевой платформе) они задаются явно.

Package - пакет

В контексте YP пакетом называется результат выполнения задания, созданный BitBake (выполненное задание), - обычные двоичные файлы, созданные из исходных кодов задания.

С использованием термина «пакет» связаны некоторые нюансы. Например, пакеты, упоминаемые в параграфе 1.2. Пакеты, требуемые на сборочном хосте, являются скомпилированными двоичными файлами, установка которых расширяет функциональность вашего дистрибутива Linux. Исторически в рамках YP пакетами назывались задания и некоторые переменные BitBake в результате сохранили не вполне корректные имена (например PR, PV, PE).

Package Groups - группа пакетов

Произвольная группа заданий для программных пакетов. Группы служат для объединения заданий сборка которых обычно выполняется в виде одной задачи. Например, группа может содержать задания для сборки фирменных программ или приложений с расширенными функциями. Другим вариантом может служить группа, включающая поддержку графики. По сути группа заданий является ещё одним заданием, поэтому файлы групп используют расширение .bb.

Poky

Poky (произносится *Pock*-ee) является эталонным дистрибутивом для встраиваемых систем и тестирования. Возможности Poky приведены ниже.

- Базовый дистрибутив для иллюстрации настройки своего дистрибутива.
- Средства и способы тестирования компонент YP (т. е. Poky служит для проверки YP).
- Средство для загрузки YP.

Poky не является дистрибутивом для реального применения, это просто стартовая точка для создания дистрибутива. Проект poky начинался с открытой системы, разработанной OpenedHand на основе имевшейся системы сборки OE для создания коммерчески поддерживаемой системы сборки Linux для встраиваемых систем. После покупки OpenedHand корпорацией Intel проект poky стал основой для системы сборки YP.

Recipe - задание

Набор инструкций для сборки пакетов. Задание указывает местоположение исходного кода, применяемые исправления, настройку конфигурации, параметры компиляции и т. п. Задания также указывают зависимости от библиотек или других заданий. Задание является логическим блоком выполнения, программой или образом для сборки и задаётся в файле .bb.

Reference Kit - эталонный комплект

Работающий пример системы, которая включает BSP, а также сборочный хост и другие компоненты и может функционировать на указанном оборудовании.

Source Directory - каталог исходных кодов

Этот термин обозначает структуру (дерево) каталогов, созданную как локальная копия репозитория poky Git git://git.yoctoproject.org/poky или распакованную из архива poky. Создание локальной копии репозитория poky Git рекомендуется в качестве метода организации Source Directory. Иногда для структуры используется название «каталог poky».

Система OE не поддерживает имена каталогов и файлов, содержащие пробелы, поэтому следует убедиться в их отсутствии в имени Source Directory.

Структура Source Directory содержит BitBake, документацию, метаданные и другие файлы, нужные для YP. Следовательно, Source Directory является необходимой частью YP.

При создании локальной копии репозитория Git ему можно дать любое имя. В документации обычно используется poky в качестве имени каталога верхнего уровня для локальной копии репозитория poky Git. Обычное клонирование репозитория poky Git без указания локального имени будет создавать копию в каталоге poky.

Если вы будете создавать Source Directory из архива, имя каталога верхнего уровня также будет взято из этого архива. Например, при загрузке и распаковке архива poky-warrior-21.0.0.tar.bz2 вы получите Source Directory с корневым каталогом poky-warrior-21.0.0.

Важно понимать различия между Source Directory из архива и путём клонирования git://git.yoctoproject.org/poky. При распаковке архива вы получите копии файлов конкретного выпуска и внесённые вами изменения будут сохраняться лишь локально. При работе с локальным репозиторием poky Git вы будете иметь свежие копии

файлов, а внесённые изменения можно будет представить в находящиеся в разработке ветви репозитория poky Git.

Дополнительную информацию о работе с репозиториями Git, ветвями и тегами можно найти в разделе Repositories, Tags, and Branches [1].

Task - задача

Блок выполнения в BitBake (например, do_compile, do_fetch, do_patch и т. п.).

Toaster

Web интерфейс для системы сборки OpenEmbedded, позволяющий настраивать и запускать сборку. Информация о сборке собирается и храниться в базе данных [8].

Upstream

Ссылка на исходный код или репозитории, которые не находятся в системе разработки, а расположены в области, поддерживаемой разработчиками (maintainer) исходного кода. Например, для работы с определённой частью кода его нужно сначала получить из «восходящего» источника.

Глава 3. Выпуски YP и создание стабильного выпуска

Процесс выпуска YP предсказуем и включает важные и второстепенные изменения. В этой главе кратко описано именование выпусков, их жизненный цикл и стабильность.

3.1. Основные и второстепенные выпуски

Основные выпуски YP (например, 2.7.1) выходят с интервалом около 6 месяцев в апреле и октябре каждого года. Ниже приведены несколько основных выпусков YP с кодовыми именами (см. 3.2. Кодовые имена основных выпусков).

2.2 (Morty);

2.1 (Krogoth);

2.0 (Jethro).

Временные интервалы никогда не бывают точными, но полугодовой интервал облегчает регулярные выпуски с жёстким контролем качества (QA[1]) без перегрузки пользователей частой сменой выпусков. Месяцы выпуска выбраны с учётом продолжительных праздников в разных странах.

Второстепенные выпуски YP (младшая цифра) выходят нерегулярно и обычно вызваны накоплением достаточно важных изменений или улучшений последнего основного выпуска. Ниже приведены примеры второстепенных выпусков.

2.1.1;

2.1.2;

2.2.1.

Второстепенный выпуск указывает точку ветвления от основного выпуска, где был выполнен полный цикл QA и проверка содержимого новой ветви.

Могут также выпускаться исправления (patch) к стабильному выпуску по мере их появления.

3.2. Кодовые имена основных выпусков

Каждый основной выпуск получает кодовое имя, указывающее его в репозитории Yocto Project. Ветви метаданных с одинаковым кодовым именем скорей всего будут совместимы и будут работать вместе.

Кодовое имя связывается с основным выпуском, поскольку номер выпуска YP (например, 2.7.1) может конфликтовать с данным уровнем или схемой нумерации версий компании. Кодовые имена уникальны и легко узнаваемы.

Выпуски тоже имеют номинальную версию и по этой причине в репозиториях используется кодовое имя. Информацию о выпусках и кодовых именах YP можно найти по ссылке https://wiki.yoctoproject.org/wiki/Releases.

3.3. Процесс подготовки стабильного выпуска

После выхода нового стабильного выпуска назначается ответственный за его поддержку. Этот человек отслеживает связанные с выпуском события, исследуя и обрабатывая предложенные изменения. Для передачи в стабильный выпуск рассматриваются только исправления и улучшения в ветви master (текущая разрабатываемая ветвь).

Текущая политика YP в части переноса в стабильные выпуски принимает во внимание лишь исправления ошибок и связанные с безопасностью изменения. Это означает, что обновления версии базового задания вряд ли будут приняты для стабильных выпусков. Исключением может служить наличие веских причин, таких как исправления, которые имеет смысл принять и в разрабатываемой версии.

Стабильные выпуски сопровождаются в течение года с момента публикации. Если будут обнаружены серьёзные проблемы, исправления будут перенесены в прошлые выпуски независимо от их возраста. Для проблем, решения которых не были перенесены в прошлые выпуски, имеются деревья и ветви Community LTS, где участники делятся patch-файлами для прошлых выпусков. Однако эти исправления не проходят строгого контроля. Информацию о поддержке стабильных ветвей можно найти по ссылке https://wiki.yoctoproject.org/wiki/Stable_branch_maintenance.

3.4. Тестирование и контроль качества

Частью процесса разработки и выпуска YP является контроль качества, основанный на тестировании. Стратегия тестирования позволяет команде YP проверить готовность выпуска. Кроме того, стратегия тестирования доступна для разработчиков, что позволяет вам проверить свои проекты. В этом разделе приведен обзор инфраструктуры тестирования YP. Дополнительную информацию о доступных тестах для ваших проектов можно найти в разделе Performing Automated Runtime Testing [2].

[1]Quality Assurance - гарантия (контроль) качества.

Инфраструктура тестирования и контроля качества интегрирована в проект и включает несколько частей.

- bitbake-selftest - автономная команда для запуска тестирования основных частей BitBake и его сборщиков.
- sanity.bbclass - автоматически включаемый класс для проверки полноты инструментов среды сборки и ошибок в базовой конфигурации (например, некорректная установка переменной MACHINE).
- insane.bbclass - проверка вывода сборки на здравомыслие. Например, при сборке для платформы ARM проверяется назначение выходных двоичных файлов именно для этой платформы.
- testimage.bbclass - тестирование работы созданных при сборке образов. Тесты обычно применяются с QEMU для загрузки образа и проверки его работы. Однако тест может также выполняться для машины, указанной адресом IP.
- ptest запускает тесты для пакетов, созданных при сборке программ. Тесты выполняются на целевом образе.
- oe-selftest тестирует вызовы BitBake. Эти тесты выполняются вне системы сборки OE. Команда oe-selftest по умолчанию запускает все тесты, но можно указать набор нужных тестов. Для использования oe-selftest на хосте нужно установить дополнительные пакеты (см. параграф 1.2. Пакеты, требуемые на сборочном хосте).

Изначально многие из этих тестов выполнялись вручную. Были приложены существенные усилия по автоматизации тестов, чтобы больше людей могли пользоваться ими, а команда YP могла быстрее и эффективней вести разработки.

Основной автосборщик YP (autobuilder.yoctoproject.org) тестирует код каждого выпуска YP из репозиториев OE-Core, Poky и BitBake. Тестирование выполняется для текущего состояния ветви master и предложенных исправлений. Для исправлений тестирование обычно проходит в ветви ross/mut репозитория poky-contrib (ветвь master-under-test) или master-next в репозитории poky.

Тестирование этих общедоступных ветвей обеспечивает для всех желающих информацию о проверке предложенных вариантов архитектуры и заданий OE-Core. В тестах QA проверяются разные свойства, такие как multilib, субархитектура (например, x32, poky-tiny, musl, no-x11), bitbake-selftest, oe-selftest. Полное тестирование и проверка выпуска в Autobuilder занимает несколько часов. Тестирование выполняется в разных дистрибутивах Linux на системах QEMU, а не на реальном оборудовании. В дополнение к тестам Autobuilder команда YP QA выполняет тесты на разных платформах.

Глава 4. Переход на новые выпуски YP

В этой главе представлена информация о переходе с одного выпуска YP на другой. Дополнительные сведения можно получить из заметок (release notes) к соответствующему выпуску.

4.1. Общие вопросы

Некоторые вопросы перехода не связаны с конкретным выпуском YP и В этом разделе представлена информация, актуальная для любого перехода на новый выпуск YP.

Работа с пользовательскими заданиями

Могут возникать проблемы при использовании заданий, которые были изменены в прежнем выпуске и просто скопированы в новый. Например, задание на вашем уровне может содержать изменённое задание core из прежнего выпуска, а не использовать файл добавления. При переходе на новый выпуск YP метаданные (например, включённые в задание файлы) могут измениться так, что ваше задание не будет работать. Это может быть, например, следствием удаления из включённого файла функции, которую задание пытается использовать.

Вы может скорректировать все настройки в своём задании для работы с новым выпуском, однако это решение не будет оптимальным, поскольку процесс придётся повторять для каждого нового выпуска, если проблемы будут возникать. Более эффективным решением будет использование файлов добавления (*.bbappend) для фиксации ваших настроек задания. Это изолирует внесённые вами изменений от основного задания и сделает процесс более контролируемым. Однако на практике использование файлов добавления удобно не всегда. Вариантом решения проблемы может быть создание нового задания или перенос прежнего на другой уровень.

Обновление файлов дополнения (Append)

Поскольку файлы добавления обычно содержат лишь пользовательские настройки, из часто не нужно настраивать для новых выпусков. Однако, если файл .bbappend относится к конкретной версии задания (т. е. в имени не используется шаблон %) и версия задания будет изменена, потребуется по меньшей мере переименовать файл добавления в соответствии с новым файлом задания. Несоответствие имени файла дополнения и задания (.bb) приведёт к ошибке при анализе.

В зависимости от типа настроек в файле добавления при обновлении выпуска могут возникать и другие несовместимости. Например, если файл добавления применяет исправления, а дополненное им задание обновлено до новой версии, patch-файла может не оказаться. В таких случаях нужно обновить patch-файл, если необходимость его сохраняется.

4.2. Переход на YP 1.3

4.2.1. Локальная конфигурация

Изменена переменная SSTATE_MIRRORS и файл bblayers.conf.

4.2.1.1. SSTATE_MIRRORS

Кэш общих состояний (sstate-cache), указанный переменной SSTATE_DIR, по умолчанию использует двухсимвольные имена каталогов для предотвращения проблем, связанных с чрезмерным числом файлов в одном каталоге. Кроме того, естественные пакеты sstate-cache, которые собираются для работы на сборочном хосте, будут помещаться в подкаталог, имя которого содержит идентификатор дистрибутива. Если вы копируете недавно структурированные данные sstate-cache на зеркало (локальное или удалённое), и указываете его в переменной SSTATE_MIRRORS, нужно добавлять в конце URL зеркала строку PATH, используемую BitBake перед добавлением пути к зеркалу. Например,

```
SSTATE_MIRRORS = "file://.* http://someserver.tld/share/sstate/PATH"
```

4.2.1.2. bblayers.conf

Уровень meta-yocto состоит из 2 частей, соответствующих дистрибутиву Poky и пакетам поддержки плат BSP[1] - meta-yocto и meta-yocto-bsp, соответственно. При первом запуске BitBake после обновления файл conf/bblayers.conf будет обновлён с учётом этих изменений и будет выведено предложение о перезапуске для учёта изменений.

4.2.2. Задания

Изменения в заданиях включают:

- пробелы в функциях Python;
- proto= в SRC_URI;
- nativesdk;
- задания для задач;
- IMAGE_FEATURES;
- удалённые задания.

4.2.2.1. Пробелы в функциях Python

Все функции Python должны использовать отступы из 4 пробелов. Ранее допускалось произвольное смешивание пробелов и символов табуляции, что усложняло расширение этих функций с помощью _append или _prepend с учётом того, что Python считает пробелы синтаксически значимыми. При определении или расширении функций Python (например, populate_packages, do_unpack, do_patch) в пользовательских заданиях или классах нужно использовать отступы из 4 пробелов.

4.2.2.2. proto= в SRC_URI

Все включения proto= в SRC_URI должны быть заменены на protocol=. Это относится к URI типов svn://, bzr://, hg:// и osc://, поскольку в остальных уже используется protocol=.

4.2.2.3. nativesdk

Суффикс nativesdk сейчас реализован в виде префикса что значительно упрощает упаковочный код для заданий nativesdk. Все пользовательские задания nativesdk, которые являются перемещаемыми пакетами и естественны для SDK_ARCH, а также все ссылки нужно обновить с использованием nativesdk-* вместо *-nativesdk.

4.2.2.4. Задания для задач

Задания для задач сейчас называются группами пакетов (Package group) и переименованы - вместо task-*.bb используется packagegroup-*.bb. Имеющиеся ссылки на имена прежних задач task-* будут в большинстве случаев работать, поскольку для большинства пакетов существует путь автоматического обновления. Однако нужно будет обновить ссылки в ваших заданиях и конфигурациях, поскольку в будущих выпусках они могут уже не работать. Следует переименовать пользовательские задания task-* в to packagegroup-* и изменить их для наследования packagegroup вместо task, а также удалить все, что обрабатывалось классом packagegroup.bbclass (например, предоставление пакетов -dev и -dbg, установка переменной LIC_FILES_CHKSUM и т. п.). Дополнительная информация приведена в параграфе 6.94. packagegroup.bbclass.

4.2.2.5. IMAGE_FEATURES

Заданиям для образов, включавшим apps-console-core в переменной IMAGE_FEATURES, следует вместо этого указывать splash для включения заставки при загрузке. При сохранении apps-console-core заставка будет включаться, но с выдачей предупреждения. Свойства apps-x11-core и apps-x11-games для IMAGE_FEATURES были удалены.

4.2.2.6. Удалённые задания

В этом выпуске были удалены перечисленные ниже задания, для большинства которых маловероятны ссылки из ваших метаданных. Однако следует проверить наличие таких ссылок.

- *libx11-trim* заменено на libx11, практически не отличающийся по размеру от современного Xorg.
- *xserver-xorg-lite* - используйте взамен xserver-xorg, который практически не отличается по размеру, когда модули DRI и GLX не установлены.
- *xserver-kdrive* - уже много лет не поддерживался.
- *mesa-xlib* - больше не нужно.
- *galago* - заменено на telepathy.
- *gail* - функционально интегрировано в GTK+ 2.13.
- *eggdbus* - больше не нужно.
- *gcc-*-intermediate* - сборка реструктурирована для исключения этого этапа.
- *libgsmd* - уже много лет не поддерживался. Функциональной заменой служит ofono.
- *contacts, dates, tasks, eds-tools* - практически не поддерживаемый набор приложений PIM. Перемещено в meta-gnome уровня meta-openembedded.

В дополнение к перечисленным изменениям, каталог meta-demoapps был удалён, поскольку содержащиеся в нем задания не поддерживались и многие из них устарели или были запрошены. Кроме того, эти задания не

[1]Board Support Package.

анализировались в принятой по умолчанию конфигурации. Многие из этих заданий уже заменены и поддерживаются в уровнях сообщества OE, таких как meta-oe и meta-gnome. Остальные можно найти в репозитории meta-extras на YP.

4.2.3. Именование ядер Linux

Схема именования двоичных файлов ядра изменена с включением переменной PE как части имени KERNEL_IMAGE_BASE_NAME ?= "${KERNEL_IMAGETYPE}-${PE}-${PV}-${PR}-${MACHINE}-${DATETIME}". Поскольку переменная PE по умолчанию не задана, двоичные файлы по умолчанию будут включать в имени два символа дефиса (--). Например, bzImage--3.10.9+git0+cd502a8814_7144bcc4b8-r0-qemux86-64-20130830085431.bin

4.3. Переход на YP 1.4

4.3.1. BitBake

- *Многострочные комментарии.* Если строка комментария завершается символом \, следующая строка также будет являться комментарием. Все нарушения этого правила будут приводить к выводу предупреждений. В текст файлов с такими нарушения следует внести соответствующие правки.
- *Переопределение имён пакетов.* Переменным, относящимся к именам пакетов во время выполнения (RDEPENDS, RRECOMMENDS, RSUGGESTS, RPROVIDES, RCONFLICTS, RREPLACES, FILES, ALLOW_EMPTY), а также функциям сценариев, связанных с установкой (pkg_preinst, pkg_postinst, pkg_prerm, pkg_postrm), всегда следует переопределять имена пакетов. Например, для основного пакета следует использовать RDEPENDS_${PN}, а не просто RDEPENDS. BitBake использует более строгую проверку при анализе заданий.

4.3.2. Поведение при сборке

- *Код общего состояния* был оптимизирован для предотвращения работы ненужных задач. Например, приведённая ниже команда больше не заполняет sysroot, поскольку это не требуется.

```
$ bitbake -c rootfs some-image
```

Вместо этого системе нужно просто извлечь содержимое выходных пакетов, заново создать пакеты и организовать корневую файловую систему. Это изменение не вызовет каких-либо проблем, если соблюдены объявленные зависимости.

- *Имена сканируемых каталогов.* При сканировании для поиска файлов из SRC_URI система сборки использует для имён каталогов переменную FILESOVERRIDES вместо OVERRIDES. Обычно значение OVERRIDES совпадают с FILESOVERRIDES, однако при использовании дополнительного значения в OVERRIDES может потребоваться его добавление в FILESOVERRIDES, если оно уже не добавлено в переменную MACHINEOVERRIDES или DISTROOVERRIDES (Глава 13. Переменные).

4.3.3. Прокси и извлечение исходного кода

Добавлен сценарий oe-git-proxy для извлечения источников из репозиториев Git через прокси. Работа со сценарием описана в файле meta-yocto/conf/site.conf.sample.

4.3.4. Пользовательские интерфейсные файлы (изменение netbase)

Если вы создали свой файл etc/network/interfaces с помощью добавления к заданию netbase, вместо него потребуется создать файл добавления к заданию init-ifupdown, которое можно найти в каталоге meta/recipes-core/init-ifupdown дерева исходных кодов (см. раздел Using .bbappend Files [2]).

4.3.5. Удалённая отладка

Поддержка удалённой отладки с использованием Eclipse IDE выделена в свойство образа (eclipse-debug), которое соответствует группе пакетов packagegroup-core-eclipse-debug. Ранее отладка включалась через свойство tools-debug из группы packagegroup-core-tools-debug.

4.3.6. Переменные

- SANITY_TESTED_DISTROS использует идентификатор дистрибутива, состоящий из идентификатора хоста, за которым следует идентификатор выпуска. Ранее переменная SANITY_TESTED_DISTROS содержала поле описания. Например, значение Ubuntu 12.10 становится Ubuntu-12.10. Это изменение не затронет вас, если переменная не задавалась или содержала пустое значение ("").
- SRC_URI. Каталоги ${PN}, ${PF}, ${P} и FILE_DIRNAME исключены из принятого по умолчанию значения переменной FILESPATH, которая служит для указания пути поиска файлов, заданных в SRC_URI. Если ваши задания используют указанные каталоги нужно внести изменения в задание или переместить файлы. Часто используемые каталоги включены в переменные ${BP}, ${BPN} и files, которые сохранены в принятом по умолчанию значении FILESPATH.

4.3.7. Управление пакетами целевой платформы с помощью RPM

Управление пакетами в процессе работы разрешено и используется механизм RPM. Вместо Zypper устанавливается пакет Smart для загрузки программ, контроля зависимостей и обновления. Информация о работе со Smart выводится по команде smart --help.

4.3.8. Перенос заданий

Перечисленные ниже задания были перемещены, поскольку они больше не используются в OpenEmbedded-Core.

- *clutter-box2d* на уровень meta-oe.
- *evolution-data-server* на уровень meta-gnome.

- *gthumb* на уровень meta-gnome.
- *gtkhtml2* на уровень meta-oe.
- *gupnp* на уровень meta-multimedia.
- *gypsy* на уровень meta-oe.
- *libcanberra* на уровень meta-gnome.
- *libgdata* на уровень meta-gnome.
- *libmusicbrainz* на уровень meta-multimedia.
- *metacity* на уровень meta-gnome.
- *polkit* на уровень meta-oe.
- *zeroconf* на уровень meta-networking.

4.3.9. Удаления и переименования

- *evieext* удалено по причине удаления из xserver в 2008 г.
- *Gtk+ DirectFB* удалено по причине исключения из Gtk+ начиная с версии 2.18.
- *libxfontcache, xfontcacheproto* удалены по причине удаления из сервера Xorg в 2008 г.
- *libxp, libxprintapputil, libxprintutil, printproto* удалены по причине удаления сервера XPrint из Xorg в 2008 г.
- *libxtrap, xtrapproto* удалены по причине отказа от этой функциональности.
- *Ядро linux-yocto 3.0* заменено ядром linux-yocto 3.8, ядра linux-yocto 3.2 и linux-yocto 3.4 сохранены в выпуске.
- *lsbsetup* удалено, поскольку функциональность обеспечивается lsbtest.
- *matchbox-stroke* удалено, поскольку служило лишь для проверки концепции.
- *matchbox-wm-2, matchbox-theme-sato-2* удалены, поскольку больше не поддерживаются. Однако matchbox-wm и matchbox-theme-sato сохранены.
- *mesa-dri* переименовано в mesa.
- *mesa-xlib* удалено по причине бесполезности.
- *mutter* удалено по причине ненужности.
- *orinoco-conf* удалено, поскольку устарело.
- *update-modules* удалено, поскольку больше не применяется. Сценарии postinstall и postrm для модулей ядра работают без этого сценария.
- *web* удалено, поскольку больше не поддерживаются и заменено на web-webkit.
- *xf86bigfontproto* удалено по причине исключения из разработки в 2007 г. Сейчас применяется xf86bigfontproto.
- *xf86rushproto* удалено, поскольку зависимость от xserver была ошибочной и удалена в 2005 г.
- *zypper, libzypp, sat-solver* удалены и функционально заменены на Smart (python-smartpm) при использовании пакетов RPM и включённом на целевой платформе менеджере пакетов.

4.4. Переход на YP 1.5

4.4.1. Изменения зависимостей для сборочного хоста

Система сборки OE предъявляет приведённые ниже требования к программам сборочного хоста.

- Python 2.7.3 или выше;
- Tar 1.24 или выше;
- Git 1.7.8 или выше;
- исправленная версия make при использовании 3.82 (применяется в большинстве дистрибутивов с make 3.82).

Если в дистрибутиве Linux на сборочном хосте нет указанных версий программ, можно воспользоваться архивом Buildtools, включающим эти программы (см. параграф 1.3. Требуемые версии Git, tar и Python).

4.4.2. atom-pc BSP

Эталонный аппаратный пакет BSP для atom-pc заменён BSP generic86. Он не обязательно будет работать на всех системах x86, но пригоден для работы на большинстве систем, где работал atom-pc. Кроме того, добавлен BSP generic86-64 для 64-битовых систем Atom.

4.4.3. BitBake

- BitBake теперь поддерживает оператор _remove, что потребует переименовать все элементы (функции, переменные) в пространстве задания, содержащие _remove_ или заканчивающиеся _remove, для предотвращения неожиданного поведения.
- Удалён глобальный (global) метод извлечения (pool) для BitBake, который был практически бесполезен и приводил к конфликтам с заданиями, содержащими функции с таким же именем.

- Удалён backend-сервер none, поскольку по умолчанию уже давно применяется сервер process.
- Удалён сценарий bitbake-runtask.
- Переменные ${P} и ${PF} больше не добавляются по умолчанию в переменную PROVIDES файла bitbake.conf. Эти зависимые от версии элементы PROVIDES применялись редко и попытка использовать их может приводить к одновременной сборке двух версий в результате используемого в BitBake метода контроля зависимостей.

4.4.4. Предупреждения QA

- При установке значений ERROR_QA или WARN_QA в вашей конфигурации следует убедиться, что там указаны все проблемы, которые вы хотите видеть. В предыдущих выпусках YP была ошибка, в результате которой объекты, не указанные в ERROR_QA или WARN_QA, считались предупреждениями и некоторые важные элементы не включались по умолчанию в WARN_QA. Проверки QA описаны в параграфе 6.56. insane.bbclass.
- Добавлена проверка установки /usr/share/info/dir. В вашем задании следует удалить этот файл в задаче do_install, если его устанавливает команда make install.
- При использовании класса buildhistory проверка возврата версии пакетов выполняется стандартной процедурой QA. Если вы изменили значение ERROR_QA или WARN_QA и по-прежнему хотите выполнять эту проверку, следует добавить значение version-going-backwards переменной (см. параграф 6.56. insane.bbclass).

4.4.5. Изменение схемы каталогов

- Выходные файлы установщика SDK имеют имена, включающие имя образа и архитектуру через переменную SDK_NAME.
- Образы и связанные с ними файлы устанавливаются в каталог, определяемый машиной, а не в родительский каталог с выходными файлами для разных машин. Переменная DEPLOY_DIR_IMAGE по-прежнему указывает каталог с образами для конкретного значения MACHINE и её следует применять везде, где нужно указывать этот каталог. Сценарий runqemu теперь использует эту переменную для поиска образов и двоичных файлов ядра и будет применять BitBake для определения каталога. Дополнительно можно установить переменную DEPLOY_DIR_IMAGE во внешней среде.
- При включённом классе buildhistory его вывод будет записываться в каталог сборки, а не в TMPDIR. Это позволяет удалить TMPDIR с сохранением истории. Кроме того, данные для создаваемых SDK разделены по IMAGE_NAME.
- Данные pkgdata, создаваемые в процессе упаковки, сжаты до одного каталога, определяемого машиной. Этот каталог размещается в sysroots и использует определяемое машиной имя (tmp/sysroots/*machine*/pkgdata).

4.4.6. Сокращённые значения Git SRCREV

BitBake сокращает выпуски из репозиториев Git в переменной SRCPV с обычных 40 до 10 для удобочитаемости в именах файлов и каталогов. Это изменение не должно создавать проблем в контексте, где применяются эти выпуски, поскольку вероятность конфликтов очень мала. Совпадение значений в разном контексте проблем не вызывает.

4.4.7. IMAGE_FEATURES

- Значение IMAGE_FEATURES сейчас проверяется, чтобы предотвратить добавление непригодных элементов. Некоторые пользователи ошибочно добавляют имена пакетов в эту переменную вместо использования IMAGE_INSTALL для добавления пакетов в образ. Пригодное значение IMAGE_FEATURES выводится из определений PACKAGE_GROUP, COMPLEMENTARY_GLOB и нового флага validitems в IMAGE_FEATURES. Изменение validitems позволяет добавить свойства если они не включены двумя предыдущими механизмами.
- Ранее отменённый элемент IMAGE_FEATURES apps-console-core больше не поддерживается. Если нужно включить экран заставки, добавьте splash в переменную IMAGE_FEATURES (это все, что делал элемент apps-console-core).

4.4.8. /run

Был добавлен каталог /run в соответствии с Filesystem Hierarchy Standard 3.0. Последствия этого описаны на сайте. Изменение также привело к изменению заданий, устанавливающих файлы в каталог (рекомендации по внесению изменений доступны по ссылке.

4.4.9. Удаление базы данных менеджера пакетов из заданий для образов

Образ core-image-minimal больше не добавляет remove_packaging_data_files в переменную ROOTFS_POSTPROCESS_COMMAND. Это добавление происходит автоматически при отсутствии package-management в IMAGE_FEATURES. Если у вас есть задания с таким добавлением, нужно удалить соответствующие строки, поскольку они больше не нужны и могут создавать проблемы при выполнении пост-установочных сценариев.

4.4.10. Повторная сборка образов только при наличии изменений

Задача do_rootfs и другие задачи, связанные с созданием образов, больше не помечаются как nostamp, поэтому будут выполняться повторно лишь при наличии реальных изменений на их входе. В прежних выпусках система OE всегда заново собирала образ по запросу, е не по необходимости.

4.4.11. Задания для задач

Ранее отменённый класс task.bbclass удалён. Для заданий, наследовавших этот класс следует заменить task-* на packagegroup-* и наследовать класс packagegroup.

4.4.12. BusyBox

Пакет BusyBox разделен на два двоичных исполняемых файла, один из которых использует для тех компонент, где это требуется, а второй служит для остальных задач. Такое разделение сделало возможной оптимизацию, которая исключает задание tinylogin, как было рекомендовано разработчиками. Расщепление можно отключить, установив значение 0 для переменной BUSYBOX_SPLIT_SUID.

4.4.13. Автоматизированное тестирование образов

Добавлена новая модель автоматизированного тестирования образа с использованием класса testimage.bbclass взамен старой схемы imagetest-qemu. Автоматизированное тестирование образов рассмотрено в разделе Performing Automated Runtime Testing [2].

4.4.14. История сборки

- В файле installed-package-sizes.txt сейчас записываются размеры установленных файлов каждого пакета, а не размер сжатого архива.
- В графах зависимостей (depends*.dot) используются реальные имена пакетов без замены символов дефиса, точек, знаков + и символов подчёркивания.
- В утилитах buildhistory-diff и buildhistory-collect-srcrevs улучшена обработка параметров командной строки, которые можно посмотреть с помощью опции --help.

Работа с историей сборок описана в разделе Maintaining Build Output Quality [2].

4.4.15. udev

- udev больше не приводит автоматически в udev-extraconf через переменную RRECOMMENDS, поскольку это изначально предполагалось необязательным. Если нужны дополнительные правила, используйте udev-extraconf для своего образа.
- udev больше не приводит в pciutils-ids или usbutils-ids через переменную RRECOMMENDS. Они не нужны и их удаление экономит около 350 кбайт.

4.4.16. Удалённые и переименованные задания

- Удалено ядро linux-yocto 3.2.
- libtool-nativesdk переименовано с nativesdk-libtool.
- Удалено tinylogin с заменой на suid-часть Busybox (см. параграф 4.4.12. BusyBox).
- external-python-tarball переименован в buildtools-tarball.
- web-webkit удалён с заменой на midori.
- imake удалено за ненадобностью.
- transfig-native удалено за ненадобностью.
- anjuta-remote-run удалено. Интеграция с Anjuta IDE не поддерживается уже несколькими выпусками.

4.4.17. Прочие изменения

- run-postinsts является базовым.
- Из base-files удалены ненужные каталоги.
- alsa-state обеспечивает по умолчанию пустой файл asound.conf.
- classes/image гарантирует для BAD_RECOMMENDATIONS заранее переименованных пакетов.
- classes/rootfs_rpm реализует BAD_RECOMMENDATIONS для RPM.
- systemd - удаляется systemd_unitdir, если systemd нет в DISTRO_FEATURES.
- systemd - удаляется каталог init.d, если имеется файл инициализации systemd и sysvinit не является свойством дистрибутива.
- libpam отвергает все службы для записей OTHER.
- image.bbclass - перемещено runtime_mapping_rename для предотвращения конфликтов с multilib (см. YOCTO #4993).
- linux-dtb использует систему сборки ядра для генерации файлов dtb.
- kern-tools переключен с guilt на новый инструмент kgit-s2q.

4.5. Переход на YP 1.6

4.5.1. Класс archiver

Класс archiver был переписан с упрощением конфигурации. См. раздел Maintaining Open Source License Compliance During Your Product's Lifecycle [2].

4.5.2. Изменения в пакетах

- Задание binutils больше не создает пакет binutils-symlinks и применяется update-alternatives при обработке предпочтительного варианта binutils для целевой платформы.

- Утилиты tc (управление трафиком) выделены из пакета iproute2 и помещены в пакет iproute2-tc.
- Схемы gtk-engines перенесены в отдельный пакет gtk-engines-schemas.
- Изменён суффикс архитектуры для пакетов armv7a, который теперь для пакетов с оптимизацией имеет значение t2, как и должно быть. Этот суффикс не применялся в выпуске 1.5. Результатом является изменение имён архитектуры для пакетов.

4.5.3. BitBake

В следующих параграфах рассмотрены изменения в BitBake.

4.5.3.1. Требование к соответствию ветвей для сборщика Git

При извлечении исходных кодов из репозитория Git с использованием переменной SRC_URI программа BitBake будет сравнивать значение SRCREV с ветвью. Для указания ветви можно использовать приведённую ниже форму.

```
SRC_URI = "git://server.name/repository;branch=branchname"
```

Если ветвь не задана, BitBake будет обращаться к принятой по умолчанию ветви master.

Если нужно обойти проверку (например, при извлечении версии по тегу, не относящемуся к ветвям), можно добавить ";nobranch=1" в конце URL в переменной SRC_URI.

4.5.3.2. Подстановки определений Python

BitBake включал некоторые устаревшие определения Python в удалённом модуле bb, вместо которых следует применять указанные ниже аналоги.

- bb.MalformedUrl заменяется bb.fetch.MalformedUrl.
- bb.encodeurl заменяется bb.fetch.encodeurl.
- bb.decodeurl заменяется bb.fetch.decodeurl
- bb.mkdirhier заменяется bb.utils.mkdirhier.
- bb.movefile заменяется bb.utils.movefile.
- bb.copyfile заменяется bb.utils.copyfile.
- bb.which заменяется bb.utils.which.
- bb.vercmp_string заменяется bb.utils.vercmp_string.
- bb.vercmp заменяется bb.utils.vercmp.

4.5.3.3. Сборщик SVK

Сборщик SVK исключён из BitBake.

4.5.3.4. Перенаправление консольного вывода ошибок

Консольный интерфейс BitBake будет направлять ошибки вывода на устройство stderr вместо stdout. Поэтому при использовании конвейеров, перенаправляющих вывод bitbake, и желании видеть ошибки, нужно добавить 2>&1 (или что-то похожее) в конце командной строки bitbake.

4.5.3.5. Переопределения task-taskname

Переопределения task-taskname скорректированы так, что задачи, чьи имена содержат символы подчёркивания были переопределены в заменой этих символов дефисами для обеспечения корректной работы. Например, задача do_populate_sdk переопределена как task-populate-sdk.

4.5.4. Изменения переменных

Ниже указаны изменённые переменные. Описаниям переменных системы сборки посвящена Глава 13. Переменные.

4.5.4.1. TMPDIR

Переменная TMPDIR больше не может указывать файловые системы NFS, поскольку они не обеспечивают блокировки OSI и согласованности inode, что может вызывать проблемы. Проверка выполняется при запуски и выводится сообщение об ошибке, если TMPDIR указывает NFS.

4.5.4.2. INC

Переменная INC устарела и будет вызывать предупреждения в процессе сборки. Для инкрементирования при изменениях следует использовать переменную PR. Дополнительная информация приведена в разделе Working With a PR Service [2].

4.5.4.3. IMAGE_FSTYPES

Опция sum.jffs2 для IMAGE_FSTYPES заменена опцией jffs2.Hu, с сохранением порядка обработки.

4.5.4.4. COPY_LIC_MANIFEST

Для включения переменной COPY_LIC_MANIFEST теперь используется значение 1.

4.5.4.5. COPY_LIC_DIRS

Для включения переменной COPY_LIC_DIRS теперь используется значение 1.

4.5.4.6. PACKAGE_GROUP

Переменная PACKAGE_GROUP переименована в FEATURE_PACKAGES, чтобы точнее отражать её назначение. Переменную PACKAGE_GROUP можно использовать, но система сборки OE будет выдавать предупреждения.

4.5.4.7. Поведение команд предварительной и пост-обработки

Указанные ниже команды ожидают списка разделенных точками с запятой переменных, а не произвольных команд.

```
ROOTFS_PREPROCESS_COMMAND
ROOTFS_POSTPROCESS_COMMAND
SDK_POSTPROCESS_COMMAND
POPULATE_SDK_POST_TARGET_COMMAND
POPULATE_SDK_POST_HOST_COMMAND
IMAGE_POSTPROCESS_COMMAND
IMAGE_PREPROCESS_COMMAND
ROOTFS_POSTUNINSTALL_COMMAND
ROOTFS_POSTINSTALL_COMMAND
```

Для перехода можно просто «обернуть» команды в функции командного процессора (shell) и затем вызывать функции, как показано ниже.

```
my_postprocess_function() {
   echo "hello" > ${IMAGE_ROOTFS}/hello.txt
}
ROOTFS_POSTPROCESS_COMMAND += "my_postprocess_function; "
```

4.5.5. Тестирование пакетов (ptest)

Тестирование пакетов (ptest) встроено, но по умолчанию не устанавливается. Работа с тестами описана в разделе Testing Packages with ptest [2], а класс ptest - в параграфе 6.104. ptest.bbclass.

4.5.6. Изменение сборки

Раздельные каталоги для сборки и исходных кодов разрешены по умолчанию для отдельных заданий, где это известно («белый список»), а также для всех заданий, наследующих класс cmake. В будущих выпусках класс autotools будет разрешать также отделение каталога сборки по умолчанию. Программы сборки заданий на основе autotools, которые не могут работать с отдельным каталогом сборки следует изменить, чтобы они могли наследовать класс autotools-brokensep вместо autotools или autotools_stageclasses.

4.5.7. qemu-native

Задание qemu-native сейчас собирается по умолчанию без графического интерфейса на базе SDL. Для включения графического интерфейса в файле local.conf должны быть приведённые ниже строки[1].

```
PACKAGECONFIG_pn-qemu-native = "sdl"
ASSUME_PROVIDED += "libsdl-native"
```

4.5.8. core-image-basic

Образ core-image-basic переименован в core-image-full-cmdline, а группа packagegroup-core-basic - в packagegroup-core-full-cmdline.

4.5.9. Лицензирование

Файл LICENSE верхнего уровня изменён для более точного описания лицензий различных компонент OE-Core без изменения самого лицензирования. Обычно это изменение не вызывает побочных эффектов. Однако некоторые задания указывают этот файл в LIC_FILES_CHKSUM (как ${COREBASE}/LICENSE), поэтому нужно изменить сопровождающую контрольную сумму 3f40d7994397109285ec7b81fdeb3b58 на 4d92cd373abda3937c2bc47fbc49d690. Лучше указывать в LIC_FILES_CHKSUM файл, описывающий лицензию, распространяемую с исходным кодом для сборки задания, а не ${COREBASE}/LICENSE.

4.5.10. Опции CFLAGS

Опция -fpermissive удалена из принятого по умолчанию значение CFLAGS. Это нужно учитывать в заданиях, где возникает отказ при сборке без этой опции - задания нужно изменить для устранения ошибок, сообщаемых компилятором, или добавить опцию -fpermissive в переменную CFLAGS для этого задания.

4.5.11. Типы вывода пользовательских образов

Пользовательские типы образов, указанные в переменной IMAGE_FSTYPES, должны объявлять свои зависимости (при наличии) через новую переменную IMAGE_TYPEDEP.

4.5.12. Задачи

Задача do_package_write удалена за ненадобностью.

4.5.13. Поставщик update-alternative

Принятый по умолчанию поставщик update-alternatives сменен с opkg на opkg-utils, что позволило решить некоторые проблемы с закольцованными зависимостями. Используемый при работе пакет (runtime) update-alternatives-cworth переименован в update-alternatives-opkg.

[1] В установленном по умолчанию файле local.conf эти строки присутствуют, поэтому при сборке образов для систем без монитора следует их закомментировать.

4.5.14. Переопределение virtclass

Переопределения virtclass устарели и взамен используются эквиваленты (например, virtclass-native вместо class-native).

4.5.15. Удалённые и переименованные задания

- packagegroup-toolset-native больше не применяется.
- linux-yocto-3.8 - поддержка ядра Linux yocto 3.8 исключена, ядра 3.10 и 3.14 добавлены как задания linux-yocto-3.10 и linux-yocto-3.14.
- ocf-linux функционально заменено заданием cryptodev-linux.
- genext2fs больше не используется системой сборки и не поддерживается в восходящих разработках.
- js - древняя версия движка Mozilla javascript, которая больше не применяется.
- zaurusd перенесено на уровень meta-handheld.
- eglibc 2.17 заменено заданием eglibc 2.19.
- gcc 4.7.2 заменено новым стабильным gcc 4.8.2.
- external-sourcery-toolchain поддерживается на уровне meta-sourcery.
- linux-libc-headers-yocto 3.4+git — сейчас по умолчанию применяется the linux-libc-headers версии 3.10.
- meta-toolchain-gmae отменено.
- packagegroup-core-sdk-gmae отменено.
- packagegroup-core-standalone-gmae-sdk-target отменено.

4.5.16. Удалённые классы

- module_strip;
- pkg_metainfo;
- pkg_distribute;
- image-empty.

4.5.17. Эталонные BSP

- Эталонный пакет BeagleBoard ARM (beagleboard) заменён BeagleBone (beaglebone).
- Эталонный пакет RouterStation Pro MIPS (routerstationpro) заменён EdgeRouter Lite (edgerouter).

Прежние эталонные BSP для машин beagleboard и routerstationpro сохранены на уровне meta-yocto-bsp-old в дереве исходных кодов репозитория http://git.yoctoproject.org/cgit/cgit.cgi/meta-yocto-bsp-old/.

4.6. Переход на YP 1.7

4.6.1. Изменение опций QEMU PACKAGECONFIG в файле local.conf

Задание QEMU сейчас применяет множество опций PACKAGECONFIG для управления дополнительными возможностями. Используемый для задания принятых по умолчанию значений этих опций требует изменения имеющегося файла local.conf с целью добавления PACKAGECONFIG для qemu-native и nativesdk-qemu вместо их установки. Иными словами, для добавления графического вывода в QEMU следует включить в файл local.conf приведённые ниже строки

```
PACKAGECONFIG_append_pn-qemu-native = " sdl"
PACKAGECONFIG_append_pn-nativesdk-qemu = " sdl"
```

4.6.2. Минимальная версия Git

Сейчас требуется версия Git не ниже 1.7.8, поскольку сборщику Git нужна опция --list. Как обычно, если сборочный хост не соответствует требованиям, можно загрузить и установить архив buildtools (см. параграф 1.3. Требуемые версии Git, tar и Python.

4.6.3. Изменение класса autotools

- *По умолчанию применяется отдельный каталог сборки.* Класс autotools изменён для работы с каталогом сборки (B), отделенным от дерева исходных кодов (S). Это указывается как B != S или сборка вне дерева.

  Если собираемая программа уже позволяет сборку вне дерева, ничего изменять не нужно. Если же программа не поддерживает такой сборки, нужно её исправить соответствующим образом или изменить задание для наследования класса autotools-brokensep вместо autotools или autotools_stage.

- *Опция --foreign больше не передаётся automake при работе autoconf.* Эта опция говорит automake, что определённый пакет не соответствует стандартам GNU и поэтому не следует ожидать распространения в нем таких файлов как ChangeLog, AUTHORS и т. п. Поскольку большинство программ уже говорят automake, что они сами включают режим foreign, эта опция стала ненужной. Однако некоторые задания придётся всё-таки изменить. Это легко сделать, указав в файле configure.ac передачу foreign в AM_INIT_AUTOMAKE().

4.6.4. Отмена сценариев настройки конфигурации

В некоторых заданиях, упаковывающих двоичные сценарии настройки, эти сценарии отключены по причине возникновения ошибок при подстановке путей. Программам, связанным с библиотеками, использующими такие сценарии, следует применять более надёжный пакет pkg-config. Список заданий, изменённых в этой версии (и их сценариев настройки) включает directfb (directfb-config), freetype (freetype-config), gpgme (gpgme-config), libassuan (libassuan-config), libcroco (croco-6.0-config), libgcrypt (libgcrypt-config), libgpg-error (gpg-error-config), libksba (ksba-config), libpcap (pcap-config), libpcre (pcre-config), libpng (libpng-config, libpng16-config), libsdl (sdl-config), libusb-compat (libusb-config), libxml2 (xml2-config), libxslt (xslt-config), ncurses (ncurses-config), neon (neon-config), npth (npth-config), pth (pth-config), taglib (taglib-config). В дополнение к этому была добавлена поддержка pkg-config в некоторые задания из этого списка, если пакет ещё не включал её.

4.6.5. Замена eglibc 2.19 на glibc 2.20

Поскольку eglibc и glibc уже довольно близки, замена не должна потребовать существенных изменений в программах, связанных с eglibc. Однако в glibc 2.20 внесено множество мелких изменений, которые могут потребовать исправлений в ряде программ (например, удаление макроса тестирования свойств _BSD_SOURCE). Для glibc 2.20 требуется ядро Linux версии 2.6.32 или выше. Дополнительная информация об изменениях в glibc 2.20 доступна по ссылке.

4.6.6. Автоматическая загрузка модулей ядра

Переменные module_autoload_* отменены и вместо них следует использовать новую переменную KERNEL_MODULE_AUTOLOAD. Переменные module_conf_* должны сейчас применяться вместе с новой переменной KERNEL_MODULE_PROBECONF. Эти новые переменные не требуют указания имени модуля как части имени переменной, что не только упрощает использование, но и позволяет аккуратно встраивать значения этих переменных в подписи задач и активизировать повторное выполнение задач при появлении изменений. Следует заменить все включения module_autoload_* на KERNEL_MODULE_AUTOLOAD и добавить все модули, для которых заданы переменные module_conf_*, в KERNEL_MODULE_PROBECONF.

4.6.7. Изменения проверки качества (QA)

- Добавлены проверки file-rdeps и build-deps для контроля соблюдения зависимостей (например, пакетов, содержащих сценарии, которым требуется /bin/bash) и корректности указания зависимостей при сборке (Глава 10. Ошибки и предупреждения тестов QA).
- Проверка качества пакетов выполняется новой задачей do_package_qa, а не do_package task. Это изменение не должно вызывать проблем за исключением сложных пользовательских заданий, которые отключают задачу упаковки, помечая её как noexec. Для таких пакетов нужно отключать задачу do_package_qa.
- Файлы, переписанные при выполнении задачи do_populate_sysroot, теперь вызывают ошибку, а не предупреждение. Заданиям не следует переписывать файлы, записанные в sysroot другими заданиями, и при наличии таких заданий их следует должным образом изменить.

  Сообщение о такой ошибке может появиться в результате изменения конфигурации или метаданных, приводящего к появлению потерянных файлов в sysroot. При возникновении такой ошибки для её устранения можно удалить или переименовать каталог TMPDIR, а затем повторить сборку. Все, что было собрано до возникновения ошибки, сохранится в кэше состояний и не будет собираться заново.

4.6.8. Удалённые задания

- *x-load* заменено U-boot SPL для всех Cortex TI SoC. Для более старых плат следует использовать уровень meta-ti.
- *ubootchart* устарело. Добавлено задание bootchart2, служащее функциональной заменой.
- *Linux-yocto 3.4* - поддержка ядра linux-yocto 3.4 прекращена, поддержка ядер 3.10 и 3.14 сохранена, добавлено ядро 3.17.
- *eglibc* заменено на glibc (см. параграф 4.6.5. Замена eglibc 2.19 на glibc 2.20).

4.6.9. Прочие изменения

Функции записи истории сборки теперь создает вместо build-id файл build-id.txt, который включает полный заголовок сборки, выводимый BitBake при старте. Имеет смысл удалить старые файлы build-id из имеющихся репозиториев сборки для предотвращения путаницы. Функция истории сборки описана в разделе Maintaining Build Output Quality [2].

4.7. Переход на YP 1.8

4.7.1. Удалённые задания

- *owl-video* функционально заменено gst-player.
- *gaku* функционально заменено gst-player.
- *gnome-desktop* сейчас доступно в meta-gnome и отдельное задание не нужно.
- *gsettings-desktop-schemas* сейчас доступно в meta-gnome и отдельное задание не нужно.
- *python-argparse* - модуль уже предоставляется в используемом по умолчанию дистрибутиве Python в пакете python-argparse, поэтому отдельное задание уже не нужно.
- *telepathy-python, libtelepathy, telepathy-glib, telepathy-idle, telepathy-mission-control* перенесены с meta-oe и поэтому больше не нужны заданиям из OpenEmbedded-Core.
- *linux-yocto_3.10 и linux-yocto_3.17* - поддержка linux-yocto 3.10 и 3.17 прекращена. Ядро 3.14 поддерживается и добавлено ядро 3.19.

- *poky-feed-config-opkg* устарело и больше не требуется. Вместо него применяется distro-feed-config их meta-oe.
- *libav 0.8.x* - сейчас используется libav 9.x.
- *sed-native* больше не нужно. Предполагается обеспечение рабочей версии sed дистрибутивом хоста.

4.7.2. Выбор BlueZ 4.x или 5.x

имеется встроенная поддержка BlueZ 5.x предпочтительная по сравнению с используемой по умолчанию 4.x. Для использования BlueZ 5.x достаточно добавить bluez5 в переменную DISTRO_FEATURES. Если ранее выбор был включён в файл добавления (*.bbappend), его можно удалить. Кроме того, был добавлен класс bluetooth для более удобного выбора подходящей поддержки bluetooth в задании. Для использования этого класса в задании добавьте строки по приведённому ниже образцу.

```
inherit bluetooth
PACKAGECONFIG ??= "${@bb.utils.contains('DISTRO_FEATURES', 'bluetooth', '${BLUEZ}', '', d)}
PACKAGECONFIG[bluez4] = "--enable-bluetooth,--disable-bluetooth,bluez4"
PACKAGECONFIG[bluez5] = "--enable-bluez5,--disable-bluez5,bluez5"
```

4.7.3. Изменение сборки ядра

Процесс сборки ядра был изменён, чтобы поместить исходные коды в общую рабочую область и отделить элементы сборки (artifact) в дереве исходных кодов. В теории пути перехода были предоставлены для наиболее распространённых вариантов заданий для сборки ядер, но это может работать не во всех случаях. В частности нужно обеспечить корректность использования переменных ${S} (исходные коды) и ${B} (элементы сборки) в таких функциях, как do_configure и do_install. Для заданий, не наследующих kernel-yocto и не включающих linux-yocto.inc, можно использовать файл linux.inc уровня meta-oe для внесения всех требуемых изменений. Для справки можно воспользоваться ссылкой на обновление файла linux.inc для уровня meta-oe.

Задания, основанные на исходном коде ядра и не наследующие класс module, могут требовать явного указания зависимостей в задаче ядра do_shared_workdir. Например, do_configure[depends] += "virtual/kernel:do_shared_workdir".

4.7.4. Запрет SSL 3.0 в OpenSSL

SSL 3.0 сейчас отключён при сборке OpenSSL. Это позволяет избежать уязвимости POODLE. Если нужно всё-таки включить SSL 3.0, это можно сделать в файле добавления (*.bbappend) для задания openssl, удалив -no-ssl3 из переменной EXTRA_OECONF.

4.7.5. Изменение принятого по умолчанию каталога sysroot

Принятые по умолчанию в gcc каталоги sysroot и include перенаправлены в несуществующее место, чтобы отследить использование каталогов хоста в результате отсутствия корректно переданных опций. Это применяется для кросс-компиляторов, используемых при сборке, и создаваемых в SDK. Если изменение вызывает отказ при сборке, это может означать, что флаги и команды компилятора были некорректно переданы базовым программам. В таких случаях нужно внести соответствующие правки.

4.7.6. Улучшение повторной сборки

Изменены классы base, autotools и cmake для очистки созданных файлов при необходимости повтора do_configure. Одним из улучшения является попытка выполнить команду make clean в задаче do_configure, если имеется файл Makefile. Некоторые программы не поддерживают очистки в своих файлах make. Для таких заданий нужно установить CLEANBROKEN = "1".

4.7.7. Изменения в проверках QA

- Использование INC ранее вызывало предупреждения, а сейчас приводит к ошибке. Следует удалить использование INC из всех заданий и файлов добавления.
- Добавлена проверка QA использования ${D} в значениях FILES там, где применять D не следует. Эта проверка гарантирует использование $D в функциях pkg_preinst, pkg_postinst, pkg_prerm, pkg_postrm вместо ${D}.
- В переменной S нужно устанавливать действительное для задания значение. Если переменная S в задании не установлена, каталог автоматически не создаётся. Если S не указывает каталог, существующий к моменту завершения задачи do_unpack, выдаётся предупреждение.
- Проверяется переменная LICENSE на предмет корректности форматирования множества лицензий. Если формат не пригоден (например, указано множество лицензий без оператора, определяющего их взаимодействие), выдаётся предупреждение.

4.7.8. Прочие изменения

- Сценарий send-error-report ожидает опцию -s перед адресом сервера (это предполагает указание сервера).
- Сценарий oe-pkgdata-util ожидает опции -"перед каталогом pkgdata, который сейчас можно не указывать. При отсутствии pkgdata сценарий будет запускать BitBake для запроса PKGDATA_DIR в среду сборки.

4.8. Переход на YP 2.0

4.8.1. GCC 5

По умолчанию используется компилятор GCC 5.2, потребовавшийся для устранения ошибок компиляции во многих заданиях. Одним из важных примеров служит ситуация, когда система «зависала» при сборке ядра Linux для ARM с компилятором GCC 5. При использовании своих заданий, собирающих ядро для ARM, вам может потребоваться внести исправление. В стандартном дереве ядра linux-yocto эти исправления уже внесены. Дополнительная информация

приведена на странице https://gcc.gnu.org/gcc-5/changes.html, рекомендации по переносу - на странице https://gcc.gnu.org/gcc-5/porting_to.html.

Можно вернуться к использованию GCC версии 4.9 или 4.8 с помощью строки вида GCCVERSION = "4.9%" в файле конфигурации.

4.8.2. Удаление Gstreamer 0.10

Поддержка Gstreamer 0.10 была заменена поддержкой Gstreamer 1.x. В результате этого изменения задания для Gstreamer 0.10 и связанных программ размещены сейчас в meta-multimedia. Это ведёт к тому, что в Qt4 поддержка Phonon и Gstreamer до умолчанию отключена для QtWebkit.

4.8.3. Удалённые задания

- *bluez4* устарело и перемещено в meta-oe в результате полной интеграции с bluez5.
- *gamin* устарело и удалено.
- *gnome-icon-theme* функционально заменено заданием adwaita-icon-theme.
- *Задания Gstreamer 0.10* удалены в пользу поддержки заданий для Gstreamer 1.x.
- *insserv* устарело и удалено.
- *libunique* больше не используется и перенесено в meta-oe.
- *midori* функционально заменено заданием epiphany.
- *python-gst* устарело и было удалено, поскольку требовалось только для Gstreamer 0.10.
- *qt-mobility* устарело и было удалено, поскольку требует наличия Gstreamer 0.10 (удалено).
- *subversion* удалены версии 1.6.x этого задания.
- *webkit-gtk* старая версия 1.8.3 удалена с заменой на webkitgtk.

4.8.4. Улучшение хранилища данных BitBake

Изменён метод обработки переопределений хранилищем BitBake. Переопределения сейчас применяются динамически, а bb.data.update_data() не используется. На практике это изменение вряд ли потребует какого-либо изменения метаданных. Однако имеются незначительные изменения в поведении, указанные ниже.

- Все возможные переопределения теперь доступны в истории по команде bitbake -e.
- d.delVar('VARNAME') и d.setVar('VARNAME', None) приводят к очистке переменной и всех её переопределений. До изменения очищались лишь не переопределенные значения.

4.8.5. Смена поведения shell-функций

Консольные (shell) версии функций сообщений BitBake (bbdebug, bbnote, bbwarn, bbplain, bberror, bbfatal) связаны с их эквивалентами в BitBake - bb.debug(), bb.note(), bb.warn(), bb.plain(), bb.error(), bb.fatal(), соответственно. Таким образом, функции сообщений, ожидаемые в пользовательском интерфейсе BitBake UI, будут выводиться фактически. На практике это изменение имеет два проявления, описанных ниже.

- Если на консоли появляются сообщения, которых ранее (до изменения) не было, может потребоваться очистка вызовов bbwarn, bberror и т. п. Можно просто удалить эти вызовы.
- Функция bbfatal подавляет полный вывод журнала ошибок в UI, поэтому при необходимости видеть этот вывод полностью нужно заменить bbfatal на die или bbfatal_log.

4.8.6. Очистка дополнительных пакетов для разработки и отладки

Удалены использованные при разработке и отладке задания для acl, apmd, aspell, attr, augeas, bzip2, cogl, curl, elfutils, gcc-target, libgcc, libtool, libxmu, opkg, pciutils, rpm, sysfsutils, tiff, xz. Для всех перечисленных заданий сейчас применяется стандартная схема, где для задания имеется один пакет -dev, -dbg, -staticdev.

4.8.7. Перенос данных отслеживания заданий в OE-Core

Поддержка данных отслеживания для заданий, которые были частью meta-yocto, перенесена на уровень OE-Core. Это относится к файлам package_regex.inc и distro_alias.inc, которые обычно включаются при использовании класса distrodata. Кроме того, содержимое файла upstream_tracking.inc теперь разделено по заданиям.

4.8.8. Автоматическая очистка устаревших файлов в sysroot

Файлы из заданий, которых уже нет в текущей конфигурации, автоматически удаляются из sysroot и других мест, управляемых общим состоянием. Эта автоматическая очистка означает, что система сборки корректно обрабатывает такие ситуации, как переименования сборочной стороны заданий, удаление уровней из bblayers.conf и изменение DISTRO_FEATURES.

В дополнение к этому очищаются каталоги старых версий заданий. Если нужно отключить такую очистку, можно установить в конфигурации переменную SSTATE_PRUNE_OBSOLETEWORKDIR = "0".

4.8.9. Репозиторий метаданных linux-yocto отделен от исходного кода

Дерево linux-yocto до этого представляло набор изменений в ядре и конфигурации (метаданных). Хотя такой формат эффективен для синхронизации конфигураций ядра и изменений в исходных кодах, разработчикам не всегда понятно, как работать с метаданными применительно к исходным кодам.

Обработка метаданные перенесена из класса kernel-yocto и применяется внешний репозиторий метаданных yocto-kernel-cache, который всегда использовался для заполнения ветви meta в linux-yocto. Отдельный репозиторий кэша linux-yocto сейчас служит основным местом хранения для этих данных. В результате linux-yocto больше не может обрабатывать комбинированные деревья. Поэтому при необходимости иметь свой комбинированный репозиторий ядра нужно будет разделить его и соответственно обновить свои задания. Примером может служить задание meta/recipes-kernel/linux/linux-yocto_4.1.bb.

4.8.10. Дополнительные проверки QA

- Добавлена проверка host-user-contaminated для контроля вопросов владения файлами за пределами /home. Проверка ищет файлы, некорректно принадлежащие пользователю, запустившему BitBake, а не предусмотренному пользователю в целевой системе.
- Добавлена проверка invalid-chars для обнаружения недействительных (не-UTF8) символов в значениях переменных метаданных задания (DESCRIPTION, SUMMARY, LICENSE, SECTION), поскольку некоторые менеджеры пакетов не поддерживают такие символы.
- Добавлена проверка invalid-packageconfig опций PACKAGECONFIG на соответствие опциям задания.

4.8.11. Прочие изменения

- gtk-update-icon-cache переименовано в gtk-icon-utils.
- Элемент tools-profile в IMAGE_FEATURES, а также соответствующие packagegroup и packagegroup-core-tools-profile больше не включаются в oprofile. Внедрение в oprofile изначально было предназначено для облегчения компиляции для платформ с ограниченными ресурсами. Однако это не получило широкого распространения и вряд ли будет применяться в будущем по причине расширения возможностей платформ и усовершенствования инструментов кросс-компиляции.
- Принятое по умолчанию значение переменной IMAGE_FSTYPES включает ext4 вместо ext3.
- Удалена поддержка переменной PRINC.
- Группа пакетов packagegroup-core-full-cmdline больше не используется в lighttpd по причине того, что это не соответствует назначению packagegroup, которое заключается в добавлении консольных инструментов, которые по умолчанию обеспечиваются busybox.

4.9. Переход на YP 2.1

4.9.1. Преобразование переменных в функциях Python

Выражения для переменных вида ${VARNAME} больше не преобразуются автоматически в функциях Python. Это сделано для того, чтобы позволить функциям Python создавать shell-сценарии и другой код для ситуаций, где не нужны такие преобразования. В имеющемся коде, который применяет такие преобразования, нужно изменить их для преобразования отдельных переменных с помощью d.getVar() или d.expand() для более сложных выражений.

4.9.2. Нижний регистр для переменной OVERRIDES

В соглашении о переопределениях всегда предполагались символы нижнего регистра. Это стало сейчас требованием, поскольку хранилище BitBake предполагает нижний регистр символов для некоторого ускорения процесса анализа. На практике это означает, что все завершающееся в OVERRIDES должно указываться символами нижнего регистра (например, значения MACHINE, TARGET_ARCH, DISTRO, а также имена заданий, если действуют переопределения _pn-recipename).

4.9.3. Параметр раскрытия функций getVar() и getVarFlag() стал обязательным

Параметр раскрытия для функций getVar() и getVarFlag() ранее по умолчанию имел значение False, однако теперь этого нет и параметр нужно задавать. Требуется заменить все вызовы getVar(), где параметр не был указан, вызовами с заданным параметром. Ниже приведен пример команды, позволяющей автоматизировать эту замену.

```
sed -e 's:\(\.getVar([^,()]*\)):\1, False):g' -i `grep -ril getVar *`
sed -e 's:\(\.getVarFlag([^,()]*, [^,()]*\)):\1, False):g' -i `grep -ril getVarFlag *`
```

Причина этого изменения заключается в том, что в будущих выпусках YP планируется по умолчанию задавать значение True. Однако изменение требуется вносить постепенно, поскольку внезапная смена может вызвать сложные в обнаружении побочные эффекты.

4.9.4. Изменение окружения Makefile

Переменная EXTRA_OEMAKE сейчас имеет по умолчанию значение "" вместо прежнего "-e MAKEFLAGS=", что было исторической случайностью, требовавшей переопределения во множестве классов (например, autotools, module) и заданий. При обновлении на этот выпуск нужно отредактировать все задания, основанные на прежнем значении, путём возврата "-e MAKEFLAGS=" или явной установки нужного значения EXTRA_OEMAKE, что обычно требуется только в случаях установки Makefile по умолчанию не подходящего для кросс-компиляции значения переменной с использованием оператора =, а не ?=.

4.9.5. Возвращение libexecdir к ${prefix}/libexec

Использование ${libdir}/${BPN} в качестве libexecdir отличается от всех основных дистрибутивов, применяющих ${prefix}/libexec или ${libdir}, и противоречит стандартам кодирования GNU, задающим использование ${prefix}/libexec, а также практике задания используемой пакетом вложенности в самом пакете. Кроме того, различия libexecdir в разных заданиях осложняют этим заданиям вызов двоичных файлов, установленных в libexecdir. Стандарт FHS[1] признает использование ${prefix}/libexec/, предоставляя приложениям выбор между ${prefix}/lib и ${prefix}/libexec.

[1] Filesystem Hierarchy Standard - стандарт иерархии файловых систем.

4.9.6. ac_cv_sizeof_off_t больше не кэшируется в файлах сайта

Для заданий, наследующих класс autotools, значение ac_cv_sizeof_off_t больше не кэшируется в файлах сайта для autoconf. Причина этого заключается в том, что ac_cv_sizeof_off_t не обязательно является неизменным для архитектуры, как считалось ранее, и меняется в зависимости от поддержки больших файлов. Для большинства программ, использующих autoconf, внесённое изменение не создает проблем. Однако задания, которые обходят стандартную задачу do_configure из класса autotools, и программ использующих очень старые версии autoconf, задание может не определить корректный размер off_t в процессе выполнения задачи do_configure.

Лучшим вариантом действий является исправление программ при необходимости, чтобы принятая по умолчанию реализация из класса autotools работала так, чтобы выполнялась утилита autoreconf, создающая работоспособный сценарий configure, и переопределение задачи do_configure для использования принятой по умолчанию реализации.

4.9.7. Генерация образа отделена от генерации файловой системы

Ранее задача do_rootfs для образов собирала файловую систему и помещала в неё созданные образы. С этого выпуска YP генерация образов вынесена в отдельные задачи do_image_*, чтобы отделить операции от кода.

В большинстве случаев это изменение не вызовет проблем. Однако при наличии настроек, непосредственно меняющих задачу do_rootfs или или ссылку на неё, может потребоваться внесение изменений. В частности, при добавлении любых задач после do_rootfs следует изменить эти задачи так, чтобы они выполнялись после do_image_complete, а не после do_rootfs для соблюдения корректного порядка выполнения.

Ещё одна часть этого изменения заключается в переносе определений и функций пост-обработки из класса image в класс rootfs-postcommands. Функционально это ничего не меняет.

4.9.8. Удалённые задания

- *gcc версии 4.8* - версии 4.9 и 5.3 сохранены;
- *qt4* - поддержка Qt 4.x перенесена в отдельный уровень meta-qt4;
- *x11vnc* перенесено на уровень meta-oe;
- *linux-yocto-3.14*;
- *linux-yocto-3.19*;
- *libjpeg* заменено заданием libjpeg-turbo;
- *pth* признано устаревшим;
- *liboil* больше не нужно и перенесено на уровень meta-multimedia;
- *gtk-theme-torturer* больше не нужно и перенесено на уровень meta-gnome;
- *gnome-mime-data* больше не нужно и перенесено на уровень meta-gnome;
- *udev* заменено заданием eudev для совместимости при использовании sysvinit с новыми ядрами;
- *python-pygtk* признано устаревшим;
- *adt-installer* признано устаревшим (см. параграф 4.9.11. Удаление ADT).

4.9.9. Изменения классов

- *autotools_stage* удалён в результате обеспечения нужной функциональности классом autotools. Заданиям, наследовавшим autotools_stage сейчас нужно наследовать autotools.
- *boot-directdisk* слит с классом image-vm. Класс boot-directdisk применялся редко и изменение не должно вызывать проблем.
- *bootimg* слит с классом image-live. Класс bootimg применялся редко и изменение не должно вызывать проблем.
- *packageinfo* удалён, поскольку применялся лишь в удалённом интерфейсе Hob UI.

4.9.10. Изменения пользовательского интерфейса системы сборки

- *Интерфейс Hob UI на базе GTK+* удалён, поскольку больше не поддерживается и основан на устаревшей библиотеке GTK+ 2. Интерфейс Toaster UI на основе web более эффективен и активно поддерживается (см. раздел Using the Toaster Web Interface [8]).
- *Интерфейс "puccho" BitBake UI* удалён, поскольку больше не поддерживается и малополезен.

4.9.11. Удаление ADT

Пакет разработки приложений ADT[1] был удалён, поскольку его функциональность практически полностью перекрывалась со стандартным SDK и расширенным SDK [7]. Плагин Yocto Project Eclipse IDE не затронут изменением.

4.9.12. Изменения в эталонном дистрибутиве Poky

- Уровень meta-yocto переименован в meta-poky, что точнее отражает его назначение - эталонный дистрибутив Poky. Уровень meta-yocto-bsp сохранил своё имя, поскольку он содержит эталонные машины для YP и больше никак не связан с Poky. Ссылки на meta-yocto в файле conf/bblayers.conf должны обновиться автоматически.
- Класс uninative включён в Poky по умолчанию. Этот класс пытается изолировать систему сборки от библиотеки C дистрибутива хоста и делает возможным использование естественных элементов общего состояния

[1] Application Development Toolkit.

разными дистрибутивами хоста. При включённом класса загружается архив собранной библиотеки C в начале сборки. Класс uninative включается в файле meta/conf/distro/include/yocto-uninative.inc, что позволяет управлять им независимо от использования дистрибутива Poky.

Если нужно собрать свой архив uninative, это можно сделать с помощью задания uninative-tarball и сделать архив доступным для сборочных машин (например, по протоколу HTTP/HTTPS), а также установить похожую на yocto-uninative.inc конфигурацию.

- Генерация статических библиотек сейчас по умолчанию отключена в Poky для большинства классов. Это сократило время сборки и также размер области сборки. Запрет создания таких библиотек задан в файле meta/conf/distro/include/no-static-libs.inc. В задания, которым не нужна опция --disable-static в команде configure, следует включить переменную DISABLE_STATIC = "".
- В отдельном компактном дистрибутиве poky-tiny сейчас применяется библиотека musl C вместо сильно урезанной glibc. Использование musl уменьшает размер дистрибутива и упрощает поддержку. При использовании poky-tiny со своей конфигурацией glibc нужно будет изменить настройки в новом выпуске.

4.9.13. Изменения в пакетах

- Двоичные файлы runuser и mountpoint из пакета util-linux перенесены в util-linux-runuser и util-linux-mountpoint.
- Пакет python-elementtree объединён с пакетом python-xml.

4.9.14. Изменения в файлах настройки

- Функция настройки no-thumb-interwork удалена из включаемых файлов настройки ARM, поскольку для ARM EABI требуется взаимодействие, эта функция не имеет смысла.
- Отменена поддержка ARM OABI в gcc 4.7.
- Удалены файлы tune-cortexm*.inc и tune-cortexr4.inc, поскольку они не были достаточно протестированы. Пока система сборки OE не будет официально поддерживать CPU без MMU, эти файлы лучше держать на отдельном уровне, если они нужны.

4.9.15. Поддержка GObject Introspection

Этот выпуск поддерживает генерацию файлов GIR[1] с помощью самоанализа GObject, который является стандартным механизмом доступа к программам на основе GObject. Можно включать, отключать и тестировать генерацию таких данных, как описано в разделе Enabling GObject Introspection Support [2].

4.9.16. Прочие изменения

- Минимальная версия Git заменена на 1.8.3.1. Если сборочных хост имеет более старую версию, следует установить нужные пакеты, как указано в параграфе 1.3. Требуемые версии Git, tar и Python.
- Ошибочная и неполная поддержка менеджера пакетов RPM версии 4 была удалена. Проверенная и поддерживаемая версия 5 для RPM сохранена.
- Ранее было указано, что удаляются пакеты update-rc.d, base-passwd, shadow, update-alternatives, run-postinsts, если package-management нет в IMAGE_FEATURES. В выпуске YP 2.1 эти пакеты удаляются лишь при наличии read-only-rootfs в IMAGE_FEATURES, поскольку они могут требоваться для образов с доступом на чтение и запись даже при отсутствии менеджера пакетов (например, для добавления, изменения или удаления пользователей в процессе работы).
- Команда devtool modify по умолчанию извлекает исходный код в соответствии с ожиданиями. Опции -x и --extract больше не работают. Если нужно указать своё дерево исходных кодов, следует задать опцию -n или --no-extract для команды devtool modify.
- Если форм-фактор для машины не указан или не говорит о наличии клавиатуры, это наличие предполагается по умолчанию. Основное влияние это изменение оказывает на Sato UI.
- Каталоги упаковки .debug создаются автоматически. Если задание собирает пакеты, которые устанавливают двоичные файлы в нестандартные каталоги, больше не нужно заботиться об установке FILES_${PN}-dbg для указания каталогов .debug.
- Неточные данные о процентах занятости диска и CPU удалены из вывода buildstats с заменой данными getrusage() и корректной статистикой ввода-вывода. Возможно потребуется обновление пользовательского кода для чтения данных buildstats.
- Файл meta/conf/distro/include/package_regex.inc отменен с переносом содержимого в отдельные задания. Поскольку этот файл скорей всего будет удалён в новых выпусках YP, предлагается убрать все ссылки на него из вашей конфигурации.
- Файл v86d/uvesafb удалён из эталонных машин genericx86 и genericx86-64 уровня meta-yocto-bsp. Большинство современных плат x86 не использует этот файл и он лишь добавляет сообщения об ошибках ядра при запуске. Если поддержка uvesafb нужна, можно добавить v86d в образ.
- Пути сборки sysroot удалены из файлов с отладочными символами. Это означает, что удалённый отладчик GDB, использующий невырезанные символы сборки sysroot, больше не сможет работать (хотя такая работа никогда и не документировалась). Поддерживаемым методом для выполнения похожих операций является установка IMAGE_GEN_DEBUGFS = "1", при которой будет создаваться сопровождающий отладочный образ вместе с данными для отладки.

[1] GLib Introspective Repository - репозиторий самоанализа Glib.

4.10. Переход на YP 2.2

4.10.1. Минимальная версия ядра

Минимальная версия ядра для целевых платформ и SDK сейчас 3.2.0 в результате перехода на glibc 2.24. Для платформ AArch64 нужна версия 3.14, для Nios II - 3.19. Для систем x86 и x86_64 при необходимости можно сбросить OLDEST_KERNEL до версии 2.6.32.

4.10.2. Упрощено упорядочение каталогов в sysroot

Упорядочение каталогов в sysroot упрощено и добавлены переменные SYSROOT_DIRS, SYSROOT_DIRS_NATIVE и SYSROOT_DIRS_BLACKLIST. Дополнительная информация приведена в списке рассылки OE-Core (v2 patch series).

4.10.3. Включено удаление старых образов и других файлов из tmp/deploy

Удаление старых образов и других файлов из tmp/deploy/ включено по умолчанию в результате использования для этих файлов нового метода подготовки. В результате переменная RM_OLD_IMAGE стала избыточной.

4.10.4. Изменения Python

4.10.4.1. BitBake требует Python 3.4+

BitBake требует Python версии 3.4 или выше.

4.10.4.2. На сборочном хосте требуется UTF-8 Locale

Python 3 требует на сборочном хосте установки UTF-8. Поскольку C.UTF-8 не является стандартом, по умолчанию применяется en_US.UTF-8.

4.10.4.3. Метаданные должны использовать синтаксис Python 3

Для метаданных теперь требуется синтаксис Python 3. Рекомендации по подготовке метаданных можно найти в любом из руководств по переносу в Python 3. Можно также согласиться на фиксации (commit) преобразования для Bitbake и использовать OE-Core в качестве руководства. Ниже перечислены наиболее важные области.

- Конвейеры командной строки субпроцессов требуют декодирования locale.
- Изменён синтаксис восьмеричных значений.
- Изменены имена функции iter*().
- Итерации возвращают представления (view), а не списки.
- Изменены имена модулей Python.

4.10.4.4. Задания Python для целевых платформ переключены на Python 3

Большинство заданий Python для целевых платформ переведены на Python 3. К сожалению, системы с менеджером пакетов RPM и его поддержкой через SMART продолжают использовать Python 2.

Python 2 и применяющие его задания могут пока собираться для целевых платформ как в прежних версиях.

4.10.4.5. Архив buildtools включает Python 3

Архив buildtools сейчас включает Python 3.

4.10.5. Замена uClibc на musl

Библиотека uClibc была заменена библиотекой musl, которая лучше разработана, широко поддерживается и совместима с большим числом приложений, нежели uClibc.

4.10.6. ${B} больше не служит для задач рабочим каталогом по умолчанию

${B} больше не является принятым по умолчанию рабочим каталогом для задач. Поэтому все пользовательские задачи должны иметь флаг [dirs] или в задачах рабочий каталог должен быть указан вручную (например, командой cd). Предпочтительно использовать флаг [dirs].

4.10.7. Сценарий runqemu переписан на Python

Сценарий runqemu был переписан на Python и поведение в некоторых случаях изменилось. Прежние шаблоны поведения поддерживаются.

В новом сценарии runqemu на Python информация о машине уже не задана жёстко и можно выбрать использование конфигурационного файла qemuboot, чтобы определить собственные аргументы BSP и сделать его загружаемым с помощью runqemu. Конфигурационный файл имеет форму image-name-machine.qemuboot.conf.

Файл конфигурации включает тонкую настройку опций, передаваемых QEMU без жёсткого кодирования сценария runqemu и информации о разных машинах. Использование конфигурационного файла удобно, в частности, при попытке использования QEMU с машинами, отличающимися от qemu* в OE-Core. Файл qemuboot.conf создаётся классом qemuboot при сборке корневой файловой системы (rootfs). Аргументы загрузки QEMU могут быть заданы в конфигурационном файле BSP и класс qemuboot сохранит их в qemuboot.conf.

При использовании runqemu без файла конфигурации применяется команда вида

```
$ runqemu machine rootfs kernel [options]
```

Поддерживаются машины qemuarm, qemuarm64, qemux86, qemux86-64, qemuppc, qemumips, qemumips64, qemumipsel, qemumips64el.

Рассмотрим пример использования машины qemux86-64, обеспечивающий корневую файловую систему, образ и использование опции nographic.

```
$ runqemu qemux86-64 tmp/deploy/images/qemux86-64/core-image-minimal-qemux86-64.ext4 tmp/deploy/images/qemux86-64/bzImage
nographic
```

Ниже приведен список переменных, которые могут быть установлены в конфигурационном файле (например, bsp.conf) для загрузки BSP с помощью runqemu.

```
"QB" means "QEMU Boot".
QB_SYSTEM_NAME: QEMU name (e.g. "qemu-system-i386")
QB_OPT_APPEND: Options to append to QEMU (e.g. "-show-cursor")
QB_DEFAULT_KERNEL: Default kernel to boot (e.g. "bzImage")
QB_DEFAULT_FSTYPE: Default FSTYPE to boot (e.g. "ext4")
QB_MEM: Memory (e.g. "-m 512")
QB_MACHINE: QEMU machine (e.g. "-machine virt")
QB_CPU: QEMU cpu (e.g. "-cpu qemu32")
QB_CPU_KVM: Similar to QB_CPU except used for kvm support (e.g. "-cpu kvm64")
QB_KERNEL_CMDLINE_APPEND: Options to append to the kernel's -append
               option (e.g. "console=ttyS0 console=tty")
QB_DTB: QEMU dtb name
QB_AUDIO_DRV: QEMU audio driver (e.g. "alsa", set it when support audio)
QB_AUDIO_OPT: QEMU audio option (e.g. "-soundhw ac97,es1370"), which is used
               when QB_AUDIO_DRV is set.
QB_KERNEL_ROOT: Kernel's root (e.g. /dev/vda)
QB_TAP_OPT: Network option for 'tap' mode (e.g. "-netdev tap,id=net0,ifname=@TAP@,script=no,downscript=no
               -device virtio-net-device,netdev=net0").
               runqemu will replace "@TAP@" with the one that is used, such as tap0, tap1 ...
QB_SLIRP_OPT:  Network option for SLIRP mode (e.g. "-netdev user,id=net0 -device virtio-net-
device,netdev=net0")
QB_ROOTFS_OPT: Used as rootfs (e.g.
               "-drive id=disk0,file=@ROOTFS@,if=none,format=raw -device virtio-blk-device,drive=disk0").
               runqemu will replace "@ROOTFS@" with the one which is used, such as
               core-image-minimal-qemuarm64.ext4.
QB_SERIAL_OPT: Serial port (e.g. "-serial mon:stdio")
QB_TCPSERIAL_OPT: tcp serial port option (e.g.
               " -device virtio-serial-device -chardev socket,id=virtcon,port=@PORT@,host=127.0.0.1 -
               device      virtconsole,chardev=virtcon"
               runqemu will replace "@PORT@" with the port number which is used.
```

Для использования runqemu, устанавливается переменная IMAGE_CLASSES += "qemuboot" и запускается runqemu.

Синтаксис командной строки можно узнать с помощью команды runqemu help.

4.10.8. Изменён стиль хэширования принятого по умолчанию компоновщика

Компоновщик теперь по умолчанию применяет для кросс-компиляции gcc стиль хэширования sysv, чтобы перехватывать задания, создающие программы без использования LDFLAGS. Это изменение может создавать некоторые QA при сборке таких заданий в виде сообщений No GNU_HASH1. Для решения проблемы нужно изменить эти задания, чтобы они применяли LDFLAGS. В зависимости от способа сборки (например, Makefile) может потребоваться исправление системы сборки. Иногда достаточно добавить TARGET_CC_ARCH += "${LDFLAGS}".

4.10.9. KERNEL_IMAGE_BASE_NAME больше не использует KERNEL_IMAGETYPE

Переменная KERNEL_IMAGE_BASE_NAME больше не использует KERNEL_IMAGETYPE для создания базового имени образа. Поскольку система сборки OE может создавать множество типов образов ядра, эта часть базового имени была удалена, а сохранено лишь KERNEL_IMAGE_BASE_NAME ?= "${PKGE}-${PKGV}-${PKGR}-${MACHINE}-${DATETIME}. При наличии заданий или классов, использующих KERNEL_IMAGE_BASE_NAME напрямую, может потребоваться обновление ссылок.

4.10.10. BitBake

- Инструменты goggle UI и автономный image-writer удалены, поскольку им нужна поддержка GTK+ 2.0.
- Сборщик Perforce сейчас поддерживает SRCREV для указания используемого выпуска исходного кода (${AUTOREV}, номер списка изменений, p4date или метка) вместо отдельных параметров SRC_URI. Это изменение больше соответствует другим сборщикам, работающим с системами контроля версий. Задания, использующие Perforce, следует обновить для работы с SRCREV вместо SRC_URI.
- Нужно изменить некоторые внутренние структуры кода BitBake для доступа к кэшу заданий с целью поддержки новой функциональности с множеством конфигураций. Эти изменения будут влиять на внешние инструменты, применяемые в BitBake модулем tinfoil. Информация об этих изменениях доступна в описаниях изменений сценариев из состава OpenEmbedded-Core (1 и 2).
- Переписан код управления задачами для предотвращения использования косвенных идентификаторов, что позволило повысить производительность. Это изменение не должно вызывать проблем для большинства пользователей. Однако для проверки подписей нужна функция setscene, указываемая переменной BB_SETSCENE_VERIFY_FUNCTION. Поэтому добавлена переменная BB_SETSCENE_VERIFY_FUNCTION2, разрешающая нескольким версиям BitBake работать с подходящими метаданными, включая OpenEmbedded-Core и Poky. При использовании своих планировщиков задач BitBake может потребоваться обновление кода для работы с новой структурой.

4.10.11. Удалён инструмент Swabber

Инструмент Swabber, предназначенный для обнаружения «мусора» в процессе сборки, был удалён, поскольку некоторое время не поддерживается и никогда не был достаточно эффективным. Система сборки OE включает ряд механизмов (в том числе проверки QA), позволяющих обойтись без этого инструмента.

4.10.12. Удалённые задания

- *augeas* больше не требуется и перенесено в meta-oe.
- *directfb* не поддерживается и перенесено в meta-oe.
- *gcc* - удалена версия 4.9 с сохранением версий 5.4 и 6.2.
- *gnome-doc-utils* больше не требуется.
- *gtk-doc-stub* заменено gtk-doc.
- *gtk-engines* больше не требуется и перенесено в meta-gnome.
- *gtk-sato-engine* устарело.
- *libglade* больше не требуется и перенесено в meta-oe.
- *libmad* не поддерживается и функционально заменено libmpg123 с переносом libmad в meta-oe.
- *libowl* устарело.
- *libxsettings-client* больше не требуется.
- *oh-puzzles* функционально заменено puzzles.
- *oprofileui* устарело. OProfile в значительной степени вытеснен perf.
- *packagegroup-core-directfb.bb.*
- *core-image-directfb.bb.*
- *pointercal* больше не требуется и перенесено в meta-oe.
- *python-imaging* больше не требуется и перенесено в meta-python
- *python-pyrex* больше не требуется и перенесено в meta-python.
- *sato-icon-theme* устарело.
- *swabber-native* - утилита удалена Swabber (параграф 4.10.11. Удалён инструмент Swabber).
- *tslib* больше не требуется и перенесено в meta-oe.
- *uclibc* удалено с заменой на musl.
- *xtscal* больше не требуется и перенесено в meta-oe.

4.10.13. Удалённые классы

- *distutils-native-base* больше не требуется.
- *distutils3-native-base* больше не требуется.
- *sdl* устанавливает лишь переменные DEPENDS и SECTION, которые лучше назначать в задании.
- *sip* практически не применялся.
- *swabber* (см. параграф 4.10.11. Удалён инструмент Swabber).

4.10.14. Второстепенные изменения в пакетах

- *grub* - grub-editenv перенесён в отдельный пакет.
- *systemd* - связанные с container и vm блоки выделены в новый пакет systemd-container.
- *util-linux* - prlimit выделен в пакет util-linux-prlimit.

4.10.15. Прочие изменения

- Файл *package_regex.inc* удалён в результате переноса определений в соответствующие задания.
- Команды *devtool add* и *recipetool create* используют по умолчанию фиксированное значение SRCREV при сборке из репозиториев Git. Значение можно переопределить с помощью опции -a или --autorev для использования во всех случаях ${AUTOREV}.
- *distcc* интерфейс GTK+ UI по умолчанию отключён.
- *packagegroup-core-tools-testapps* - удалено Piglit.
- *image.bbclass* - COMPRESS(ION) переименована в CONVERSION. Это означает замену COMPRESSIONTYPES, COMPRESS_DEPENDS и COMPRESS_CMD на CONVERSIONTYPES, CONVERSION_DEPENDS и CONVERSION_CMD. Имена переменных COMPRESS* будут работать в выпуске 2.2, но метаданные, которым не нужна совместимость с прежними версиями, следует переименовать, поскольку COMPRESS* в будущих выпусках не планируется поддерживать.
- *gtk-doc* - доступна полная версия. Однако некоторые старые программы могут оказаться не способными применять новую версию для создания документации. Нужно изменить задания, собирающие такие программы, для явного запрета создания документации с помощью gtk-doc.

4.11. Переход на YP 2.3

4.11.1. Sysroot для задания

Система сборки OE теперь использует один каталог sysroot на задание для решения давних проблем с автоматическим обнаружением не объявленных зависимостей. Поэтому возможно обнаружение в написанных ранее пользовательских заданиях отсутствия объявленных зависимостей, в частности, случайно заданных ранее типовым процессом сборки, и уже присутствующих в общем каталоге sysroot из предыдущих выпусков.

- *Объявление зависимостей при сборке.* Поскольку это новая возможность, нужно явно объявить все зависимости при сборке задания. Если зависимости не указать, они не будут включены в sysroot для задания.
- *Указание зависимостей естественных инструментов предустановки и пост-установки.* Нужно конкретно указать все зависимости от естественных инструментов для сценариев pkg_preinst и pkg_postinst с помощью переменной PACKAGE_WRITE_DEPS. Это обеспечит доступность инструментов при необходимости запуска сценариев на хосте сборки при выполнении задачи do_rootfs.

  Рассмотрим, например, задание dbus, которое имеет сценарий pkg_postinst, вызывающий systemctl, если systemd имеется в переменной DISTRO_FEATURES. В этом случае systemd-systemctl-native добавляется в переменную PACKAGE_WRITE_DEPS с учётом наличия systemd в переменной DISTRO_FEATURES.
- *Проверка заданий, использующих SSTATEPOSTINSTFUNCS.* Функции, добавленные в SSTATEPOSTINSTFUNCS, вызываются как в прежних выпусках YP. Однако в результате того, что sysroot сейчас заполняется отдельно для каждого задания при наличии перемещений в функциях, вызываемых через SSTATEPOSTINSTFUNCS нужно будет внести изменения для использования сценария пост-установки, установленного функцией, добавленной в SYSROOT_PREPROCESS_FUNCS. Примером может служить класс pixbufcache в каталоге meta/classes/ Yocto Project Source Repositories.

  Сама переменная SSTATEPOSTINSTFUNCS сейчас отменена с заменой на задачу do_populate_sysroot[postfuncs]. Поэтому при наличии функций, которые нужно вызывать после создания sysroot для задания, разумно будет предпринять шаги для использования сценария пост-установки, как описано выше. Это подготовит код к удалению SSTATEPOSTINSTFUNCS в будущем выпуске YP.
- *Указание sysroot при использовании некоторых внешних сценариев.* Поскольку общий каталог sysroot сейчас отсутствует, сценарии oe-find-native-sysroot и oe-run-native были изменены так, что требуется указать какое задание используется в STAGING_DIR_NATIVE.

Дополнительная информация о работе sysroot для заданий приведена в параграфе 6.127. staging.bbclass.

4.11.2. Переменная PATH

В среде, используемой для работы задач сборки, переменная PATH теперь очищается с удалением естественных путей (/bin, /sbin, /usr/bin и т. п.) и символьные ссылки указывают только на двоичные файлы сборочного хоста, указанные в переменных HOSTTOOLS и HOSTTOOLS_NONFATAL, добавленных в PATH. Поэтому все естественные двоичные файлы, предоставляемые хостом, должны включаться в эти две переменные на уровне настройки.

Другим вариантом является добавление задания -native в переменную DEPENDS. Переменная PATH в этом случае не очищается в devshell таким же способом, поскольку это создало бы проблемы при запуске инструментов хоста в среде разработки.

4.11.3. Изменение сценариев

- *oe-find-native-sysroot* - применение сценария изменилось и имеет форму

  ```
  $ . oe-find-native-sysroot recipe
  ```

  Сейчас задание должно указываться как часть команды, что до версии YP 2.7.1 не требовалось.
- *oe-run-native* - применение сценария изменилось и имеет форму

  ```
  $ oe-run-native native_recipe tool
  ```

  Сейчас естественное задание должно указываться как часть команды, что до версии YP 2.7.1 не требовалось.
- *cleanup-workdir* - удалён в результате обнаружения наносимых им повреждений дерева сборки (удаление файлов). Вместо удаления части TMPDIR рекомендуется удалять TMPDIR полностью и восстанавливать из общего состояние (sstate) при последующей сборки.
- *wipe-sysroot* удалён, поскольку больше не требуется.

4.11.4. Изменения функций

Ранее отменённые функции bb.data.getVar(), bb.data.setVar() удалены с заменой на d.getVar(), d.setVar() и т. п. Следует исправить ссылки на старые функции.

4.11.5. BitBake

- *Каменен интерфейс depexp.* Графический интерфейс исследования зависимостей BitBake depexp заменён интерфейсом taskexp (Task Explorer), обеспечивающим графическое представление файла task-depends.dot. Представляемые этим интерфейсом данные более точны, нежели данные depexp. Визуализация данных является часто запрашиваемой функцией, поскольку стандартные средства просмотра файлов *.dot зачастую не справляются с размером task-depends.dot.
- *Изменён вывод bitbake -g.* Файлы package-depends.dot и pn-depends.dot, которые раньше создавались командой bitbake -g, удалены. Сейчас генерируется файл recipe-depends.dot, являющийся сокращённым вариантом task-depends.dot. Это изменение вызвано тем, что файлы package-depends.dot и pn-depends.dot в значительной степени относятся ко времени до выполнения на основе задач и не учитывают зависимостей между заданиями, что может вводить в заблуждение.

- *Изменение переменных для зеркал.* Переменные для зеркал, включая MIRRORS, PREMIRRORS и SSTATE_MIRRORS позволяют теперь разделять значения пробелами, поэтому больше не нужно использовать \\n. BitBake ищет пары значений, что упрощает использование. Для имеющихся переменных не должно требоваться каких-либо изменений.

- *Сборщик SVN использует параметр ssh, а не rsh.* Этот новый необязательный параметр используется при установке protocol = svn+ssh. Параметр может применяться лишь для указания программы ssh, используемой SVN. Сборщик SVN передаёт параметр через переменную SVN_SSH при выполнении задачи do_fetch. Дополнительная информация приведена в разделе Subversion (SVN) Fetcher (svn://).

- *Удалены BB_SETSCENE_VERIFY_FUNCTION и BB_SETSCENE_VERIFY_FUNCTION2*, поскольку применявший их механизм больше не используется для связанных с заданиями систем sysroot.

4.11.6. Абсолютные символьные ссылки

Абсолютные символьные ссылки (symlink) больше не разрешены в промежуточных (stage) файлах и вызывают ошибку. Для явного создания символьных ссылок может использоваться сценарий lnr, который заменяет команду ln -r.

Если сценарий сборки в создаваемой заданием программе создает абсолютные символьные ссылки, которые нужно исправлять, можно наследовать в задании класс relative_symlinks для преобразования ссылок в относительные.

4.11.7. Версии GPLv2 для заданий GPLv3 перенесены

Старые GPLv2-версии заданий GPLv3 перенесены на отдельный уровень meta-gplv2. При использовании INCOMPATIBLE_LICENSE для исключения GPLv3 или PREFERRED_VERSION для подстановки GPLv2-версий заданий GPLv3 нужно добавить в конфигурацию уровень meta-gplv2, доступный по ссылке.

Перемещённые задания GPLv2 не получают такой же поддержки, как другие базовые задания. Для них не выпускаются исправления, связанные с безопасностью, и разработка уже не ведётся. Фактически сообщество разработчиков негативно относится к использованию старых версий заданий. Перенос их на отдельный уровень делает яснее потребности других заданий и более чётко определяет их.

Долгосрочным решением может быть переход на компоненты с лицензией BSD, заменяющие компоненты GPLv3, если требуется исключить лицензии GPLv3 из системы. Это решение будет исследовано в новых выпусках YP.

4.11.8. Изменение управления пакетами

- Менеджер пакетов Smart был заменён на DNF. Smart больше не разрабатывается и не перенесён на Python 3.x. Единственным кандидатом на замену оказался пакет DNF.

  Изменение функциональности заключается в том, что управление пакетами на целевой платформе во время работы с удалённого хранилища пакетов выполняется другим инструментом с иным набором команд. При наличии сценариев, вызывавших инструмент напрямую или через API, их нужно изменить. Информация о менеджере DNF доступна по ссылке.

- Rpm 5.x заменён на Rpm 4.x по двум основным причинам:
  - DNF не совместим на уровне API с 5.x, а перенос и поддержка являются сложными задачами;
  - Rpm 5.x имеет ограниченную поддержку и YP является одним из немногих применений.

- Поддержка Berkeley DB 6.x удалена и по умолчанию используется Berkeley DB 5.x.
  - Версия 6.x Berkeley DB была отвергнута значительной частью сообщества по причине лицензии AGPLv3. В результате большинство проектов с открытым кодом, использующих DB, продолжает работать с DB 5.x.
  - В OE-core использование DB 6.x нужно было лишь Rpm 5.x и нет причин сохранять DB 6.x в OE-core.

- Инструмент createrepo заменён createrepo_c, который является современным средством генерации метаданных удалённого репозитория и работает быстрее, чем createrepo (язык с вместо Python).

- Независимые от архитектуры пакеты RPM теперь используют суффикс noarch вместо all.

  Это обусловлено тем, во многих случаях использования DNF/RPM4 такой переход уже произошёл. Меняются лишь имена файлов и теги архитектуры. Никаких иных изменений не вносится в OE-core и класс allarch.bbclass.

- Подписание удалённых хранилищ пакетов с помощью PACKAGE_FEED_SIGN не поддерживается. Проблема будет полностью решена в новом выпуске YP (см. defect 11209).

- OPKG сейчас использует по умолчанию библиотеку libsolv для учёта зависимостей пакетов, что значительно превосходит использовавшийся ранее внутренний инструмент OPKG. Изменение ведёт к незначительному росту размера на диске (около 500 кбайт) и расхода оперативной памяти (см. commit message).

4.11.9. Удалённые задания

- *linux-yocto 4.8* - версия 4.8 удалена, версии 4.1 (LTSI), 4.4 (LTS), 4.9 (LTS/LTSI) и 4.10 сохранены.
- *python-smartpm* - функционально заменено dnf.
- *createrepo* - заменено createrepo-c.
- *rpmresolve* - больше не требуется в результате перехода на RPM 4 по причине использования самого RPM.
- *gstreamer* - удалены задания GStreamer Git, поскольку они устарели. Задания 1.10.x сохранены.
- *alsa-conf-base* - объединено с alsa-conf, поскольку libasound зависит от обоих.

- *tremor* - перенесено в meta-multimedia. Целочисленное декодирование Vorbis не требуется современному оборудованию. Поэтому подключаемый модуль GStreamer ivorbis по умолчанию отключён., что избавляет от необходимости включать задание tremor в OE-Core.

- *gummiboot* - заменено systemd-boot.

4.11.10. Изменения Wic

- *Используемый по умолчанию выходной каталог.* Сейчас Wic использует текущий каталог, а не /var/tmp/wic. Опции -o и --outdir сохранились и служат для задания предпочтительного выходного каталога.

- *Удалён подключаемый модуль fsimage*. Плагин Wic fsimage дублировал функциональность rawcopy.

Описание Wic доступно в разделе Creating Partitioned Images Using Wic [2].

4.11.11. Изменения QA

- *Проверка unsafe-references-in-binaries*, отключённая по умолчанию, была удалена. Эта проверка была предназначена для обнаружения двоичных файлов в каталоге /bin, связанных с библиотеками в /usr/lib для случая наличия у пользователя каталога /usr на отдельной от / файловой системе.

  Удалённая проверка имела ошибки. Каталог /usr сейчас редко размещается в разделе отдельном от /.

- *Проверка file-rdeps* сейчас по умолчанию возвращает ошибку, а не предупреждение. Поэтому требуется решить проблемы с невыполненными зависимостями. Дополнительная информация приведена в параграфах 6.56. insane.bbclass и 10.2. Ошибки и предупреждения.

4.11.12. Прочие изменения

- В этом выпуске изменено множество заданий для игнорирования элемента largefile в DISTRO_FEATURES, включающего безусловную поддержку больших файлов. Свойство всегда было включено по умолчанию, а его отключение не тестировалось достаточно широко. В будущих выпусках YP возможность отключения свойства поддержки больших файлов будет упразднена совсем.

- Если значение DISTRO_VERSION включает значение переменной DATE, которое принято по умолчанию для выпусков Poky, значение DATE явно исключается из /etc/issue и /etc/issue.net, которые выводятся в приглашении на регистрацию в системе, чтобы избежать конфликтов при использовании нескольких библиотек (Multilib). В любом случае значение DATE не является точным, если задание base-files восстановлено из общего состояния (sstate), а не собрано заново.

  Если нужно записать дату сборки в /etc/issue* или иное место образа, лучше всего задать для этого функцию пост-обработки и вызвать её из команды ROOTFS_POSTPROCESS_COMMAND. Это обеспечит соответствие значения моменту создания образа.

- Сценарий инициализации Dropbear сейчас по умолчанию запрещает DSA-ключи хоста. Это изменение соответствует файлу службы systemd, который поддерживает только ключи RSA, и свежим версиям OpenSSH, где не поддерживаются ключи DSA для хоста.

- Класс buildhistory сейчас корректно использует символы табуляции в качестве разделителей колонок в файле installed-package-sizes.txt, что позволяет импортировать этот файл.

- Переменная USE_LDCONFIG заменена свойством ldconfig в переменной DISTRO_FEATURES. Дистрибутивам, которые раньше задавали USE_LDCONFIG = "0", следует вместо этого указывать DISTRO_FEATURES_BACKFILL_CONSIDERED_append = " ldconfig".

- Принятое по умолчанию значение COPYLEFT_LICENSE_INCLUDE сейчас включает все версии лицензий AGPL в дополнение к GPL и LGPL. Заданный по умолчанию список не гарантирует полноты и следует обратиться к юридическим документам с учётом дистрибутива, если нет полной уверенности.

- Пакеты модулей ядра сейчас имеют суффиксы с номером версии для возможности сосуществования на целевой системе нескольких версий ядра. Для возврата к прежней схеме именования без суффиксов следует указать KERNEL_MODULE_PACKAGE_SUFFIX = "".

- Удаление файлов libtool *.la сейчас включено по умолчанию, поскольку эти файлы реально не нужны в Linux и их перемещение создает дополнительную нагрузку. Если нужно сохранить такие файлы, следует изменить переменную INHERIT_DISTRO, чтобы она не включала значение "remove-libtool".

- Расширяемые SDK, собранные для GCC 5+, сейчас отвергают установку в дистрибутивы с версиями GCC на хосте 4.8 или 4.9. Это изменение вызвано тем, что известно об отказе при инсталляции из-за способа сборки общего состояния uninative (sstate). Дополнительные сведения приведены в параграфе 6.140. uninative.bbclass.

- Все естественные и nativesdk задания теперь используют раздельные значения DISTRO_FEATURES вместо общего для заданий платформы, чтобы избежать неоправданной повторной сборки. В качестве DISTRO_FEATURES для естественных заданий служит DISTRO_FEATURES_NATIVE, добавленная к пересечению переменных DISTRO_FEATURES и DISTRO_FEATURES_FILTER_NATIVE. Для заданий nativesdk соответствующими переменными являются DISTRO_FEATURES_NATIVESDK и DISTRO_FEATURES_FILTER_NATIVESDK.

- Переменная FILESDIR, которая ранее была отменена и применялась редко, сейчас удалена совсем. Следует заменить в заданиях эту переменную на FILESPATH.

- Переменная MULTIMACH_HOST_SYS удалена, поскольку уже не нужна для связанных с заданиями sysroot.

4.12. Переход на YP 2.4

4.12.1. Режим присутствия в памяти

Постоянный режим теперь доступен в принятой по умолчанию работе BitBake вместо прежнего резидентного режима (memory resident mode, т. е. oe-init-build-env-memres). Сейчас нужно лишь установить тайм-аут в переменной BB_SERVER_TIMEOUT (секунды) и сервер BitBake будет оставаться в памяти заданное время между вызовами. Сценарий oe-init-build-env-memres удалён за ненадобностью.

4.12.2. Изменения в пакетах

- *python3*
  - Основной пакет python3 теперь содержит весь стандартный дистрибутив Python 3. Это соответствует поведению, ожидаемому в традиционных дистрибутивах Linux. Если нужно установить лишь часть Python 3, указывается python-core и один или несколько отдельных пакетов, которые нужны.
  - *python3* - сценарии bz2.py, lzma.py, и _compression.py перенесены из python3-misc в python3-compression.
- *binutils* - библиотека libbfd помещена в отдельный пакет libbfd, что экономит пространство при установке некоторых инструментов (например, perf), когда нужна лишь библиотека libbfd а не весь пакет binutils.
- *util-linux*
  - Программа su помещена в отдельный пакет util-linux-su, который собирается только при наличии pam в переменной DISTRO_FEATURES. Пакет util-linux should не следует устанавливать, пока он не требуется, поскольку su обычно предоставляется через файл shadow. Основной пакет util-linux зависит при работе (RDEPENDS) от util-linux-su, если pam присутствует в DISTRO_FEATURES.
  - Программа switch_root program помещена в отдельный пакет util-linux-switch-root для небольших образов initramfs, которым не нужен весь пакет util-linux или исполняемый файл busybox, которые значительно больше switch_root. В основном пакете util-linux имеется рекомендуемая зависимость (RRECOMMENDS) от пакета util-linux-switch-root.
  - Программа ionice помещена в отдельный пакет util-linux-ionice. В основном пакете util-linux имеется рекомендуемая зависимость (RRECOMMENDS) от пакета util-linux-ionice.
- *initscripts* - программа sushell помещена в отдельный пакет initscripts-sushell, что позволяет системам получать sushell при включённом selinux. Это избавляет от необходимости устанавливать весь пакет initscripts. Основной пакет initscripts зависит при работе (RDEPENDS) от sushell, если selinux присутствует в DISTRO_FEATURES.
- *glib-2.0* сейчас имеет рекомендованную зависимость (RRECOMMENDS) от пакета shared-mime-info, поскольку значительная часть GIO бесполезна при отсутствии базы данных MIME. Можно исключить зависимость через переменную BAD_RECOMMENDATIONS, если пакет shared-mime-info слишком велик и не требуется.
- *Стандартный запуск Go.* Стандартная среда запуска Go выделена из основного задания go в go-runtime.

4.12.3. Удалённые задания

- *acpitests* не поддерживается.
- *autogen-native* больше не требуется для Grub, oe-core, meta-oe.
- *bdwgc* уже не требуется в OpenEmbedded-Core и перенесено в meta-oe.
- *byacc* нужно лишь для rpm 5.x и перенесено в meta-oe.
- *gcc (5.4)* - версия 5.4 не поддерживается с заменой на 6.3/7.2.
- *gnome-common* не поддерживается и больше не требуется.
- *go-bootstrap-native* - Go 1.9 выполняет загрузку самостоятельно, поэтому задание удалено.
- *guile* требовалось только в autogen-native и remake, а теперь не нужно этим программам.
- *libclass-isa-perl* ранее требовалось для LSB 4, а сейчас не нужно.
- *libdumpvalue-perl* ранее требовалось для LSB 4, а сейчас не нужно.
- *libenv-perl* ранее требовалось для LSB 4, а сейчас не нужно.
- *libfile-checktree-perl* ранее требовалось для LSB 4, а сейчас не нужно.
- *libi18n-collate-perl* ранее требовалось для LSB 4, а сейчас не нужно.
- *libiconv* требовалось только для uclibc, удалённой в предыдущем выпуске. Библиотеки glibc и musl имеют свои реализации, meta-mingw требуется для libiconv, поэтому перенесено в meta-mingw.
- *libpng12* ранее требовалось для LSB. Сейчас применяется libpng 1.6.x.
- *libpod-plainer-perl* ранее требовалось для LSB 4, а сейчас не нужно.
- *linux-yocto (4.1)* удалено с заменой на 4.4, 4.9, 4.10, 4.12.
- *mailx* ранее требовалось для совместимости с LSB, разработка прекращена.
- *mesa (только версия git)* - git-версия задания устарела по сравнению с версией выпуска.
- *ofono (только версия git)* - git-версия задания устарела по сравнению с версией выпуска.

- *portmap* устарело и заменено rpcbind.
- *python3-pygpgme* устарело и не поддерживается. Ранее требовалось для пакета dnf, который переключен сейчас на gpgme Python.
- *python-async* удалено с переносом на Python 3.
- *python-gitdb* удалено с переносом на Python 3.
- *python-git* удалено с переносом на Python 3.
- *python-mako* удалено с переносом на Python 3.
- *python-pexpect* удалено с переносом на Python 3.
- *python-ptyprocess* удалено с переносом на Python 3.
- *python-pycurl* не применяется в OpenEmbedded-Core (meta-oe).
- *python-six* удалено с переносом на Python 3.
- *python-smmap* удалено с переносом на Python 3.
- *remake* в качестве провайдера virtual/make больше не применяется, поэтому не нужно в OpenEmbedded-Core.

4.12.4. Перенос дерева устройств ядра

Поддержку дерева устройств ядра сейчас проще включить в задании для ядра, поскольку код дерева устройств перенесён в класс kernel-devicetree. Функциональность включается автоматически для любого задания, наследующего класс kernel и установившего переменную KERNEL_DEVICETREE. Прежний механизм с файлом meta/recipes-kernel/linux/linux-dtb.inc продолжает работать, но вызывает предупреждения. В будущих выпусках YP файл meta/recipes-kernel/linux/linux-dtb.inc предполагается удалить. Рекомендуется исключить все обязательные операторы, запрашивающие meta/recipes-kernel/linux/linux-dtb.inc, из заданий для ядер.

4.12.5. Изменения QA для пакетов

- Проверка unsafe-references-in-scripts удалена.
- При указании ${COREBASE}/LICENSE в LIC_FILES_CHKSUM выдаётся предупреждение, поскольку файл является описанием лицензии для OE-Core. Следует применять ${COMMON_LICENSE_DIR}/MIT, если задание использует лицензию MIT и нет возможности задать предпочтительный метод указания файла в дереве кода.

4.12.6. Изменения в файлах README

- Основной файл Poky README перенесён на уровень meta-poky и переименован в README.poky. Для прежнего местоположения создана символьная ссылка.
- Файл README.hardware перенесён на уровень meta-yocto-bsp с символьной ссылкой на прежнее место.
- Создан файл README.qemu для машин qemu*.

4.12.7. Прочие изменения

- Удалена переменная ROOTFS_PKGMANAGE_BOOTSTRAP и все ссылки на неё, поэтому следует исключить переменную из своих заданий.
- Удалён каталог meta-yocto. В выпуске YP 2.1 уровень meta-yocto был переименован в meta-poky, а подкаталог meta-yocto сохранили для предотвращения проблем в имеющихся конфигурациях.
- Файл maintainers.inc, который указывал сопровождающих путём указания ответственного за каждое задание в OE-Core, перенесён из meta-poky в OE-Core (из meta-poky/conf/distro/include в meta/conf/distro/include).
- Класс buildhistory делает одну фиксацию (commit) на сборку, а не на подкаталог репозитория. Это предполагает, что фиксация разрешена (BUILDHISTORY_COMMIT = "1"), что является обычным делом. Ранее класс buildhistory делал фиксацию на подкаталог репозитория для упрощения просмотра изменений в каталоге. Сейчас для просмотра таких изменений следует указывать каталог в качестве последнего параметра команды git show или git diff.
- Файл x86-base.inc, включаемый во все конфигурации машин x86, устанавливает IMAGE_FSTYPES ?= "live", вместо использования оператора +=. Это упрощает переопределение принятого по умолчанию значения.
- BitBake выдаёт множество событий BuildStarted при включённом множестве конфигураций (одно событие на конфигурацию). Дополнительная информация приведена в разделе Events [4].
- По умолчанию файл security_flags.inc устанавливает переменную GCCPIE с опцией включения PIE[1] в gcc, что осложняет организацию атак ROP[2].
- OE-Core предоставляет подключаемый модуль bitbake-layers с командой create-layer, реализация которой привела к ненужности сценария yocto-layer, который может быть удалён из будущих выпусков YP.
- Типы vmdk, vdi, qcow2 для файлов образов сейчас применяются вместе с типом образа wic через переменную CONVERSION_CMD. Эквивалентные типы образов имеют значения wic.vmdk, wic.vdi и wic.qcow2.
- do_image_<type>[depends] было заменено переменной IMAGE_DEPENDS_<type>. При наличии классов с пользовательскими типами образов их следует обновить.

[1]Position Independent Executables - независимые от положения исполняемые файлы.
[2]Return Oriented Programming - ориентированное на возврат программирование.

- Добавлена поддержка OpenSSL 1.1, однако 1.0.x продолжает по умолчанию использоваться в переменной PREFERRED_VERSION. Это сохранено из-за остающихся проблем совместимости с другими программами. Переменная PROVIDES в задании openssl 1.0 сейчас включает openssl10 как маркер, который может использоваться DEPENDS в заданиях, собирающих программы, зависимые от OpenSSL 1.0.
- Для согласованного поведения опций BitBake -r и -R (prefile и postfile), используемых для чтения или пост-чтения дополнительного конфигурационного файла из командной строки, они сейчас действуют лишь в текущей команде BitBake command. раньше указанные опции BitBake сохранялись при следующих вызовах.

4.13. Переход на YP 2.5

4.13.1. Изменения в пакетах

- *bind-libs* - библиотеки подготавливаются заданием bind в отдельном пакете bind-libs.
- *libfm-gtk* - привязки libfm GTK+ перенесены в отдельный пакет libfm-gtk.
- *flex-libfl* - задание flex выделено из libfl в пакет flex-libfl для снижения числа зависимостей при установке только библиотеки.
- *grub-efi* - конфигурация grub-efi перенесена в отдельное задание grub-bootconf. Однако зависимость от grub-efi указывается через провайдера virtual/grub-bootconf, что позволяет иметь своё задание для обеспечения зависимости. Можно использовать файл добавления BitBake для возврата конфигурации в задание grub-efi.
- *Поддержка устаревших хранилищ пакетов armv7a* удалена для перехода с armv7a на armv7a-vfp-neon в хранилищах пакетов, который раньше включался переменной PKGARCHCOMPAT_ARMV7A. Переход произошёл в 2011 г. и активные хранилища пакетов должны быть обновлены с указанием новых имён.

4.13.2. Удалённые задания

- *gcc* - версия 6.4 заменена 7.x.
- *gst-player* переименовано в gst-examples.
- *hostap-utils* - пакет устарел.
- *latencytop* больше не поддерживается (последний выпуск был в 2009 г.).
- *libpfm4* требовалось лишь удалённому заданию oprofile.
- *linux-yocto* - версии 4.4, 4.9, 4.10 удалены, версии 4.12, 4.14, 4.15 сохраняются.
- *man* заменено современным man-db
- *mkelfimage* - программа удалена из проекта coreboot и больше не нужна после удаления типа образов ELF.
- *nativesdk-postinst-intercept* не поддерживается.
- *neon* - программа не поддерживается и больше не нужна в OpenEmbedded-Core.
- *oprofile* - функциональность заменена perf, а совместимость с musl может быть затруднена.
- *pax* - программа устарела.
- *stat* - программа не развивается, а функции stat обеспечивает пакет coreutils.
- *zisofs-tools-native* больше не требуется после исключения сжатых образов ISO.

4.13.3. Изменения сценариев и инструментов

- *yocto-bsp, yocto-kernel, yocto-layer*, ранее входившие в poky, но не в OpenEmbedded-Core, удалены. Эти сценарии устарели и не поддерживаются, что во многих случаях ограничивало их применимость. Команда bitbake-layers create-layer служит их прямой заменой для yocto-layer (см. раздел BSP Kernel Recipe Example [6]).
- *devtool finish* сейчас завершает работу с ошибкой при наличии непредставленных изменений или в процессе выполнения rebase в репозитории задания. Причиной ошибки могут быть незафиксированные изменения, не включённые в обновления к правкам, которые применены заданием. Если эти изменения несущественны, можно форсировать выполнение команды с помощью опции -f или --force.
- *scripts/oe-setup-rpmrepo* - функциональность заменена командой bitbake package-index.
- *scripts/test-dependencies.sh* - устарел и заменён функциональностью sysroot для заданий, добавленной YP 2.4.

4.13.4. BitBake

- Изменена опция --runall, которая имеет 2 варианта поведения:
  - *вариант A* - для данной цели (набора целей) просматривается граф задач и задача X запускается лишь при её наличии и включении в сборку.
  - *вариант B* - для данной цели (набора целей) просматривается граф задач и задача X запускается, если любое задание в графе задач имеет такую цель, даже если этого задания нет в исходном графе задач.

  Опция --runall сейчас работает по варианту B, а ранее работала подобно варианту A. Опция --runonly добавлена для сохранения возможности использовать вариант A.
- Удалены несколько явных задач, выполнявшихся для всех заданий в дереве зависимостей (например, fetchall, checkuriall и все задачи классов distrodata и archiver). Имеется опция BitBake, позволяющая выполнить это для любой задачи. Например, bitbake <target> -c fetchall следует заменить на bitbake <target> --runall=fetch.

4.13.5. Изменения Python и Python 3

Управляемые сценариями файлы python-*-manifest.inc, применявшиеся для генерации пакетов Python и Python 3, заменены файлом JSON для удобства чтения и поддержки. Доступна новая задача для сопровождающих задания Python, которая переключает на файл JSON при переходе на новые версии Python. Файл можно редактировать напрямую вместо редактирования сценария и использования его для обновления файла.

Ещё одно изменение, которое следует отметить, связано с тем, что задания Python больше не имеют провайдеров во время сборки пакетов. Это предполагает, что python является одним из пакетов, обеспечиваемых заданием Python. Можно больше не запускать bitbake python-foo и не иметь DEPENDS для python-foo, но нужно добавить IMAGE_INSTALL_append = " python-foo" или RDEPENDS_${PN} = "python-foo".

Поведение прежних провайдеров при сборке было причудой способа создания манифестов Python. Более подробные сведения об изменениях доступны по ссылке.

4.13.6. Прочие изменения

- Класс kernel поддерживает сборку пакетов для нескольких ядер. Если задание для ядра или файл .bbappend включает упаковку, следует заменить ссылку на ядро в именах пакетов значением ${KERNEL_PACKAGE_NAME}. Например, при запрете автоматической установки ядра с помощью RDEPENDS_kernel-base = "" можно избежать предупреждений, задав RDEPENDS_${KERNEL_PACKAGE_NAME}-base = "".
- Класс buildhistory по умолчанию фиксирует изменения в репозитории и больше не нужно задавать BUILDHISTORY_COMMIT = "1". Если нужно отключить фиксацию, установите BUILDHISTORY_COMMIT = "0".
- Эталонная машина beaglebone переименована в beaglebone-yocto. BSP beaglebone-yocto является эталонной реализацией, использующей лишь компоненты OpenEmbedded-Core и meta-yocto-bsp, тогда как Texas Instruments поддерживает полнофункциональный BSP на уровне meta-ti. Переименование избавило от имевшегося конфликта имён. двух BSP.
- Класс update-alternatives больше не работает со сценариями инициализации SysV, поскольку это стало проблематичным. Задание sysklogd больше не применяет update-alternatives по причине несовместимости с другими реализациями.
- По умолчанию класс cmake использует при сборке пакетов ninja вместо make для повышения производительности сборки. Если задание не работает с ninja, можно установить OECMAKE_GENERATOR = "Unix Makefiles" для возврата к make.
- Ранее отменённые функции base_* удалены с заменами из meta/lib/oe и bitbake/lib/bb. Эти функции обычно использовались в заданиях и классах и все ссылки на удалённые функции следует изменить в соответствии с приведённой ниже таблицей.

Удалено	*Замена*
base_path_join()	oe.path.join()
base_path_relative()	oe.path.relative()
base_path_out()	oe.path.format_display()
base_read_file()	oe.utils.read_file()
base_ifelse()	oe.utils.ifelse()
base_conditional()	oe.utils.conditional()
base_less_or_equal()	oe.utils.less_or_equal()
base_version_less_or_equal()	oe.utils.version_less_or_equal()
base_contains()	bb.utils.contains()
base_both_contain()	oe.utils.both_contain()
base_prune_suffix()	oe.utils.prune_suffix()
oe_filter()	oe.utils.str_filter()
oe_filter_out()	oe.utils.str_filter_out() (или оператор _remove).

- Использование exit 1 для явной отсрочки сценария пост-установки до первой перезагрузки отменено, поскольку этот неочевидный механизм может скрывать ошибки. Если во время создания rootfs нужно явно отложить пост-установку до первой перезагрузки, следует использовать pkg_postinst_ontarget() или вызов postinst-intercepts defer_to_first_boot from pkg_postinst(). Любой отказ сценария pkg_postinst(), включая exit 1, будет вызывать предупреждение при работе задачи do_rootfs.

  Дополнительная информация приведена в разделе Post-Installation Scripts [2].
- Образы типа elf исключены, поскольку инструмент mkelfimage, который нужен для их создания, больше не поддерживается в coreboot и требует обновления при каждом изменении binutils.
- Удалена поддержка сжатия образов .iso (ранее включаемая через COMPRESSISO = "1"). Инструменты пользовательского пространства (zisofs-tools) не поддерживаются, а squashfs обеспечивает более эффективное сжатие. Для сборки live-образов с компрессией squashfs+lz4 следует задать LIVE_ROOTFS_TYPE = "squashfs-lz4" и включить live в IMAGE_FSTYPES.
- Заданием с безусловной зависимостью от libpam было лишь buildable с pam в DISTRO_FEATURES. Если зависимость не является обязательной, лучше сделать её условной по наличию pam в DISTRO_FEATURES.
- Для машин на базе EFI загрузчик (по умолчанию grub-efi) устанавливается в образе как /boot. Можно использовать Wic для разделения загрузчика на разделы boot и rootfs.
- Patch-файлы, контекст которых не имеет точного соответствия (исправление даёт fuzz при наложении), будут вызывать предупреждения. Пример доступен по ссылке.
- Предполагается, что уровни устанавливают LAYERSERIES_COMPAT_layername в соответствии с версией OpenEmbedded-Core, с которой они совместимы. Это указывается как кодовое имя с использованием пробелов

для разделения значений (например, "rocko sumo"). Если уровень не устанавливает LAYERSERIES_COMPAT_layername, выводится предупреждение. Если уровень устанавливает значение, не включающее текущей версии ("sumo" для выпуска 2.5), выдаётся ошибка.

- В переменной окружения TZ устанавливается значение UTC в среде сборки для предотвращения проблем воспроизводимости в некоторых заданиях.

4.14. Переход на YP 2.6

4.14.1. По умолчанию применяется GCC 8.2

GNU Compiler Collection версии 8.2 сейчас по умолчанию используется для компиляции. Дополнительная информация об изменениях в выпуске GCC 8.x доступна по ссылке https://gcc.gnu.org/gcc-8/changes.html. Для случаев потребности в компиляторе версии 7.x сохранен GCC 7.3. Этот компилятор можно выбрать, установив в конфигурации для переменной GCCVERSION значение "7.%".

4.14.2. Удалённые задания

- *beecrypt* не требуется в связи с переходом на RPM 4.
- *bigreqsproto* заменено xorgproto.
- calibrateproto удалено с заменой на xinput.
- *compositeproto*, *damageproto*, *dmxproto*, *dri2proto*, *dri3proto* заменены xorgproto.
- *eee-acpi-scripts* устарело.
- *fixesproto*, *fontsproto* заменены xorgproto.
- *fstests* устарело.
- *gccmakedep* больше не применяется.
- *glproto* заменено xorgproto.
- *gnome-desktop3* больше нет требуется. Перенесено в meta-oe.
- *icon-naming-utils* больше не применяется в результате удаления темы Sato в 2016 г.
- *inputproto*, *kbproto* заменены xorgproto.
- *libusb-compat* устарело.
- *libuser* устарело.
- *libnfsidmap* больше не требуется с переходом на nfs-utils 2.2.1.
- *libxcalibrate* больше не требуется.
- *mktemp* устарело. Команда mktemp предоставляется заданиями busybox и coreutils.
- *ossp-uuid* не будет поддерживаться и в основном заменено uuid.h в util-linux.
- *pax-utils* больше не нужно. Прежние тесты QA использовали это задание при сборке.
- *pcmciautils* устарело.
- *pixz* больше не требуется, поскольку xz поддерживает многопотоковое сжатие.
- *presentproto*, *randrproto*, *recordproto*, *renderproto*, *resourceproto*, *scrnsaverproto* заменены xorgproto.
- *trace-cmd* устарело, функциональность заменена perf.
- *wireless-tools* устарело и заменено iw.
- *xcmiscproto* заменено xorgproto.
- *xextproto* заменено xorgproto.
- *xf86dgaproto* заменено xorgproto.
- *xf86driproto* заменено xorgproto.
- *xf86miscproto* заменено xorgproto.
- *xf86-video-omapfb* устарело, взамен используется драйвер ядра modesetting.
- *xf86-video-omap* устарело, взамен используется драйвер ядра modesetting.
- *xf86vidmodeproto* заменено xorgproto.
- *xineramaproto* заменено xorgproto.
- *xproto* заменено xorgproto.
- *yasm* больше не требуется, поскольку прежнее использование заменено nasm.

4.14.3. Изменения в пакетах

- *cmake* - файлы cmake.m4 и инструментарий перенесены в основной пакет.
- *iptables* - модули размещены в отдельных пакетах.

- *alsa-lib* - библиотека libasound перенесена в основной пакет alsa-lib из libasound.
- *glibc* - libnss-db теперь в своём пакете вместе со сценарием /var/db/makedbs.sh для обновления баз данных.
- *python и python3* - основной пакет удалён из задания. Нужно установить конкретные пакеты python-modules или python3-modules для остального.
- *systemtap* - systemtap-exporter перемещён в отдельный пакет.

4.14.4. Зависимости протокола XOrg

Находящиеся в разработке репозитории *proto объединены в xorgproto, поэтому соответствующие задания также были объединены в xorgproto. Задания, зависимые от старых заданий *proto, следует исправить с заменой на xorgproto. Имена удалённых при объединении заданий приведены в параграфе 4.14.2. Удалённые задания.

4.14.5. Предотвращение зависимости сборщиков в do_configure

Ранее для заданий Python, наследующих классы distutils и distutils3, при выполнении задачи do_configure можно было выполнить зависимости, заданные в setup.py, если они не были указаны в sysroot (т. е. задания, указывающие зависимости не были включены в DEPENDS).

Это изменение влияет не только на упомянутые классы distutils и distutils3, но и на все задания, наследующие distutils*, например, на задания setuptools и setuptools3.

Извлечение типов зависимостей, не указанных в sysroot, снижает производительность сборки, поэтому такая выборка была явно отключена. В результате все отсутствующие в заданиях Python зависимости при использовании этих классов будут давать ошибку в задаче do_configure.

4.14.6. Решена проблема настройки аудита в linux-yocto

В результате ошибки функции аудита в ядре не записывались предупреждения при сборке. Сейчас эта проблема решена. Предупреждения можно видеть в пользовательских конфигурациях с заданием для ядра linux-yocto.

4.14.7. Именование элементов образов и ядер

- Имена переменных (например, IMAGE_NAME) используют новую переменную IMAGE_VERSION_SUFFIX вместо DATETIME, что упрощает изменения. Переменная IMAGE_VERSION_SUFFIX задаётся файлом bitbake.conf в форме IMAGE_VERSION_SUFFIX = "-${DATETIME}".
- Имена некоторых переменных изменены для согласованности, как показано в таблице.

Старое имя	*Новое имя*
KERNEL_IMAGE_BASE_NAME	KERNEL_IMAGE_NAME
KERNEL_IMAGE_SYMLINK_NAME	KERNEL_IMAGE_LINK_NAME
MODULE_TARBALL_BASE_NAME	MODULE_TARBALL_NAME
MODULE_TARBALL_SYMLINK_NAME	MODULE_TARBALL_LINK_NAME
INITRAMFS_BASE_NAME	INITRAMFS_NAME

- Переменная MODULE_IMAGE_BASE_NAME удалена, имя архива контролируется напрямую переменной MODULE_TARBALL_NAME.
- Переменные KERNEL_DTB_NAME и KERNEL_DTB_LINK_NAME добавлены для управления элементами дерева устройств ядра (DTB[1]) вместо переменных KERNEL_IMAGE_*.
- Добавлены переменные KERNEL_FIT_NAME и KERNEL_FIT_LINK_NAME для задания имени плоского дерева образа (FIT[2]) для образов ядра подобно другим развёрнутым элементам.
- Значения переменных MODULE_TARBALL_NAME и MODULE_TARBALL_LINK_NAME больше не включают префиксов module- и суффиксов .tgz. Эти части сейчас кодируются жёстко для согласованности с другими переменными именования элементов.
- Добавлена переменная INITRAMFS_LINK_NAME для контроля символьных ссылок подобно другим элементам.
- INITRAMFS_NAME сейчас использует ${PKGE}-${PKGV}-${PKGR}-${MACHINE}${IMAGE_VERSION_SUFFIX} вместо ${PV}-${PR}-${MACHINE}-${DATETIME} для соответствия другим переменным.

4.14.8. Отмена SERIAL_CONSOLE

Переменная SERIAL_CONSOLE функционально заменена переменной SERIAL_CONSOLES некоторое время назад. С выпуска YP 2.6 переменная SERIAL_CONSOLE отменена официально. SERIAL_CONSOLE будет работать как в прежних выпусках, но для совместимости с новыми версиями рекомендуется заменить её на SERIAL_CONSOLES. Различия между этими переменными заключается в том, что в SERIAL_CONSOLES элементы записи разделяются точкой с запятой вместо пробела, а также возможно указание нескольких консолей, разделенных пробелами.

4.14.9. Сценарий настройки считает неизвестные опции ошибкой

Если конфигурационный сценарий находит незнакомую опцию, это вызывает ошибку a QA вместо предупреждения. Это может потребовать внесения изменений в имеющиеся задания.

4.14.10. Изменение переопределений

- Удалены virtclass-native и virtclass-nativesdk, отменённые в 2012 г. с заменой на class-native и class-nativesdk. Переопределения virtclass-multilib- для multilib сохраняются.

[1]Device Tree Binary.
[2]Flattened image tree.

- *Более высокий приоритет forcevariable по сравнению с libc* был документирован достаточно давно, однако давняя причуда с установкой OVERRIDES давала переопределениям libc (например libc-glibc, libc-musl и т. п.) более высокий приоритет. Сейчас это устранено.

  Это изменение не должно вызывать проблем, однако в некоторых необычных конфигурациях возможно иное поведение, нежели предполагалось ранее. Нужно проверить переопределения forcevariable и libc-* на пользовательских уровнях в части их осмысленности.

- *Удалено build-${BUILD_OS},* которое обычно имело форму build-linux, поскольку сборка в ОС хоста, отличной от последней версии Linux не поддерживается и не рекомендуется. Отказ от переопределения позволяет избежать ложного впечатления о поддержке других версий операционной системы.

- *Оператор _remove сохраняет пробелы,* поэтому при указании списка удаляемых элементов следует помнить, что пробелы в начала и в конце не удаляются (см. примеры в разделе Removal (Override Style Syntax) [4]).

4.14.11. Конфигурация systemd выделена в systemd-conf

Конфигурация задания systemd перенесена в system-conf, что позволяет избавить systemd от машинозависимости при необходимости применить зависимую от машину конфигурацию (например, для машин qemu*). Задание включает:

```
${sysconfdir}/machine-id
${sysconfdir}/systemd/coredump.conf
${sysconfdir}/systemd/journald.conf
${sysconfdir}/systemd/logind.conf
${sysconfdir}/systemd/system.conf
${sysconfdir}/systemd/user.conf
```

если ранее применялись файлы bbappend для добавления systemd с целью изменить какой-то из приведённых выше файлов, сейчас это нужно делать в задании systemd-conf.

4.14.12. Изменение автоматического тестирования

- *Удалена переменная TEST_IMAGE*, которая при установке значения 1 включала автоматическое тестирование сборки образа. Сейчас вместо этого используется значение переменной TESTIMAGE_AUTO.

- *Наследование классов testimage и testsdk*. Опыт диктует использование переменной IMAGE_CLASSES, а не INHERIT при наследовании классов testimage и testsdk для автоматического тестирования.

4.14.13. Изменение OpenSSL

Пакет OpenSSL обновлён с версии 1.0 до 1.1. Обновление может создать проблемы для заданий, использующих обе версии в своих цепочках зависимостей, поскольку обе версии не могут быть установлены одновременно при сборке.

При работе могут применяться обе версии библиотеки.

4.14.14.Изменение BitBake

Журнальный файл сервера bitbake-cookerdaemon.log всегда присутствует в каталоге сборки, а не в текущем каталоге.

4.14.15. Изменение защиты

Дистрибутив Poky по умолчанию использует флаги компилятора для защиты, которые могут вызывать проблемы в результате более строгой проверки возможных проблем безопасности в коде.

4.14.16. Изменение пост-установки

Нужно явно указать задержку пост-установки до целевой платформы. Если нужно явно отложить пост-установку на целевой платформе до перезагрузки после инсталляции, следует использовать pkg_postinst_ontarget() или вызов postinst-intercepts defer_to_first_boot из pkg_postinst(). Любой отказ сценария pkg_postinst(), включая exit 1, будет вызывать ошибку задачи do_rootfs. Информация о пост-установке приведена в разделе Post-Installation Scripts [2].

4.14.17. Оптимизация на основе профилей Python 3

Задание python3 сейчас включает оптимизацию по профилю, что несколько увеличивает время сборки для повышения производительности работы на целевой платформе. Оптимизация включается лишь при поддержке текущей машиной MACHINE пользовательского режима эмуляции в QEMU ("qemu-usermode" присутствует в MACHINE_FEATURES, как принято по умолчанию).

Если нужно отключить оптимизацию для любого значения MACHINE_FEATURES, следует убедиться в отсутствии pgo в переменной PACKAGECONFIG для задания python3. Это можно сделать, указав на уровне конфигурации PACKAGECONFIG_remove_pn-python3 = "pgo" или задав PACKAGECONFIG в файле добавления для python3.

4.14.18. Прочие изменения

- По умолчанию используется набор инструкций Thumb-2 для armv7a и выше. При наличии заданий для сборки программ, которым нужен набор инструкций ARM, его можно задать в виде ARM_INSTRUCTION_SET = "arm".

- run-postinsts больше не использует /etc/*-postinsts для dpkg/opkg, применяя встроенную поддержку пост-установки. Поведение RPM не изменилось.

- Переменные NOISO и NOHDD больше не применяются. Можно управлять сборкой образов *.iso и *.hddimg напрямую через переменную IMAGE_FSTYPES.

- От сценария scripts/contrib/mkefidisk.sh отказались в пользу Wic.

- Задание kernel-modules удалено из RRECOMMENDS для машин qemumips и qemumips64. Исключено также влияние файла x86-base.inc. Для машин genericx86 и genericx86-64 задание kernel-modules сохранено в переменной RRECOMMENDS.
- Удалена переменная LGPLv2_WHITELIST_GPL-3.0. При наличии её в конфигурации следует вместо неё установить или дополнить переменную WHITELIST_GPL-3.0.
- ${ASNEEDED} сейчас напрямую включается в переменную TARGET_LDFLAGS. Другие определения из meta/conf/distro/include/as-needed.inc перенесены в соответствующие задания.
- Поддержка ключей DSA для хоста исключена из заданий OpenSSH. Если такие ключи продолжают использоваться, следует перейти на более защищённые алгоритмы, рекомендуемые OpenSSH.
- Задание dhcp сейчас применяет конфигурационный файл dhcpd6.conf в dhcpd6.service для IPv6 DHCP вместо файла dhcpd.conf, который сейчас служит для IPv4.

4.15. Переход на YP 2.7

4.15.1. BitBake

- BitBake проверяет анонимные (anonymous) и чистые (pure) функции Python (например, def funcname:) в метаданных на предмет использования символов табуляции с выдачей предупреждений при их наличии.
- Bitbake проверяет BBFILE_COLLECTIONS на предмет дубликатов с возвратом ошибки при обнаружении.

4.15.2. Исключена поддержка Eclipse™

Поддержка Eclipse IDE исключена, но сохраняется для выпусков до 2.7. Выпуск 2.7 не включает плагин Eclipse Yocto.

4.15.3. Разделение системной и пользовательской части в qemu-native

Системная и пользовательская часть задания qemu-native разделены. Задание qemu-native обеспечивает пользовательские компоненты, а qemu-system-native - системные. При наличии заданий, зависящих от функциональности системы эмуляции QEMU при сборке, следует заменить в них qemu-native на qemu-system-native.

4.15.4. Удаление файла upstream-tracking.inc

Ранее отменённый файл upstream-tracking.inc удалён. Все переменные UPSTREAM_TRACKING* устанавливаются в соответствующих заданиях. Следует удалить в своей конфигурации все ссылки на файл upstream-tracking.inc.

4.15.5. Удаление переменной DISTRO_FEATURES_LIBC

Переменная DISTRO_FEATURES_LIBC больше не применяется. Возможность настройки glibc с помощью kconfig была удалена некоторое время назад и свойства libc-* утратили силу. Следует удалить DISTRO_FEATURES_LIBC из своих заданий.

4.15.6. Корректировки значений лицензий

- *Socat* - значение LICENSE изменено на GPLv2 вместо GPLv2+.
- *libgfortran* - задана лицензия GPL-3.0-with-GCC-exception.
- *elfutils* - удалено значение Elfutils-Exception и установлено GPLv2 для общих библиотек.

4.15.7. Изменение подготовки пакетов

- *bind* - двоичный файл nsupdate перенесён в пакет bind-utils.
- *Разделение отладки*, принятое по умолчанию, было изменено для создания отдельных пакетов с исходным кодом (package_name-dbg и package_name-src). При использовании dbg-pkgs в IMAGE_FEATURES для получения отладочных символов можно добавить в переменную src-pkgs, если нужны исходные коды. Пакеты с исходным кодом по умолчанию сохранены в SDK, если в SDKIMAGE_FEATURES не задано их исключение.
- *Монтирование с помощью util-linux*. Файл /etc/default/mountall перенесён в субпакет mount.
- Разделение двоичных файлов util-linux - сейчас каждый двоичный исполняемый файл выделен в свой пакет для более тонкого управления. Основной пакет util-linux втягивает отдельные двоичные пакеты с помощью переменных RRECOMMENDS и RDEPENDS. В результате образы не будут иметь видимых изменений, пока не установлена переменная NO_RECOMMENDATIONS.
- *netbase/base-files* - файл /etc/hosts перенесён из netbase в base-files.
- *tzdata* - основной пакет преобразован в пустой мета-пакет, который по умолчанию втягивает все пакеты tzdata.
- *lrzsz* удалён из packagegroup-self-hosted и packagegroup-core-tools-testapps. Потребность в X/Y/ZModem маловероятна в современных системах. Если вы включали пакет lrzsz на основе упомянутых групп, потребуется добавить его явно.

4.15.8. Удалённые задания

- *gcc* - задания версии 7.3 исключены, версия 8.3 сохраняется.
- *linux-yocto* - исключены задания версий 4.14 и 4.18, версии 4.19 и 5.0 сохранены.
- *go* - исключены задания версии 1.9, версии 1.11 и 1.12 сохранены.
- *xvideo-tests* устарело.

- *libart-lgpl* устарело.
- *gtk-icon-utils-native* сейчас предоставляется заданием gtk+3-native
- *gcc-cross-initial* больше не нужно, взамен используется gcc-cross/gcc-crosssdk.
- *gcc-crosssdk-initial* больше не нужно, взамен используется gcc-cross/gcc-crosssdk.
- *glibc-initial* удалено, поскольку преимущества наличия для site_config перевешиваются сложностью сборки.

4.15.9. Удалённые классы

- *distutils-tools* никогда не использовался.
- *bugzilla.bbclass* устарел.
- *distrodata* - функциональность заменена более современной реализацией на основе tinfoil.

4.15.10. Прочие изменения

- *Каталог distro* репозитория Poky удалён из каталога сценариев верхнего уровня.
- Perl сейчас выполняет сборку для целевых систем с использованием perl-cross для улучшения сопровождаемости и повышения производительности сборки. Это изменение не должно вызывать проблем, если вы не меняли сильно своё задание Perl.
- *arm-tunes* - удалена опция -march, если уже добавлено mcpu.
- *update-alternatives* - файл преобразования переименован в PACKAGE_PREPROCESS_FUNCS.
- *base/pixbufcache* - удалён устаревший код sstatecompletions.
- *native class* - включена обработка RDEPENDS.
- *inetutils* - в задании отключена оболочка rsh.

Глава 5. Структура каталога исходных кодов

Дерево исходных кодов содержит множество компонент и понимание их роли важно для работы с YP. В этой главе описана структура дерева, а также назначение файлов и каталогов. Рекомендации по организации локального дерева исходного кода даны в разделе Locating Yocto Project Source Files [2].

Система сборки OE не поддерживает имена файлов и каталогов с пробелами.

5.1. Компоненты верхнего уровня

В этом разделе описаны компоненты верхнего уровня в дереве исходных кодов.

5.1.1. bitbake/

Этот каталог включает копию BitBake для простоты использования. Копия обычно соответствует текущему стабильному выпуску BitBake. Интерпретатор метаданных BitBake считывает YP Metadata и запускает задачи, определённые в них. Отказы обычно связаны с метаданными, а не с BitBake, поэтому большинству пользователей не нужно думать о BitBake.

Можно запускать исполняемый файл bitbake, хранящийся в каталоге bitbake/bin/. Сценарий настройки окружения oe-init-build-env помещает каталоги сценариев и bitbake/bin (в указанном порядке) в переменную окружения PATH.

Информация о BitBake приведена в [4].

5.1.2. build/

Этот каталог содержит пользовательские файлы конфигурации и вывод системы сборки OE в её стандартной конфигурации, где вывод размещается вместе с деревом исходных кодов. Каталог сборки создаётся при запуске сценария настройки окружения OE (oe-init-build-env).

Можно отделить вывод и конфигурационные файлы от каталога исходных кодов, указав имя каталога при запуске установочного сценария, описанного в параграфе 5.1.9.1 oe-init-build-env.

5.1.3. documentation/

Этот каталог содержит исходные файлы документации YP, а также шаблоны и инструменты для создания руководств в формате PDF и HTML. Каждое руководство размещено в своём каталоге, например, файлы для этого документа расположены в каталоге ref-manual/.

5.1.4. meta/

В этом каталоге содержатся метаданные OpenEmbedded-Core, задания, базовые классы, конфигурации машин для эмулируемых платформ (qemux86, qemuarm и т. п.).

5.1.5. meta-poky/

Конфигурация для эталонного дистрибутива Poky.

5.1.6. meta-yocto-bsp/

Этот каталог содержит эталонные пакеты поддержки плат (BSP). Дополнительные сведения о BSP можно найти в [6].

5.1.7. meta-selftest/

Каталог с дополнительными заданиями и файлами добавления, используемые при самотестировании OE для контроля процесса сборки. Если самотестирование не требуется, не следует добавлять этот уровень в свой файл bblayers.conf.

5.1.8. meta-skeleton/

Каталог с шаблонами заданий для разработки BSP и ядер.

5.1.9. scripts/

Этот каталог содержит сценарии, расширяющие функциональность среды YP (например, QEMU). Сценарий oe-init-build-env добавляет этот каталог в переменную окружения PATH. Каталог включает сценарии, помогающие внести свой вклад в развитие YP, например, create-pull-request и send-pull-request.

5.1.9.1 oe-init-build-env

Этот сценарий организует среду сборки OE. Сценарий запускается с помощью команды source и меняет значение переменной PATH, а также устанавливает переменные BitBake с учётом текущего каталога. Сценарий установки окружения нужно запускать до использования команд BitBake. Сценарий использует другие сценарии из каталога scripts для выполнения большого объёма работ. При запуске сценария создаётся рабочая среда YP и каталог сборки (Build Directory), который делается рабочим, а также выводится список базовых целей BitBake, как показано ниже.

```
$ source oe-init-build-env

### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    core-image-sato
    meta-toolchain
    meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'
```

Сценарий получает принятые по умолчанию значения из файла conf-notes.txt в каталоге meta-poky внутри каталога исходных кодов (Source Directory). Если у вас нестандартный дистрибутив, легко изменить этот конфигурационный файл для включения нужных целей для вашего дистрибутива (Creating a Custom Template Configuration Directory [2]).

По умолчанию при работе этого сценария без аргумента Build Directory создаётся каталог build в текущем рабочем каталоге. Если аргумент Build Directory задан при запуске сценария, система сборки OE создаст указанный каталог. Например, приведённая ниже команда создаст каталог сборки mybuilds вне дерева исходных кодов.

```
$ source oe-init-build-env ~/mybuilds
```

Система сборки OE использует шаблон конфигурационных файлов, который по умолчанию расположен в каталоге meta-poky/conf каталога исходных кодов (см. раздел Creating a Custom Template Configuration Directory [2]).

Система сборки OE не поддерживает имена файлов и каталогов с пробелами. При попытке запуска сценария oe-init-build-env из каталога с пробелами в имени каталога или содержащихся в нем файлов, сценарий будет завершён с ошибкой, указывающей отсутствие каталога. Убедитесь в том, что имя Source Directory не содержит пробелов.

5.1.10. LICENSE, README и README.hardware

Эти файлы являются стандартными для верхнего уровня.

5.2. Каталог сборки build/

Система сборки OE создает сборочный каталог при запуске сценария организации среды (oe-init-build-env). Если сценарию не указан каталог сборки, будет создан каталог build в текущем каталоге. Переменная TOPDIR указывает на каталог сборки.

5.2.1. build/buildhistory

Система сборки OE создает этот каталог при включении функции истории сборки. Каталог отслеживает данные сборки по образам, пакетам и SDK (см. раздел Maintaining Build Output Quality [2]).

5.2.2. build/conf/local.conf

Этот файл конфигурации содержит все локальные пользовательские настройки для среды сборки. Опции в файле прокомментированы. Любая переменная, задаваемая здесь, переопределяет заданную в другом месте переменную, если та не использует жёсткое кодирование (например, = вместо ?=). Такое кодирование применяется для некоторых переменных, но это происходит достаточно редко.

Следует установить в этом файле для переменной MACHINE значения, соответствующее сборке, указать типы пакетов (PACKAGE_CLASSES) и мест, откуда нужно загружать файлы (DL_DIR).

Если файла local.conf нет, система OE создает его по образцу local.conf.sample при настройке окружения (oe-init-build-env). Используемый образец зависит от переменной $TEMPLATECONF, которая по умолчанию указывает meta-poky/conf при сборке из среду разработки YP и meta/conf при сборке из среды OpenEmbedded-Core. Поскольку переменная в сценарии указывает файл local.conf.sample, это позволяет настроить среду сборки из любого уровня, указав в сценарии настройки окружения на верхнем уровне сборки TEMPLATECONF=your_layer/conf. Когда процесс сборки получает файл-образец, используется утилита sed для подстановки значения ${OEROOT} вместо всех ##OEROOT##.

С использованием переменной TEMPLATECONF можно ознакомиться в файле scripts/oe-setup-builddir дерева исходных кодов. Версия файла local.conf.sample для YP находится в каталоге meta-poky/conf.

5.2.3. build/conf/bblayers.conf

Этот файл определяет уровни, являющиеся деревьями каталогов, через которые проходит BitBake. Переменная BBLAYERS в этом файле содержит список уровней, которые BitBake пытается найти.

Если файла bblayers.conf нет при запуске сборки, система OE создает его по образцу bblayers.conf.sample при выполнении сценария настройки среды сборки (oe-init-build-env). Используемый образец зависит от переменной $TEMPLATECONF, которая по умолчанию для среды YP содержит значение meta-poky/conf, а для OpenEmbedded-Core - meta/conf. Поскольку переменная в сценарии указывает файл bblayers.conf.sample file, это позволяет настроить среду сборки из любого уровня, указав в сценарии настройки окружения на верхнем уровне сборки TEMPLATECONF=your_layer/conf. Когда процесс сборки получает файл-образец, используется утилита sed для подстановки значения ${OEROOT} вместо всех ##OEROOT##.

С использованием переменной TEMPLATECONF можно ознакомиться в файле scripts/oe-setup-builddir дерева исходных кодов. Версия файла local.conf.sample для YP находится в каталоге meta-poky/conf.

5.2.4. build/conf/sanity_info

Этот файл показывает статус проверки работоспособности и создаётся в процессе сборки.

5.2.5. build/downloads/

Этот каталог содержит загруженные архивы исходных кодов. Каталог можно применять для множества сборок или перенести в другое место, указываемое переменной DL_DIR.

5.2.6. build/sstate-cache/

Этот каталог содержит файлы общего кэша состояний. Каталог можно применять для множества сборок или перенести в другое место, указываемое переменной SSTATE_DIR.

5.2.7. build/tmp/

Система сборки OE создает и использует этот каталог для вывода всех результатов сборки. Каталог указывается переменной TMPDIR. При отсутствии каталога BitBake создает его. При необходимости очистить результаты сборки и начать её с нуля (все кроме загрузки источников) можно удалить содержимое каталога tmp или сам каталог. В этом случае следует также удалить каталог общего кэша состояний build/sstate-cache.

5.2.8. build/tmp/buildstats/

Каталог с данными статистики сборки.

5.2.9. build/tmp/cache/

При разборе метаданных (задания и файлы конфигурации) BitBake кэширует результаты в каталоге build/tmp/cache/ для ускорения сборки в будущем. Результаты организованы по машинам. При последующей сборке BitBake проверяет каждое задание (вместе с включёнными и добавленными к нему файлами) на предмет изменений. Изменения могут детектироваться по времени редактирования файлов (mtime) или хэш-значениям их содержимого. Если файлы не менялись, используется кэшированный результат, а при наличии изменений файлы анализируются заново.

5.2.10. build/tmp/deploy/

В этом каталоге содержаться «конечные результаты» процесса сборки OE. Каталог указывается переменной DEPLOY_DIR. Содержимое каталога описано в разделах Images и Application Development SDK [1].

5.2.11. build/tmp/deploy/deb/

В этом каталоге хранятся файлы .deb, созданные при сборке. Пакеты сортируются в хранилища по типу архитектуры.

5.2.12. build/tmp/deploy/rpm/

В этом каталоге хранятся файлы .rpm, созданные при сборке. Пакеты сортируются в хранилища по типу архитектуры.

5.2.13. build/tmp/deploy/ipk/

В этом каталоге хранятся файлы . ipk, созданные при сборке.

5.2.14. build/tmp/deploy/licenses/

Этот каталог получает данные о лицензировании пакетов. Например, он может содержать подкаталоги для bash, busybox, glibc и т. п., содержащие файлы COPYING с описанием лицензирования. Информация о лицензировании приведена в разделе Maintaining Open Source License Compliance During Your Product's Lifecycle [2]).

5.2.15. build/tmp/deploy/images/

Этот каталог получает окончательные образы файловых систем. Если полученный образ нужно записать на устройство, его следует искать здесь. Следует соблюдать осторожность при удалении файлов из этого каталога. Можно безопасно удалять старые образы (например, core-image-*), однако загрузчики ядра (*zImage*, *uImage* и т. п.) и дополняющие их файлы могут быть развёрнуты в каталоге до создания образов. Поскольку эти файлы не создаются непосредственно из образа, при их удалении они не будут автоматически созданы заново при последующей сборке.

При нечаянном удалении таких файлов их придётся принудительно создавать заново. Для этого нужно знать цели, с которыми связаны файлы. Например, можно заново собрать файлы командами

```
$ bitbake -c clean virtual/kernel
$ bitbake virtual/kernel
```

5.2.16. build/tmp/deploy/sdk/

Система сборки OE создает этот каталог для размещения сценариев установки инструментов, выполнение которых обеспечивает инсталляцию файловой системы sysroot, соответствующей целевой платформе. Описание сценариев установки дано в разделе Building an SDK Installer [7].

5.2.17. build/tmp/sstate-control/

Система сборки OE использует этот каталог для файлов манифеста общего состояния. Код общего состояния использует эти файлы для записи установленных каждой задачей sstate файлов, чтобы эти файлы можно было удалять при очистке задания или установке новой версии. Система сборки также использует файлы манифестов для обнаружения перекрытий между задачами и выдачи предупреждений.

5.2.18. build/tmp/sysroots-components/

Этот каталог содержит компоненты sysroot, которые задача do_prepare_recipe_sysroot указывает (link) или копирует в зависимые от заданий каталоги sysroot каждого задания из DEPENDS. Заполнение этого каталога выполняется через общее состояние, а путь указывается переменной COMPONENTS_DIR. За исключением нескольких экзотических случаев обработка sysroots-components происходит автоматически и заданиям не нужно ссылаться напрямую на build/tmp/sysroots-components.

5.2.19. build/tmp/sysroots/

В прежних версиях OE каталог служил для создания глобального общего корня sysroot на уровне машины вместе с естественным sysroot. Начиная с YP 2.7.1, каталоги sysroot размещаются в рабочих каталогах заданий (WORKDIR) и build/tmp/sysroots/ не используется. Однако build/tmp/sysroots/ можно по-прежнему заполнять с помощью команды bitbake build-sysroots и использовать для совместимости. В общем случае использование этого каталога не рекомендуется, следует применять отдельные sysroot для заданий.

5.2.20. build/tmp/stamps/

В этом каталоге хранится информация, которую BitBake использует для отслеживания выполняемых задач. Каталог содержит подкаталоги по архитектуре, имени и версии пакета. Например,

```
stamps/all-poky-linux/distcc-config/1.0-r0.do_build-2fdd....2do
```

Хотя файлы в этой структуре не содержат данных, BitBake может отслеживать задачи по именам и временным меткам (см. раздел Stamp Files and the Rerunning of Tasks [1]).

5.2.21. build/tmp/log/

Этот каталог содержит журналы сборки, например, вывод задач do_check_pkg и do_distro_check. Запуск сборки не обязательно создает этот каталог.

5.2.22. build/tmp/work/

Этот каталог содержит зависимые от архитектуры рабочие каталоги для пакетов, собираемых BitBake. Все задачи выполняются из соответствующих рабочих каталогов. Например, исходные коды для конкретного пакета распаковываются, правятся (patch), настраиваются и компилируются в своём рабочем каталоге. Внутри таких каталогов организация основана на группах пакетов и версиях, для которых будет компилироваться исходный код в соответствии с WORKDIR.

В качестве примера рабочего каталога рассмотрим linux-yocto-kernel-3.0 для машины qemux86. Рабочим каталогом для этого пакета будет tmp/work/qemux86-poky-linux/linux-yocto/3.0+git1+<.....>, указанный в переменной WORKDIR. В этом каталоге распаковываются исходные коды для linux-qemux86-standard-build, а затем накладываются исправления с помощью Quilt. (см. раздел Using Quilt in Your Workflow [2]). В каталоге linux-qemux86-standard-build создаются стандартные каталоги Quilt linux-3.0/patches и linux-3.0/.pc и могут применяться стандартные команды Quilt.

В WORKDIR создаются и другие каталоги, наиболее важным из которых является WORKDIR/temp/, где хранятся журналы (log-файлы) для каждой задачи в форме log.do_*.pid, а также сценарии BitBake для каждой задачи в форме run.do_*.pid. В каталоге WORKDIR/image/ команда make install размещает выходные файлы, которые затем делятся по пакетам в WORKDIR/packages-split/.

5.2.23. build/tmp/work/tunearch/recipename/version/

Рабочий каталог задания - ${WORKDIR}. Как описано в параграфе 5.2.19. build/tmp/sysroots/, начиная с выпуска YP 2.7.1, система OE собирает каждое задание в своём каталоге (WORKDIR). Путь к рабочему каталогу создаётся с использованием архитектуры для данной сборки (например, TUNE_PKGARCH, MACHINE_ARCH или allarch), имени задания и его версии (например, PE:PV-PR).

В рабочем каталоге каждого задания имеется множество подкаталогов:

- *${WORKDIR}/temp* содержит журналы сборки для каждой задачи, файлы run для каждой выполненной задачи и файл log.task_order, указывающий порядок выполнения задач;
- *${WORKDIR}/image* содержит вывод задачи do_install, соответствующей её переменной ${D};
- *${WORKDIR}/pseudo* содержит псевдобазу данных и журнал для всех задач pseudo в задании.
- *${WORKDIR}/sysroot-destdir* содержит вывод задачи do_populate_sysroot;
- *${WORKDIR}/package* содержит вывод задачи do_package да разделения по отдельным пакетам;

- *${WORKDIR}/packages-split* содержит вывод задачи do_package после разделения по пакетам с каталогами для каждого пакета, созданного заданием;
- *${WORKDIR}/recipe-sysroot* заполняется зависимостями для цели задания и выглядит подобно целевой файловой системе, включая библиотеки, с которыми заданию может потребоваться связь (например, библиотека C);
- *${WORKDIR}/recipe-sysroot-native* заполняется естественными зависимостями задания и содержит инструменты, требуемые для сборки задания (например, компилятор autoconf, libtool и т. п.).
- *${WORKDIR}/build* применяется лишь для заданий, поддерживающих сборки, где исходный код отделен от результатов; система OE использует этот каталог в качестве отдельного каталога сборки (${B}).

5.2.24. build/tmp/work-shared/

Для эффективности система сборки OE создает и использует этот каталог для хранения заданий, использующих общий с другими заданиями рабочий каталог. На практике это применяется для gcc и вариантов (gcc-cross, libgcc, gcc-runtime и т. п.).

5.3. Метаданные - meta/

5.3.1. meta/classes/

Этот каталог содержит файлы классов *.bbclass, включающие многократно используемый пакетами код. Каждый пакет наследует класс base.bbclass. Примером другого важного наследуемого класса является autotools.bbclass, теоретически позволяющий любому пакету, основанному на autotools работать с YP без больших усилий. Класс kernel.bbclass содержит базовый код и функции для работы с ядром Linux. Такие функции как генерация образов или подготовка пакетов имеют свои классы (image.bbclass, rootfs_*.bbclass и package*.bbclass). Описанию классов посвящена Глава 6. Классы.

5.3.2. meta/conf/

Этот каталог содержит основной набор конфигурационных файлов (начиная с bitbake.conf) в которые включены все остальные файлы конфигурации. Операторы include в конце файла bitbake.conf указывают даже файл local.conf. Хотя bitbake.conf устанавливает принятые по умолчанию значения, их можно переопределить в файле local.conf, а также в файле конфигурации машины или дистрибутива.

5.3.3. meta/conf/machine/

Этот каталог содержит файлы конфигурации машины. Если установлено MACHINE = "qemux86", система сборки OE будет искать файл qemux86.conf в этом каталоге. Каталог include содержит общие данные для разных машин. Если нужна поддержка новой машины в YP, следует заглянуть в этот каталог.

5.3.4. meta/conf/distro/

Содержимое этого каталога определяет конфигурацию дистрибутива. Для YP основным файлом служит defaultsetup.conf. Каталог включает версии и определения SRCDATE для настроенных здесь приложений. Примером дополнительной конфигурации может служить poky-bleeding.conf, хотя он в основном наследует конфигурацию Poky.

5.3.5. meta/conf/machine-sdk/

Система сборки OE ищет в этом каталоге файлы конфигурации, соответствующие значению SDKMACHINE. По умолчанию в YP включены файлы для 32- и 64-битовых систем x86, поддерживающие некоторые хосты SDK. Можно расширить поддержку SDK, добавив файлы конфигурации в каталоги соответствующих уровней.

5.3.6. meta/files/

Этот каталог содержит базовые файлы лицензий и несколько текстовых файлов, используемых системой сборки. В текстовых файлах приведена минимальная информация об устройствах, а также список файлов и каталогов с известными правами доступа.

5.3.7. meta/lib/

Этот каталог содержит код библиотек OE Python, используемых при сборке.

5.3.8. meta/recipes-bsp/

Каталог содержит привязки к конкретному оборудованию или конфигурации для оборудования, такие как u-boot и grub.

5.3.9. meta/recipes-connectivity/

Каталог содержит библиотеки и приложения для коммуникаций с другими устройствами.

5.3.10. meta/recipes-core/

Каталог содержит компоненты, требуемые для сборки базового образа Linux, включая основные зависимости.

5.3.11. meta/recipes-devtools/

Каталог с инструментами, используемыми системой сборки, которые могут применяться и целевой платформой.

5.3.12. meta/recipes-extended/
Каталог со второстепенными приложениями, добавляющими свойства к основному образу. Эти приложения могут быть нужны для выполнения требований LSB[1].

5.3.13. meta/recipes-gnome/
Каталог, содержащий все, что требуется для приложений GTK+.

5.3.14. meta/recipes-graphics/
Каталог с X-библиотеками и другими библиотеками, относящимися к графике.

5.3.15. meta/recipes-kernel/
Каталог с ядром и базовыми приложениями и библиотеками, от которых зависит ядро.

5.3.16. meta/recipes-lsb4/
Задания, добавленные для поддержки LSB версии 4.x.

5.3.17. meta/recipes-multimedia/
Кодеки и утилиты поддержки для звука, изображений и видео.

5.3.18. meta/recipes-rt/
Каталог с заданиями для пакетов и образов, применяемых при использовании и тестировании ядер PREEMPT_RT.

5.3.19. meta/recipes-sato/
Эталонный дистрибутив Sato, а также связанные с ним приложения и данные конфигурации.

5.3.20. meta/recipes-support/
Каталог с заданиями, используемыми другими заданиями, но не включаемыми напрямую в образ (зависимости).

5.3.21. meta/site/
Список кэшированных результатов для разных вариантов архитектуры. Поскольку некоторые результаты тестов autoconf не могут быть определены при кросс-компиляции по причине недоступности запуска в live-системе, данные из этого каталога передаются autoconf для разных вариантов архитектуры.

5.3.22. meta/recipes.txt
Файл с описанием содержимого recipes-*.

Глава 6. Классы
Файлы классов служат абстракцией функциональности и совместно используются множеством файлов заданий (.bb). Для использования файла класса нужно просто наследовать соответствующий класс в задании. В большинстве случаев наследования класса заданием достаточно для включения функций класса. Однако в некоторых случаях может потребоваться также задание переменных или переопределение принятого по умолчанию поведения.

Любые метаданные, обычно встречающиеся в задании, могут также помещаться в файл класса. Файлы классов используют расширение .bbclass и обычно размещаются в каталоге classes/ каталога meta*/ в дереве исходных кодов. Файлы классов также указываются переменной BUILDDIR (например, build/), как и файлы .conf в каталоге conf. Поиск файлов классов в BBPATH выполняется так же, как и файлов .conf.

В этой главе рассмотрены наиболее важные и полезные классы.

6.1. allarch.bbclass
Класс allarch наследуется заданиями, не создающими зависящего от архитектуры вывода. Класс отключает функциональность, которая обычно нужна для создания исполняемых двоичных файлов (например, создание кросс-компилятора и библиотеки C в качестве предварительного условия, а также выделение отладочных символов).

В отличие от заданий для дистрибутивов (например, Debian), задания OE создают пакеты, которые зависят от настройки переменных RDEPENDS и TUNE_PKGARCH и не должны настраиваться под все архитектуры с помощью allarch. Это происходит даже в тех случаях, когда задание не создает зависимого от архитектуры вывода.

Настройка таких заданий для всех вариантов архитектуры заставляет задачи do_package_write_* tasks иметь разные подписи для машин с разными настройками. Кроме того, происходят ненужные повторы сборки при создании образа для другого значения MACHINE даже при отсутствии изменений в задании.

По умолчанию все задания наследуют классы base и package, которые обеспечивают функциональность, требуемую для создания исполняемых файлов. Если ваше задание, например, производит лишь пакеты с файлами конфигурации, медиа-файлами или сценариями (например, Python и Perl), им нужно наследовать класс allarch.

6.2. archiver.bbclass
Класс archiver поддерживает выпуск исходных кодов и других материалов в двоичных файлах (архивах). Дополнительная информация об архивах исходных кодов приведена в разделе Maintaining Open Source License

[1]Linux Standard Base.

Compliance During Your Product's Lifecycle [2]. В описании переменной ARCHIVER_MODE представлена информация о флагах переменных (varflags), помогающих создавать архивы.

6.3. autotools*.bbclass

Классы autotools* поддерживают пакеты с автоматической настройкой. Пакеты autoconf, automake и libtool обеспечивают стандартизацию. Классы определяют набор задач (настройка конфигурации, компиляция и т. п.), которые работают с автоматическими инструментами. Обычно этого достаточно для задания нескольких стандартных переменных и простого наследования автоматических инструментов. Эти классы могут также работать с эмулируемыми автоматическими инструментами (см. раздел Autotooled Package [2]).

По умолчанию классы autotools* используют сборки за пределами дерева (т. е. сборки autotools.bbclass с B != S). Если собираемая заданием программа не поддерживает сборку вне дерева, следует иметь задание, наследующее класс autotools-brokensep, который ведёт себя как класс autotools но выполняет сборку с B == S. Этот метод полезен, когда поддержка сборки вне дерева отсутствует или не работает. Однако рекомендуется исправить сборку вне дерева и применять её по возможности.

Полезно иметь некоторое представление о работе задач, определяемых классами autotools*.

- *do_configure* регенерирует сценарий configure (используя autoreconf) и запускает его со стандартным набором инструментов, применяемых при кросс-компиляции. Можно передать в configure дополнительные параметры через переменную EXTRA_OECONF или PACKAGECONFIG_CONFARGS.
- *do_compile* запускает make с аргументами, задающими компилятор и компоновщик. Можно передать дополнительные параметры через переменную EXTRA_OEMAKE.
- *do_install* запускает make и передаёт ${D} как DESTDIR.

6.4. base.bbclass

Класс base отличается тем, что его неявно наследует каждый файл .bb. Этот класс содержит определения стандартных базовых задач для извлечения исходных файлов, их распаковки, настройки (пусто по умолчанию), компиляции (при наличии Makefile), установки (пусто по умолчанию) и упаковки (пусто по умолчанию). Эти классы часто переопределяются или расширяются другими классами, такими как autotools и package. Класс включает также некоторые функции общего назначения (такие как oe_runmake), запускающие make с аргументами из переменной EXTRA_OEMAKE, а также с аргументами, переданными напрямую в oe_runmake.

6.5. bash-completion.bbclass

Организует упаковку и зависимости в соответствии с заданиями, собирающими программы, которые включают данные bash-completion[1].

6.6. bin_package.bbclass

Класс bin_package является вспомогательным для заданий, извлекающих содержимое двоичных пакетов (например, RPM) и устанавливающих его. Двоичный пакет распаковывается и создаются новые пакеты в соответствии с заданным выходным форматом. Примером использования этого класса является извлечение и установка фирменных пакетов.

Для RPM и других пакетов, не включающих подкаталогов, следует указать соответствующий параметр сборщика, указывающий подкаталог. Например при использовании в BitBake сборщика Git (git://) параметр subpath ограничивает выбор конкретного пути в дереве. Ниже приведен пример, где применяется ${BP} для извлечения файлов в каталог, ожидаемый заданным по умолчанию значением S:

```
SRC_URI = "git://example.com/downloads/somepackage.rpm;subpath=${BP}"
```

Информация о поддерживаемых BitBake сборщиках приведена в разделе Fetchers [4].

6.7. binconfig.bbclass

Класс binconfig помогает корректировать пути к сценариям командного процессора. До широкого распространения pkg-config библиотеки распространялись со сценариями (обычно LIBNAME-config) для получения информации о библиотеках и путях к включаемым файлам, нужным программам. Этот класс помогает заданиям применять такие сценарии. При подготовке система сборки OE устанавливает эти сценарии в каталог sysroots/. Наследование этого класса приводит к тому, что все пути в этих сценариях изменяются так, чтобы корректно указывать каталог sysroots/, чтобы все сборки, использующие сценарий, применяли корректные пути для кросс-компиляции (см. описание переменной BINCONFIG_GLOB.

6.8. binconfig-disabled.bbclass

Дополнительный вариант класса binconfig, отключающий двоичные сценарии настройки путём возврата ошибки (в пользу применения pkg-config для запроса информации). Отключаемые сценарии должны быть указаны в переменной BINCONFIG наследующего класс задания.

6.9. blacklist.bbclass

Класс blacklist предотвращает сборку системой OE конкретных пакетов («чёрный список»). Для использования класса нужно наследовать его глобально и установить переменную PNBLACKLIST для каждого задания, которое вы хотите включить в «чёрный список». Задайте значение PN как флаг переменной (varflag) и укажите причину отказа от сборки, которая указывается, если сборка пакета была запрошена. Например, если вы не хотите собирать задание exoticware, можно указать приведённые ниже строки в файле local.conf или конфигурации дистрибутива

```
INHERIT += "blacklist"
PNBLACKLIST[exoticware] = "Not supported by our organization."
```

[1]Дополнение частично введённых команд при нажатии клавиши Tab.

6.10. bluetooth.bbclass

Класс bluetooth определяет переменную которая расширяет задание (пакет) базовой поддержкой bluetooth для платформы. Описание работы класса приведено в файле meta/classes/bluetooth.bbclass дерева исходных кодов YP.

6.11. buildhistory.bbclass

Класс buildhistory записывает историю сборки по метаданным вывода, которую можно использовать для поиска регрессии и анализа вывода сборки. Информация об истории сборки дана в разделе Maintaining Build Output Quality [2].

6.12. buildstats.bbclass

Класс buildstats записывает статистику производительности (время, загрузка процессора, ввод-вывод) для каждой задачи в процессе сборки. При использовании этого класса вывод осуществляется в каталог BUILDSTATS_BASE (по умолчанию ${TMPDIR}/buildstats/). Время работы можно анализировать с помощью сценария scripts/pybootchartgui/pybootchartgui.py, который выдаёт график процесса сборки и позволяет находить узкие места.

Сбор статистики по умолчанию включается переменной USER_CLASSES в файле local.conf. Для отключения следует удалить buildstats из списка USER_CLASSES.

6.13. buildstats-summary.bbclass

При глобальном наследовании этот класс выводит статистику в конце сборки. Для работы нужен класс buildstats.

6.14. ccache.bbclass

Класс ccache включает кэш компилятора C/C++ для сборки и служит для небольшого повышения производительности сборки. Однако использование класса может давать побочные эффекты, поэтому применять его не рекомендуется (см. http://ccache.samba.org/).

6.15. chrpath.bbclass

Класс chrpath служит «оболочкой» для утилиты chrpath, используемой при сборке заданий nativesdk, cross и cross-canadian для изменения записей RPATH в двоичных файлах с целью сделать их перемещаемыми.

6.16. clutter.bbclass

Класс clutter объединяет старший и младший номер версии, а также другие общие элементы, используемые Clutter и связанными заданиями. В отличие от других классов, связанных с конкретными библиотеками, заданиям, создающим другие программы, которые используют Clutter, не требуется наследовать этот класс, если они не применяют такую же схему версий, как Clutter и связанные задания.

6.17. cmake.bbclass

Класс cmake разрешает рецепты, для которых применяется система сборки CMake. Можно применять переменную EXTRA_OECMAKE для указания дополнительных опций конфигурации, передаваемых команде cmake.

6.18. cml1.bbclass

Класс cml1 обеспечивает базовую поддержку для систем настройки конфигурации в стиле ядра Linux.

6.19. compress_doc.bbclass

Включает компрессию страниц man и info. Класс предназначен для глобального наследования. По умолчанию применяется сжатие gz (gzip), но можно указать иной механизм с помощью переменной DOC_COMPRESS.

6.20. copyleft_compliance.bbclass

Класс copyleft_compliance представляет исходные коды для целей соответствия лицензиям. Этот класс служит альтернативой классу archiver и продолжает применяться, несмотря на отказ от него в пользу archiver.

6.21. copyleft_filter.bbclass

Класс, используемый классами archiver и copyleft_compliance для фильтрации лицензий. Класс copyleft_filter является внутренним и не предназначен для использования напрямую.

6.22. core-image.bbclass

Класс core-image обеспечивает базовые определения для заданий core-image-*, такие как поддержка дополнений в IMAGE_FEATURES.

6.23. cpan*.bbclass

Классы cpan* поддерживают модули Perl, задания для которых просты и обычно нужны лишь для указания архива исходных кодов и наследования нужного класса. Сборка применяет 2 метода в зависимости от типа модуля:

- для модулей на основе старой системы сборки Makefile.PL нужен класс cpan.bbclass;
- для модулей, применяющих систему сборки на основе Build.PL, нужен класс cpan_build.bbclass.

В обоих случаях наследуется класс cpan-base для базовой поддержки Perl.

6.24. cross.bbclass

Класс cross обеспечивает поддержку заданий для сборки инструментов кросс-компиляции.

6.25. cross-canadian.bbclass

Класс cross-canadian обеспечивает поддержку заданий, собирающих инструменты канадской кросс-компиляции для SDK (см раздел Cross-Development Toolchain Generation [1]).

6.26. crosssdk.bbclass

Класс crosssdk обеспечивает поддержку заданий, собирающих инструменты кросс-компиляции для сборки SDK (см раздел Cross-Development Toolchain Generation [1]).

6.27. debian.bbclass

Класс debian переименовывает выходные пакеты в соответствии с политикой именования Debian (т. е. glibc становится libc6, а glibc-devel - libc6-dev.). Переименования включает имя библиотеки и версию как часть имени пакета. Если задание создает пакеты для нескольких библиотек (файлы общих объектов типа .so), следует использовать в задании переменную LEAD_SONAME для указания библиотеки, к которой применяется схема именования.

6.28. deploy.bbclass

Класс deploy обслуживает развёртывание файлов в каталоге DEPLOY_DIR_IMAGE. Основной задачей класса является ускорение этапа развёртывания за счёт использования общего состояния. Наследующие этот класс задания должны определять свою функцию do_deploy для копирования файлов, разворачиваемых в DEPLOYDIR и использовать addtask для добавления этой задачи в нужное место (обычно после задачи do_compile или do_install). Затем класс обеспечивает размещение файлов из DEPLOYDIR в DEPLOY_DIR_IMAGE.

6.29. devshell.bbclass

Класс devshell добавляет задачу do_devshell. Включение этого класса определяется политикой дистрибутива. Использование класса описано в разделе Using a Development Shell [2].

6.30. devupstream.bbclass

Класс devupstream использует BBCLASSEXTEND для добавления варианта задания с выборкой из другого URI (например, Git) вместо архива.

```
BBCLASSEXTEND = "devupstream:target"
SRC_URI_class-devupstream = "git://git.example.com/example"
SRCREV_class-devupstream = "abcd1234"
```

Добавление этих операторов в задание создаст вариант с DEFAULT_PREFERENCE = "-1", поэтому нужно будет выбрать используемый вариант задания. Можно внести нужные изменения путём переопределения class-devupstream.

```
DEPENDS_append_class-devupstream = " gperf-native"

do_configure_prepend_class-devupstream() {
   touch ${S}/README
}
```

В настоящее время класс поддерживает лишь создание development-вариантов заданий для целевых платформ (не native или nativesdk). Синтаксис BBCLASSEXTEND (devupstream:target) обеспечивает поддержку вариантов native и nativesdk, которая будет добавлена в новых выпусках.

Поддержка других систем контроля версий (например, Subversion) ограничена зависимостями автоматической сборки BitBake (subversion-native).

6.31. distro_features_check.bbclass

Класс distro_features_check позволяет отдельным заданиям проверять требуемые и конфликтующие свойства DISTRO_FEATURES. Этот класс обеспечивает поддержку переменных REQUIRED_DISTRO_FEATURES и CONFLICT_DISTRO_FEATURES. Если какое-либо из условий, указанных в задании с помощью этих переменных не выполняется, задание будет пропущено.

6.32. distutils*.bbclass

Классы distutils* поддерживают задания для расширений Python версий 2.x. Обычно эти задания нужны лишь для указания архива исходного кода и наследования подходящего класса. При сборке возможны 3 метода в зависимости от выбора автора модуля.

- Расширения с автоматизированной сборкой, требующие Autotools и классы на основе distutils в задании.
- Расширения со сборкой на основе distutils, требующие класс distutils в своих заданиях.
- Расширения со сборкой на основе setuptools, требующие класс setuptools в своих заданиях.

Некоторым классам distutils* нужен класс distutils-common-base для поддержки Python2.

6.33. distutils3*.bbclass

Классы distutils3* поддерживают задания для расширений Python версий 3.x. Обычно эти задания нужны лишь для указания архива исходного кода и наследования подходящего класса. При сборке возможны 3 метода в зависимости от выбора автора модуля.

- Расширения с автоматизированной сборкой, требующие Autotools и классы на основе distutils в задании.
- Расширения со сборкой на основе distutils, требующие класс distutils в своих заданиях.
- Расширения со сборкой на основе setuptools3, требующие класс setuptools3 в своих заданиях.

Классы distutils3* наследуют соответствующий класс distutils* или реплицируют его с использованием Python3 (например, distutils3-base наследует класс distutils-common-base, который повторяет distutils-base, но наследует python3native вместо pythonnative).

6.34. externalsrc.bbclass

Класс externalsrc поддерживает сборку программ из исходных кодов, находящихся за пределами системы сборки OE. Сборка из внешних источников исключает обычную выборку, распаковку и применение правок. По умолчанию система сборки OE использует переменные S и B для нахождения распакованных исходных кодов и сборки. При наследовании заданием класса externalsrc переменные EXTERNALSRC и EXTERNALSRC_BUILD в конечном итоге задают S и B.

По умолчанию этот класс предполагает, что исходный код поддерживает сборку с использованием переменной B для указания каталога, куда система сборки OE будет помещать созданные объекты. По умолчанию каталог B отделен от дерева исходных кодов (S) и задаётся в форме ${WORKDIR}/${BPN}/{PV}/.

Дополнительная информация о классе externalsrc приведена в комментариях файла meta/classes/externalsrc.bbclass в дереве исходного кода, а использование класса описано в разделе Building Software from an External Source [2].

6.35. extrausers.bbclass

Класс extrausers позволяет применить дополнительные конфигурации пользователей и групп на уровне образа. Наследование этого класса задаётся глобально или в задании для образа разрешается выполнение дополнительных пользовательских и групповых операций с использованием переменной EXTRA_USERS_PARAMS.

Пользовательские и групповые операции, добавленные с использованием extrausers, не привязаны к конкретным заданиям за пределами задания для образа. Таким образом, операции могут выполняться для образа в целом. Класс useradd добавляет конфигурацию пользователей и групп к конкретному заданию.

Ниже приведен пример использования класса в задании для образа.

```
inherit extrausers
EXTRA_USERS_PARAMS = "\
    useradd -p '' tester; \
    groupadd developers; \
    userdel nobody; \
    groupdel -g video; \
    groupmod -g 1020 developers; \
    usermod -s /bin/sh tester; \
    "
```

Следующий пример добавляет пользователей tester-jim и tester-sue, а также назначает для них пароль tester01.

```
inherit extrausers
EXTRA_USERS_PARAMS = "\
    useradd -P tester01 tester-jim; \
    useradd -P tester01 tester-sue; \
    "
```

Ещё один пример устанавливает для пользователя root пароль 1876*18:

```
inherit extrausers
EXTRA_USERS_PARAMS = "\
    usermod -P 1876*18 root; \
    "
```

6.36. fontcache.bbclass

Класс fontcache генерирует нужные сценарии пост-установки и пост-удаления (postinst и postrm) для шрифтовых пакетов. Эти сценарии вызывают команду fc-cache (часть Fontconfig) для добавления шрифтов в кэш информации. Поскольку кэш-файлы зависят от архитектуры, fc-cache запускается с использованием QEMU, если сценарии postinst нужно запускать на сборочном хосте при создании образа. Если шрифты устанавливаются не в основном пакете задания, нужно указать в переменной FONT_PACKAGES пакеты, содержащие шрифты.

6.37. fs-uuid.bbclass

Класс fs-uuid извлекает UUID из значения ${ROOTFS}, которое должно присутствовать в момент вызова функции. Класс работает только с файловыми системами ext и зависит от tune2fs.

6.38. gconf.bbclass

Класс gconf обеспечивает базовую функциональность для заданий, которым нужно устанавливать схемы GConf. Схемы будут помещаться в отдельный пакет (${PN}-gconf), создаваемый автоматически при наследовании этого класса. Этот пакет использует подходящие сценарии пост-установки и пост-удаления (postinst и postrm) для регистрации и дерегистрации схем в целевом образе.

6.39. gettext.bbclass

Класс gettext обеспечивает поддержку сборки программ, использующих систему поддержки языков GNU gettext. Всем заданиям, собирающим программы с использованием gettext, нужно наследовать этот класс.

6.40. gnome.bbclass

Класс gnome поддерживает задания, собирающие программы из стека GNOME. Этот класс наследует классы gnomebase, gtk-icon-cache, gconf и mime, а также отключает самоанализ Gobject, когда это приемлемо.

6.41. gnomebase.bbclass

Класс gnomebase является базовым для заданий, собирающих программы из стека GNOME. Класс устанавливает SRC_URI для загрузки исходного кода с зеркал GNOME, а также расширяет переменную FILES типовыми путями установки GNOME.

6.42. gobject-introspection.bbclass

Обеспечивает поддержку заданий, собирающих программы с самоанализом GObject. Эта функциональность доступна только при наличии свойства gobject-introspection-data в DISTRO_FEATURES, а также qemu-usermode в MACHINE_FEATURES.

Эта функциональность по умолчанию включена и в случае неприменимости её следует отключать в переменной DISTRO_FEATURES_BACKFILL_CONSIDERED или MACHINE_FEATURES_BACKFILL_CONSIDERED.

6.43. grub-efi.bbclass

Класс grub-efi задаёт относящиеся к режиму grub-efi функции для сборки загружаемых образов. Класс поддерживает перечисленные ниже переменные.

- INITRD - указывает список образов файловых систем для конкатенации и применения в качестве стартового RAM-диска (initrd). Необязательна.
- ROOTFS - указывает образ файловой системы для включения в качестве корневой. Необязательна.
- GRUB_GFXSERIAL - значение 1 задаёт графический режим и консоль serial.
- LABELS - список целей для автоматической настройки конфигурации.
- APPEND - список переопределения добавленных строк для каждой метки LABEL.
- GRUB_OPTS - дополнительные опции конфигурации, разделённые символом ;. Необязательна.
- GRUB_TIMEOUT - время ожидания перед выбором заданной по умолчанию метки. Необязательна.

6.44. gsettings.bbclass

Класс gsettings обеспечивает базовую функциональность для заданий, которым требуется устанавливать схемы GSettings (glib). Схемы предполагаются частью основного пакета. Для регистрации и исключения схем из целевого образа добавляются соответствующие сценарии пост-установки и пост-удаления (postinst/postrm).

6.45. gtk-doc.bbclass

Класс gtk-doc является вспомогательным для извлечения подходящих зависимостей gtk-doc и отключения gtk-doc.

6.46. gtk-icon-cache.bbclass

Класс gtk-icon-cache генерирует соответствующие сценарии пост-установки и пост-удаления (postinst/postrm) для пакетов, использующих GTK+ и пиктограммы установки. Эти сценарии вызывают gtk-update-icon-cache для добавления шрифтов в кэш пиктограмм GTK+. Поскольку кэш-файлы зависят от архитектуры, класс gtk-update-icon-cache работает на основе QEMU, если сценарии пост-установки нужно запускать на сборочном хосте в процессе создания образа.

6.47. gtk-immodules-cache.bbclass

Класс gtk-immodules-cache генерирует соответствующие сценарии пост-установки и пост-удаления (postinst/postrm) для пакетов, использующих методы ввода GTK+ для виртуальной клавиатуры. Эти сценарии могут вызывать gtk-update-icon-cache для добавления в кэш методов ввода. Поскольку кэш-файлы зависят от архитектуры, класс gtk-update-icon-cache работает на основе QEMU, если сценарии пост-установки нужно запускать на сборочном хосте в процессе создания образа.

Если модули методов вводе будут устанавливаться не в основном пакете, следует указать эти пакеты в переменной GTKIMMODULES_PACKAGES.

6.48. gzipnative.bbclass

Класс gzipnative включает использование естественных версий gzip и pigz вместо версий, доступных на хосте сборки.

6.49. icecc.bbclass

Класс icecc поддерживает программу Icecream, упрощающую компиляцию заданий и распределяющую их между удалёнными машинами. Класс представляет каталоги с символьными ссылками gcc и g++ на icecc как для естественных, так и для кросс-компиляторов. В зависимости от конфигурации или компиляции система сборки OE добавляет каталоги в начало переменной PATH, а затем использует переменные ICECC_CXX и ICEC_CC, которые указывают на компиляторы g++ и gcc, соответственно. Для кросс-компиляторов класс создает файл tar.gz с инструментарием YP и устанавливает в переменной ICECC_VERSION используемую версию компилятора, используемого для кросс-разработки. Класс обрабатывает эти три этапа компиляции (native, cross-kernel, target) и создает требуемую среду в файле tar.gz для использования удалёнными машинами. Класс также поддерживает генерацию SDK.

Если переменная ICECC_PATH не задана в файле local.conf, класс пытается найти двоичный файл icecc для использования. Если переменная ICECC_ENV_EXEC установлена в local.conf, ей следует указывать пользовательский сценарий icecc-create-env. Если переменная не указывает такой сценарий, система сборки использует принятый по умолчанию из задания icecc-create-env-native.bb. Этот сценарий изменён по сравнению с распространяемым в icecc.

Если не нужно применять распределенную компиляцию Icecream к определенным заданиям или классам, можно поместить их в «чёрный список» с помощью переменных ICECC_USER_PACKAGE_BL и ICECC_USER_CLASS_BL в файле local.conf. Это приведёт к локальной обработке таких заданий и классов системой сборки OE.

Кроме того, можно указать с помощью переменной ICECC_USER_PACKAGE_WL в local.conf задания для форсированного применения icecc в заданиях с пустой переменной PARALLEL_MAKE.

Наследование класса icecc меняет все подписи sstate, поэтому при наличии в команде разработки выделенной системы сборки, которая заполняет STATE_MIRRORS и желании многократно использовать sstate из STATE_MIRRORS, наследовать класс icecc должны все разработчики или никто из них.

На распределенном уровне при наследовании класса icecc нужно обеспечить начало работы всех сборщиков с общими подписями sstate. После наследования класса можно отключить это свойство, как показано ниже.

```
INHERIT_DISTRO_append = " icecc"
ICECC_DISABLED ??= "1"
```

Такой подход обеспечивает использование всеми одних подписей, но требует от каждого, кто хочет применять Icecream включить это свойство в файле local.conf с помощью строки ICECC_DISABLED = "".

6.50. image.bbclass

Класс image помогает создавать образы в разных форматах. Сначала создаётся корневая файловая система с использованием одного из файлов rootfs*.bbclass (в зависимости от используемого формата пакетов), затем создаётся один или несколько файлов с образами. Типы образов определяются переменной IMAGE_FSTYPES, а список пакетов, включаемых в образ, задаёт переменная IMAGE_INSTALL.

Настройка образов описана в разделе Customizing Images [2], а их создание - в разделе Images [1].

6.51. image-buildinfo.bbclass

Класс image-buildinfo записывает информацию в каталог /etc/build целевой файловой системы.

6.52. image_types.bbclass

Класс image_types определяет все типы стандартного вывода образов, которые могут быть указаны в переменной IMAGE_FSTYPES. Этот класс может служить в качестве справки о добавлении поддержки вывода пользовательских типов образов.

По умолчанию класс image автоматически включает image_types. Класс image использует переменную IMGCLASSES, как показано ниже.

```
IMGCLASSES = "rootfs_${IMAGE_PKGTYPE} image_types ${IMAGE_CLASSES}"
IMGCLASSES += "${@['populate_sdk_base', 'populate_sdk_ext']['linux' in d.getVar("SDK_OS")]}"
IMGCLASSES += "${@bb.utils.contains_any('IMAGE_FSTYPES', 'live iso hddimg', 'image-live', '', d)}"
IMGCLASSES += "${@bb.utils.contains('IMAGE_FSTYPES', 'container', 'image-container', '', d)}"
IMGCLASSES += "image_types_wic"
IMGCLASSES += "rootfs-postcommands"
IMGCLASSES += "image-postinst-intercepts"
inherit ${IMGCLASSES}
```

Класс image_types также обслуживает преобразование и сжатие образов.

Для сборки образа VMware VMDK нужно добавить wic.vmdk в переменную IMAGE_FSTYPES. Похожая операция выполняется для создания образов vdi[1] и qcow2[2].

6.53. image-live.bbclass

Этот класс контролирует сборку «живых» (live) образов (HDDIMG и ISO), содержащих syslinux для унаследованной загрузки, а также загрузчик (bootloader), заданный в EFI_PROVIDER, если MACHINE_FEATURES содержит efi. Обычно этот класс напрямую не используется, а добавляется значение live в IMAGE_FSTYPES.

6.54. image-mklibs.bbclass

Класс image-mklibs разрешает использование в задаче do_rootfs утилиты mklibs, оптимизирующей размер библиотек в образе. По умолчанию класс включается в файле local.conf.template с использованием переменной USER_CLASSES.

```
USER_CLASSES ?= "buildstats image-mklibs image-prelink"
```

6.55. image-prelink.bbclass

Класс image-mklibs разрешает использование в задаче do_rootfs утилиты prelink, оптимизирующей динамическую компоновку общих библиотек для снижения времени запуска исполняемых файлов. По умолчанию класс включается в файле local.conf.template с использованием переменной USER_CLASSES.

```
USER_CLASSES ?= "buildstats image-mklibs image-prelink"
```

6.56. insane.bbclass

Класс insane добавляет в процесс генерации пакетов этап проверки качества системой сборки OE. Выполняется набор тестов для обнаружения основных проблем, возникающих в процессе работы. Включение этого класса обычно определяет политика дистрибутива.

Можно настроить проверку работоспособности с выдачей при отказах предупреждений или сообщений об ошибках. Обычно при первом отказе конкретного теста выдаётся предупреждение, а последующие считаются ошибками, когда

[1]Virtual Box Virtual Disk Image - образ виртуального диска Virtual Box.
[2]QEMU Copy On Write Version 2.

метаданные известны и корректны. Глава 10. Ошибки и предупреждения тестов QA описывает все предупреждения и сообщения об ошибках, которые могут выдаваться в принятой по умолчанию конфигурации.

Переменные WARN_QA и ERROR_QA управляют поведением при проверке на глобальном уровне (уровень конфигурации дистрибутива). Можно пропустить одну или несколько проверок, указав их в переменной INSANE_SKIP. Например, для пропуска проверки символьных ссылок для файлов .so в задании для основного пакета добавляется INSANE_SKIP_${PN} += "dev-so". Следует помнить об переопределениях имён пакетов (в примере ${PN}).

Проверки QA служат для обнаружения имеющихся и возможных проблем в упакованном выводе. Поэтому при отключении проверок следует соблюдать осторожность.

Ниже приведен список всех тестов, которые можно указывать в переменных WARN_QA и ERROR_QA.

already-stripped
Проверяет удаление отладочных символов до их исключения системой сборки. Вырезание отладочных символов нормально для обычных пакетов. Для того, чтобы отладка работала на платформах, использующих пакеты -dbg, вырезание отладочных символов должно быть отключено.

arch
Проверяет тип ELF[1], размер, порядок битов всех двоичных файлов на предмет соответствия целевой архитектуре. Отказ теста говорит о несовместимости двоичного файла. Тест может показывать использование непригодного компилятора или опция компиляции. Для некоторых программ (например, загрузчиков) может потребоваться обход этого теста.

buildpaths
Проверяет пути к местам на хосте сборки, указанные в выходных файлах. Сейчас тест даёт очень много ложных срабатываний и по этой причине обычно отключён.

build-deps
Проверяет соответствие зависимостей при сборке, заданных в DEPENDS (кроме RDEPENDS), или зависимостей на уровне задачи зависимостям, заданным для работы (runtime). Такая проверка полезна, в частности, для определения моментов обнаружения и добавления runtime-зависимостей в процессе упаковки. Если в метаданных не задано явных зависимостей, на этапе упаковки будет поздно обеспечивать создание зависимостей и процесс может установки пакета в образ завершиться ошибкой при выполнении задачи do_rootfs, поскольку обнаруженные автоматически зависимости не выполнены. Примером может служить автоматическое добавление зависимостей классом update-rc.d в пакете initscripts-functions для пакетов, которые устанавливают сценарии инициализации, ссылающиеся на файл /etc/init.d/functions. Заданию следует включать явные RDEPENDS для пакетов от initscripts-functions, чтобы система сборки OE могла гарантировать, что задание initscripts действительно собрано и пакет initscripts-functions доступен.

compile-host-path
Проверяет журнал do_compile на предмет наличия путей на сборочном хосте, используемых при сборке заданий. Наличие таких путей может приводить к засорению хоста выводом сборки.

debug-deps
Проверяет независимость всех пакетов, кроме -dbg, от отладочных пакетов -dbg (зависимость может привести к ошибкам упаковки).

debug-files
Проверяет наличие в каталогах .debug посторонних (не относящихся к пакетам -dbg) файлов. Все отладочные файлы должны находиться в пакетах -dbg.

dep-cmp
Проверяет наличие недействительных операторов сравнения версий в зависимостях между пакетами при работе (т. е. в переменных RDEPENDS, RRECOMMENDS, RSUGGESTS, RPROVIDES, RREPLACES, RCONFLICTS). Некорректное сравнение может вызывать отказы и нежелательное поведение при передаче менеджеру пакетов.

desktop
Запускает программу desktop-file-validate для проверки всех файлов .desktop на предмет соответствия их содержимого спецификации файлов .desktop.

dev-deps
Проверяет независимость всех пакетов, кроме -dev и -staticdev от пакетов -dev, для предотвращения ошибок упаковки.

dev-so
Проверяет, что символьные ссылки .so размещаются в пакетах dev, а не в каких-либо иных. В общем случае такие ссылки полезны лишь для пакетов разработки, поэтому их корректно размещать в пакетах -dev. В некоторых редких случаях для динамически загружаемых модулей такие ссылки применяются в основном пакете.

file-rdeps
Проверяет выполнение зависимостей на уровне файлов, идентифицированных системой сборки OE. Например, сценарий оболочки может начинаться со строки #!/bin/bash, которая будет транслироваться в зависимость от файла /bin/bash. Из трёх менеджеров пакетов, поддерживаемых системой сборки OE, лишь RPM напрямую обрабатывает зависимости на уровне файлов, автоматически преобразуя их в зависимости от пакетов, содержащих нужные файлы. Однако отсутствие такой функциональности в других менеджерах пакетов не означает необходимости преобразования зависимостей. Эта проверка QA пытается обеспечить наличие явно созданных RDEPENDS для обработки всех зависимостей на уровне файлов, найденных в файлах пакета.

files-invalid
Проверяет наличие в переменной FILES наличие недопустимых значений с //.

host-user-contaminated
Проверяет, что ни один из созданных заданием пакетов не содержит файлов за пределами /home с идентификаторами пользователя и группы, соответствующими пользователю, применяющему BitBake. Наличие таких файлов обычно означает их установку с некорректными UID/GID, поскольку идентификаторы на целевой платформе не зависят от идентификаторов на хосте. Дополнительная информация приведена в описании задачи do_install task.

incompatible-license
Сообщает о пакетах, исключённых из сборки по причине наличия лицензии, указанной в INCOMPATIBLE_LICENSE.

[1]Executable and Linkable Format - исполняемый и компонуемый формат.

install-host-path
Проверяет журнал do_install для указания использования путей к местоположению на сборочном хосте. Использование таких путей может приводить в «загрязнению» хоста выводом сборки.

installed-vs-shipped
Сообщает о файлах, установленных задачей do_install, но не включённых ни в один пакет через переменную FILES. Не входящие в пакеты файлы не могут присутствовать в образе на последующих этапах сборки. В идеальном случае все установленные файлы должны включаться в те или иные пакеты. Не включённые в пакеты файлы можно удалить в конце задачи do_install, если они не нужны ни одному из пакетов.

invalid-chars
Проверяет, что переменные метаданных DESCRIPTION, SUMMARY, LICENSE и SECTION в задании не содержат символов, отличных от UTF-8. Некоторые менеджеры пакетов не поддерживают такие символы.

invalid-packageconfig
Проверяет отсутствие не определённых свойств в переменной PACKAGECONFIG. Например, любых имён foo, для которых не существует форма PACKAGECONFIG[foo] = "..."

la
Проверяет файлы .la на предмет наличия в путях TMPDIR. Любой файл .la в таком пути является некорректным, поскольку libtool добавляет корректный префикс sysroot при автоматическом использовании самих файлов.

ldflags
Обеспечивает компоновку двоичных файлов с опциями LDFLAGS, предоставленными системой сборки. Если этот тест не проходит, следует проверить передачу переменной LDFLAGS команде компоновщика.

libdir
Проверяет установку библиотек в некорректные (возможно, жёстко заданные) пути. Например, этот тест будет находить задание, которое устанавливает /lib/bar.so при ${base_libdir} lib32. Другим примером является задание, устанавливающее /usr/lib64/foo.so при ${libdir} /usr/lib.

libexec
Проверяет наличие в пакете файлов в /usr/libexec. Эта проверка не выполняется, если переменная libexecdir явно указывает /usr/libexec.

packages-list
Проверяет многократное включение пакета в переменную PACKAGES, что может вызывать проблемы при подготовке пакетов.

perm-config
Указывает строки с непригодным форматом в fs-perms.txt.

perm-line
Указывает строки с непригодным форматом в fs-perms.txt.

perm-link
Указывает строки в файле fs-perms.txt задающие link, когда указанная цель уже существует.

perms
В настоящее время не используется (резерв).

pkgconfig
Проверяет файлы .pc на предмет наличия путей TMPDIR/WORKDIR. Любой файл .pc file, содержащий такие пути, является некорректным, поскольку pkg-config добавляет корректный префикс sysroot при обращении к файлам.

pkgname
Проверяет отсутствие у всех пакетов из PACKAGES имён с недопустимыми символами (кроме 0-9, a-z, ., +, и -).

pkgv-undefined
Проверяет, установлена ли переменная PKGV во время выполнения задачи do_package.

pkgvarcheck
Проверяет переменные RDEPENDS, RRECOMMENDS, RSUGGESTS, RCONFLICTS, RPROVIDES, RREPLACES, FILES, ALLOW_EMPTY, pkg_preinst, pkg_postinst, pkg_prerm и pkg_postrm на предмет установок, не являющихся специфическими для пакета. Использование этих переменных без связанного с пакетом суффикса может приводить к неоправданному усложнению зависимостей других пакетов в том же задании и другим побочным эффектам.

pn-overrides
Проверяет отсутствие значения имени (PN) задания в переменной OVERRIDES. Если задание названо так, что его значение PN соответствует имеющемуся в OVERRIDES (например, PN совпадает с MACHINE или DISTRO), это может приводить к неожиданным последствиям, например, назначения вида FILES_${PN} = "xyz" фактически превратятся в FILES = "xyz".

rpaths
Проверяет rpaths в исполняемых файлах на предмет наличия путей системы сборки, таких как TMPDIR. При наличии таких путей будут переданы неверные опции -rpath в команды компоновщика и в исполняемых файлах могут возникнуть проблемы безопасности.

split-strip
Сообщает об отказах при выделении или удалении отладочных символов из исполняемых файлов.

staticdev
Проверяет наличие статических библиотек (*.a) а пакетах, не являющихся staticdev.

symlink-to-sysroot
Проверяет наличие в пакетах символьных ссылок на каталог TMPDIR сборочного хоста. Такие ссылки будут не пригодны при работе на целевой системе.

textrel
Проверяет наличие в исполняемых файлах ELF перемещений (relocation) в их разделах .text, что может влиять на производительность работы. При наличии таких перемещений система сборки выдаёт сообщение, описанное в параграфе 10.2. Ошибки и предупреждения.

useless-rpaths
Проверяет в исполняемых файлах пути загрузки динамических библиотек (rpath), которые по умолчанию на стандартных системах ищет компоновщик (например, /lib и /usr/lib). Хотя эти пути не будут вызывать проблем, они не требуются и будут занимать пространство.

var-undefined

Сообщает о важных для подготовки пакетов переменных (WORKDIR, DEPLOY_DIR, D, PN, PKGD), которые не определены во время задачи do_package.

version-going-backwards

При включённой истории сборки сообщается, когда записываемый пакет имеет более низкую версию, чем записанный ранее одноимённый пакет. Если выходные пакеты помещаются в хранилище и обновляете с его помощью целевую систему, снижение версии может привести к тому, что целевая система не «обновит» этот пакет. Если в целевой системе не применяется обновление через сеть, можно не думать об этой проблеме.

xorg-driver-abi

Проверяет во всех пакетах с драйверами Xorg наличие зависимости ABI. Задание xserver-xorg обеспечивает имена драйверов ABI. Все драйверы должны зависеть от версии ABI, с которой они собраны. Задания для драйверов, включающие файл xorg-driver-input.inc или xorg-driver-video.inc, будут получать версию автоматически. Поэтому нужно лишь явно добавить зависимости для заданий.

6.57. insserv.bbclass

Класс insserv использует утилиту insserv для обновления других символьных ссылок в /etc/rc?.d/ внутри образа на основе зависимостей, заданных заголовками LSB в сценариях init.d.

6.58. kernel.bbclass

Класс kernel отвечает за сборку ядер Linux и содержит код для сборки деревьев ядра. Нужные заголовки собираются в каталоге STAGING_KERNEL_DIR, чтобы можно было собрать внешние модули с помощью класса module.

Это означает, что каждый собираемый модуль ядра пакуется отдельно и создаются зависимости между модулями путём анализа modinfo. Если требуются все модули, установка пакета kernel-modules обеспечивает инсталляцию всех пакетов с модулями и другими пакетами ядра, такими как kernel-vmlinux.

Логика класса kernel позволяет встраивать образ файловой системы initramfs при сборке образа ядра (см. раздел Building an Initial RAM Filesystem (initramfs) Image [2]).

Классы kernel и module используют внутри себя другие классы, включая kernel-arch, module-base и linux-kernel-base.

6.59. kernel-arch.bbclass

Класс kernel-arch устанавливает переменную окружения ARCH для компиляции ядра Linux (включая модули).

6.60. kernel-devicetree.bbclass

Класс kernel-devicetree, наследующий kernel, поддерживает генерацию дерева устройств.

6.61. kernel-fitimage.bbclass

Класс kernel-fitimage обеспечивает поддержку образов zImage.

6.62. kernel-grub.bbclass

Класс kernel-grub обновляет область и меню загрузки с ядром в качестве приоритетного механизма загрузки при установке RPM обновления ядра на развёрнутой платформе.

6.63. kernel-module-split.bbclass

Класс kernel-module-split обеспечивает базовые функции для разделения модулей ядра Linux в отдельные пакеты.

6.64. kernel-uboot.bbclass

Класс kernel-uboot поддерживает сборку образов из репозиториев ядра в стиле vmlinux.

6.65. kernel-uimage.bbclass

Класс kernel-uimage поддерживает упаковку uImage.

6.66. kernel-yocto.bbclass

Класс kernel-yocto обеспечивает базовую функциональность для сборки из репозиториев со стилем ядра linux-yocto.

6.67. kernelsrc.bbclass

Класс kernelsrc устанавливает исходные коды и версию ядра Linux.

6.68. lib_package.bbclass

Класс lib_package поддерживает задания, собирающие библиотеки и создающие двоичные исполняемые файлы, которые по умолчанию не следует устанавливать вместе с библиотекой. Эти исполняемые двоичные файлы добавляются в отдельный пакет ${PN}-bin.

6.69. libc*.bbclass

Классы libc* поддерживают задания, собирающие пакеты с libc:

- libc-common обеспечивает базовую поддержу сборки с использованием libc;
- libc-package обеспечивает поддержу сборки с использованием glibc и eglibc.

6.70. license.bbclass

Класс license обеспечивает создание манифеста лицензий и исключение лицензий. Класс включён по умолчанию через принятое по умолчанию значение переменной INHERIT_DISTRO.

6.71. linux-kernel-base.bbclass

Класс linux-kernel-base обеспечивает базовую функциональность для заданий, которые собираются из дерева исходных кодов ядра Linux, не включая сборку самого ядра. Например, задание perf наследует этот класс.

6.72. linuxloader.bbclass

Обеспечивает функцию linuxloader(), сообщающую динамический загрузчик/компоновщик, обеспечиваемый платформой. Значение функции используется множеством других классов.

6.73. logging.bbclass

Обеспечивает стандартные функции оболочки, используемые для записи в журнал сообщений BitBake с различными уровнями важности (bbplain, bbnote, bbwarn, bberror, bbfatal, bbdebug). Класс включён по умолчанию, поскольку наследуется классом base.

6.74. meta.bbclass

Класс meta наследуется заданиями, которые сами не собирают пакетов, но служат мета-целью для других заданий.

6.75. metadata_scm.bbclass

Класс metadata_scm обеспечивает функциональность для запроса ветви и выпуска репозитория SCM. Класс base применяет этот класс для печати выпуска каждого задания перед его сборкой. Класс metadata_scm включён по умолчанию, поскольку наследуется классом base.

6.76. migrate_localcount.bbclass

Класс migrate_localcount проверяет и инкрементирует данные localcount в задании.

6.77. mime.bbclass

Класс mime создает нужные сценарии пост-установки и пост-удаления (postinst/postrm) для пакетов, устанавливающих файлы типов MIME. Эти сценарии вызывают update-mime-database для добавления типов MIME в общую базу данных.

6.78. mirrors.bbclass

Класс mirrors организует некоторые стандартные записи MIRRORS для зеркал исходного кода, которые обеспечивают резервные пути при недоступности репозитория, заданного в SRC_URI. Класс включён по умолчанию, поскольку наследуется классом base.

6.79. module.bbclass

Класс module обеспечивает поддержку сборки внешних (out-of-tree) модулей ядра Linux. Этот класс наследует классы module-base и kernel-module-split, а также реализует задачи do_compile и do_install. Класс обеспечивает все, что нужно для сборки и упаковки модулей ядра.

Базовая информация о сборке внешних модулей ядра приведена в разделе Incorporating Out-of-Tree Modules [3].

6.80. module-base.bbclass

Класс module-base обеспечивает базовые функции сборки модулей ядра Linux. Обычно задание для сборки программы, включающей один или несколько модулей ядра и имеющей свои средства сборки, наследует этот класс , а не класс module.

6.81. multilib*.bbclass

Класс multilib* обеспечивает поддержку сборки библиотек с оптимизацией для разных платформ и для разной архитектуры с установкой всех библиотек в один образ. Информация о работе с множеством библиотек приведена в разделе Combining Multiple Versions of Library Files into One Image [2].

6.82. native.bbclass

Класс native обеспечивает базовую функциональность для заданий, собирающих инструменты, которые будут работать на хосте сборки. Задание для сборки инструментов, работающих на этом хосте, можно создать двумя способами.

- *Задание myrecipe-native.bb, наследующее класс native*. В этом случае нужно указать оператор inherit для класса native в задании после всех других операторов inherit, чтобы класс native наследовался последним. Имя для такого задания должно иметь форму myrecipe-native.bb, поскольку другие формы именования могут создавать конфликты с имеющимся кодом, который зависит от согласованного именования.
- *Задание для целевой платформы, содержащее строку BBCLASSEXTEND = "native"*. Внутри этого задания указываются переопределения _class-native и _class-target для уточнения функциональности, относящейся к заданию для хоста или целевой платформы.

В обоих вариантах используется класс native. Преимущество второго метода заключается в том, что он не требует двух раздельных заданий для хоста сборки и целевой платформы. Совпадающие части задания автоматически становятся общими.

6.83. nativesdk.bbclass

Класс nativesdk обеспечивает базовую функциональность для заданий, собирающих инструменты, которые будут работать как часть SDK (т. е. на SDKMACHINE). Задание можно создать двумя разными способами.

- *Задание nativesdk-myrecipe.bb, наследующее класс nativesdk*. В этом случае нужно указать оператор inherit для класса nativesdk в задании после всех других операторов inherit, чтобы этот класс наследовался последним. Имя для такого задания должно иметь форму nativesdk-myrecipe.bb, поскольку другие формы именования могут создавать конфликты с имеющимся кодом, который зависит от согласованного именования.
- *Задание, содержащее строку BBCLASSEXTEND = "nativesdk"*. Внутри этого задания указываются переопределения _class-nativesdk и _class-target для уточнения функциональности, относящейся к заданию для SDK или целевой платформы.

В обоих вариантах используется класс nativesdk. Преимущество второго метода заключается в том, что он не требует двух раздельных заданий для SDK и целевой платформы. Совпадающие части задания автоматически становятся общими.

6.84. nopackages.bbclass

Отключает подготовку пакетов для заданий и классов, где пакеты не требуются.

6.85. npm.bbclass

Обеспечивает поддержку сборки программ Node.js из NPM[1]. В настоящее время задания, наследующие этот класс, должны применять сборщик npm:// для установки зависимостей и автоматической подготовки пакетов. Работа с пакетами NPM описана в разделе Creating Node Package Manager (NPM) Packages [2].

6.86. oelint.bbclass

Класс oelint задаёт устаревший инструмент проверки lint из meta/classes в каталоге исходных кодов. Имеется много классов, которые могут быть полезны в OE-Core, но в реальности никогда не применяются на этом уровне. Класс oelint является одним из таких классов. Однако информация о нем может снизить число разных версий похожих классов на разных уровнях.

6.87. own-mirrors.bbclass

Класс own-mirrors упрощает настройку своих зеркал в переменной PREMIRRORS, в соответствии с которой выполняется выборка до обращения к SRC_URI в каждом задании. Для использования этого класса его нужно наследовать глобально и задать переменную SOURCE_MIRROR_URL, как показано ниже.

```
INHERIT += "own-mirrors"
SOURCE_MIRROR_URL = "http://example.com/my-source-mirror"
```

В переменной SOURCE_MIRROR_URL можно указать лишь один идентификатор URL.

6.88. package.bbclass

Этот класс поддерживает базовую функциональность создания пакетов из вывода сборки. Код, относящийся к конкретным типам упаковки, содержится в классах package_deb, package_rpm, package_ipk, and package_tar. Последний из этих классов применять не рекомендуется.

Список форматов упаковки можно задать с помощью переменной PACKAGE_CLASSES в файле conf/local.conf каталога сборки. Переменная может включать несколько типов упаковки. Поскольку образы создаются из пакетов, класс упаковки требуется задать для создания образа. Система сборки будет выбирать из списка указанный первым класс.

При необязательной настройке репозитория (источника пакетов) на хосте разработки, который может использоваться DNF, можно устанавливать пакеты из этого источника при работе образа на целевой платформе (см. раздел Using Runtime Package Management [2]).

Выбранный класс для конкретного пакета может влиять на производительность сборки и требуемое пространство. В общем случае сборка пакетов с использованием IPK примерно на треть быстрее сборки с использованием RPM. Это сравнение учитывает полную сборку пакета со всеми зависимостями, собранными заранее. Причина этого заключается в том, что менеджер пакетов RPM создает и обрабатывает больше метаданных, нежели менеджер IPK. Поэтому для небольших систем целесообразно выбрать для PACKAGE_CLASSES значение "package_ipk".

При выборе менеджера пакетов следует рассмотреть использование RPM с учётом отмеченного ниже.

- RPM обеспечивает больше возможностей по сравнению с IPK за счёт обработки большего объёма метаданных. Например, метаданные включают типы отдельных файлов, контрольные суммы, поддержку «разбросанных» файлов, обнаружение и разрешение конфликтов в системах Multilib, обновление в стиле ACID, и возможность перепаковки для откатов назад.
- Для небольших систем дополнительное пространство, занятое Berkeley Database и объем метаданных RPM могут влиять на возможности обновления.

Дополнительные сведения о влиянии класса package приведены в рассылках https://lists.yoctoproject.org/pipermail/poky/2011-May/006362.html и https://lists.yoctoproject.org/pipermail/poky/2011-May/006363.html

6.89. package_deb.bbclass

Класс package_deb поддерживает создание пакетов в формате Debian (.deb), обеспечивая их запись в каталог ${DEPLOY_DIR_DEB}. Этот класс наследует класс package и включается переменной PACKAGE_CLASSES в local.conf.

[1]Node package manager - менеджер пакетов узла.

6.90. package_ipk.bbclass

Класс package_ipk поддерживает создание пакетов в формате IPK (.ipk), обеспечивая их запись в каталог ${DEPLOY_DIR_IPK}. Этот класс наследует класс package и включается переменной PACKAGE_CLASSES в local.conf.

6.91. package_rpm.bbclass

Класс package_rpm поддерживает создание пакетов в формате RPM (.rpm), обеспечивая их запись в каталог ${DEPLOY_DIR_RPM}. Этот класс наследует класс package и включается переменной PACKAGE_CLASSES в local.conf.

6.92. package_tar.bbclass

Класс package_tar поддерживает создание пакетов в форме архивов (tarball), обеспечивая их запись в каталог ${DEPLOY_DIR_TAR}. Этот класс наследует класс package и включается переменной PACKAGE_CLASSES в local.conf.

Класс package_tar нельзя указывать первым в переменной PACKAGE_CLASSES, для образов и SDK должен применяться формат .deb, .ipk или .rpm.

6.93. packagedata.bbclass

Класс packagedata обеспечивает базовую функциональность для чтения файлов pkgdata из переменной PKGDATA_DIR. Эти файлы содержат информацию о каждом пакете, создаваемом системой сборки OE.

Этот класс включён по умолчанию, поскольку он наследуется классом package.

6.94. packagegroup.bbclass

Класс packagegroup устанавливает принятые по умолчанию значения, пригодные для заданий групп пакетов (PACKAGES, PACKAGE_ARCH, ALLOW_EMPTY и т. п.). Настоятельно рекомендуется наследовать этот класс всем заданиям для групп. Использование класса описано в разделе Customizing Images Using Custom Package Groups [2].

Ранее этот класс носил имя task.

6.95. patch.bbclass

Класс patch обеспечивает функциональность наложения исправлений в задаче do_patch. Класс включён по умолчанию, поскольку наследуется классом base.

6.96. perlnative.bbclass

Класс perlnative поддерживает использование естественной версии Perl, созданной системой сборки вместо версии, предоставляемой сборочным хостом.

6.97. pixbufcache.bbclass

Класс pixbufcache создает нужные сценарии пост-установки и пост-удаления (postinst/postrm) для пакетов, устанавливающих загрузчики pixbuf, которые применяются с gdk-pixbuf. Эти сценарии вызывают update_pixbuf_cache для добавления загрузчиков pixbuf в кэш. Поскольку файлы кэширования зависят от архитектуры, update_pixbuf_cache запускается с использованием QEMU если сценарии пост-установки нужно запускать на сборочном хосте в процессе создания образа. Если загрузчики pixbuf устанавливаются не в основном пакете, следует установить переменную PIXBUF_PACKAGES для указания этих пакетов.

6.98. pkgconfig.bbclass

Класс pkgconfig обеспечивает стандартный способ получения информации о заголовках и библиотеках с помощью pkg-config. Этот класс нацелен на интеграцию pkg-config с использующими программу библиотеками. В процессе подготовки BitBake устанавливает данные pkg-config в каталог sysroot. За счёт использования функциональности sysroot в пакете pkg-config классу pkgconfig больше не требуются манипуляции с файлами.

6.99. populate_sdk.bbclass

Класс populate_sdk обеспечивает поддержку заданий, содержащих только SDK. Описание сборки инструментов кросс-разработки с применением задачи do_populate_sdk приведено в разделе Building an SDK Installer [7].

6.100. populate_sdk_*.bbclass

Классы populate_sdk_* поддерживают создание SDK:

- *populate_sdk_base* - базовый класс для создания SDK со всеми менеджерами пакетов (DEB, RPM, opkg);
- *populate_sdk_deb* - создание SDK с помощью менеджера пакетов Debian;
- *populate_sdk_rpm* - создание SDK с помощью менеджера пакетов RPM;
- *populate_sdk_ipk* - создание SDK с помощью менеджера пакетов opkg (IPK);
- *populate_sdk_ext* - создание расширяемых SDK с помощью со всеми менеджерами пакетов.

Класс populate_sdk_base наследует подходящий класс populate_sdk_* (deb, rpm, ipk) в соответствии с IMAGE_PKGTYPE.

Базовый класс убеждается в наличии всех исходных и целевых каталогов, затем заполняет SDK. После этого класс populate_sdk_base создает две файловых системы sysroot - ${SDK_ARCH}-nativesdk содержит кросс-компилятор и связанные с ним инструменты, а также цель, включающую корневую файловую систему, которая настроена для образа SDK. Эти два образа размещаются в SDK_OUTPUT, как показано ниже.

```
${SDK_OUTPUT}/${SDK_ARCH}-nativesdk-pkgs
${SDK_OUTPUT}/${SDKTARGETSYSROOT}/target-pkgs
```

В заключение базовый класс заполнения SDK создает сценарий настройки среды инструментов, архив SDK и установщик. Классы populate_sdk_deb, populate_sdk_rpm и populate_sdk_ipk поддерживают соответствующие типы SDK. Эти классы наследуются и применяются классом populate_sdk_base.

Дополнительные сведения о создании инструментария кросс-разработки приведены в разделе Cross-Development Toolchain Generation [1]. Преимущества сборки инструментария кросс-разработки с использованием задачи do_populate_sdk описаны в разделе Building an SDK Installer [7].

6.101. prexport.bbclass

Класс prexport обеспечивает функциональность поддержки экспорта значений PR. Класс не предназначен для использования на прямую и включается применением bitbake-prserv-tool export.

6.102. primport.bbclass

Класс primport обеспечивает функциональность поддержки импорта значений PR. Класс не предназначен для использования на прямую и включается применением bitbake-prserv-tool import.

6.103. prserv.bbclass

Класс prserv обеспечивает функциональность для использования сервиса PR при автоматическом управлении переменной PR в каждом задании. Этот класс включён по умолчанию, поскольку он наследуется классом package, однако система сборки OE не включает эту функциональность, пока не установлена переменная PRSERV_HOST.

6.104. ptest.bbclass

Класс ptest обеспечивает функции для упаковки и установки тестов заданий, собирающих программы для этих тестов.

Этот класс предназначен для наследования индивидуальными заданиями, однако функции класса отключаются, если ptest отсутствует в DISTRO_FEATURES. Тестирование описано в разделе Testing Packages With ptest [2].

6.105. ptest-gnome.bbclass

Включает тесты пакетов (ptest) для пакетов GNOME, предназначенных для выполнения с gnome-desktop-testing.

Тестирование описано в разделе Testing Packages With ptest [2].

6.106. python-dir.bbclass

Класс python-dir указывает базовую версию и расположение Python, а также расположение пакетов на сайте.

6.107. python3native.bbclass

Класс python3native поддерживает применение системой сборки естественной версии Python 3 вместо версии, предоставляемой сборочным хостом.

6.108. pythonnative.bbclass

При наследовании заданием класс pythonnative поддерживает использование естественной версии Python, созданной системой сборки, вместо версии, предоставляемой сборочным хостом.

6.109. qemu.bbclass

Класс qemu обеспечивает функциональность заданий, которым требуется QEMU или нужно проверить наличие этого эмулятора. Обычно класс применяется при запуске программ для целевых систем на сборочном хосте с использованием режима эмуляции QEMU.

6.110. recipe_sanity.bbclass

Класс recipe_sanity проверяет выполнение на хосте предварительных условий, которые могут влиять на сборку (например, установку переменных или наличие программ).

6.111. relocatable.bbclass

Класс relocatable обеспечивает перемещение (relocation) для исполняемых файлов, установленных в sysroot. Класс применяется классом chrpath и сам использует классы cross и native.

6.112. remove-libtool.bbclass

Класс remove-libtool добавляет в задачу do_install функцию удаления всех файлов .la, установленных libtool, и эти файлы не будут включены в sysroot и целевые пакеты. Если файлы .la нужно установить, задание может переопределить удаление, установив REMOVE_LIBTOOL_LA = "0". По умолчанию класс remove-libtool не включён.

6.113. report-error.bbclass

Файл report-error поддерживает включение инструмента отчётов об ошибках, который позволяет сохранять сведения об ошибках сборки в единой базе данных. Этот класс собирает отладочную информацию для задания, включая версию задания, систему сборки, целевую систему, ветвь, фиксацию и журнальный файл. Эти данные сохраняются в формате JSON в базе ${LOG_DIR}/error-report.

6.114. rm_work.bbclass

Класс rm_work class поддерживает удаление временных рабочих каталогов для экономии дискового пространства. Система сборки OE может занимать значительный объем диска в процессе сборки. Частью занятого пространства являются файлы в каталогах ${TMPDIR}/work каждого задания. Как только система сборки создаст для задания пакеты, эти фалы становятся ненужными. Однако по умолчанию система сборки сохраняет их для проверки и возможной отладки. Если файлы нужно удалить для экономии дискового пространства в процессе сборки, можно включить rm_work by добавлением в файл local.conf каталога сборки строки INHERIT += "rm_work".

Если исходный код размещается вне рабочего каталога задания, включение rm_work может приводить к утрате внесённых в исходный код изменений. Для исключения некоторых заданий из процесса удаления с помощью rm_work можно указать имена этих заданий в файле local.conf с помощью, например, RM_WORK_EXCLUDE += "busybox glibc".

6.115. rootfs*.bbclass

Классы rootfs* поддерживают создание корневой файловой системы для образа и включают:

- *rootfs-postcommands* с определением функций пост-обработки в заданиях для образов;
- *rootfs_deb* для создания корневых файловых систем в образах, собираемых с использованием пакетов .deb;
- *rootfs_rpm* для создания корневых файловых систем в образах, собираемых с использованием пакетов .rpm;
- *rootfs_ipk* для создания корневых файловых систем в образах, собираемых с использованием пакетов .ipk;
- *rootfsdebugfiles* для установки дополнительных файлов со сборочного хоста непосредственно в корневую файловую систему.

Корневая файловая система создаётся с использованием одного из файлов rootfs*.bbclass в соответствии с переменной PACKAGE_CLASSES. Информация о создании образов корневой файловой системы приведена в разделе Image Generation [1].

6.116. sanity.bbclass

Класс sanity проверяет наличие нужных для работы программ на сборочном хосте, чтобы уведомить пользователя о возможных проблемах при сборке. Этот класс также проверяет пользовательскую конфигурацию в файле local.conf для предотвращения ошибок при сборке. Включение этого класса обычно определяется политикой распространения.

6.117. scons.bbclass

Класс scons поддерживает задания для сборки программ, использующих систему сборки SCons. переменная EXTRA_OESCONS позволяет задать дополнительные параметры конфигурации, передаваемые команде scons.

6.118. sdl.bbclass

Класс sdl поддерживает задания для сборки программ, использующих библиотеку Simple DirectMedia Layer (SDL).

6.119. setuptools.bbclass

Класс setuptools поддерживает расширения Python версии 2.x, используемые системами сборки на основе setuptools. Такие задания должны наследовать класс setuptools.

6.120. setuptools3.bbclass

Класс setuptools3 поддерживает расширения Python версии 3.x, используемые системами сборки на основе setuptools. Такие задания должны наследовать класс setuptools3.

6.121. sign_rpm.bbclass

Класс sign_rpm поддерживает создание подписанных пакетов RPM.

6.122. sip.bbclass

Класс sip поддерживает задания для сборки или упаковки привязок Python на основе SIP.

6.123. siteconfig.bbclass

Класс siteconfig поддерживает функциональность настройки конфигурации сайта и применяется классом autotools для ускорения задачи do_configure.

6.124. siteinfo.bbclass

Класс siteinfo обеспечивает информацию о целевых системах, которая может быть нужна другим классам и заданиям.

В качестве примера рассмотрим инструменты Autotools, которым может потребоваться проверка, выполняемая на целевой аппаратной платформе. Поскольку в общем случае это невозможно при кросс-компиляции, используется информация о сайте для предоставления кэшированных результатов тестирования, что эти проверки можно было пропустить с сохранением корректных значений переменных. Результаты тестов хранятся в каталоге meta/site с сортировкой по таким категориям, как архитектура, порядок битов, используемая библиотека libc. Информация о сайте предоставляет список файлов, содержащих данные, относящиеся к конкретной сборке, в переменной CONFIG_SITE, которая автоматически выбирается инструментами Autotools.

Класс также предоставляет такие переменные, как SITEINFO_ENDIANNESS и SITEINFO_BITS, которые могут применяться в метаданных.

6.125. spdx.bbclass

Класс spdx объединяет в реальном масштабе времени сканирование лицензий, генерацию стандартного вывода SPDX и проверку лицензий в процессе сборки.

6.126. sstate.bbclass

Класс sstate обеспечивает поддержку общего состояния (sstate). По умолчанию класс включён через заданное по умолчанию значение INHERIT_DISTRO. Дополнительные сведения о sstate даны в разделе Shared State Cache [1].

6.127. staging.bbclass

Класс staging устанавливает файлы в отдельные рабочие каталоги заданий для sysroot.

- Задача do_populate_sysroot отвечает за обработку файлов, попадающих в sysroot заданий.
- Задача do_prepare_recipe_sysroot устанавливает файлы в рабочие каталоги отдельных заданий (WORKDIR).

Код класса staging достаточно сложен и обычно работает в 2 этапа, описанных ниже.

- На первом этапе обрабатываются задания, имеющие файлы, которые хотят использовать другие задания, зависящие от данного. Обычно эти зависимости указываются через задачу do_install в ${D}. Задача do_populate_sysroot копирует часть этих файлов в ${SYSROOT_DESTDIR}. Эти файлы определяются переменными SYSROOT_DIRS, SYSROOT_DIRS_NATIVE и SYSROOT_DIRS_BLACKLIST. Кроме того, задание может дополнительно настроить файлы, объявляя функцию обработки в SYSROOT_PREPROCESS_FUNCS.

  Из этих файлов создаётся объект общего состояния, который помещается в каталог tmp/sysroots-components/. Файлы сканируются на предмет жёстко заданных путей с местам исходной установки. Если местоположение найдено в текстовом файле, оно заменяется маркерами и создаётся список файлов, для которых нужны такие замены. Эти корректировки называют FIXME. Список сканируемых файлов определяет SSTATE_SCAN_FILES.

- На втором этапе обрабатываются задания, желающие использовать что-либо из других заданий и задающие зависимость в переменной DEPENDS. Задание будет иметь задачу do_prepare_recipe_sysroot, при выполнении которой создаются recipe-sysroot и recipe-sysroot-native в рабочем каталоге (WORKDIR). Система сборки OE создает жёсткие ссылки на копии соответствующих файлов из компонент sysroot в рабочий каталог. Если создание жёсткой ссылки невозможно, система сборки использует копии. Затем система сборки преобразует FIXME в пути в соответствии со списком, определенным на первом этапе. В заключение выполняются все файлы из ${bindir} в sysroot, имеющие префикс postinst-. Хотя такие сценарии пост-установки не рекомендуются для общего пользования, они позволяют решать некоторые проблемы, такие как создание пользователей или индексы модулей.

  Поскольку у заданий могут быть зависимости вне DEPENDS (например, do_unpack[depends] += "tar-native:do_populate_sysroot"), функция создания sysroot extend_recipe_sysroot добавляется в качестве предварительной для задач, чьи зависимости не указаны в DEPENDS, но работают аналогично.

  При установке зависимостей в sysroot код просматривает граф зависимостей и обрабатывает их так же, как зависимости из sstate. Это означает, например, что для естественного инструмента будут добавлены естественные зависимости, но не библиотеки для целевой системы. Применяется тот же код обработки зависимостей sstate, поэтому сборки будут идентичными, независимо от использования sstate. Более точное описание приведено для функции setscene_depvalid() в классе sstate.

  Система сборки аккуратно поддерживает манифесты устанавливаемых файлов, чтобы можно было установить любую нужную зависимость. Хэш sstate для установленных элементов также сохраняется, чтобы в случае изменения система сборки могла заново установить элемент.

6.128. syslinux.bbclass

Класс syslinux обеспечивает функции syslinux для сборки загружаемых образов и включает несколько переменных.

- Необязательная переменная INITRD указывает список образов файловых систем для объединения и использования в качестве начального RAM-диска (initrd).
- Необязательная переменная ROOTFS указывает образ файловой системы для включения как корневой.
- AUTO_SYSLINUXMENU включает создание автоматического меня при установке значения "1".
- LABELS указывает цели для автоматической настройки конфигурации.
- APPEND указывает строку append переопределяемую для каждой цели (label).
- SYSLINUX_OPTS указывает опции для добавления в файл syslinux (разделяются символом ;).
- SYSLINUX_SPLASH указывает фон для загрузочного меню VGA при использовании меню загрузки.
- SYSLINUX_DEFAULT_CONSOLE указывает принятую по умолчанию консоль (console=ttyX) для загрузки ядра.
- SYSLINUX_SERIAL задаёт дополнительный последовательный порт или выключает консоль при пустом имени.
- SYSLINUX_SERIAL_TTY задаёт дополнительный аргумент загрузки ядра "console=tty...".

6.129. systemd.bbclass

Класс systemd поддерживает задания, устанавливающие файлы модулей systemd. Класс не включается, пока значение systemd не добавлено в DISTRO_FEATURES.

В этом классе задание или Makefile (что вызывается в задаче do_install) устанавливает файлы модулей в каталог ${D}${systemd_unitdir}/system. Если устанавливаемые модули входят в пакеты, отличные от основного, в задании нужно установить переменную SYSTEMD_PACKAGES для указания пакетов, в которых будут установлены файлы.

В переменной SYSTEMD_SERVICE нужно указать имя службы, а также следует использовать переопределение имени пакета, к которому применяется значение. Если значение применяется к основному пакету задания, используется ${PN}. Например, для задания connman указывается SYSTEMD_SERVICE_${PN} = "connman.service". Для служб устанавливается автоматический запуск при загрузке, пока не задано SYSTEMD_AUTO_ENABLE = "disable". Дополнительная информация о systemd приведена в разделе Selecting an Initialization Manager [2].

6.130. systemd-boot.bbclass

Класс systemd-boot обеспечивает функции для загрузчика systemd-boot при сборке загружаемых образов. Это внутренний класс, не предназначенный для использования напрямую. Этот класс является результатом слияния класса gummiboot из прежних выпусков YP с проектом systemd.

Для использования класса устанавливается EFI_PROVIDER = "systemd-boot", в результате чего создаётся автономный загрузчик EFI, независимый от systemd. Этот класс использует и поддерживает переменные SYSTEMD_BOOT_CFG, SYSTEMD_BOOT_ENTRIES и SYSTEMD_BOOT_TIMEOUT. См. также документацию Systemd-boot.

6.131. terminal.bbclass

Класс terminal обеспечивает поддержку запуска терминальных сессий. Выбор терминала для сессии задаёт переменная OE_TERMINAL. Класс terminal применяется другими классами при необходимости запуска терминальной сессии. Например, это делает класс patch в предположении PATCHRESOLVE = user, а также классы cml1 и devshell.

6.132. testimage*.bbclass

Классы testimage* поддерживают автоматизированное тестирование образов с использованием QEMU и реального оборудования. Классы обслуживают загрузку теста и запуск образа. Для использования классов нужно их настроить в среде разработки. Рекомендуется использовать переменную IMAGE_CLASSES (а не INHERIT) для наследования класса testimage при автоматизированном тестировании образов.

Тестами служат команды, которые запускаются на целевой системе с помощью ssh. Тесты написаны на языке Python и используют модуль unittest.

Класс testimage.bbclass запускает тесты образа при вызове с помощью команды

```
$ bitbake -c testimage image
```

Класс testimage-auto выполняет тестирование образа после его создания (требуется TESTIMAGE_AUTO = 1). Дополнительная информация о тестировании представлена в разделе Performing Automated Runtime Testing [2].

6.133. testsdk.bbclass

Поддерживает запуск автоматизированных тестов для SDK при вызове

```
$ bitbake -c testsdk image
```

Рекомендуется использовать переменную IMAGE_CLASSES (а не INHERIT) для наследования класса testimage при автоматизированном тестировании SDK.

6.134. texinfo.bbclass

Этот класс нужно наследовать заданиям, при сборке которых вызываются утилиты texinfo. Созданы естественные и кросс-задания для использования фиктивных сценариев, обеспечиваемых texinfo-dummy-native, для повышения производительности. Задания для целевой архитектуры используют настоящие утилиты Texinfo (по умолчанию предоставляются хостом). Если нужно использовать задание Texinfo из состава системы сборки, можно удалить texinfo-native из переменной ASSUME_PROVIDED и makeinfo из SANITY_REQUIRED_UTILITIES.

6.135. tinderclient.bbclass

Класс tinderclient представляет результаты сборки внешнему экземпляру Tinderbox. Этот класс не поддерживается.

6.136. toaster.bbclass

Класс toaster собирает информацию о пакетах и образах, передавая её как события, которые может получать пользовательский интерфейс BitBake. Класс включается при работе пользовательского интерфейса Toaster и не предназначен для использования напрямую.

6.137. toolchain-scripts.bbclass

Класс toolchain-scripts обеспечивает сценарии, используемые при организации среды для установленных SDK.

6.138. typecheck.bbclass

Класс typecheck обеспечивает возможность проверки соответствия значений переменных в конфигурации уровня заданным для них типам. Система сборки OE позволяет определять типы с помощью флагов переменных type, например, IMAGE_FEATURES[type] = "list".

6.139. uboot-config.bbclass

Класс uboot-config обеспечивает поддержку настройки U-Boot для машины, которая указывается в задании как

```
UBOOT_CONFIG ??= <default>
UBOOT_CONFIG[foo] = "config,images"
```

Можно также указать машину в форме UBOOT_MACHINE = "config".

6.140. uninative.bbclass

Пытается изолировать систему сборки от библиотеки C дистрибутива сборочного хоста, чтобы обеспечить возможность многократного использования элементов общего состояния в разных дистрибутивах хоста. Если этот класс включён, в начале сборки загружается архив с собранной библиотекой C. В эталонном дистрибутиве Poky этот класс включён по умолчанию в файле meta/conf/distro/include/yocto-uninative.inc. Для других дистрибутивов, не основанных на poky также может применяться файл require conf/distro/include/yocto-uninative.inc. Другим вариантом является самостоятельная организация задания uninative-tarball, публикация полученного архива (например, по HTTP) и соответствующая установка переменных UNINATIVE_URL и UNINATIVE_CHECKSUM. Примером может служить файл meta/conf/distro/include/yocto-uninative.inc.

Класс uninative также безоговорочно применяется в eSDK. При сборке расширяемого SDK собирается uninative-tarball и полученный архив включается в SDK.

6.141. update-alternatives.bbclass

Класс update-alternatives помогает выбору вариантов при наличии нескольких источников, представляющих одну команду. Такие ситуации возникают, когда несколько команд, обеспечивающих одну или близкие функции, устанавливаются с одним именем. Например, команда ar доступна в busybox, binutils и elfutils. Класс update-alternatives обрабатывает переименование двоичных файлов, позволяя установить множество пакетов без конфликта. Команда ar будет работать независимо от того, какие пакеты установлены или впоследствии удалены. Класс переименовывает двоичные файлы в каждом пакете и создает символьную ссылку на пакет с высшим приоритетом при установке или удалении пакетов.

Для использования класса нужно определить несколько переменных:

- ALTERNATIVE;
- ALTERNATIVE_LINK_NAME;
- ALTERNATIVE_TARGET;
- ALTERNATIVE_PRIORITY.

Эти переменные указывают варианты команд, нужные пакету, обеспечивают пути для ссылок, используемые по умолчанию ссылки для целей и т. п. Информация по использованию переменных приведена в комментариях к файлу update-alternatives.bbclass.

Можно использовать команду update-alternatives непосредственно в заданиях, однако с классом в большинстве случаев работать проще.

6.142. update-rc.d.bbclass

Класс update-rc.d использует update-rc.d для безопасной установки сценариев инициализации от имени приложений. Система сборки OE заботится о таких деталях, как гарантии остановки сценария перед удалением пакета и запуска при установке. Управляют классом переменные INITSCRIPT_PACKAGES, INITSCRIPT_NAME и INITSCRIPT_PARAMS.

6.143. useradd*.bbclass

Классы useradd* поддерживают добавление пользователей или групп для пакетов на целевой платформе. Например, при наличии пакетов с системными службами, которые следует запускать от имени пользователя или группы, можно воспользоваться этими классами для возможности включения этих пользователей или групп. Задание meta-skeleton/recipes-skeleton/useradd/useradd-example.bb в дереве исходных кодов содержит простой пример, показывающий добавление трёх пользователей и групп для двух пакетов.

Класс useradd_base обеспечивает базовую функциональность для работы с пользователями и группами. Классы useradd* поддерживают переменные USERADD_PACKAGES, USERADD_PARAM, GROUPADD_PARAM и GROUPMEMS_PARAM. Класс useradd-staticids поддерживает добавление пользователей и групп со статическими значениями идентификаторов uid и gid. По умолчанию система сборки OE назначает значения uid и gid при добавлении пакетами пользователей и групп динамически во время установки пакета. Это хорошо работает для программ, которые не заботятся об окончательных значениях идентификаторов пользователей и групп. Однако при возникновении проблем с использованием недетерминированных значений uid и gid можно переопределить поведение, задав статические идентификаторы. В этом случае система сборки OE смотрит нужные значения в файлах files/passwd и files/group по пути BBPATH. Для использования статических значений uid и gid нужно задать несколько переменных (USERADDEXTENSION, USERADD_UID_TABLES, USERADD_GID_TABLES, USERADD_ERROR_DYNAMIC).

Класс useradd-staticids не используется напрямую, а включается или отключается через переменную USERADDEXTENSION. Если включить или отключить класс в настроенной системе, в TMPDIR могут оказаться некорректные значения uid и gid, однако удаление TMPDIR позволяет решить эту проблему.

6.144. utility-tasks.bbclass

Класс utility-tasks обеспечивает поддержку вспомогательных задач, применяемых для всех заданий, таких как do_clean и do_listtasks. Класс включён по умолчанию, поскольку он наследуется классом base.

6.145. utils.bbclass

Класс utils обеспечивает некоторые полезные функции Python, которые обычно применяются в inline-выражениях Python (например, ${@...}). Класс включён по умолчанию, поскольку он наследуется классом base.

6.146. vala.bbclass

Класс vala поддерживает задания, которым нужно собирать программы, написанные с использованием языка Vala.

6.147. waf.bbclass

Класс waf поддерживает задания, собирающие программы с использованием системы сборки Waf. Переменная EXTRA_OECONF или PACKAGECONFIG_CONFARGS позволяет задать дополнительные параметры конфигурации, передаваемые Waf.

Глава 7. Задачи

Задачи являются блоками выполнения BitBake. Задания (файлы .bb) применяют задачи для настройки, компиляции и упаковки программ. В этой главе приведен справочник по задачам, определенным в системе сборки OE.

7.1. Обычные задачи сборки заданий

В последующих параграфах рассмотрены обычные задачи, связанные со сборкой заданий. Дополнительные сведения о задачах и зависимостях приведены в разделах Tasks и Dependencies[4].

7.1.1. do_build

Используемая по умолчанию задача для всех сборочных заданий, зависящая от других задач, нужных для сборки.

7.1.2. do_compile

Компиляция исходного кода. Задача выполняется из каталога, заданного переменной ${B}. По умолчанию эта задача вызывает функцию oe_runmake, если найден файл сборки (Makefile, makefile или GNU makefile). В противном случае do_compile не делает ничего.

7.1.3. do_compile_ptest_base

Компилирует набор используемых при работе (runtime) тестов, который включён в собираемое задание.

7.1.4. do_configure

Настраивает исходные коды выбирая опции конфигурации и сборки для собираемой программы. Задача выполняется в текущем рабочем каталоге, заданном переменной ${B}. По умолчанию эта задача вызывает функцию oe_runmake, если найден файл сборки (Makefile, makefile или GNU makefile) и не установлено CLEANBROKEN = "1". В противном случае do_configure не делает ничего.

7.1.5. do_configure_ptest_base

Настраивает набор runtime-тестов, включаемых в собираемые программы.

7.1.6. do_deploy

Записывает выходные файлы для развёртывания в ${DEPLOY_DIR_IMAGE}. Задача выполняется из текущего рабочего каталога, указанного ${B}. Заданиям, применяющим эту задачу, нужно наследовать класс deploy и записывать вывод в каталог ${DEPLOYDIR} (не следует путать его с ${DEPLOY_DIR}). Класс deploy устанавливает do_deploy как задачу с общим состоянием (sstate), которую можно ускорить с помощью механизма sstate, обеспечивающего копирование вывода из ${DEPLOYDIR} в ${DEPLOY_DIR_IMAGE}. Не записывайте вывод напрямую в ${DEPLOY_DIR_IMAGE}, поскольку это нарушит работу sstate.

Задача do_deploy не добавляется по умолчанию и должна указываться вручную. Если нужно запустить задачу после do_compile, можно выполнить

```
addtask deploy after do_compile
```

Добавление do_deploy после других задач выполняется так же. Не нужно добавлять before do_build в команду addtask (хотя это безвредно), поскольку класс base содержит

```
do_build[recrdeptask] += "do_deploy"
```

Дополнительная информация о развёртывании приведена в разделе Dependencies [4].

При повторном выполнении задачи do_deploy предыдущий вывод удаляется (очищается).

7.1.7. do_fetch

Извлекает исходный код, используя переменную SRC_URI и префикс аргумента для определения нужного сборщика.

7.1.8. do_image

Запускает процесс создания образа после того, как система сборки OE выполнит задачу do_rootfs, в которой указываются выбранные для установки приложения, создаётся корневая файловая система и выполняется пост-обработка. Задача do_image выполняет предварительную обработку образа с помощью IMAGE_PREPROCESS_COMMAND и динамически генерирует при необходимости задачи do_image_*. Дополнительные сведения о создании образов приведены в разделе Image Generation [1].

7.1.9. do_image_complete

Завершает процесс генерации образа. Эта задача выполняется после того, как система сборки OE запустит задачу do_image, в которой выполняется предварительная обработка и создаётся образ с помощью динамически генерируемых задач do_image_*.

Задача do_image_complete выполня пост-обработку образа с помощью IMAGE_POSTPROCESS_COMMAND. Создание образов описано в разделе Image Generation [1].

7.1.10. do_install

Копирование файлов, которые должны быть включены в пакет, в область ${D}. Задача выполняется из каталога, заданного переменной ${B}, который служит каталогом компиляции. Задача do_install, как и другие задачи, напрямую или косвенно зависящие от установленных файлов (например do_package, do_package_write_*, do_rootfs), запускаются в fakeroot [1].

При установке файлов нужно соблюдать осторожность в выборе идентификаторов владельца и группы, чтобы для них не были указаны неопределённые значения. Некоторые методы копирования файлов, особенно команда cp с рекурсией, могут сохранять UID и/или GID исходного файла, что обычно нежелательно. Проверка host-user-contaminated позволяет контролировать владельца и группу для файлов.

Ниже перечислены некоторые методы установки файлов.

- Утилита install (предпочтительный вариант).
- Команда cp с опцией --no-preserve=ownership.
- Команда tar с опцией --no-same-owner (см. файл bin_package.bbclass в каталоге meta/classes дерева кодов).

7.1.11. do_install_ptest_base

Копирует файлы набора тестов runtime из области компиляции в область хранения.

7.1.12. do_package

Анализирует содержимое области ${D} и разделяет его на части по доступным пакетам и файлам. Задача использует переменные PACKAGES и FILES.

Задача do_package вместе с do_packagedata сохраняет некоторые важные метаданные пакетов (см. описание переменной PKGDESTWORK и раздел Automatically Added Runtime Dependencies [1]).

7.1.13. do_package_qa

Запускает проверки QA для файлов пакета (см. параграф 6.56. insane.bbclass).

7.1.14. do_package_write_deb

Создает пакеты Debian (файлы *.deb) и помещает их в каталог ${DEPLOY_DIR_DEB} в области источника пакетов. Дополнительные сведения об упаковке приведены в разделе Package Feeds [1].

7.1.15. do_package_write_ipk

Создает пакеты IPK (файлы *.ipk) и помещает их в каталог ${DEPLOY_DIR_IPK} в области источника пакетов. Дополнительные сведения об упаковке приведены в разделе Package Feeds [1].

7.1.16. do_package_write_rpm

Создает пакеты RPM (файлы *.rpm) и помещает их в каталог ${DEPLOY_DIR_RPM} в области источника пакетов. Дополнительные сведения об упаковке приведены в разделе Package Feeds [1].

7.1.17. do_package_write_tar

Создает архивы (tarball) и помещает их в каталог ${DEPLOY_DIR_TAR} (см. раздел Package Feeds [1]).

7.1.18. do_packagedata

Сохраняет метаданные пакеты, созданные задачей do_package в каталоге PKGDATA_DIR для глобального доступа.

7.1.19. do_patch

Находит patch-файлы и применяет их к исходному коду. После извлечения и распаковки исходных файлов система сборки использует операторы SRC_URI из задания для поиска и применения исправлений к исходному коду. Система сборки использует переменную FILESPATH для определения принятого по умолчанию набора каталогов поиска patch-файлов. Файлы исправлений по умолчанию используют расширение *.patch или *.diff и хранятся в подкаталоге каталога с файлом задания. Например, задание bluez5 уровня OE-Core (poky/meta) использует каталог poky/meta/recipes-connectivity/bluez5 и включает два patch-файла. В задании bluez5 операторы SRC_URI указывают на исходные файлы и исправления, нужные для сборки пакета.

В случае задания bluez5_5.48.bb операторы применяются SRC_URI из включаемого файла bluez5.inc.

Как отмечено выше, система сборки считает файлы с расширениями .patch и .diff файлами исправлений. Однако можно использовать параметр apply=yes с оператором SRC_URI для указания в качестве исправлений любых файлов.

```
SRC_URI = " \
     git://path_to_repo/some_package \
     file://file;apply=yes \
"
```

И наоборот, если у вас есть целый каталог patch-файлов и нужно исключить применение части их задачей do_patch, следует использовать параметр apply=no с оператором SRC_URI.

```
SRC_URI = " \
     git://path_to_repo/some_package \
     file://path_to_lots_of_patch_files \
```

```
        file://path_to_lots_of_patch_files/patch_file5;apply=no \
    "
```

Если все файлы в каталоге имеют расширение .patch или .diff, будут применены все исправления, кроме patch_file5.

Дополнительная информация о применении patch-файлов дана в разделах Patching [1] и Patching Code [2].

7.1.20. do_populate_lic

Записывает лицензионную информацию для задания, которая затем собирается при создании образа.

7.1.21. do_populate_sdk

Создает структуру файлов и каталогов для устанавливаемого SDK (см. раздел SDK Generation [1]).

7.1.22. do_populate_sysroot

Помещает (копирует) часть файлов, установленных задачей do_install в соответствующий каталог sysroot. Информация о доступе к этим файлам из других заданий приведена в описаниях переменных STAGING_DIR*. Каталоги, которые обычно не нужны другим заданиям (например, /etc), по умолчанию не копируются. Копируемые по умолчанию каталоги приведены в описаниях переменных SYSROOT_DIRS*. Значения этих переменных можно изменить в задании, если нужно добавить или исключить каталоги, доступные другим заданиям в процессе сборки.

Задача do_populate_sysroot относится к задачам с общим состоянием (sstate), что позволяет ускорить её с помощью sstate. При повторном выполнении задачи предшествующий вывод удаляется (очистка).

7.1.23. do_prepare_recipe_sysroot

Устанавливает файлы в sysroot отдельных заданий (т. е. recipe-sysroot и recipe-sysroot-native в каталоге ${WORKDIR} на основе зависимостей из DEPENDS). Дополнительная информация приведена в параграфе 6.127. staging.bbclass.

7.1.24. do_rm_work

Удаляет рабочие файлы после завершения их использования системой сборки OE (см. параграф 6.114. rm_work.bbclass).

7.1.25. do_unpack

Распаковывает исходный код в рабочий каталог, указанный ${WORKDIR}. Переменная S тоже играет роль в выборе окончательного места размещения распакованных файлов. Дополнительная информация о распаковке файлов исходного кода приведена в разделе Source Fetching [1] а также описаниях переменных WORKDIR и S.

7.2. Вызываемые вручную задачи

Описанные в этом разделе задачи обычно запускаются вручную, например, с помощью bitbake -c.

7.2.1. do_checkpkg

Обеспечивает информацию о задании, включая «восходящую» версию и статус. Для проверки версии и статуса задания служат приведённые ниже команды devtool (Глава 8. Краткий справочник по работе с devtool).

```
$ devtool latest-version
$ devtool check-upgrade-status
```

Информация о проверке статуса обновления задания приведена в параграфе 8.9. Проверка статуса обновления задания. Для сборки задачи checkpkg используется опция -c с именем задачи, как показано ниже.

```
$ bitbake core-image-minimal -c checkpkg
```

По умолчанию результат сохраняется в $LOG_DIR (например, $BUILD_DIR/tmp/log).

7.2.2. do_checkuri

Проверяет пригодность значения SRC_URI.

7.2.3. do_clean

Удаляет все выходные файлы для цели от задачи do_unpack вперёд (do_unpack, do_configure, do_compile, do_install, do_package). Задача запускается командой bitbake -c clean recipe. Запуск задачи не удаляет кэш-файлы sstate, поэтому при отсутствии изменений и пересборке задания после очистки, выходные файлы будут просто восстановлены из кэша sstate. Если нужно удалить кэш для задания, должна применяться задача do_cleansstate (bitbake -c cleansstate recipe).

7.2.4. do_cleanall

Удаляет все выходные файлы, кэш общего состояния (sstate) и загруженные файлы для цели (содержимое DL_DIR). Задача do_cleanall идентична do_cleansstate за исключением того, что удаляет загруженные исходные файлы. Задача запускается командой bitbake -c cleanall recipe. Обычно задача cleanall не применяется, если не нужно начать сборку заново с задачи do_fetch.

7.2.5. do_cleansstate

Удаляет все выходные файлы и кэш общего состояния (sstate) файлы для цели. Задача do_cleansstate идентична do_clean, но дополнительно удаляет кэш общего состояния (sstate). Задача запускается командой bitbake -c cleansstate recipe. При запуске do_cleansstate система сборки OE больше не применяет sstate, поэтому гарантируется сборка задания «с нуля».

Задача do_cleansstate не может удалять файлы sstate с удалённого зеркала sstate. Если нужно собрать цель с нуля при использовании удалённых зеркал, нужно задать опцию -f, как показано ниже.

```
$ bitbake -f -c do_cleansstate target
```

7.2.6. do_devpyshell

Запускает оболочку, в которой интерактивный интерпретатор Python позволяет взаимодействовать со средой сборки BitBake. Из этой оболочки можно напрямую проверять и устанавливать биты из хранилища данных, а также вызывать функции как в среде BitBake. Описание devpyshell приведено в разделе Using a Development Python Shell [2].

7.2.7. do_devshell

Запускает оболочку со средой, настроенной на разработку и/или отладку. Работа с оболочкой devshell описана в разделе Using a Development Shell [2].

7.2.8. do_listtasks

Перечисляет определённые для цели задачи.

7.2.9. do_package_index

Создает или обновляет индекс в области хранилищ пакетов. Эта задача не вызывается командой bitbake -c как другие задачи этого раздела. Поскольку задача предназначена для задания package-index, она вызывается командой bitbake package-index.

7.3. Задачи, связанные с образами

Перечисленные ниже задачи относятся к заданиям для образов.

7.3.1. do_bootimg

Создает загружаемый live-образ. Типы образов рассмотрены в описании переменной IMAGE_FSTYPES.

7.3.2. do_bundle_initramfs

Объединяет образ initramfs и ядро в один образ (см. CONFIG_INITRAMFS_SOURCE).

7.3.3. do_rootfs

Создает корневую файловую систему (см. раздел Image Generation [1]).

7.3.4. do_testimage

Загружает образ и выполняет тесты на нем (см. раздел Performing Automated Runtime Testing [2]).

7.3.5. do_testimage_auto

Загружает образ и выполняет тесты на нем сразу после сборки. Задача включается установкой TESTIMAGE_AUTO = 1.

Дополнительную информацию можно найти в разделе Performing Automated Runtime Testing [2].

7.4. Задачи, связанные с ядром

Описанные ниже задачи относятся к заданиям для ядра. Некоторые задачи (например, do_menuconfig) применимы также к заданиям, использующим настройку в стиле ядра Linux (например BusyBox).

7.4.1. do_compile_kernelmodules

Запускает этап сборки модулей ядра (если нужно), включающий 1) сборку ядра (vmlinux) и 2) модулей (make modules).

7.4.2. do_diffconfig

При вызове пользователем эта задача создает файл, содержащий различия между исходной конфигурацией, созданной задачей do_kernel_configme и пользовательской конфигурацией (например, результат do_kernel_menuconfig). Этот файл может служить для создания фрагмента конфигурации, содержащего лишь различия. Задачу можно вызвать командой вида bitbake linux-yocto -c diffconfig.

Дополнительную информацию можно найти в разделе Creating Configuration Fragments [3].

7.4.3. do_kernel_checkout

Преобразует недавно распакованные коды ядра в форму, подходящую для системы сборки OE. Поскольку исходные коды ядра могут извлекаться разными способами, задача do_kernel_checkout task обеспечивает чёткую рабочую копию ядра с корректным выбором ветви.

7.4.4. do_kernel_configcheck

Служит для проверки конфигурации, созданной задачей do_kernel_menuconfig и выводит предупреждения в случае отсутствия запрошенной конфигурации в финальном файле .config или переопределения правил конфигурации во фрагменте аппаратной конфигурации. Задачу можно запустить явно для просмотра вывода с помощью команды bitbake linux-yocto -c kernel_configcheck -f. Дополнительные сведения можно найти в разделе Validating Configuration [3].

7.4.5. do_kernel_configme

После наложения правок на ядро с помощью задачи do_patch задача do_kernel_configme собирает и объединяет все фрагменты конфигурации ядра, которые затем могут быть переданы этапу настройки конфигурации. Здесь также применяются опции из заданных пользователем файлов defconfig и режимы настройки --allnoconfig.

7.4.6. do_kernel_menuconfig
Вызывается пользователем для работы с файлом .config, используемым для сборки задания linux-yocto и запускает инструмент настройки конфигурации ядра Linux, который позволяет настроить параметры ядра. Эту задачу можно запустить командой bitbake linux-yocto -c menuconfig. Информация о настройке конфигурации ядра приведена в разделе Using menuconfig [3].

7.4.7. do_kernel_metadata
Собирает все свойства, нужные для сборки данного ядра, независимо от их источника (SRC_URI или репозиторий Git). После сбора задача преобразует свойства в набор фрагментов конфигурации и patch-файлов, применяемых последующими задачами, такими как do_patch и do_kernel_configme.

7.4.8. do_menuconfig
Запускает команду make menuconfig для ядра (см. раздел Using menuconfig [3]).

7.4.9. do_savedefconfig
При вызове пользователем создает файл defconfig, который может использоваться вместо принятого по умолчанию. Сохранённый файл defconfig содержит различия между заданным по умолчанию файлом defconfig и внесённые пользователем изменения (с использованием других методов, таких как задача do_kernel_menuconfig). Задачу можно вызвать командой bitbake linux-yocto -c savedefconfig.

7.4.10. do_shared_workdir
После компиляции ядра, но до компиляции модулей задача копирует файлы, требуемые для сборки модулей и созданные при сборке ядра, в общий рабочий каталог. После копирования задача do_compile_kernelmodules может собрать модули на следующем этапе.

7.4.11. do_sizecheck
После сборки ядра эта задача проверяет размер ядра с вырезанными символами, сравнивая его с переменной KERNEL_IMAGE_MAXSIZE. Если переменная задана и размер ядра превышает её значение, выдаётся предупреждение.

7.4.12. do_strip
При установленной переменной KERNEL_IMAGE_STRIP_EXTRA_SECTIONS эта задача вырезает заданные переменной разделы из файла vmlinux. Обычно это применяется для исключения таких разделов как .comment для снижения размера ядра.

7.4.13. do_validate_branches
После распаковки ядра и перед наложением правок (patch) эта задача проверяет наличие ветвей машины и метаданных, указанных переменной SRCREV. Если ветви не найдены, а переменная AUTOREV не используется, задача do_validate_branches вызывает ошибку сборки.

7.5. Прочие задачи

7.5.1. do_spdx
Этап сборки, принимающий исходный код и сканирующий его на удалённом сервере FOSSOLOGY для создания документа SPDX. Задача применима только к классу spdx.

Глава 8. Краткий справочник по работе с devtool
Консольный инструмент devtool обеспечивает множество функций для сборки, тестирования и подготовки пакетов. Эта команда доступная вместе с bitbake и является важнейшей частью расширяемых SDK. Здесь приведено краткое описание devtool, а более полные сведения содержатся в разделе Using the Extensible SDK [7].

8.1. Получение справки
Команды devtool организованы подобно Git и включают множество субкоманд для разных функций, которые можно увидеть по команде devtool —help.

```
$ devtool -h
NOTE: Starting bitbake server...
usage: devtool [--basepath BASEPATH] [--bbpath BBPATH] [-d] [-q]
               [--color COLOR] [-h]
               <subcommand> ...

OpenEmbedded development tool

options:
  --basepath BASEPATH   Base directory of SDK / build directory
  --bbpath BBPATH       Explicitly specify the BBPATH, rather than getting it
                        from the metadata
  -d, --debug           Enable debug output
  -q, --quiet           Print only errors
  --color COLOR         Colorize output (where COLOR is auto, always, never)
  -h, --help            show this help message and exit

subcommands:
  Beginning work on a recipe:
```

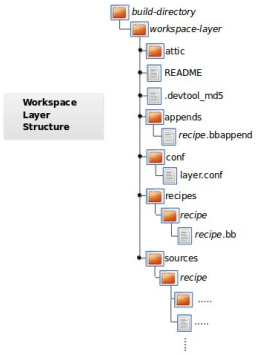
```
    add                   Add a new recipe
    modify                Modify the source for an existing recipe
    upgrade               Upgrade an existing recipe
  Getting information:
    status                Show workspace status
    search                Search available recipes
    latest-version        Report the latest version of an existing recipe
    check-upgrade-status  Report upgradability for multiple (or all) recipes
  Working on a recipe in the workspace:
    build                 Build a recipe
    rename                Rename a recipe file in the workspace
    edit-recipe           Edit a recipe file
    find-recipe           Find a recipe file
    configure-help        Get help on configure script options
    update-recipe         Apply changes from external source tree to recipe
    reset                 Remove a recipe from your workspace
    finish                Finish working on a recipe in your workspace
  Testing changes on target:
    deploy-target         Deploy recipe output files to live target machine
    undeploy-target       Undeploy recipe output files in live target machine
    build-image           Build image including workspace recipe packages
  Advanced:
    create-workspace      Set up workspace in an alternative location
    export                Export workspace into a tar archive
    import                Import exported tar archive into workspace
    extract               Extract the source for an existing recipe
    sync                  Synchronize the source tree for an existing recipe
Use devtool <subcommand> --help to get help on a specific command
```

Как указано на справочной странице можно получить дополнительные сведения о конкретных командах, указав имя команды и опцию --help.

```
$ devtool add --help
NOTE: Starting bitbake server...
usage: devtool add [-h] [--same-dir | --no-same-dir] [--fetch URI]
                   [--fetch-dev] [--version VERSION] [--no-git]
                   [--srcrev SRCREV | --autorev] [--srcbranch SRCBRANCH]
                   [--binary] [--also-native] [--src-subdir SUBDIR]
                   [--mirrors] [--provides PROVIDES]
                   [recipename] [srctree] [fetchuri]

Adds a new recipe to the workspace to build a specified source tree. Can
optionally fetch a remote URI and unpack it to create the source tree.

arguments:
  recipename            Name for new recipe to add (just name - no version,
                        path or extension). If not specified, will attempt to
                        auto-detect it.
  srctree               Path to external source tree. If not specified, a
                        subdirectory of
                        /home/scottrif/poky/build/workspace/sources will be
                        used.
  fetchuri              Fetch the specified URI and extract it to create the
                        source tree

options:
  -h, --help            show this help message and exit
  --same-dir, -s        Build in same directory as source
  --no-same-dir         Force build in a separate build directory
  --fetch URI, -f URI   Fetch the specified URI and extract it to create the
                        source tree (deprecated - pass as positional argument
                        instead)
  --fetch-dev           For npm, also fetch devDependencies
  --version VERSION, -V VERSION
                        Version to use within recipe (PV)
  --no-git, -g          If fetching source, do not set up source tree as a git
                        repository
  --srcrev SRCREV, -S SRCREV
                        Source revision to fetch if fetching from an SCM such
                        as git (default latest)
  --autorev, -a         When fetching from a git repository, set SRCREV in the
                        recipe to a floating revision instead of fixed
  --srcbranch SRCBRANCH, -B SRCBRANCH
                        Branch in source repository if fetching from an SCM
                        such as git (default master)
  --binary, -b          Treat the source tree as something that should be
                        installed verbatim (no compilation, same directory
                        structure). Useful with binary packages e.g. RPMs.
  --also-native         Also add native variant (i.e. support building recipe
                        for the build host as well as the target machine)
  --src-subdir SUBDIR   Specify subdirectory within source tree to use
  --mirrors             Enable PREMIRRORS and MIRRORS for source tree fetching
                        (disable by default).
  --provides PROVIDES, -p PROVIDES
                        Specify an alias for the item provided by the recipe.
                        E.g. virtual/libgl
```

8.2. Структура уровня workspace

Команда devtool использует уровень Workspace, в котором выполняется сборка. Этот уровень не связан с какой-либо конкретной командой devtool, а служит рабочей областью для инструмента. Структура workspace показана на рисунке.



attic
Каталог, создаваемый в случаях, когда devtool хочет что-либо сохранить при выполнении команды devtool reset. Например, можно ввести команду devtool add, изменить задание, а затем вызвать devtool reset и devtool отметит изменённые файлы и перенесёт их в каталог attic.

README
Информация об уровне workspace и управлении им.

.devtool_md5
Файл контрольной суммы, используемый devtool.

appends
Каталог с файлами *.bbappend, указывающими внешний источник.

conf
Конфигурационный каталог с файлом layer.conf.

recipes
Каталог с заданиями, каждое из которых размещается с своём каталоге, имя которого соответствует имени задания. Программа devtool помещает в эти каталоги файлы .bb для заданий.

sources
Каталог с рабочей копией файлов исходного кода для сборки задания. По умолчанию этот каталог используется в качестве дерева источников, если не задано иного пути к ним. Каталог включает подкаталоги с файлами соответствующих заданий.

8.3. Добавление задания на уровень workspace

Команда devtool add добавляет задание на уровень workspace. Задание не нужно создавать заранее, это сделает devtool. Файлы исходных кодов для задания должны размещаться во внешней области.

Ниже приведен пример, добавляющий задание jackson на уровень workspace и исходными кодами из каталога /home/user/sources/jackson.

```
$ devtool add jackson /home/user/sources/jackson
```

Если в момент добавления задания уровня workspace ещё не было, команда создаст его и заполнит в соответствии с параграфом 8.2. Структура уровня workspace. Если уровень уже создан, команда devtool add добавит файлы задания и дополнения, а также файлы исходного кода в существующую структуру. Создаётся файл .bbappend для указания внешнего дерева источников.

Если для задания указаны зависимости при работе, нужно убедиться в наличии соответствующих пакетов на целевой платформе перед попыткой запустить приложение. Если нужных пакетов (например, библиотек) нет на целевой платформе, приложение не сможет работать (8.14. Развёртывание программ на целевой машине).

По умолчанию devtool add использует последний выпуск (т. е. master) при извлечении файлов из удалённого URI. В некоторых случаях можно задать выпуск по ветви, тегу или хэш-значению фиксации опциями команды devtool add.

- Для задания ветви служит опция --srcbranch (выбирает ветвь warrior)

```
$ devtool add --srcbranch warrior jackson /home/user/sources/jackson
```

- Для задания тега или хэш-значения служит опция --srcrev.

```
$ devtool add --srcrev yocto-2.7.1 jackson /home/user/sources/jackson
$ devtool add --srcrev some_commit_hash /home/user/sources/jackson
```

В этих примерах выбирается тег yocto-2.7.1 и фиксация с хэшем some_commit_hash.

Если нужно использовать наиболее свежую версию при каждой сборке, следует указать опцию --autorev или -a.

8.4. Извлечение исходного кода для имеющегося задания

Команда devtool extract служит для извлечения исходных кодов имеющегося задания, имя которого (без версии, пути и расширения) должно быть указано в команде вместе с каталогом, куда следует поместить извлечённые файлы. Опции команды позволяют указать ветвь, в которую будет извлекаться код, и управлять сохранением временного каталога.

8.5. Синхронизация дерева исходного кода

Команда devtool sync позволяет синхронизировать исходный код для имеющегося задания. В команде нужно указать имя задания (без версии, пути и расширения) вместе с каталогом, куда следует поместить извлечённые файлы. Опции команды позволяют указать ветвь, в которую будет извлекаться код, и управлять сохранением временного каталога.

8.6. Изменение задания

Команда devtool modify служит для изменения источников имеющегося задания. Она похожа на команду add, но не создает нового задания на уровне workspace, поскольку задание уже присутствует в другом уровне. Команда извлекает файлы исходного кода, организуя их в локальный репозиторий Git, если источник ещё не был получен из Git, выбирает ветвь для разработки и применяет все правки, зафиксированные (commit) в задании. При вводе команды devtool modify recipe программа devtool будет использовать оператор SRC_URI в имеющемся задании для указания восходящего источника и извлечёт исходные файлы в заданное по умолчанию место, используя по умолчанию ветвь devtool.

8.7. Редактирование задания

Команда devtool edit-recipe запускает редактор, указанный в переменной EDITOR внутри файла задания. В команде нужно указывать корневое имя задания (без версии и расширения), а файл задания должен размещаться в структуре workspace. Однако эти требования можно переопределить с помощью опции -a или --any-recipe.

8.8. Обновление задания

Команда devtool update-recipe служит для обновления задания правками (patch), отражающими изменения в исходных файлах. Например, при намерении работать с тем или иным кодом можно сначала использовать команду devtool modify для извлечения кода и организации рабочего пространства. Затем можно изменить, скомпилировать и протестировать код. Когда результат будет достигнут и изменения зафиксированы (commit) в репозитории Git, можно воспользоваться командой devtool update-recipe для создания patch-файлов и обновления задания.

```
$ devtool update-recipe recipe
```

Команда devtool update-recipe будет игнорировать незафиксированные изменения.

Зачастую программные настройки будут меняться в пользовательском уровне а не в исходном задании. В таких случаях можно использовать опцию -a или --append в команде devtool update-recipe, позволяющую указать уровень.

```
$ devtool update-recipe recipe -a base-layer-directory
```

При этом создаётся файл дополнения *.bbappend в нужном каталоге указанного уровня, который может (не обязательно) быть включён в файл bblayers.conf. Если файл дополнения уже имеется, команда обновит его.

8.9. Проверка статуса обновления задания

Восходящие задания время от времени меняются, поэтому может возникнуть потребность в их обновлении. Для проверки статуса задания служит команда devtool check-upgrade-status, которая выводит таблицу текущих версий заданий, последних версий этих заданий, адреса электронной почты сопровождающих и другую информацию, такую как хэш-значения фиксаций и возможные причины обновления. Для уровня oe-core адреса сопровождающих берутся из файла maintainers.inc. Если задание использует сборщик Git fetcher, а не архив, хэш фиксации указывается для тега последней версии.

Как для всех команд devtool можно получить справку

```
$ devtool check-upgrade-status -h
NOTE: Starting bitbake server...
usage: devtool check-upgrade-status [-h] [--all] [recipe [recipe ...]]

Prints a table of recipes together with versions currently provided by
recipes, and latest upstream versions, when there is a later version available

arguments:
  recipe      Name of the recipe to report (omit to report upgrade info for
              all recipes)

options:
 -h, --help  show this help message and exit
  --all, -a   Show all recipes, not just recipes needing upgrade
```

Если в команде не указано конкретное задание, проверяться будут все заданий для всех уровней в конфигурации. Ниже приведена часть выходной таблицы для всех заданий. Отметим, что в списке указана причина не обновлять задание base-passwd, связанная с тем, что для этого требуется обновить также cdebconf, что достаточно сложно. Когда причина отказа от обновления указана, она обычно записывается в файл задания (RECIPE_NO_UPDATE_REASON). Пример можно найти в файле base-passwd.bb.

```
$ devtool check-upgrade-status
    ...
    NOTE: acpid                     2.0.30          2.0.31
Ross Burton <ross.burton@intel.com>
    NOTE: u-boot-fw-utils           2018.11         2019.01
Marek Vasut <marek.vasut@gmail.com>
d3689267f92c5956e09cc7d1baa4700141662bff
    NOTE: u-boot-tools              2018.11         2019.01
Marek Vasut <marek.vasut@gmail.com>
d3689267f92c5956e09cc7d1baa4700141662bff
     ...
    NOTE: base-passwd               3.5.29          3.5.45
Anuj Mittal <anuj.mittal@intel.com>  cannot be updated due to: Version
3.5.38 requires cdebconf for update-passwd utility
    NOTE: busybox                   1.29.2          1.30.0
Andrej Valek <andrej.valek@siemens.com>
    NOTE: dbus-test                 1.12.10         1.12.12
Chen Qi <Qi.Chen@windriver.com>
```

8.10. Обновление задания

По мере развития программ задания обновляются с новыми версиями. Разработчику следует поддерживать актуальность своих локальных заданий, для чего имеется несколько методов, описанных в разделе Upgrading Recipes [2]. Здесь описано обновление с помощью команды devtool upgrade. Перед обновлением задания следует проверить его статус, как описано в параграфе 8.9. Проверка статуса обновления задания.

Команда devtool upgrade обновляет имеющееся задание до последней версии восходящего источника. Файл обновлённого задания и связанные с ним файлы помещаются в workspace и при необходимости распаковываются в указанное место дерева источников. В процессе обновления связанные с заданием patch-файлы также обновляются или добавляются.

При использовании команды devtool upgrade нужно указывать базовое имя задания (без версии, пути и расширения), а также каталог, в который нужно извлечь исходный код. Опции команды позволяют управлять номером версии для обновления (PV), выпуском (SRCREV), применением patch-файлов и т. п.

Дополнительную информацию о работе с devtool можно найти в разделе Use devtool upgrade to Create a Version of the Recipe that Supports a Newer Version of the Software [7], а примеры использования devtool upgrade - в Using devtool upgrade [2].

8.11. Сброс задания

Команда devtool reset удаляет задание и его конфигурацию (например, файл .bbappend) с уровня workspace. Следует понимать, что команда не переносит файлов задания, поэтому нужно поместить завершённое задание и файл дополнения в нужное место за пределами workspace перед использованием команды devtool reset. Если команда devtool reset обнаруживает, что файл задания или дополнения был изменён, эти изменённые файлы сохраняются в каталоге attic на уровне workspace. Ниже приведен пример сброса каталога workspace, содержащего задание mtr.

```
$ devtool reset mtr
NOTE: Cleaning sysroot for recipe mtr...
NOTE: Leaving source tree /home/scottrif/poky/build/workspace/sources/mtr as-is; if you no
  longer need it then please delete it manually
```

8.12. Сборка задания

Команда devtool build служит для сборки заданий и эквивалентна команде bitbake -c populate_sysroot. При использовании команды нужно указывать базовое имя задания (без версии, пути и расширения). Опция -s или --disable-parallel-make позволяет отключить при сборке параллельные операции.

8.13. Сборка образа

Команда devtool build-image служит для сборки образа со включением в него пакетов из workspace. Команда удобна для создания образов в процессе тестирования приложений. Для окончательной установки пакета в образ следует отредактировать задание для образа. Команда требует указать образ, куда включаются пакеты и имеет вид

```
$ devtool build-image image
```

8.14. Развёртывание программ на целевой машине

Команда devtool deploy-target служит для развёртывания вывода сборки «вживую» на целевой машине.

```
$ devtool deploy-target recipe target
```

Цель указывается адресом или именем и должна поддерживать сервер SSH (user@hostname[:destdir]).

Команда размещает все файлы, установленные задачей do_install. На целевой платформе не требуется менеджер пакетов, а при наличии он обходится. Команда deploy-target предназначена лишь для разработки и её не следует применять для создания окончательного образа.

Поведение развёрнутых приложений в некоторых случаях может оказаться неожиданным, если развёртывается новое приложение, задание для сборки которого учитывает все рабочие зависимости, на целевой платформе нет пакетов, от которых приложение зависит. Некорректное поведение объясняется тем, что команда devtool deploy-target не развёртывает пакеты (например, библиотеки), от которых зависит новое приложение. Поэтому при вызове новым приложением отсутствующей функции результаты могут быть неожиданными. Для предотвращения подобных проблем следует заранее установить все нужные пакеты на целевой платформе.

8.15. Удаление программ с целевой машины

Команда devtool undeploy-target позволяет удалить развёрнутый на целевой машине вывод сборки, перенесённый туда ранее командой devtool deploy-target. Цель указывается адресом или именем и должна поддерживать сервер SSH (user@hostname[:destdir]).

```
$ devtool undeploy-target recipe target
```

8.16. Создание уровня рабочего пространства в другом месте

Команда devtool create-workspace создает новый уровень workspace в каталоге сборки, в котором создаётся файл README и каталог conf. Приведённая ниже команда создает уровень workspace в текущем рабочем уровне.

```
$ devtool create-workspace
```

Можно создать уровень workspace в другом месте, а также изменить его имя, например devtool create-workspace /home/scottrif/new-workspace, создаст уровень new-workspace в домашнем каталоге пользователя scottrif.

8.17. Получение статуса заданий в рабочем пространстве

Команда devtool status выводит список заданий, размещённых на уровне workspace, включая пути к соответствующим внешним деревьям кода. Приведённая ниже команда показывает задание mtr_0.86.bb и путь к его файлам.

```
$ devtool status
mtr: /home/scottrif/poky/build/workspace/sources/mtr
     (/home/scottrif/poky/build/workspace/recipes/mtr/mtr_0.86.bb)
```

8.18. Поиск доступных заданий для целей

Команда devtool search позволяет найти доступные задания для цели по имени задания или пакета, а также по описанию и установленным файлам.

Глава 9. Справочник по OpenEmbedded Kickstart (.wks)

9.1. Введение

Текущая реализация Wic поддерживает только базовые команды kickstart для разделов - partition (сокращённо part) и bootloader. В новых версиях будут добавлены другие команды и опции. Использование неподдерживаемых команд ведёт к непредсказуемым результатам.

В этой главе описаны доступные команды kickstart, их синтаксис и опции. Команды основаны на версиях Fedora kickstart с изменениями, учитывающими возможности Wic. Полная документация по командам доступна по ссылке.

9.2. Команда part (partition)

Каждая из этих команд создает в системе раздел, используя приведённый ниже синтаксис.

```
part [mntpoint]
partition [mntpoint]
```

Если точка монтирования mntpoint не указана, Wic создает раздел, не монтируя его. Параметр mntpoint должен принимать одну из двух форм:

- /*path* - например, "/", "/usr", "/home";
- *swap* - раздел, используемый для подкачки (swap).

Указание mntpoint приводит к автоматическому монтированию раздела. Wic обеспечивает это путём добавления записей в таблицу файловых систем (fstab) при генерации образа. Для генерации Wic корректного файла fstab нужно указать также опцию раздела --ondrive, --ondisk или --use-uuid partition как часть команды.

Программа mount должна понимать синтаксис PARTUUID, используемый с --use-uuid и не корневыми точками монтирования, включая swap. Версии этой команды в busybox в настоящее время не умеют этого.

Ниже приведен пример с точкой монтирования /. Опция --ondisk служит для указания раздела на диске sdb.

```
part / --source rootfs --ondisk sdb --fstype=ext3 --label platform --align 1024
```

Ниже приведен список поддерживаемых командой part (partition) опций.

- *--size* задаёт минимальный размер раздела в мегабайтах целым числом без суффикса MB. Опция нужна при использовании --source.
- *--fixed-size* задаёт точный размер раздела в мегабайтах и не может использоваться вместе с --size. Если собранный образ больше заданного опцией размера, возникает ошибка.
- *--source* является опцией Wic, которая задаёт источник данных для заполнения раздела. Чаще всего эта опция имеет значение rootfs, но могут применяться и другие значения, отображаемые на пригодный подключаемый модуль (source plug-in, см. раздел Using the Wic Plug-Ins Interface [2]).

  При использовании --source rootfs Wic создает раздел требуемого размера и заполняет его содержимым корневой файловой системы, указанным опцией -r или эквивалентом rootfs, полученным из опции -e. Тип файловой системы для создания раздела выводится из опции --fstype.

  При задании --source plugin-name Wic создает раздел требуемого размера и заполняет его содержимым, которое создаётся заданным подключаемым модулем с использованием данных, указанных опцией -r или эквивалентом rootfs, полученным из опции -e. Содержимое и тип файловой системы зависят от плагина.

  Если опция --source не применяется, команда wic создает пустой раздел, поэтому нужна опция --size для точного задания размера.
- *--ondisk* или *--ondrive* задаёт создание раздела на определённом диске.
- *--fstype* задаёт тип файловой системы для раздела (ext4, ext3, ext2, btrfs, squashfs, swap).
- *--fsoptions* задаёт строку опций в произвольной форме для использования при монтировании файловой системы. Строка копируется в файл /etc/fstab установленной системы и заключается в кавычки. Если опция не задана, используется значение defaults.
- *--label label* задаёт метку файловой системы (label) для раздела. Если такая метка уже применяется другой файловой системой, создаётся новая метка.
- *--active* помечает раздел как активный (загрузка).
- *--align* (кбайт) - специальная опция Wic, указывающая начало раздела на границе, кратной заданному числу.
- *--no-table* - специальная опция Wic, резервирующая пространство для раздела без добавления записи в таблицу разделов.
- *--exclude-path* - специальная опция Wic, исключающая указанный путь из результирующего образа. Работает только с плагином rootfs.

- *--extra-space* - специальная опция Wic, добавляющая пространство (по умолчанию 10 Мбайт) к занятому разделом месту. Окончательный размер раздела превышает заданный параметром --size.
- *--overhead-factor* - специальная опция Wic, умножающая размер раздела на значение опции, которое должно быть больше 1 (по умолчанию 1.3).
- *--part-name* - специальная опция Wic, задающая имя раздела GPT.
- *--part-type* - специальная опция Wic, задающая тип идентификатора GUID для раздела GPT. Список типов GUID можно найти на странице http://en.wikipedia.org/wiki/GUID_Partition_Table#Partition_type_GUIDs.
- *--use-uuid* - специальная опция Wic, заставляющая Wic генерировать случайный идентификатор GUID для раздела, используемый в конфигурации загрузчика для указания корневого раздела.
- *--uuid* - специальная опция Wic, задающая UUID для раздела.
- *--fsuuid* - специальная опция Wic, задающая UUID файловой системы. Можно генерировать или обновлять WKS_FILE с помощью этой опции, если заранее заданный идентификатор UUID добавлен в команду загрузки ядра в конфигурационном файле загрузчика перед запуском Wic.
- *--system-id* - специальная опция Wic, задающая системный идентификатор раздела (1-битовое шестнадцатеричное значение с необязательным префиксом 0x.
- *--mkfs-extraopts* задаёт опции для передачи утилите mkfs, в дополнение к принятым по умолчанию опциям, которые на некоторых файловых системах могут не работать. Доступные опции можно увидеть по команде wic help kickstart.

9.3. Команда bootloader

Эта команда управляет конфигурацией загрузчика с помощью описанных ниже параметров. Функциональность загрузчика и загрузочные разделы реализуются подключаемыми модулями (--source). Команда bootloader по сути обеспечивает способ изменения конфигурации загрузчика.

- *--timeout* задаёт время ожидания (секунды) загрузчика перед выбором заданного по умолчанию варианта.
- *--append* задаёт параметры ядра, добавляемые к команде syslinux APPEND или grub.
- *--configfile* указывает пользовательский файл конфигурации для загрузчика. Если файл не находится в каталоге canned-wks, нужно указать полный путь. Этот файл переопределяет все опции загрузчика.

Глава 10. Ошибки и предупреждения тестов QA

10.1. Введение

При сборке задания система OE выполняет разные проверки QA, выдавая предупреждения и ошибки. Иногда новое задание для сборки программ не создает проблем, однако зачастую они возникают и приходится вносить изменения.

Хотя соблазнительно не обращать внимания на сообщения QA или даже отключить проверки, лучше попытаться решить обнаруженные при проверке проблемы. Эта глава посвящена сообщениям QA и содержит краткие рекомендации по устранению проблем.

В следующем параграфе представлены все сообщения QA в принятой по умолчанию конфигурации. Каждая запись содержит текст сообщения и комментарии.

- В конце каждого сообщения указан связанный с ним тест QA (как указано в параграфе 6.56. insane.bbclass).
- Список ошибок и предупреждений относится лишь к тестам QA и не включает другие возможные ошибки.
- Поскольку некоторые тесты QA по умолчанию отключены, они не учтены здесь.

10.2. Ошибки и предупреждения

<packagename>: <path> is using libexec please relocate to <libexecdir> [libexec]

Указанный пакет содержит файлы в /usr/libexec, тогда как конфигурация дистрибутива указывает другой путь к <libexecdir> (по умолчанию $prefix/libexec, но может быть измене, например, ${libdir}).

package <packagename> contains bad RPATH <rpath> in file <file> [rpaths]

Указанный двоичный файл, созданный заданием, содержит пути загрузки динамических библиотек (rpaths) с путями сборки (такими как TMPDIR), которые некорректны для цели и могут создавать проблемы безопасности. Следует проверить опции -rpath, передаваемые компоновщику, в журнале задачи do_compile. В зависимости от используемой системы сборки, могут быть опции для полного запрета использования rpath при сборке задания.

<packagename>: <file> contains probably-redundant RPATH <rpath> [useless-rpaths]

Указанный двоичный файл, созданный заданием, содержит пути загрузки динамических библиотек (rpaths), которые на стандартных системах просматриваются компоновщиком по умолчанию (например, /lib и /usr/lib). Такие пути не вызывают неполадок, но занимают место, будучи ненужными. В зависимости от используемой системы сборки, могут быть опции для полного запрета использования rpath при сборке задания.

<packagename> requires <files>, but no providers in its RDEPENDS [file-rdeps]

В указанных файлах пакета <packagename> были обнаружены зависимости на уровне файлов, не соответствующие явно записям в RDEPENDS. Если файлы нужны при работе, RDEPENDS в задании следует указывать пакеты, предоставляющие эти файлы.

<packagename1> rdepends on <packagename2>, but it isn't a build dependency? [build-deps]

Между указанными пакетами имеется зависимость при работе, но в задании ничто явно не указывает системе сборки OE необходимость выполнения этой зависимости. Это обычно связано с добавлением значения

RDEPENDS на этапе подготовки пакета вместо того, чтобы автоматически сделать это раньше на основе содержимого пакета. В большинстве случаев в задание следует явно добавить RDEPENDS для зависимости.

non -dev/-dbg/nativesdk- package contains symlink .so: <packagename> path '<path>' [dev-so]

Символьные ссылки на файлы .so, предназначенные лишь для разработки, которые нужно перенести в пакет -dev. Это может возникать при добавлении *.so* вместо *.so.* в пакеты, не относящиеся к разработке. Следует изменить FILES (возможно и PACKAGES), чтобы указанные файлы .so перешли в пакет -dev.

non -staticdev package contains static .a library: <packagename> path '<path>' [staticdev]

Файлам статической библиотеки .a следует быть в пакете -staticdev. Следует изменить FILES (возможно и PACKAGES), чтобы указанные файлы .a перешли в пакет -staticdev.

<packagename>: found library in wrong location [libdir]

Указанный файл может быть установлен в некорректно (возможно заданное жёстко) место. Например, проверка будет указывать задания, которые устанавливают файлы /lib/bar.so, когда ${base_libdir} = "lib32". Другим случаем является установка заданием /usr/lib64/foo.so при ${libdir} = "/usr/lib". Возможны ложные срабатывания и в таких случаях следует добавить "libdir" в INSANE_SKIP для пакета.

non debug package contains .debug directory: <packagename> path <path> [debug-files]

Указанный пакет содержит каталог .debug, которому не следует появляться вне пакетов -dbg. Это может быть связано с добавлением пути, содержащего каталог .debug, и неявным добавлением .debug в пакет -dbg. В таких случаях следует явно добавить .debug в переменную FILES_${PN}-dbg.

Architecture did not match (<machine_arch> to <file_arch>) on <file> [arch]

По умолчанию система сборки OE проверяет тип ELF, размер слова и порядок битов во всех двоичных файлах на предмет соответствия целевой архитектуре. Отрицательный результат проверки может говорить о выборе неверного компилятора или опций. Иногда для программ (например, загрузчиков) требуется обход этого теста. Если файл, для которого указываются ошибки, относится к микрокоду, не предназначенному для выполнения в целевой операционной системе, следует добавить arch в переменную INSANE_SKIP для пакета. Помощь в обнаружении источника проблем может оказать просмотр журнала задачи do_compile в части неверных опций.

Bit size did not match (<machine_bits> to <file_bits>) <recipe> on <file> [arch]

По умолчанию система сборки OE проверяет тип ELF, размер слова и порядок битов во всех двоичных файлах на предмет соответствия целевой архитектуре. Отрицательный результат проверки может говорить о выборе неверного компилятора или опций. Иногда для программ (например, загрузчиков) требуется обход этого теста. Если файл, для которого указываются ошибки, относится к микрокоду, не предназначенному для выполнения в целевой операционной системе, следует добавить arch в переменную INSANE_SKIP для пакета. Помощь в обнаружении источника проблем может оказать просмотр журнала задачи do_compile в части неверных опций.

Endianness did not match (<machine_endianness> to <file_endianness>) on <file> [arch]

По умолчанию система сборки OE проверяет тип ELF, размер слова и порядок битов во всех двоичных файлах на предмет соответствия целевой архитектуре. Отрицательный результат проверки может говорить о выборе неверного компилятора или опций. Иногда для программ (например, загрузчиков) требуется обход этого теста. Если файл, для которого указываются ошибки, относится к микрокоду, не предназначенному для выполнения в целевой операционной системе, следует добавить arch в переменную INSANE_SKIP для пакета. Помощь в обнаружении источника проблем может оказать просмотр журнала задачи do_compile в части неверных опций.

ELF binary '<file>' has relocations in .text [textrel]

Указанный двоичный файл ELF содержит перемещения (relocation) в своих разделах .text. Это влияет на производительность работы. Обычно проблему можно решить добавлением опции компилятора -fPIC или -fpic. Например, для программы, читающей при сборке переменную CFLAGS можно добавить в задание строку CFLAGS_append = " -fPIC ". Дополнительная информация о перемещениях в процессе работы доступна по ссылке.

No GNU_HASH in the elf binary: '<file>' [ldflags]

Двоичный файл, созданный при сборке задания не был скомпонован с опциями LDFLAGS, предоставленными системой сборки. Следует проверить передачу переменной LDFLAGS команде компоновщика. Для обхода проблемы в этой ситуации обычно применяется передача LDFLAGS с помощью TARGET_CC_ARCH внутри задания в форме TARGET_CC_ARCH += "${LDFLAGS}".

Package <packagename> contains Xorg driver (<driver>) but no xorg-abi- dependencies [xorg-driver-abi]

Указанный пакет содержит драйвер Xorg, но не имеет соответствующей зависимости ABI. Имена драйверов ABI предоставляет пакет xserver-xorg и все драйверы должны зависеть от версии ABI, с которой они были собраны. Задания для драйверов, включающие xorg-driver-input.inc или xorg-driver-video.inc получают эту версию автоматически, поэтому следует лишь явно добавить зависимости в задания для двоичных драйверов.

The /usr/share/info/dir file is not meant to be shipped in a particular package. [infodir]

Каталог /usr/share/info/dir не следует включать в пакеты. Для этого следует добавить в задачу do_install или do_install_append строку rm ${D}${infodir}/dir.

Symlink <path> in <packagename> points to TMPDIR [symlink-to-sysroot]

Символьная ссылка указывает на каталог TMPDIR хоста сборки и будет некорректной на целевой платформе. Следует исправить ссылку, указав в ней относительный путь, либо удалить эту ссылку.

<file> failed sanity test (workdir) in path <path> [la]

Указанный файл .la содержит пути TMPDIR. Это некорректно, поскольку libtool добавляет префикс sysroot при использовании файлов.

<file> failed sanity test (tmpdir) in path <path> [pkgconfig]

Указанный файл .pc содержит пути TMPDIR/WORKDIR. Это некорректно, поскольку pkg-config добавляет префикс sysroot при доступе к файлам.

<packagename> rdepends on <debug_packagename> [debug-deps]

Имеется зависимость между указанным пакетом, не относящимся к отладке (имя пакета не имеет суффикса -dbg), и пакетом dbg. Пакеты dbg содержат отладочные символы и могут создаваться несколькими способами:

- включение dbg-pkgs в IMAGE_FEATURES;
- использование IMAGE_INSTALL;
- как зависимость другого пакета dbg, созданного одним из предыдущих способов.

Зависимости могут добавляться автоматически в результате ошибочного включения ненужных фалов в пакет dbg (например, файл .so вместо символьной ссылки) или вручную (например, через переменную RDEPENDS).

<packagename> rdepends on <dev_packagename> [dev-deps]

Имеется зависимость между указанным пакетом, не относящимся к разработке (имя пакета не имеет суффикса -dev), и пакетом для разработки. Пакеты dev включают файлы заголовков и могут создаваться разными способами:

- включение dev-pkgs в IMAGE_FEATURES;
- использование IMAGE_INSTALL;
- как зависимость другого пакета dev, созданного одним из предыдущих способов.

Зависимости могут добавляться автоматически в результате ошибочного включения ненужных фалов в пакет dev (например, файл .so вместо символьной ссылки) или вручную (например, через переменную RDEPENDS).

<var>_<packagename> is invalid: <comparison> (<value>) only comparisons <, =, >, <=, and >= are allowed [dep-cmp]

При добавлении зависимости с учётом версии в одну из переменных RDEPENDS, RRECOMMENDS, RSUGGESTS, RPROVIDES, RREPLACES, RCONFLICTS должны применяться лишь указанные операторы сравнения. Следует изменить значения зависимостей с учётом версий в соответствии с сообщением.

<recipename>: The compile log indicates that host include and/or library paths were used. Please check the log '<logfile>' for more information. [compile-host-path]

Журнал задачи do_compile показывает, что файлы отыскивались по путям на хосте - это неприемлемо для кросс-компиляции. Следует найти эти имена по строке «is unsafe for cross-compilation» или «CROSS COMPILE Badness».

<recipename>: The install log indicates that host include and/or library paths were used. Please check the log '<logfile>' for more information. [install-host-path]

Журнал задачи do_install показывает, что файлы отыскивались по путям на хосте - это неприемлемо для кросс-компиляции. Следует найти эти имена по строке «is unsafe for cross-compilation» или «CROSS COMPILE Badness».

This autoconf log indicates errors, it looked at host include and/or library paths while determining system capabilities. Rerun configure task after fixing this. The path was '<path>'

Журнал задачи do_ configure показывает, что файлы отыскивались по путям на хосте - это неприемлемо для кросс-компиляции. Следует найти эти имена по строке «is unsafe for cross-compilation» или «CROSS COMPILE Badness».

<packagename> doesn't match the [a-z0-9.+-]+ regex [pkgname]

Внутреннее соглашение системы сборки OE (иногда вынуждаемое менеджером пакетов) требует указывать имена пакетов в нижнем регистре. Если задание не следует этому или добавляются пакеты, не соответствующие соглашению, в переменную PACKAGES, выдаётся данное сообщение. В таких случаях следует переименовать задание или имена пакетов, включённых в PACKAGES.

<recipe>: configure was passed unrecognized options: <options> [unknown-configure-option]

Сценарий configure сообщает о непонятной опции, что может быть следствием изменения набора поддерживаемых параметров или возникновения ошибки в имени опции. Следует внимательно посмотреть вывод команды ./configure --help и журнал изменений в репозитории (change log), а затем должным образом изменить переменные EXTRA_OECONF, PACKAGECONFIG_CONFARGS или отдельные опции в PACKAGECONFIG.

Recipe <recipefile> has PN of "<recipename>" which is in OVERRIDES, this can result in unexpected behavior. [pn-overrides]

Имя указанного задания (PN) присутствует в OVERRIDES. Если задание названо так, что его имя соответствует чему-либо включённому в OVERRIDES (например, PN совпадает с MACHINE или DISTRO), могут возникать неожиданные последствия. Например, назначение вида FILES_${PN} = "xyz" будет приводить к FILES = "xyz". Следует переименовать задание (или PN, если переменная задана явно) для устранения конфликта.

<recipefile>: Variable <variable> is set as not being package specific, please fix this. [pkgvarcheck]

Некоторые переменные (RDEPENDS, RRECOMMENDS, RSUGGESTS, RCONFLICTS, RPROVIDES, RREPLACES, FILES, pkg_preinst, pkg_postinst, pkg_prerm, pkg_postrm, ALLOW_EMPTY) всегда следует устанавливать для конкретного пакета (т. е. использовать переопределением вида RDEPENDS_${PN} = "value", а не RDEPENDS = "value"). Следует исправить назначение переменных в задании.

File '<file>' from <recipename> was already stripped, this will prevent future debugging! [already-stripped]

Из созданных двоичных файлов уже вырезаны отладочные символы перед тем, как система сборки попыталась их извлечь. В разрабатываемых проектах часто вырезают отладочные символы по умолчанию. Для отладки пакетов на целевой платформе с использованием пакетов -dbg удаление символов должно быть отключено. В некоторых системах сборки это можно отключить указанием в конфигурации дополнительной опции. В иных случаях может потребоваться исправление сценариев сборки. Для этого следует насти в команде компоновки строки strip или STRIP и опции -s или -S (возможна их передача из командной строки компилятора через опцию -Wl). Запрет вырезания символов не означает их наличия в двоичных пакетах, поскольку система сборки OE выделяет эти символы в пакеты -dbg, вырезая из двоичных файлов.

<packagename> is listed in PACKAGES multiple times, this leads to packaging errors. [packages-list]

Имя пакета должно указываться в переменной PACKAGES лишь 1 раз. Следует проверить переменную.

FILES variable for package <packagename> contains '//' which is invalid. Attempting to fix this but you should correct the metadata. [files-invalid]

Строка // некорректна в пути Unix. Следует найти её в переменной FILES и заменить на /.

<recipename>: Files/directories were installed but not shipped in any package [installed-vs-shipped]

Файлы установлены задачей do_install, но не включены в переменную FILES какого-либо задания. Такие файлы не могут присутствовать в образе на последующих этапах сборки, поэтому нужно добавить их в переменную FILES соответствующего пакета (например, FILES_${PN} для основного пакета) или удалить файлы в конце задачи do_install, если они не нужны.

<oldpackage>-<oldpkgversion> was registered as shlib provider for <library>, changing it to <newpackage>-<newpkgversion> because it was built later

Указанную общую библиотеку предоставляют сразу <oldpackage> и <newpackage>. Это может быть следствием смены имени задания. Однако в иных случаях сообщение может означать, что ошибочно провайдером указана «приватная» версия библиотеки вместо общей. Тогда следует добавить файл библиотеки .so в переменную PRIVATE_LIBS задания, обеспечивающего приватную версию библиотеки.

10.3. Настройка и отключение проверок QA

Можно настроить проверки QA глобально, чтобы отказ конкретной проверки вызывал ошибку или предупреждение, с помощью переменных WARN_QA и ERROR_QA. Можно также отключить проверки в конкретном задании, используя INSANE_SKIP. Информация о проверках QA приведена в параграфе 6.56. insane.bbclass.

Следует помнить, что проверки QA предназначены для обнаружения имеющихся и возможных проблем в выводе пакетов, поэтому следует отключать эти проверки с осторожностью.

Глава 11. Образы

Система сборки OE включает несколько образов для разных случаев, имена которых можно указать в команде bitbake для сборки.

Сборка образов без компонент с лицензиями GPLv3, LGPLv3 или AGPL-3.0 поддерживается только для минимальных и базовых вариантов. Кроме того, для образов с компонентами, использующими лицензию, отличную от GPLv3 и подобных ей, нужно внести соответствующие изменения в файл local.conf до ввода команды BitBake для сборки.

1. Поместить символ комментария в начало строки EXTRA_IMAGE_FEATURES.
2. Установить INCOMPATIBLE_LICENSE = "GPL-3.0 LGPL-3.0 AGPL-3.0".

Посмотреть образы в в локальном репозитории poky Git можно с помощью команды ls meta*/recipes*/images/*.bb. Список поддерживаемых заданий приведен ниже.

build-appliance-image
Пример виртуальной машины, содержащей все компоненты, требуемые для запуска сборки, использующей систему сборки, а также самой системы сборки. Можно собрать и запустить образ с помощью VMware Player или VMware Workstation. Дополнительная информация приведена на странице Build Appliance сайта YP.

core-image-base
Образ с консольным интерфейсом и полной поддержкой оборудования целевой платформы.

core-image-clutter
Образ с поддержкой инструментов Clutter на основе Open GL, обеспечивающих разработку графических интерфейсов с анимацией.

core-image-full-cmdline
Образ с консольным интерфейсом и расширенной функциональностью Linux.

core-image-lsb
Образ, соответствующий спецификации LSB, требующий конфигурации дистрибутива в соответствии с LSB (например, poky-lsb). При сборке core-image-lsb без такой конфигурации образ не будет соответствовать LSB.

core-image-lsb-dev
Образ core-image-lsb с поддержкой разработки на хосте, включающий файлы заголовков и библиотеки, нужные для разработки. Образ требует конфигурации дистрибутива в соответствии с LSB (например, poky-lsb). При сборке core-image-lsb-dev без такой конфигурации образ не будет соответствовать LSB.

core-image-lsb-sdk
Образ core-image-lsb с поддержкой кросс-разработки, включающий файлы заголовков и библиотеки, нужные для формирования автономного SDK. Образ требует конфигурации дистрибутива в соответствии с LSB (например, poky-lsb). При сборке core-image-lsb-sdk без такой конфигурации образ не будет соответствовать LSB. Образ пригоден для разработки на целевой платформе.

core-image-minimal
Компактный образ, пригодный для загрузки на устройстве.

core-image-minimal-dev
Образ core-image-minimal с поддержкой разработки на хосте, включающий файлы заголовков и библиотеки, нужные для разработки.

core-image-minimal-initramfs
Образ core-image-minimal с файловой системой initramfs как частью ядра, которая позволяет более эффективно находить первую программу init (см. описание переменной PACKAGE_INSTALL).

core-image-minimal-mtdutils
Образ core-image-minimal с поддержкой утилит Minimal MTD, позволяющий пользователю взаимодействовать с подсистемой MTD в ядре для операций с flash-устройствами.

core-image-rt
Образ core-image-minimal с набором тестов и инструментов для работы в реальном масштабе времени.

core-image-rt-sdk
Образ core-image-rt с кросс-инструментами, заголовочными файлами и библиотеками для создания автономного SDK, пригодного для разработки на целевой платформе.

core-image-sato
Образ с поддержкой среды Sato для мобильных устройств. Образ поддерживает X11 с темой Sato, термина, редактор, менеджер файлов, проигрыватель и т. п.

core-image-sato-dev
Образ core-image-sato для разработки на хосте. Включает библиотеки для сборки приложений на устройстве, инструменты тестирования и профилирования, отладочные символы (прежнее название core-image-sdk).

core-image-sato-sdk
Образ core-image-sato с включением кросс-инструментов. Включает файлы заголовков и библиотеки для формирования автономного SDK, поддерживающего разработку на целевой платформе.

core-image-testmaster
базовый образ для автоматизированного тестирования, разворачиваемый на своём разделе, что позволяет развернуть тестируемый образ отдельно. Подробно описан в разделе Performing Automated Runtime Testing [2].

core-image-testmaster-initramfs
Образ initramfs, адаптированный для использования с core-image-testmaster.

core-image-weston
Простой образ Wayland с терминалом, включающий терминальные библиотеки и эталонный сборщик Weston (см. раздел Using Wayland and Weston [2]).

core-image-x11
Базовый образ X11 с терминалом.

Глава 12. Свойства

В этой главе описаны свойства машин и дистрибутивов, которые можно включить в образ, и описаны свойства образов. Свойства позволяют определить пакеты, включаемые в создаваемый образ. Дистрибутив позволяет выбрать включаемые свойства через переменную DISTRO_FEATURES, которая задаётся или дополняется в конфигурационном файле дистрибутива (poky.conf, poky-tiny.conf, poky-lsb.conf и т. п.). Свойства машины задаются в переменной

MACHINE_FEATURES, устанавливаемой в файле конфигурации машины и задающей аппаратные возможности. Эти две переменные определяют модули ядра, утилиты и другие пакеты для включения в образ. Дистрибутив позволяет задать набор свойств машины, которые не будут включаться, если дистрибутив их не поддерживает.

Одним из методов определения заданий, которые проверяются на предмет определённого свойства, является просмотр метаданных для свойства. Ниже приведен пример поиска заданий, сборка которых может быть изменена с учётом указанного свойства.

```
$ cd poky
$ git grep 'contains.*MACHINE_FEATURES.*feature'
```

12.1. Свойства машины

Ниже перечислены элементы, которые можно указать в MACHINE_FEATURES. Свойства не соответствуют однозначно пакетам и могут выходить за рамки простого управления пакетами. Иногда свойство может влиять на сборку нескольких заданий. Например, свойство может указывать наличие некого параметра задачи do_configure в задании. В списке свойств представлены только свойства из метаданных YP.

- *acpi* - оборудование включает ACPI (x86/x86_64).
- *alsa* - оборудование имеет драйверы ALSA.
- *apm* - оборудования использует APM (или эмуляцию APM).
- *bluetooth* - оборудование имеет встроенные элементы BT.
- *efi* - поддержка загрузки через EFI.
- *ext2* - устройство HDD или Microdrive.
- *irda* - оборудование поддерживает IrDA.
- *keyboard* - устройство имеет клавиатуру.
- *pcbios* - поддержка загрузки через BIOS.
- *pci* - оборудование имеет шину PCI.
- *pcmcia* - оборудование имеет гнезда PCMCIA или CompactFlash.
- *phone* - поддержка мобильных телефонов (голос).
- *qvga* - машина имеет дисплей QVGA (320x240).
- *rtc* - машина имеет часы RTC.
- *screen* - устройство имеет экран.
- *serial* - устройство имеет последовательный порт (обычно RS232).
- *touchscreen* - устройство имеет сенсорный экран.
- *usbgadget* - оборудование поддерживает устройства USB (гаджеты).
- *usbhost* - оборудование поддерживает хост USB.
- *vfat* - поддержка файловой системы FAT.
- *wifi* - оборудование имеет встроенные элементы WiFi.

12.2. Свойства дистрибутива

Ниже представлен список свойств, которые можно установить для дистрибутива в переменной DISTRO_FEATURES. Свойства не соответствуют однозначно пакетам и могут выходить за рамки простого управления пакетами. Иногда свойство может влиять на сборку нескольких заданий. Например, свойство может указывать наличие некого параметра задачи do_configure в задании.

Некоторые свойства дистрибутива являются и свойствами машины. Управлять такими свойствами можно на обоих уровнях (см. описание переменной COMBINED_FEATURES). В списке представлены лишь свойства из метаданных YP.

- *alsa* - включает поддержку ALSA (при доступности устанавливаются модули ядра для совместимости с OSS).
- *api-documentation* - включает создание документации API при сборке. Документация добавляется в архив SDK при использовании команды bitbake -c populate_sdk (раздел Adding API Documentation to the Standard SDK [7]).
- *bluetooth* - включает поддержку bluetooth (только встроенные BT).
- *bluez5* - включает BlueZ версии 5 с поддержкой базовых уровней и протоколов Bluetooth. По умолчанию значение DISTRO_FEATURES включает bluetooth, что вызывает включение bluez5. Если поддержка bluez5 не нужна и достаточно bluez4, нужно задать DISTRO_FEATURES_BACKFILL_CONSIDERED = "bluez5". Установка этой переменной говорит системе сборки OE, что использование bluez5 исключается в пользу bluez4.
- *cramfs* - включает поддержку CramFS.
- *directfb* - включает поддержку DirectFB.
- *ext2* - включает инструменты для поддержки устройств с HDD/Microdrive для хранения файлов.
- *ipsec* - включает поддержку IPSec.
- *ipv6* - включает поддержку IPv6.
- *irda* - включает поддержку IrDA.

- *keyboard* - включает поддержку клавиатуры (например, будет загружаться keymap).
- *ldconfig* - включает поддержку ldconfig и ld.so.conf на целевой платформе.
- *nfs* - включает поддержку клиентов NFS (для монтирования на устройстве NFS export).
- *opengl* - включает библиотеку Open GL, обеспечивающую кроссплатформенный и многоязычный интерфейс API для плоской и трёхмерной графики.
- *pci* - включает поддержку шины PCI.
- *pcmcia* - включает поддержку PCMCIA/CompactFlash.
- *ppp* - включает поддержку PPP dialup.
- *ptest* - включает сборку тестов пакетов для отдельных заданий (см. Testing Packages With ptest [2]).
- *smbfs* - включает поддержку клиентов SMB (для монтирования каталогов Samba/Microsoft Windows).
- *systemd* - включает поддержку менеджера инициализации systemd, который является полной заменой init и обеспечивает параллельный запуск служб, снижение нагрузки на оболочку и пр.
- *usbgadget* - включает поддержку USB Gadget (для разных устройств USB).
- *usbhost* - включает поддержку USB Host (подключение внешней клавиатуры, мыши, дисков и т. п.).
- *wayland* — включает серверный протокол Wayland и библиотеку для его поддержки.
- *wifi* - включает поддержку встроенных устройств WiFi.
- *x11* - включает X-сервер и библиотеки поддержки.

12.3. Свойства образа

Содержимым образов, создаваемых системой сборки OE, можно управлять с помощью переменных IMAGE_FEATURES и EXTRA_IMAGE_FEATURES, которые обычно указываются в задании для образа. Эти переменные помогают выбрать возможности из предопределённых вариантов, такие как отладка или средства разработки.

Ниже перечислены свойства (возможности) доступные для всех образов.

- *allow-empty-password* позволяет входить в Dropbear и OpenSSH с именем root и другими именами без пароля.
- *dbg-pkgs* устанавливает отладочные символы для всех пакетов, включённых в образ.
- *debug-tweaks* делает образ пригодным для отладки (см. свойства allow-empty-password, empty-root-password, post-install-logging).
- *dev-pkgs* устанавливает компоненты разработки (заголовочные файлы и дополнительные библиотеки) для всех пакетов в образе.
- *doc-pkgs* устанавливает пакеты документации для всех пакетов в образе.
- *empty-root-password* устанавливает пустой пароль для пользователя root.
- *package-management* устанавливает средства управления пакетами и сохраняет базу данных о пакетах.
- *post-install-logging* включает запись событий пост-установочных сценариев в файл /var/log/postinstall.log при первой загрузке образа на целевой системе. Переменная VOLATILE_LOG_DIR = no делает каталог /var/log в образе постоянным.
- *ptest-pkgs* устанавливает пакеты ptest для всех поддерживающих ptest пакетов.
- *read-only-rootfs* создает образ с корневой файловой системой, доступной лишь для чтения (см. раздел Creating a Read-Only Root Filesystem [2]).
- *splash* включает вывод заставки в процессе загрузки системы. По умолчанию заставка обеспечивается пакетом psplash, который не позволяет менять свою конфигурацию. Можно использовать другую заставку, указав имя пакета (пакетов) в переменной SPLASH задания для образа или на уровне конфигурации дистрибутива.
- *staticdev-pkgs* устанавливает статические библиотеки разработки (файлы *.a) для всех пакетов в образе.

Некоторые образы, доступные лишь при наследовании класса core-image, перечислены ниже.

- *hwcodecs* устанавливает кодеки аппаратного ускорения.
- *nfs-server* устанавливает сервер NFS.
- *perf* устанавливает инструменты профилирования, такие как perf, systemtap и LTTng. Информация о работе с этими инструментами приведена в [7].
- *ssh-server-dropbear* устанавливает Dropbear (минимальный сервер SSH).
- *ssh-server-openssh* устанавливает сервер OpenSSH SSH с более развитыми функциями, нежели Dropbear. При наличии обоих серверов OpenSSH и Dropbear в переменной IMAGE_FEATURES предпочтение будет отдано OpenSSH и Dropbear не будет устанавливаться.
- *tools-debug* устанавливает средства отладки, такие как strace и gdb. Информация о работе с GDB приведена в разделе Debugging With the GNU Project Debugger (GDB) Remotely [2], а трассировка описана в [9].
- *tools-sdk* устанавливает полный пакет SDK для работы на устройстве.
- *tools-testapps* устанавливает инструменты тестирования устройства (например, отладки серверного экрана).

- *x11* устанавливает X-сервер.
- *x11-base* устанавливает X-сервер с минимальной оболочкой.
- *x11-sato* устанавливает X-сервер с оболочкой OpenedHand Sato.

12.4. Отключение автоматически добавляемых свойств

Иногда системе сборки OE требуются значения MACHINE_FEATURES или DISTRO_FEATURES для контроля добавленной функциональности, которую невозможно отключить. В таких случаях нужно добавлять запись дополнительного свойства в одну из указанных переменных, но это заставит разработчиков, у которых уже заданы эти переменные, добавлять новые свойства, чтобы сохранить общий уровень функциональности. Поэтому система сборки OE включает механизм автоматического отключения (backfill) добавленных свойств в имеющиеся конфигурации дистрибутива или машины. Список свойств, для которых это выполняется, можно увидеть в переменных DISTRO_FEATURES_BACKFILL и MACHINE_FEATURES_BACKFILL файла meta/conf/bitbake.conf file.

Поскольку свойства по умолчанию включаются во все конфигурации, разработчики, желающие отключить ненужные свойства, должны указать их в переменной DISTRO_FEATURES_BACKFILL_CONSIDERED или MACHINE_FEATURES_BACKFILL_CONSIDERED.

Ниже приведены два примера.

- *Свойство дистрибутива pulseaudio.* Ранее поддержка PulseAudio включалась в Qt и GStreamer. Поэтому свойство добавляется автоматически через переменную DISTRO_FEATURES_BACKFILL и автоматически включается для всех дистрибутивов в файле meta/conf/bitbake.conf. Для отключения этого свойства без влияния на конфигурации других дистрибутивов, которым нужна поддержка PulseAudio, добавляется значение pulseaudio в переменную DISTRO_FEATURES_BACKFILL_CONSIDERED файла конфигурации дистрибутива. Добавление в эту переменную свойства, указанного также в DISTRO_FEATURES_BACKFILL, предотвратит добавление системой сборки этого свойства в DISTRO_FEATURES, т. е. к отключению в данном дистрибутиве.
- *Свойство машины rtc.* Ранее поддержка часов RTC[1] была включена для всех целевых устройств, поэтому свойство автоматически добавлялось для всех машин через переменную MACHINE_FEATURES_BACKFILL в файле meta/conf/bitbake.conf file. Однако такие часы поддерживают не все устройства. Для таких устройств можно отключить поддержку этих часов с помощью переменной MACHINE_FEATURES_BACKFILL_CONSIDERED в файле конфигурации машины .conf. Добавление свойства в переменную MACHINE_FEATURES_BACKFILL предотвратит включение свойства в переменную MACHINE_FEATURES, т. е. поддержку RTC в данной машине.

Глава 13. Переменные

В этой главе перечислены и описаны основные переменные, используемые системой сборки OE.

A

ABIEXTENSION

Расширение поля ABI[2] канонического имени GNU для архитектуры (например, "eabi"). Расширения ABI задаются во включаемых файлах для машин. Например, файл meta/conf/machine/include/arm/arch-arm.inc задаёт расширение ABIEXTENSION = "eabi".

ALLOW_EMPTY

Задаёт необходимость создания на выходе пустых пакетов, которые BitBake по умолчанию не создает. Принятое по умолчанию поведение может создавать проблемы в тех случаях, когда RDEPENDS или другое жёсткое требование во время работы указывает необходимость пакета.

Как и в других переменных управления пакетами требуется суффикс в виде имени пакета, как показано ниже.

```
ALLOW_EMPTY_${PN} = "1"
ALLOW_EMPTY_${PN}-dev = "1"
ALLOW_EMPTY_${PN}-staticdev = "1"
```

ALTERNATIVE

Перечисляет команды в пакете, которым нужна схема вариативного именования. Иногда одна команда предоставляется несколькими пакетами и в таких случаях системе сборки OE приходиться использовать систему вариантов для создания разных имён двоичных файлов, чтобы команды могли существовать совместно.

При использовании этой переменной указывается список команд пакета, которые применяются также в другом пакете, как показано ниже. Например, если пакет busybox имеет 4 таких команды, можно указать ALTERNATIVE_busybox = "sh sed test bracket". Дополнительная информация о вариантах именования приведена в параграфе 6.141. update-alternatives.bbclass.

ALTERNATIVE_LINK_NAME

Применяется системой вариантов (alternative) для отображения дублированных команд на реальные файлы. Например, если команда bracket, поддерживаемая пакетом busybox, дублируется в другом пакете, нужно использовать переменную ALTERNATIVE_LINK_NAME для указания реального местоположения файла.

```
ALTERNATIVE_LINK_NAME[bracket] = "/usr/bin/["
```

В этом примере двоичный файл команды bracket ([) из busybox размещается в /usr/bin/. Если переменная ALTERNATIVE_LINK_NAME не задана, используется ${bindir}/name. (см. 6.141. update-alternatives.bbclass).

ALTERNATIVE_PRIORITY

Применяется системой вариантов для установки приоритета дублированных команд. Переменная позволяет создать единственный принятый по умолчанию вариант, независимо от команды или пакета, а также принятые по умолчанию варианты для конкретных пакетов (6.141. update-alternatives.bbclass). Варианты синтаксиса даны ниже.

```
ALTERNATIVE_PRIORITY = "priority"
ALTERNATIVE_PRIORITY[name] = "priority"
ALTERNATIVE_PRIORITY_pkg[name] = "priority"
```

[1] Real time clock - часы в реальном масштабе времени.
[2] Application Binary Interface - интерфейс двоичных приложений.

ALTERNATIVE_TARGET

Применяется системой вариантов для создания используемого по умолчанию местоположения для дубликатов команд. Переменная позволяет создать единственный принятый по умолчанию вариант, независимо от команды или пакета, а также принятые по умолчанию варианты для конкретных пакетов (6.141. update-alternatives.bbclass). Варианты синтаксиса даны ниже.

```
ALTERNATIVE_TARGET = "target"
ALTERNATIVE_TARGET[name] = "target"
ALTERNATIVE_TARGET_pkg[name] = "target"
```

Если переменная ALTERNATIVE_TARGET не определена, наследуется значение из ALTERNATIVE_LINK_NAME.

При совпадении ALTERNATIVE_LINK_NAME и ALTERNATIVE_TARGET к цели для ALTERNATIVE_TARGET добавляется ".{BPN}". Если указанный файл не переименован, система вариантов переименует его, чтобы избежать необходимости переименования вариантов в задаче do_install, сохраняя при необходимости поддержку команды.

APPEND

Переопределяет список добавленных строк для каждой цели, указанной с LABELS (использование переменной рассмотрено в описании класса grub-efi).

AR

Сокращённая команда и аргументы для работы ar.

ARCHIVER_MODE

При использовании с классом archiver определяет тип информации, использованной для создания архива. Эту переменную можно использовать для создания архивов исправленных (patched) и оригинальных исходных кодов, настроенных кодов и т. п. С помощью описанных ниже флагов.

ARCHIVER_MODE[src] = "original"
Используются оригинальные (неупакованные) исходные файлы.

ARCHIVER_MODE[src] = "patched"
Используются исправленные (patched) исходные файлы (принято по умолчанию).

ARCHIVER_MODE[src] = "configured"
Используются настроенные исходные файлы.

ARCHIVER_MODE[diff] = "1"
Используются различия (patch) между do_unpack и do_patch.

ARCHIVER_MODE[diff-exclude] ?= "file file ..."
Список файлов и каталогов для исключения из diff.

ARCHIVER_MODE[dumpdata] = "1"
Используются данные окружения.

ARCHIVER_MODE[recipe] = "1"
Используются файлы заданий и включаемые файлы.

ARCHIVER_MODE[srpm] = "1"
Используются файлы RPM.

Информация об использовании переменной приведена в файле meta/classes/archiver.bbclass дерева исходных кодов.

AS

Сокращённая команда и аргументы для работы ассемблера.

ASSUME_PROVIDED

Список имён заданий (PN values), которые BitBake не будет пытаться собрать, предполагая, что они уже собраны. В OpenEmbedded-Core переменная ASSUME_PROVIDED задаёт естественные инструменты, которые не нужно собирать. Примером служит задание git-native, его указание позволяет применять Git хоста без сборки git-native.

ASSUME_SHLIBS

Предоставляет дополнительные данные о провайдере общих библиотек, которые дополняют или переопределяют информацию, автоматически предоставляемую системой. Элементы разделяются пробелами.

Например, приведённая ниже форма добавляет провайдера shlib с именем shlibname в packagename с необязательным указанием версии.

```
shlibname:packagename[_version]
```

Следующий пример добавляет общую библиотеку libEGL.so.1 как предоставляемую пакетом libegl-implementation.

```
ASSUME_SHLIBS = "libEGL.so.1:libegl-implementation"
```

AUTHOR

Почтовый адрес для контактов с исходным автором или авторами для передачи правок и сообщений об ошибках.

AUTO_LIBNAME_PKGS

При наследовании класса debian (принято по умолчанию) переменная задаёт пакеты, которые следует проверять на предмет библиотек и переименовывать в соответствии с правилами Debian. По умолчанию принято значение ${PACKAGES}, которое включает класс debian для всех пакетов, явно создаваемых заданием.

AUTO_SYSLINUXMENU

Включает создание автоматического меню для загрузчика syslinux. Переменная должна устанавливаться в задании, класс syslinux проверяет её.

AUTOREV

Когда для переменной SRCREV установлено значение данной переменной, это задаёт использование последнего выпуска исходных кодов из репозитория. Если используется SRCREV = "${AUTOREV}" для поиска последней версии программы, нужно убедиться, что PV содержит ${SRCPV}. Предположим, что имеется задание для ядра, наследующее класс kernel и применяется приведённый выше оператор. В этом случае ${SRCPV} не попадёт автоматически в PV и нужно изменить PV в задании для включения ${SRCPV}.

Дополнительные сведения приведены в разделе Automatically Incrementing a Binary Package Revision Number [2].

AVAILTUNES

Список определённых настроек CPU и ABI, доступных для системы сборки OE. С конкретной конфигурацией машины могут быть совместимы не все настройки и могут возникать несовместимости с разными библиотеками в конфигурации Multilib. Настройки добавляются с использованием оператора BitBake += в конец списка через пробел (см. раздел Basic Syntax [4]).

B

B

Каталог внутри каталога сборки, куда система сборки OE помещает объекты, созданные в процессе сборки задания. По умолчанию этот каталог совпадает с каталогом S, определяемым как

```
S = "${WORKDIR}/${BP}"
```

Можно разделить каталог S и каталог, указанный переменной B. Большинство заданий, применяющих автоматические инструменты поддерживают такое разделение. По умолчанию система сборки использует отдельные каталоги для gcc и некоторых заданий для ядра.

BAD_RECOMMENDATIONS

Перечисляет «лишь рекомендуемые» (recommended-only) пакеты, которые не устанавливаются. К таким пакетам относятся указанные через переменную RRECOMMENDS. Можно предотвратить установку любого из таких пакетов, указав его в переменной BAD_RECOMMENDATIONS.

```
BAD_RECOMMENDATIONS = "package_name package_name package_name ..."
```

Эту переменную можно задать глобально в файле local.conf или присоединить к заданию для конкретного образа, используя переопределение имени задания, как показано ниже.

```
BAD_RECOMMENDATIONS_pn-target_image = "package_name"
```

Важно отметить, что при отказе от установки пакетов с помощью этой переменной и наличии зависимости от них других пакетов (указанных в переменной RDEPENDS для задания), система сборки OE будет игнорировать запрос и установит пакеты для предотвращения нарушений в зависимостях.

Переменная поддерживается только для менеджеров пакетов IPK и RPM, но не поддерживается для DEB.

См. также описания переменных NO_RECOMMENDATIONS и PACKAGE_EXCLUDE.

BASE_LIB

Имя каталоги библиотек для настройки CPU или ABI, которое применяется лишь в контексте Multilib (см. раздел Combining Multiple Versions of Library Files into One Image [2]). Переменная определяется во включаемых файлах машины в дереве исходного кода. Если Multilib не используется, переменная имеет значение lib.

BASE_WORKDIR

Задаёт базу для сетевого каталога, применяемого во всех задания (по умолчанию ${TMPDIR}/work).

BB_ALLOWED_NETWORKS

Задаёт список разделенных пробелами хостов, которые сборщику разрешено использовать для получения нужных исходных кодов. Ниже рассмотрены некоторые аспекты использования этой переменной.

- Список хостов применяется лишь в при установке BB_NO_NETWORK = "0" или не заданной переменной.
- Ограниченно поддерживаются шаблоны для начальной части имени хоста. Например, приведённая ниже строка будет соответствовать git.gnu.org, ftp.gnu.org и foo.git.gnu.org.

  ```
  BB_ALLOWED_NETWORKS = "*.gnu.org"
  ```

  Символ * работает только в начальной части имени, отделённой точкой от остального имени и не допускается его использование в других местах. Например, можно указать *.foo.bar, но *aa.foo.bar не будет работать.
- Зеркала, не указанные в списке хостов, будут пропускаться с указанием в отладочном выводе.
- Попытка доступа в сети, не указанную в списке, завершится отказом.

Использование BB_ALLOWED_NETWORKS вместе с переменной PREMIRRORS очень полезно. Добавление хоста в PREMIRRORS позволяет получать исходные коды с этого хоста и предотвращает ошибки, возникающие когда неразрешенный хост указан в SRC_URI. Это связано с тем, что сборщик не пытается использовать хосты из SRC_URI после успешной выборки на основе PREMIRRORS.

BB_DANGLINGAPPENDS_WARNONLY

Определяет обработку в BitBake ситуаций, когда файл дополнения (.bbappend) не имеет соответствующего задания (.bb). Такое часто возникает при рассинхронизации уровней (например, oe-core нужно задания, для которого старой версии уже нет, а другой уровень ещё не обновлён с новой версией задания). По умолчанию в такой ситуации возникает критическая ошибка и это обеспечивает наиболее безопасный вариант. Поведение можно изменить указав для переменной BB_DANGLINGAPPENDS_WARNONLY значение "1", "yes" или "true" в файле local.conf каталога сборки.

BB_DISKMON_DIRS

Отслеживает доступное пространство и inode при сборке, позволяя контроль по этим параметрам (по умолчанию отключён.). Переменная задаётся в форме BB_DISKMON_DIRS = "<action>,<dir>,<threshold> […]", где

<action>

ABORT - незамедлительное прерывание сборки при превышении порога.

STOPTASKS - остановка сборки после завершения текущих задач в случае превышения порога.

WARN - вывод предупреждения и продолжение сборки. Последующие предупреждения выдаются в соответствии с переменной BB_DISKMON_WARNINTERVAL, которая должна быть определена.

<dir>

Любые каталоги для мониторинга, разделённые пробелами. Если каталоги расположены на одном устройстве, отслеживается лишь первый.

<threshold>

Минимальное пространство на диске или/и минимальное число inode. Можно применять суффиксы G (Гбайт), M (Мбайт), K (Кбайт, принято по умолчанию). Не следует использовать GB, MB или KB.

```
BB_DISKMON_DIRS = "ABORT,${TMPDIR},1G,100K WARN,${SSTATE_DIR},1G,100K"
BB_DISKMON_DIRS = "STOPTASKS,${TMPDIR},1G"
BB_DISKMON_DIRS = "ABORT,${TMPDIR},,100K"
```

Первый пример работает только при установке переменной BB_DISKMON_WARNINTERVAL и заставляет систему сборки немедленно прерывать процесс, если свободное пространство в ${TMPDIR} становится меньше 1 Гбайта или число inode ниже 100 Кбайт. Поскольку указаны 2 каталога, система сборки будет выдавать предупреждение при снижении свободного места в ${SSTATE_DIR} меньше 1 Гбайт или inode меньше 100 Кбайт. Следующие предупреждения будут выдаваться с интервалами, заданными BB_DISKMON_WARNINTERVAL.

Второй пример будет останавливать сборку по завершении задач, когда свободное место в ${TMPDIR} станет меньше 1 Гбайт. Число inode не отслеживается.

Третий пример задаёт немедленное прерывание сборки при числе свободных inode в ${TMPDIR} меньше 100 Кбайт без отслеживания свободного места в каталоге.

BB_DISKMON_WARNINTERVAL

Задаёт интервал предупреждений о нехватке места на диске и свободных inode. Для использования переменной нужно задать BB_DISKMON_DIRS с действием WARN. В процессе сборки при снижении свободного места будут выдаваться предупреждения с заданным интервалом. Если при установке BB_DISKMON_DIRS = "WARN" интервал не указан, используется BB_DISKMON_WARNINTERVAL = "50M,5K". При задании переменной в файле конфигурации применяется форма BB_DISKMON_WARNINTERVAL = "<disk_space_interval>,<disk_inode_interval>".

<disk_space_interval>

Интервал свободного пространства в G (Гбайт), M (Мбайт) или K (Кбайт). Не используйте GB, MB или KB.

<disk_inode_interval>

Интервал свободных inode в G (Гбайт), M (Мбайт) или K (Кбайт). Не используйте GB, MB или KB.

```
BB_DISKMON_DIRS = "WARN,${SSTATE_DIR},1G,100K"
BB_DISKMON_WARNINTERVAL = "50M,5K"
```

Эти переменные заставляют BitBake выдавать предупреждения каждый раз, когда свободное место в каталоге ${SSTATE_DIR} снижается на 50 Мбайт или число свободных inode - на 5 Кбайт. Предупреждения будут выдаваться после того, как свободное место станет меньше 1 Гбайт или число свободных inode - меньше 100 Кбайт.

BB_GENERATE_MIRROR_TARBALLS

Заставляет систему сборки размещать архивы репозиториев исходного кода (например, Git), включая метаданные, в каталоге DL_DIR. Для повышения производительности система сборки OE п умолчанию не сохраняет такие архивы.

Переменная устанавливается в файле local.conf каталога сборки.

BB_NUMBER_THREADS

Максимальное число задач, которые BitBake следует запускать параллельно. Система сборки OE автоматически устанавливает для этой переменной число ядре на хосте сборки. Например, в системе с 2-ядерным процессором и поддержкой hyper-threading по умолчанию задаётся BB_NUMBER_THREADS = "4". Для 1-процессорных систем не следует переопределять эту переменную. Оптимизация скорости сборки описана в Speeding Up a Build [2].

BB_SERVER_TIMEOUT

Указывает время (в секундах), по истечение которого сервер BitBake выключается при отсутствии активности. Например, в файле local.conf можно указать 20-секундное ожидание в виде BB_SERVER_TIMEOUT = "20". Для постоянной работы сервера следует установить BB_SERVER_TIMEOUT = "-1".

BBCLASSEXTEND

Позволяет расширить задание для сборки вариантов программ. Существуют распространённые варианты заданий, такие как «естественные» (например, quilt-native, копирующее Quilt для работы на системе сборки), «кросс-» (например, gcc-cross для создания на сборочной машине программ для работы на целевой платформе MACHINE), nativesdk для машины SDK вместо MACHINE и mulitlib в форме multilib:multilib_name. Для сборки разных вариантов задания с минимальным количеством кода обычно просто добавляются в задание приведённые ниже строки.

```
BBCLASSEXTEND =+ "native nativesdk"
BBCLASSEXTEND =+ "multilib:multilib_name"
```

Механизм BBCLASSEXTEND создает варианты задания, переписывая значения переменных и применяя переопределения, такие как _class-native. Например, для создания естественного (native) варианта задания переменная DEPENDS в задании foo переписывает как DEPENDS для foo-native.

Даже при использовании BBCLASSEXTEND задание анализируется однократно, что вносит некоторые ограничения. Например, невозможно включить разные файлы в зависимости от варианта, поскольку оператор include обрабатывается в процессе разбора задания.

BBFILE_COLLECTIONS

Список имён настроенных уровней, используемых для поиска других переменных BBFILE_*. Обычно каждый уровень добавляет своё имя в конце этой переменной в своём файле conf/layer.conf.

BBFILE_PATTERN

Переменная, которая преобразуется для сопоставления с файлами из переменной BBFILES в определённом слое. Эта переменная используется в файле conf/layer.conf и должна иметь суффикс с именем конкретного слоя (например, BBFILE_PATTERN_emenlow).

BBFILE_PRIORITY

Задаёт уровень приоритета для файлов заданий уровня. Эта переменная полезна в случаях, когда одноимённые задания указаны в разных уровнях. Установка переменной позволяет задать приоритет заданий своего уровня, позволяя сделать его предпочтительным по сравнению с другими. Предпочтения, задаваемые этой переменной взаимодействуют с версиями заданий (переменная PV). Например, уровень, имеющий задание с более высоким значением PV, может иметь меньшее значение BBFILE_PRIORITY и быть менее предпочтительным.

Большее значение BBFILE_PRIORITY означает более высокий приоритет. Если переменная не задана, её значение устанавливается на основе зависимостей уровня (см. LAYERDEPENDS). Если уровень не имеет зависимостей и приоритет не задан, определяется минимальным заданным приоритетом + 1 (т. е. 1, если приоритет не задан).

С помощью команды bitbake-layers show-layers можно посмотреть все настроенные уровни с их приоритетами.

BBFILES

Разделённый пробелами список файлов заданий, который BitBake использует для сборки программ. При указании файлов заданий можно применять синтаксис Python glob.

BBFILES_DYNAMIC

Активирует содержимое при наличии идентифицированных уровней, которые задаются коллекциями, определёнными в них. Переменная используется для исключения файлов .bbappend, для которых соответствующие файлы .bb размещены на уровне, пытающемся изменить другие уровни, но не желающем задавать жёсткую зависимость от этих уровней. Переменная BBFILES_DYNAMIC задаётся в форме collection_name:filename_pattern. Пример указывает имена двух коллекций и два шаблона имён файлов.

```
BBFILES_DYNAMIC += " \
    clang-layer:${LAYERDIR}/bbappends/meta-clang/*/*/*.bbappend \
    core:${LAYERDIR}/bbappends/openembedded-core/meta/*/*/*.bbappend \
"
```

Другой пример показывает сообщение об ошибке при обнаружении непригодной записи и разбор прерывается.

```
ERROR: BBFILES_DYNAMIC entries must be of the form <collection name>:<filename pattern>, not:
       /work/my-layer/bbappends/meta-security-isafw/*/*/*.bbappend
       /work/my-layer/bbappends/openembedded-core/meta/*/*/*.bbappend
```

BBINCLUDELOGS

Управляет отображением журнала сборки программой BitBake при возникновении отказов.

BBINCLUDELOGS_LINES

При установленной переменной BBINCLUDELOGS задаёт максимальное число строк из журнала задачи для отображения в случаях отказа. По умолчанию журнал выводится целиком.

BBLAYERS

Список уровней, включаемых в сборку. Переменная указывается в файле bblayers.conf каталога сборки. Например,

```
BBLAYERS = " \
  /home/scottrif/poky/meta \
  /home/scottrif/poky/meta-poky \
  /home/scottrif/poky/meta-yocto-bsp \
  /home/scottrif/poky/meta-mykernel \
  "
```

Здесь включены 4 уровня, один из которых (meta-mykernel) является пользовательским.

BBMASK

Управляет исключением файлов заданий и дополнений из обработки BitBake, «пряча» ненужные файлы. BitBake игнорирует задания и дополнения, соответствующие выражениям маски. Значение переменной передается компилятору регулярных выражений Python, в маске применяется синтаксис Python Regular Expression (re). Сравнение выполняется для полных путей к файлам. Приведённый ниже пример обеспечивает игнорирование BitBake всех заданий и дополнений в каталоге meta-ti/recipes-misc/.

```
BBMASK = "meta-ti/recipes-misc/"
```

Если нужно скрыть множество каталогов или заданий, можно указать несколько фрагментов регулярных выражений, как показано ниже.

```
BBMASK += "/meta-ti/recipes-misc/ meta-ti/recipes-ti/packagegroup/"
BBMASK += "/meta-oe/recipes-support/"
BBMASK += "/meta-foo/.*/openldap"
BBMASK += "opencv.*\.bbappend"
BBMASK += "lzma"
```

Для исключения каталога следует указывать символ / в конце имени.

BBMULTICONFIG

Позволяет BitBake выполнить сборку со множеством конфигураций и указывает каждую конфигурацию (multiconfig). С помощью этой переменной можно задать BitBake сборку нескольких целей, каждая из которых имеет свою конфигурацию. Переменная задаётся в файле conf/local.conf, например, BBMULTIFONFIG = "configA configB configC". Каждый конфигурационный файл должен размещаться в каталоге сборки внутри каталога conf/multiconfig (например, build_directory/conf/multiconfig/configA.conf). Использование переменной в среде с поддержкой сборки нескольких конфигураций описано в разделе Building Images for Multiple Targets Using Multiple Configurations [2].

BBPATH

Применяется BitBake для поиска файлов .bbclass и файлов конфигурации подобно системной переменной PATH. При использовании BitBake извне каталога сборки нужно обеспечить, чтобы переменная BBPATH указывала сборочный каталог. Установка выполняется так же, как для других переменных среды и переменная должна быть задана до начала работы BitBake.

```
$ BBPATH = "build_directory"
$ export BBPATH
$ bitbake target
```

BBSERVER

При определении в среде BitBake переменная BBSERVER указывает удалённый сервер BitBake. Переменная экспортируется в среду BitBake как показано ниже.

```
export BBSERVER=localhost:$port"
```

По умолчанию BBSERVER присутствует также в BB_HASHBASE_WHITELIST, поэтому переменная BBSERVER исключена из расчёта контрольной суммы и данных зависимостей.

BINCONFIG

При наследовании класса binconfig-disabled эта переменная указывает двоичные сценарии настройки, которые следует отключить в пользу запроса информации с помощью pkg-config. Класс binconfig-disabled будет изменять указанные сценарии для возврата ошибки, позволяющего найти и заменить вызовы таких сценариев. При добавлении нескольких сценариев они разделяются пробелами, как показано ниже для задания libpng.

```
BINCONFIG = "${bindir}/libpng-config ${bindir}/libpng16-config"
```

BINCONFIG_GLOB

При наследовании класса binconfig эта переменная задаёт шаблон для сценариев конфигурации, которые нужно редактировать. При редактировании изменяются пути, заданные во время компиляции, чтобы они были пригодны для установки в sysroot и могли использоваться процессами сборки других заданий. Переменная использует шаблоны оболочки при сопоставлении имён, похожие на fnmatch и glob. Информацию о работе переменной можно найти в файле meta/classes/binconfig.bbclass дерева исходных кодов, а также в параграфе 6.7. binconfig.bbclass.

BP

Базовое имя и версия задания без дополнительных суффиксов (-native, lib64- и т. п.). BP имеет вид ${BPN}-${PV}.

BPN

Это вариант переменной PN с удалёнными общими префиксами и суффиксами, такими как nativesdk-, -cross, -native, а также lib64- и lib32-. Точный список удаляемых префиксов и суффиксов задаётся переменными MLPREFIX и SPECIAL_PKGSUFFIX.

BUGTRACKER

Задаёт идентификатор URL сайта отслеживания ошибок для задания. Система сборки OE не использует эту переменную, она служит лишь указателем для разработчиков в случае необходимости сообщить об ошибке.

BUILD_ARCH

Указывает архитектуру хоста сборки (например, i686). Система сборки OE устанавливает переменную по выводу команды uname.

BUILD_AS_ARCH

Задаёт зависящие от архитектуры флаги ассемблера для хоста сборки (по умолчанию "").

BUILD_CC_ARCH

Задаёт зависящие от архитектуры флаги компилятора C для хоста сборки (по умолчанию "").

BUILD_CCLD
Задаёт команду компоновки, используемую хостом сборки в случае применения для компоновки компилятора C. По умолчанию BUILD_CCLD указывает GCC и передаёт в качестве аргумента значение BUILD_CC_ARCH.

BUILD_CFLAGS
Задаёт флаги, передаваемые компилятору C при сборке для сборочного хоста. При сборке в контексте -native по умолчанию используется значение CFLAGS.

BUILD_CPPFLAGS
Задаёт флаги, передаваемые препроцессору C (т. е. компиляторам C и C++). При сборке в контексте -native по умолчанию используется значение CPPFLAGS.

BUILD_CXXFLAGS
Задаёт флаги, передаваемые компилятору C++ при сборке для сборочного хоста. При сборке в контексте -native по умолчанию используется значение CXXFLAGS.

BUILD_FC
Задаёт флаги, передаваемые компилятору Fortran для хоста сборки. По умолчанию переменная указывает Gfortran и передаёт значение BUILD_CC_ARCH.

BUILD_LD
Задаёт флаги команду компоновщика для хоста сборки. По умолчанию переменная указывает компоновщик GNU (ld) и передаёт значение BUILD_LD_ARCH.

BUILD_LD_ARCH
Задаёт зависящие от архитектуры флаги компоновщика для хоста сборки (по умолчанию "").

BUILD_LDFLAGS
Задаёт флаги, передаваемые компоновщику при сборке для сборочного хоста. При сборке в контексте -native по умолчанию используется значение LDFLAGS.

BUILD_OPTIMIZATION
Задаёт флаги оптимизации, передаваемые компилятору C при сборке для сборочного хоста или SDK через заданные по умолчанию значения переменных BUILD_CFLAGS и BUILDSDK_CFLAGS. По умолчанию переменная BUILD_OPTIMIZATION имеет значение -O2 -pipe.

BUILD_OS
Указывает операционную систему сборочного хоста (например, linux). Система сборки OE устанавливает переменную по первому слову вывода команды uname, преобразованному в нижний регистр.

BUILD_PREFIX
Префикс двоичного файла инструментария для естественных заданий. Система сборки OE использует значение переменной TARGET_PREFIX при сборке естественных заданий.

BUILD_STRIP
Указывает команду для вырезания отладочных символов из файлов, созданных для сборочного хоста (по умолчанию ${BUILD_PREFIX}strip).

BUILD_SYS
Задаёт систему (включая архитектуру и ОС), используемую при сборке для сборочного хоста (задания native). Система сборки OE автоматически назначает эту переменную на основе значений BUILD_ARCH, BUILD_VENDOR и BUILD_OS (это значение не требуется менять).

BUILD_VENDOR
Задаёт имя производителя для использования при сборке для сборочного хоста (по умолчанию "").

BUILDDIR
Указывает местоположение каталога сборки, который можно задать опосредованно через сценарий oe-init-build-env, указав путь в команде запуска сценария. Если сценарий запущен без указания каталога сборки, по умолчанию BUILDDIR будет применять текущий каталог.

BUILDHISTORY_COMMIT
При наследовании класса buildhistory управляет фиксацией вывода истории сборки в локальном репозитории Git. При значении 1 этот локальный репозиторий будет поддерживаться автоматически классом buildhistory с фиксацией при каждой сборке изменений в каждом каталоге верхнего уровня вывода истории сборки (images, packages, sdk). Это позволяет отслеживать историю сборки. По умолчанию класс buildhistory не фиксирует вывод истории сборки в локальном репозитории Git, т. е. BUILDHISTORY_COMMIT ?= "0".

BUILDHISTORY_COMMIT_AUTHOR
При наследовании класса buildhistory указывает автора для каждой фиксации Git. Чтобы переменная работала, нужно установить BUILDHISTORY_COMMIT = "1". Git требует в переменной BUILDHISTORY_COMMIT_AUTHOR значение в формате email@host. Использование недействительного адреса или хоста не ведёт к ошибке. По умолчанию класс buildhistory задаёт BUILDHISTORY_COMMIT_AUTHOR ?= "buildhistory <buildhistory@${DISTRO}>"

BUILDHISTORY_DIR
При наследовании класса buildhistory эта переменная задаёт каталог для хранения данных истории сборки. По умолчанию устанавливается BUILDHISTORY_DIR ?= "${TOPDIR}/buildhistory".

BUILDHISTORY_FEATURES
При наследовании класса buildhistory эта переменная задаёт включённые функции истории сборки (см. Maintaining Build Output Quality [2]). Свойства указываются в форме разделённого пробелами списка элементов:
- *image* - анализ содержимого образов, включая список установленных пакетов;
- *package* - анализ содержимого отдельных пакетов;
- *sdk* - анализ содержимого SDK;
- *task* - сохранение подписей выходных файлов задач sstate (по одному файлу на задачу с контрольной суммой SHA-256 для каждого подготовленного файла).

По умолчанию класс buildhistory устанавливает BUILDHISTORY_FEATURES ?= "image package sdk".

BUILDHISTORY_IMAGE_FILES
При наследовании класса buildhistory переменная задаёт список путей к файлам, копируемым из образа в каталог истории сборки каталога image-files в каталоге для образа, чтобы можно было отследить содержимое каждого файла. По умолчанию копируются /etc/passwd и /etc/group, что позволяет контролировать изменения пользователей и групп. Можно включить в список любой файл. Задание некорректного пути в переменной не ведёт к ошибке, поэтому можно указать даже не всегда присутствующие файлы. По умолчанию класс buildhistory устанавливает BUILDHISTORY_IMAGE_FILES ?= "/etc/passwd /etc/group".

BUILDHISTORY_PUSH_REPO

При наследовании класса buildhistory переменная может задавать удалённый репозиторий, в который история сборки выталкивает (push) изменения Git. Для работы переменной BUILDHISTORY_PUSH_REPO нужно установить BUILDHISTORY_COMMIT = "1". Репозиторий должен соответствовать заданному удалённому адресу, понятному Git, или удалённому имени, установленному вручную с использованием git remote в локальном репозитории. По умолчанию класс buildhistory устанавливает BUILDHISTORY_PUSH_REPO ?= "".

BUILDSDK_CFLAGS

Указывает флаги для передачи компилятору C при сборке для SDK. При сборке в контексте nativesdk- переменная CFLAGS получает значение этой переменной по умолчанию.

BUILDSDK_CPPFLAGS

Указывает флаги для передачи препроцессору C (для компиляторов C и C++) при сборке для SDK. При сборке в контексте nativesdk- переменная CPPFLAGS получает значение этой переменной по умолчанию.

BUILDSDK_CXXFLAGS

Указывает флаги для передачи компилятору C++ при сборке для SDK. При сборке в контексте nativesdk- переменная CXXFLAGS получает значение этой переменной по умолчанию.

BUILDSDK_LDFLAGS

Указывает флаги для передачи компоновщику при сборке для SDK. При сборке в контексте nativesdk- переменная LDFLAGS получает значение этой переменной по умолчанию.

BUILDSTATS_BASE

Указывает местоположение каталога со статистикой сборки при включённом и используемом классе buildstats. По умолчанию BUILDSTATS_BASE имеет значение ${TMPDIR}/buildstats/.

BUSYBOX_SPLIT_SUID

Управляет для задания BusyBox расщеплением выходного файла на 2 части, одна из который выполняет функции, требующие setuid root, а другая делает все остальное (где не требуется setuid root).

BUSYBOX_SPLIT_SUID по умолчанию имеет значение 1, обеспечивающее на выходе два исполняемых файла. Установка значения 0 отменяет расщепление выходного файла.

C

CACHE

Задаёт каталог, используемый BitBake для хранения кэшированных метаданных чтобы не анализировать их при каждом запуске BitBake.

CC

Сокращённая команда и аргументы для запуска компилятора C.

CFLAGS

Задаёт флаги для передачи компилятору C. Переменная экспортируется в окружение и доступна программам, собираемым на этапе компиляции. Принятая по умолчанию инициализация CFLAGS зависит от собираемого объекта:

- TARGET_CFLAGS при сборке для целевой платформы;
- BUILD_CFLAGS при сборке для сборочного хоста (-native);
- BUILDSDK_CFLAGS при сборке для SDK (nativesdk-).

CLASSOVERRIDE

Внутренняя переменная, задающая переопределение класса, который следует применять (например, class-target, class-native и т. п.). Классы, использующие эту переменную (например, native, nativesdk и т. п.), устанавливают подходящее значение. Переменная CLASSOVERRIDE получает используемое по умолчанию значение class-target из файла bitbake.conf.

Приведённое ниже переопределение позволяет установить дополнительные файлы, но только при сборке для целевой платформы.

```
do_install_append_class-target() {
    install my-extra-file ${D}${sysconfdir}
}
```

Ниже приведен пример, где переменная FOO получает значение native при сборке для сборочного хоста и other - остальных случаях

```
FOO_class-native = "native"
FOO = "other"
```

Базовым механизмом использования CLASSOVERRIDE является просто включение в принятое по умолчанию значение OVERRIDES.

CLEANBROKEN

При установке в задании значения 1 для этой переменной команда make clean не будет работать для собираемой программы, поэтому система сборки OE не будет пытаться запустить её при выполнении задачи do_configure.

COMBINED_FEATURES

Указывает список аппаратных свойств, включённых в MACHINE_FEATURES и DISTRO_FEATURES. Это определяет список свойств, которыми имеет смысл управлять на уровне конфигурации машины и дистрибутива. Например, свойство bluetooth требует аппаратной поддержки, но является также опцией дистрибутива.

COMMON_LICENSE_DIR

Указывает каталог meta/files/common-licenses в дереве исходных кодов, где хранятся файлы базовых лицензий.

COMPATIBLE_HOST

Регулярное выражение, указывающее один или несколько хостов (естественное задание) или целей (неестественное задание), с которыми задание совместимо. Выражение сопоставляется с HOST_SYS. Переменную можно использовать для предотвращения сборки заданий для несовместимых классов систем. Особенно полезно это при сборке ядер. Переменная также помогает ускорить синтаксический анализ, поскольку система сборки будет пропускать несовместимые задания.

COMPATIBLE_MACHINE

Регулярное выражение, указывающее одну или несколько целевых машин, совместимых с заданием. Выражение сравнивается с MACHINEOVERRIDES. Можно использовать эту переменную для предотвращения сборки кода для машин, которые не совместимы с заданием. Особенно полезно это при сборке ядер. Переменная также помогает повысить скорость анализа, поскольку система сборки будет пропускать несовместимые с машиной задания.

COMPLEMENTARY_GLOB

Задаёт шаблоны сопоставления при установке списка дополнительных пакетов для всех пакетов, явно или неявно включённых в образ. Переменная использует шаблоны соответствия имён Unix (fnmatch), похожие на шаблоны преобразования путей Unix (glob).

Результирующий список пакетов связан с элементом, который может быть добавлен в IMAGE_FEATURES. Примером может служить элемент dev-pkgs при добавлении которого в IMAGE_FEATURES будут устанавливаться пакеты -dev (заголовки и другие файлы для разработки) для каждого пакета в образе.

Для добавления указанного шаблоном свойства служит флаг переменной и значение шаблона, например, COMPLEMENTARY_GLOB[dev-pkgs] = '*-dev'.

COMPONENTS_DIR

Сохраняет компоненты sysroot для каждого задания. Система сборки OE использует переменную при создании зависимых от задания каталогов sysroot для других заданий. По умолчанию установлено "${STAGING_DIR}-components" (т. е. "${TMPDIR}/sysroots-components").

CONF_VERSION

Отслеживает версию файла local.conf. Значение увеличивается при каждом изменении совместимости build/conf/.

CONFFILES

Указывает редактируемые или настраиваемые файлы в пакете. Если система PMS[1] применяется для управления пакетами на целевой платформе, возможна замена конфигурационных файлов после установки пакета, что может быть нежелательно. Иными словами, в пакете могут быть редактируемые файлы, которые не следует менять при обновлении пакета. Переменная CONFFILES позволяет указать эти файлы и PMS не будет обновлять их.

Для использования переменной CONFFILES имя пакета переопределяется в указанием результирующего пакета. Затем указывается список разделенных пробелами файлов, например, CONFFILES_${PN} += "${sysconfdir}/file1 ${sysconfdir}/file2 ${sysconfdir}/file3"

Переменные CONFFILES и FILES связаны между собой и файлы из CONFFILES должны быть подмножеством файлов из FILES, поскольку они являются частью пакета.

При указании путей в переменной CONFFILES хорошим тоном является использование подходящих переменных. Например, следует указывать ${sysconfdir} вместо /etc или ${bindir} вместо /usr/bin. Список этих переменных приведен в начале файла meta/conf/bitbake.conf в дереве исходных кодов.

CONFIG_INITRAMFS_SOURCE

Задаёт файлы для initramfs. Система сборки OE получает и использует эту переменную ядра Kconfig, как переменную окружения, которая по умолчанию имеет пустое значение ("").

CONFIG_INITRAMFS_SOURCE может указывать один архив cpio (.cpio) или список разделенных пробелами каталогов и файлов для сборки образа initramfs. Файл cpio должен включать архив файловой системы для использования в качестве образа initramfs. Каталоги должны включать схему файловой системы для включения в образ initramfs, а файлы - записи в соответствии с форматом, описанным программой usr/gen_init_cpio в дереве ядра.

Если задано множество каталогов и файлов, образ initramfs будет включать из. Информация о создании initramfs приведена в разделе Building an Initial RAM Filesystem (initramfs) Image [2].

CONFIG_SITE

Список файлов, содержащих результаты теста autoconf, относящиеся к текущей сборке. Переменная используется утилитами Autotools при настройке конфигурации (configure).

CONFIGURE_FLAGS

Минимальные аргументы для GNU configure.

CONFLICT_DISTRO_FEATURES

При наследовании класса distro_features_check эта переменная указывает свойства дистрибутива, которые будут вызывать конфликты при сборке образа. Иными словами, при включении свойства в CONFLICT_DISTRO_FEATURES и DISTRO_FEATURES в одной конфигурации будет возникать ошибка.

COPYLEFT_LICENSE_EXCLUDE

Список разделенных пробелами лицензий для исключения источников, архивируемых классом archiver. Если лицензия из переменной LICENSE включена в COPYLEFT_LICENSE_EXCLUDE, её источник не будет архивироваться. Переменная имеет более высокий приоритет по сравнению с COPYLEFT_LICENSE_INCLUDE.

Принятое по умолчанию значение "CLOSED Proprietary" устанавливается классом copyleft_filter, который наследуется классом archiver.

COPYLEFT_LICENSE_INCLUDE

Список разделенных пробелами лицензий для включения источников, архивируемых классом archiver. Если лицензия из переменной LICENSE включена в COPYLEFT_LICENSE_INCLUDE, её источник будет архивироваться. Принятое по умолчанию значение задаётся классом copyleft_filter, наследуемым классом archiver, и содержит GPL*, LGPL* и AGPL*.

COPYLEFT_PN_EXCLUDE

Список заданий для исключения источников, архивируемых классом archiver. Переменная переопределяет включение и исключение лицензий через COPYLEFT_LICENSE_INCLUDE и COPYLEFT_LICENSE_EXCLUDE. Принятое по умолчанию значение "" указывает отсутствие явного исключения заданий и устанавливается классом copyleft_filter, который наследуется классом archiver.

COPYLEFT_PN_INCLUDE

Список заданий для включения источников, архивируемых классом archiver. Переменная переопределяет включение и исключение лицензий через COPYLEFT_LICENSE_INCLUDE и COPYLEFT_LICENSE_EXCLUDE. Принятое по умолчанию значение "" указывает отсутствие явного включения заданий и устанавливается классом copyleft_filter, который наследуется классом archiver.

COPYLEFT_RECIPE_TYPES

Список разделенных пробелами типов заданий для включения в архив исходных кодов, создаваемый классом archiver. Задания могут иметь тип target, native, nativesdk, cross, crosssdk и cross-canadian. По умолчанию применяется тип target* и для COPYLEFT_RECIPE_TYPES оно устанавливается классом copyleft_filter, который наследуется классом archiver.

[1]Package Management System - система управления пакетами.

COPY_LIC_DIRS

При установке значения 1 для этой переменной и COPY_LIC_MANIFEST система сборки OE копирует в образ файлы лицензий из каталога /usr/share/common-licenses для каждого пакета. Файлы лицензий помещаются в образ при его сборке.

COPY_LIC_DIRS не предлагает путь добавления лицензий для вновь устанавливаемых пакетов в образах с файловой системой read-only, которая не разрешает запись (см. описание LICENSE_CREATE_PACKAGE). Дополнительная информация о предоставлении текстов лицензий приведена в разделе Providing License Text [2].

COPY_LIC_MANIFEST

При установке значения 1 система сборки OE копирует манифест лицензии в файл образа /usr/share/common-licenses/license.manifest в процессе сборки.

COPY_LIC_MANIFEST не предлагает для вновь устанавливаемых в образ пакетов путь добавления лицензий, который может лучше подходить для файловых систем без возможности записи read-only), не позволяющих вносить изменения (см. описание переменной LICENSE_CREATE_PACKAGE). Сведения о предоставлении текстов лицензий приведены также в разделе Providing License Text [2].

CORE_IMAGE_EXTRA_INSTALL

Задаёт список пакетов для добавления в образ. Переменную следует включать лишь в файл local.conf каталога сборки. Эта переменная заменила ранее применявшуюся переменную POKY_EXTRA_INSTALL.

COREBASE

Задаёт родительский каталог уровня метаданных OpenEmbedded-Core (meta).

COREBASE_FILES

Список файлов из каталога COREBASE, которые следует скопировать, если они не относятся к уровням, указанным в bblayers.conf. Переменная предназначена для копирования метаданных из системы сборки OE в расширяемый SDK. Явное указание копируемых файлов из COREBASE требуется потому, что этот каталог обычно содержит каталоги сборки и другие файлы, которые не нужны в eSDK.

CPP

Сокращённая команда и аргументы для запуска препроцессора C.

CPPFLAGS

Задаёт флаги для передачи препроцессору C (для компиляторов C и C++). Переменная экспортируется в окружение и доступна программам, собираемым на этапе компиляции. Принятая по умолчанию инициализация CPPFLAGS зависит от собираемого объекта:

- TARGET_CPPFLAGS при сборке для целевой платформы;
- BUILD_CPPFLAGS при сборке для сборочного хоста (-native);
- BUILDSDK_CPPFLAGS при сборке для SDK (nativesdk-).

CROSS_COMPILE

Префикс двоичных файлов инструментария для целевой платформы (совпадает со значением TARGET_PREFIX). Система сборки OE устанавливает CROSS_COMPILE лишь в определённом контексте (например, при сборке заданий для ядра или модулей ядра).

CVSDIR

Каталог для хранения файлов, выбранных в системе CVS.

CXX

Сокращённая команда и аргументы для запуска компилятора C++.

CXXFLAGS

Задаёт флаги для передачи компилятору C++. Переменная экспортируется в окружение и доступна программам, собираемым на этапе компиляции. Принятая по умолчанию инициализация CXXFLAGS зависит от собираемого объекта:

- TARGET_CXXFLAGS при сборке для целевой платформы;
- BUILD_CXXFLAGS при сборке для сборочного хоста (-native);
- BUILDSDK_CXXFLAGS при сборке для SDK (nativesdk-).

D

D

Каталог назначения в каталоге сборки, куда устанавливаются компоненты задачей do_install (по умолчанию ${WORKDIR}/image). Задачи, читающие каталог или записывающие в него, следует запускать в fakeroot [1].

DATE

Дата начала сборки в формате YMD (например, 20150209 для 9 февраля 2015 г.).

DATETIME

Дата и время начала текущей сборки в формате, пригодном для временных меток.

DEBIAN_NOAUTONAME

При наследовании класса debian (принято по умолчанию) переменная задаёт отмену переименования библиотек в стиле Debian для отдельного пакета. При установке переменной должно применяться имя пакета, например, DEBIAN_NOAUTONAME_fontconfig-utils = "1".

DEBIANNAME

При наследовании класса debian (принято по умолчанию) переменная позволяет переопределить имя библиотеки для отдельного пакета. Такое переопределение происходит редко и при установке переменной должно применяться имя пакета, например, DEBIANNAME_${PN} = "dbus-1".

DEBUG_BUILD

Задаёт сборку пакетов с отладочной информацией. Это влияет на значение SELECTED_OPTIMIZATION.

DEBUG_OPTIMIZATION

Опции для передачи в TARGET_CFLAGS и CFLAGS при компиляции системы с отладкой. По умолчанию переменная имеет значение -O -fno-omit-frame-pointer ${DEBUG_FLAGS} -pipe.

DEFAULT_PREFERENCE

Задаёт незначительное смещение при выборе приоритета для задания. Чаще всего для переменной устанавливают значение -1 в заданиях для development-версий программ. Это ведёт к сборке по умолчанию стабильной версии задания при отсутствии PREFERRED_VERSION. Смещение от DEFAULT_PREFERENCE невелико и часто переопределяется BBFILE_PRIORITY, если эта переменная различается в двух уровнях, содержащих разные версии одного задания.

DEFAULTTUNE

Используемые по умолчанию настройки CPU и ABI для системы сборки OE. Переменная DEFAULTTUNE помогает определить TUNE_FEATURES. Принятые по умолчанию настройки явно или неявно задаются машиной (MACHINE), однако их можно переопределить с помощью значений переменной AVAILTUNES.

DEPENDS

Список зависимостей при сборке задания. Это зависимости от других заданий (например, заголовков и общих библиотек). Например, задание foo может включать зависимость DEPENDS = "bar". Это означает, что все файлы, установленные заданием bar, будут доступны в подходящей промежуточной системе sysroot, указанной переменными STAGING_DIR*, при выполнении задачи do_configure для foo. Этот механизм реализуется за счёт зависимости do_configure от do_populate_sysroot для каждого задания, указанного в DEPENDS, через объявление [deptask] в классе base.

Иногда нужно явно указать, например, STAGING_DIR_HOST. Стандартные классы и связанные со сборкой переменные настраиваются для автоматического использования подходящей промежуточной системы sysroot.

Другим примером может служить применение DEPENDS для добавления утилит, работающих на сборочном хосте в процессе сборки. Например, задание codegen может включать DEPENDS = "codegen-native".

Дополнительная информация приведена в описании класса native class и переменной EXTRANATIVEPATH.

- DEPENDS содержит список имён заданий, точнее - список имён PROVIDES, обычно совпадающих с именами заданий. Включение в DEPENDS имён пакетов, таких как foo-dev, не имеет смысла. Вместо этого следует указать foo, поскольку это приведёт к включению всех пакетов, составляющих foo, включая foo-dev.
- Задание, указывающее в DEPENDS другое задание, само по себе не добавляет зависимость между этими заданиями при работе. Однако, как указано в разделе Automatically Added Runtime Dependencies [1], зависимости при работе часто добавляются автоматически, поэтому для большинства заданий достаточно указать DEPENDS.
- DEPENDS зачастую требуется даже для заданий, устанавливающих ранее собранные компоненты. Например, если libfoo является ранее скомпилированной библиотекой, которая связана с libbar, тогда привязка к libfoo требует наличия libfoo и libbar в sysroot. Без зависимости от libbar в переменной DEPENDS задания, устанавливающего libfoo, другие задания не смогут ссылаться на libfoo.

Информация о зависимостях при работе приведена в описании переменной RDEPENDS, а также в разделах Tasks и Dependencies [4].

DEPLOY_DIR

Указывает базовую область, куда система сборки OE помещает образы, пакеты, SDK и другие выходные файлы, предназначенные для внешнего использования. По умолчанию этот каталог размещается в каталоге сборки как ${TMPDIR}/deploy. Структура каталога сборки описана в разделе 5.2. Каталог сборки build/, а содержимое каталога развёртывания рассмотрено в разделах Images, Package Feeds, и Application Development SDK [1].

DEPLOY_DIR_DEB

Указывает область, используемую системой сборки OE для размещения пакетов Debian, готовых для развёртывания вне системы сборки. Переменная применяется лишь при наличии в PACKAGE_CLASSES значения package_deb. Конфигурационный файл BitBake изначально определяет DEPLOY_DIR_DEB как каталог DEPLOY_DIR_DEB = "${DEPLOY_DIR}/deb"

Класс package_deb использует DEPLOY_DIR_DEB для обеспечения записи задачей do_package_write_deb пакетов Debian в корректный каталог (см. раздел Package Feeds [1]).

DEPLOY_DIR_IMAGE

Указывает область, используемую системой сборки OE для размещения и связанных с ними файлов , готовых для развёртывания вне системы сборки. Переменная является машинозависимой, поскольку содержит ${MACHINE}. По умолчанию этот каталог размещается в каталоге сборки как ${DEPLOY_DIR}/images/${MACHINE}/.

Информация о структуре каталога сборки приведена в разделе 5.2. Каталог сборки build/, а содержимое каталога развёртывания описано в разделах Images и Application Development SDK [1].

DEPLOY_DIR_IPK

Указывает область, используемую системой сборки OE для размещения пакетов IPK, готовых для развёртывания вне системы сборки. Переменная применяется лишь при наличии в PACKAGE_CLASSES значения package_ipk. Конфигурационный файл BitBake изначально определяет эту переменную как DEPLOY_DIR_IPK = "${DEPLOY_DIR}/ipk"

Класс package_ipk использует DEPLOY_DIR_IPK для обеспечения записи задачей do_package_write_ipk пакетов IPK в корректный каталог (см. раздел Package Feeds [1]).

DEPLOY_DIR_RPM

Указывает область, используемую системой сборки OE для размещения пакетов RPM, готовых для развёртывания вне системы сборки. Переменная применяется лишь при наличии в PACKAGE_CLASSES значения package_rpm. Конфигурационный файл BitBake изначально определяет эту переменную как DEPLOY_DIR_RPM = "${DEPLOY_DIR}/rpm"

Класс package_rpm использует DEPLOY_DIR_RPM для обеспечения записи задачей do_package_write_rpm t пакетов RPM в корректный каталог (см. раздел Package Feeds [1]).

DEPLOY_DIR_TAR

Указывает область, которую система сборки OE использует для размещения архивов, готовых для внешнего использования. Переменная применяется при наличии в переменной PACKAGE_CLASSES строки package_tar. Конфигурационный файл BitBake изначально определяет эту переменную как подкаталог в DEPLOY_DIR - DEPLOY_DIR_TAR = "${DEPLOY_DIR}/tar". Класс package_tar использует переменную DEPLOY_DIR_TAR для гарантии того, что задача do_package_write_tar будет записывать пакеты TAR в подходящий каталог. Дополнительная информация о процессе упаковки приведена в разделе Package Feeds [1].

DEPLOYDIR

При наследовании класса deploy эта переменная указывает временную рабочую область для разворачиваемых файлов, установленную в классе deploy как DEPLOYDIR = "${WORKDIR}/deploy-${PN}". Заданиям, наследующим класс deploy, следует копировать разворачиваемые файлы в DEPLOYDIR, а класс затем скопирует их в DEPLOY_DIR_IMAGE.

DESCRIPTION

Описание пакета, используемое менеджерами пакетов. Если значение DESCRIPTION не задано, автоматически принимается значение переменной SUMMARY.

DISTRO

Короткое имя для дистрибутива. Длинные имена задаются переменной DISTRO_NAME.

Переменная DISTRO соответствует файлу конфигурации (.conf) дистрибутива, корневое имя которого совпадает со значением переменной. Например, файл конфигурации дистрибутива Poky называется poky.conf и размещается в каталоге meta-poky/conf/distro дерева исходных кодов. В этом файле переменная задана как DISTRO = "poky".

Конфигурационные файлы дистрибутивов размещаются в каталоге conf/distro метаданных с конфигурацией дистрибутива. Значение DISTRO должно быть без пробелов и обычно указывается строчными буквами. Если переменная DISTRO пуста, используется заданное по умолчанию значение, которое указано в файле meta/conf/distro/defaultsetup.conf дерева исходных кодов.

DISTRO_CODENAME

Задаёт кодовое имя для собираемого дистрибутива.

DISTRO_EXTRA_RDEPENDS

Задаёт список определяемых дистрибутивом приложений, которые включаются во все образы. Эта переменная работает через packagegroup-base, поэтому на деле применяется только к наиболее функциональным образам, включающим группу packagegroup-base. Переменная указывается в файле конфигурации дистрибутива .conf.

DISTRO_EXTRA_RRECOMMENDS

Задаёт список специфических для дистрибутива приложений, добавляемых по все образы при наличии пакетов (пакеты могут отсутствовать или быть пустыми, например, модули ядра). Пакеты из списка устанавливаются автоматически, но их можно удалить.

DISTRO_FEATURES

Поддержка программ, которую нужно включить в дистрибутив, задаваемая в конфигурационном файле. В большинстве случаев наличие или отсутствие свойства в DISTRO_FEATURES передается в соответствующую опцию сценария настройки при выполнении задачи do_configure для заданий, которые могут поддерживать это свойство. Например, указание x11 в DISTRO_FEATURES приведёт к возможности включения поддержки X11 для каждой собираемой программы. Другими примерами служат поддержка Bluetooth и NFS. Более полный список включённых в YP свойств (возможностей) представлен в параграфе 12.2. Свойства дистрибутива.

DISTRO_FEATURES_BACKFILL

Свойства для добавления в DISTRO_FEATURES при отсутствии в DISTRO_FEATURES_BACKFILL_CONSIDERED. Переменная указывается в файле meta/conf/bitbake.conf и не предназначена для настройки пользователем. Лучше просто указать переменную для свойств, включаемых во все конфигурации дистрибутива (параграф 12.4. Отключение автоматически добавляемых свойств).

DISTRO_FEATURES_BACKFILL_CONSIDERED

Свойства из переменной DISTRO_FEATURES_BACKFILL, которые не следует включать (добавлять в DISTRO_FEATURES) при сборке (параграф 12.4. Отключение автоматически добавляемых свойств).

DISTRO_FEATURES_DEFAULT

Вспомогательная переменная со списком принятых по умолчанию свойств дистрибутива за исключением свойств, относящихся к библиотеке C (libc).

При создании пользовательского дистрибутива может оказаться полезной возможность многократного использования заданных по умолчанию опций DISTRO_FEATURES без необходимости переписывать весь набор. Это можно сделать, указав в конфигурационном файле дистрибутива DISTRO_FEATURES ?= "${DISTRO_FEATURES_DEFAULT} myfeature".

DISTRO_FEATURES_FILTER_NATIVE

Задаёт список свойств, которые при их наличии в DISTRO_FEATURES следует включать в сборку заданий native. Эти свойства дополняют список свойств, отфильтрованных переменой DISTRO_FEATURES_NATIVE.

DISTRO_FEATURES_FILTER_NATIVESDK

Задаёт список свойств, которые при их наличии в DISTRO_FEATURES следует включать в сборку заданий nativesdk. Эти свойства дополняют список свойств, отфильтрованных переменой DISTRO_FEATURES_NATIVESDK.

DISTRO_FEATURES_NATIVE

Задаёт список свойств, которые следует включать в DISTRO_FEATURES при сборке естественных заданий. Эти свойства дополняют список свойств, отфильтрованных переменой DISTRO_FEATURES_FILTER_NATIVE.

DISTRO_FEATURES_NATIVESDK

Задаёт список свойств, которые следует включать в DISTRO_FEATURES при сборке заданий nativesdk. Эти свойства дополняют список свойств, отфильтрованных переменой DISTRO_FEATURES_FILTER_NATIVESDK.

DISTRO_NAME

Длинное имя дистрибутива (короткое задаёт переменная DISTRO).

Переменная DISTRO_NAME соответствует файлу конфигурации (.conf) дистрибутива, корневое имя которого совпадает со значением переменной. Например, файл конфигурации дистрибутива Poky называется poky.conf и размещается в каталоге meta-poky/conf/distro дерева исходных кодов. В этом файле poky.conf переменная DISTRO_NAME задана как DISTRO_NAME = "Poky (Yocto Project Reference Distro)".

Конфигурационные файлы дистрибутивов размещаются в каталоге conf/distro метаданных с конфигурацией дистрибутива. Если переменная DISTRO_NAME пуста, используется заданное по умолчанию значение, которое указано в файле meta/conf/distro/defaultsetup.conf дерева исходных кодов.

DISTRO_VERSION

Версия дистрибутива.

DISTROOVERRIDES

Список разделенных двоеточиями переопределений для текущего дистрибутива. По умолчанию список включает значение DISTRO. Можно расширить DISTROOVERRIDES для добавления переопределений, применяемых к дистрибутиву. Базовым механизмом для DISTROOVERRIDES является включение в принятое по умолчанию значение OVERRIDES.

DL_DIR

Основной каталог загрузок, используемый процессом сборки для хранения загруженных файлов. По умолчанию DL_DIR принимает файлы, подходящие для «зеркалирования» всего, кроме репозиториев Git. Если нужны архивы репозиториев Git, следует использовать переменную BB_GENERATE_MIRROR_TARBALLS.

Каталог загрузки DL_DIR указывается в файле conf/local.conf. Каталог поддерживает себя и не нужно его трогать. По умолчанию загрузки выполняются в каталог сборки. Если нужно использовать другой каталог, следует убрать символ комментария в строке #DL_DIR ?= "${TOPDIR}/downloads".

При первой сборке система загружает множество архивов исходного кода из разных проектов, что может продолжаться достаточно долго. Архивы сохраняются в каталоге, указанном DL_DIR и система сборки ищет там архивы исходных кодов. При очистке или повторной сборке можно сохранить каталог загрузки для ускорения процесса.

Этот каталог можно использовать для разных сборок, выполняемых на хосте разработки. Дополнительная информация о загрузке архивов исходного кода при работе через прокси или межсетевой экран приведена в параграфе 15.12. Работа через межсетевой экран, а также на странице Working Behind a Network Proxy.

DOC_COMPRESS

При наследовании класса compress_doc эта переменная задаёт правила сжатия, используемые системой сборки OE для страниц man и info. По умолчанию применяется компрессия gz (gzip), но можно выбрать xz или bz2. Информация об использовании этой переменной и правилах приведена в комментариях в файле meta/classes/compress_doc.bbclass.

E

EFI_PROVIDER

При сборке загружаемых образов (типы hddimg, iso, wic.vmdk в IMAGE_FSTYPES), переменная указывает применяемый загрузчик EFI. По умолчанию используется grub-efi, но можно задать systemd-boot (см. параграфы 6.130. systemd-boot.bbclass и 6.53. image-live.bbclass).

ENABLE_BINARY_LOCALE_GENERATION

Переменная, задающая варианты locale для glibc, которые создаются при сборке (полезно для платформ с ОЗУ, не превышающим 64 Мбайт).

ERR_REPORT_DIR

При использовании с классом report-error задаёт путь, используемый для хранения отладочных файлов, созданных инструментом отчётов об ошибках, для централизованного хранения сведений об ошибках сборки. По умолчанию переменная имеет значение ${LOG_DIR}/error-report, которое можно изменить в файле local.conf с помощью строки ERR_REPORT_DIR = "path".

ERROR_QA

Задаёт проверки качества, об отказах которых системой сборки OE будут выдаваться сообщения об ошибках. Переменная задаётся в конфигурации дистрибутива, список проверок приведен в параграфе 6.56. insane.bbclass.

EXCLUDE_FROM_SHLIBS

Запускает распознаватель общих библиотек в системе сборки OE для исключения пакета целиком при сканировании общих библиотек. Функциональность распознавателя общих библиотек частично определяется внутренней функцией package_do_shlibs, которая является частью задачи do_package task. Следует знать, что распознаватель общих библиотек может неявно определять некоторые зависимости между пакетами.

Переменная EXCLUDE_FROM_SHLIBS похожа на PRIVATE_LIBS, которая исключает отдельные библиотеки, а не весь пакет. Переменная устанавливается для конкретного пакета в форме EXCLUDE_FROM_SHLIBS = "1".

EXCLUDE_FROM_WORLD

Указывает BitBake исключить задание из сборки world (т. е. bitbake world). При сборках world программа BitBake находит, анализирует и собирает все задания, найденные в каждом уровне, указанном в файле bblayers.conf. Для исключения задания следует установить в этой переменной значение 1 в задании.

Добавленные в EXCLUDE_FROM_WORLD могут всё-таки включаться в сборку world для выполнения зависимостей других заданий. Включение в EXCLUDE_FROM_WORLD лишь блокирует явное включение задания в сборку.

EXTENDPE

применяется с файлами и именами путей для создания префикса версии задания на основе значения PE. Если для задания установлено значение PE больше 0, переменная EXTENDPE принимает это значение (т. е. при PE = "1" EXTENDPE становится "1_"). Если переменная PE в задании не установлена (принято по умолчанию) или равна 0, EXTENDPE получает значение "". Дополнительные сведения приведены в описании переменной STAMP.

EXTENDPKGV

Полная спецификация версии пакета, как она указана в финальных пакетах, создаваемых заданием. Значение переменной обычно служит для исправления зависимостей при работе от конкретной версии другого пакета в том же задании.

```
RDEPENDS_${PN}-additional-module = "${PN} (= ${EXTENDPKGV})"
```

Отношения зависимости нужны для того, чтобы менеджер пакетов обновлял эти пакеты в режиме блокировки.

EXTERNAL_KERNEL_TOOLS

Установленная переменная показывает инструменты, не включённые в дерево исходных кодов. Инструменты, доступные в дереве, обычно являются предпочтительными, но установка этой переменной позволяет системе сборки OE отдать предпочтение внешним инструментам. Использование переменной рассмотрено в параграфе 6.66. kernel-yocto.bbclass.

EXTERNALSRC

При наследовании класса externalsrc эта переменная указывает каталог исходных кодов за пределами системы сборки OE. При установке этой переменной она задаёт значение переменной S, используемой системой OE для указания распакованного исходного кода. Информация о классе externalsrc приведена в параграфе 6.34. externalsrc.bbclass, а использование переменной описано в разделе Building Software from an External Source [2].

EXTERNALSRC_BUILD

При наследовании класса externalsrc эта переменная указывает каталог, в котором собирается исходный код задания, находящийся вне системы сборки OE. При установке этой переменной она задаёт значение переменной B, используемой системой OE для указания сборочного каталога. Информация о классе externalsrc приведена в параграфе 6.34. externalsrc.bbclass, а использование переменной описано в разделе Building Software from an External Source [2].

EXTRA_AUTORECONF

Для заданий, наследующих класс autotools, эта переменная позволяет задать дополнительные опции команды autoreconf, выполняемой задачей do_configure (по умолчанию --exclude=autopoint).

EXTRA_IMAGE_FEATURES

Список разделенных пробелами дополнительных возможностей для включения в образ. Обычно эта переменная указывается в файле local.conf сборочного каталога. Возможно её указание и в других файлах, но делать этого не следует. Для включения базовых возможностей образа служит переменная IMAGE_FEATURES.

Ниже приведены примеры некоторых дополнительных возможностей.

dbg-pkgs
Добавляет пакеты -dbg для отлаживания и профилирования.
debug-tweaks
Делает образ пригодны для отладки, например, позволяя вход в систему пользователя root без пароля и включая запись в системный журнал событий после установки. Дополнительная информация представлена в описаниях свойств allow-empty-password и post-install-logging в параграфе 12.3. Свойства образа.
dev-pkgs
Добавляет в образ пакеты разработки -dev.
read-only-rootfs
Создает образ, корневая система которого доступна только для чтения (см. раздел Creating a Read-Only Root Filesystem [2]).
tools-debug
Добавляет средства отладки, такие как gdb и strace.
tools-sdk
Добавляет служебные пакеты, такие как gcc, make, pkgconfig и т. п.
tools-testapps
Добавляет средства тестирования, такие как ts_print, aplay, arecord и т. п.

Полный список возможностей, включённых в выпуск YP, представлен в параграфе 12.3. Свойства образа.
Пример настройки свойств образа с помощью этой переменной приведен в разделе Customizing Images Using Custom IMAGE_FEATURES and EXTRA_IMAGE_FEATURES [2].

EXTRA_IMAGECMD
Задаёт опции команды создания образа, указанной в IMAGE_CMD. При установке этой переменной используется переопределение соответствующего типа образа, например, EXTRA_IMAGECMD_ext3 ?= "-i 4096".

EXTRA_IMAGEDEPENDS
Список заданий для сборки, которые не предоставляют пакетов, устанавливаемых в корневую файловую систему, но от которых зависит сборка образа. Переменная EXTRA_IMAGEDEPENDS позволяет указать эти задания для выполнения зависимостей. Типичные примером является требование иметь загрузчик в конфигурации машины.
Добавление пакетов в корневую систему управляется переменными *RDEPENDS и *RRECOMMENDS.

EXTRANATIVEPATH
Список каталогов в ${STAGING_BINDIR_NATIVE}, добавляемых в начало переменной окружения PATH. Например, для добавления в путь поиска каталогов "${STAGING_BINDIR_NATIVE}/foo:${STAGING_BINDIR_NATIVE}/bar:" можно указать EXTRANATIVEPATH = "foo bar".

EXTRA_OECMAKE
Опции CMake (см. параграф 6.17. cmake.bbclass).

EXTRA_OECONF
Опции сценария configure. Передача опций зависит от переменной PACKAGECONFIG_CONFARGS.

EXTRA_OEMAKE
Опции GNU make. Поскольку переменная EXTRA_OEMAKE по умолчанию пуста, её требуется установить в соответствии с нужными опциями GNU. Переменные PARALLEL_MAKE и PARALLEL_MAKEINST также используют EXTRA_OEMAKE для передачи требуемых флагов.

EXTRA_OESCONS
При наследовании класса scons эта переменная задаёт дополнительные параметры конфигурации, которые нужно передать команде scons.

EXTRA_USERS_PARAMS
При наследовании класса extrausers эта переменная представляет пользовательские и групповые операции на уровне образа. Это более глобальный метод предоставления конфигурации пользователей и групп по сравнению с использованием класса useradd, который связывает конфигурации пользователей и групп с конкретным заданиям.
Набор команд, которые можно представить с помощью EXTRA_USERS_PARAMS, приведен в описании класса extrausers. Эти команды отображаются на одноимённые команды Unix, как показано ниже.

```
# EXTRA_USERS_PARAMS = "\
# useradd -p '' tester; \
# groupadd developers; \
# userdel nobody; \
# groupdel -g video; \
# groupmod -g 1020 developers; \
# usermod -s /bin/sh tester; \
# "
```

F

FEATURE_PACKAGES
Определяет пакеты для включения в образ, когда соответствующее свойство включено в переменную IMAGE_FEATURES. При установке значения переменной FEATURE_PACKAGES следует указывать суффикс в виде имени свойства, например,

```
FEATURE_PACKAGES_widget = "package1 package2"
```

В этом примере при наличии свойства widget в IMAGE_FEATURES в образ будут добавлены пакеты package1 и package2. Пакеты, устанавливаемые свойствами через переменную FEATURE_PACKAGES, зачастую являются группами. Несмотря на схожесть имён, не следует путать переменную FEATURE_PACKAGES с группами пакетов, встречающимися в документации.

FEED_DEPLOYDIR_BASE_URI
Указывает базовый идентификатор (URL) сервера и местоположение на нем метаданных и пакетов, требуемых OPKG для поддержки управления пакетами IPK в процессе работы. Переменная задаётся в файле local.conf, например, FEED_DEPLOYDIR_BASE_URI = "http://192.168.7.1/BOARD-dir". Здесь предполагается, что пакеты обслуживаются по протоколу HTTP, а базы данных хранятся на сервере в каталоге BOARD-dir. В этом случае система сборки OE будет создавать набор конфигурационных файлов для целевой платформы, которые подойдут для указанного хранилища.

FILES
Список файлов и каталогов, помещённых в пакет. Пакеты, созданные заданием перечисляются в переменной PACKAGES. Для использования переменной FILES имя пакета переопределяется с указанием результирующего

пакета. Затем указывается список разделенных пробелами файлов или путей, указывающих файлы, включённые в результирующий пакет.

```
FILES_${PN} += "${bindir}/mydir1 ${bindir}/mydir2/myfile"
```

- При указании файлов и путей можно применять синтаксис Python glob.
- При указании путей в переменной FILES хорошим тоном является использование подходящих переменных. Например, следует указывать ${sysconfdir} вместо /etc или ${bindir} вместо /usr/bin. Список этих переменных приведен в начале файла meta/conf/bitbake.conf в дереве исходных кодов, где также представлены принятые по умолчанию значения различных переменных FILES_*.

Если некоторые из файлов в FILES являются редактируемыми и их не следует переписывать в процессе обновления пакета системой PMS, можно указать эти файлы, чтобы PMS не переписывала их (см. описание переменной CONFFILES).

FILES_SOLIBSDEV

Задаёт спецификацию файлов для сопоставления с SOLIBSDEV. Иными словами, FILES_SOLIBSDEV определяет полный путь к символьным ссылкам на общие библиотеки для целевой платформы. Переменная задаётся в файле bitbake.conf как FILES_SOLIBSDEV ?= "${base_libdir}/lib*${SOLIBSDEV} ${libdir}/lib*${SOLIBSDEV}".

FILESEXTRAPATHS

Расширяет путь поиска OE для файлов и исправлений при обработке файлов заданий и добавлений. Используются принятые по умолчанию каталоги, используемые BitBake, определяются переменной FILESPATH, которая может быть расширена с помощью FILESEXTRAPATHS. Лучше делать это с помощью переменной FILESEXTRAPATHS из файла .bbappend file и помещать пути в начало, как показано ниже.

```
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
```

В этом примере система сборки смотрит сначала в каталоге, имя которого совпадает с именем соответствующего файла добавления.

При использовании FILESEXTRAPATHS обязательно указывайте оператор незамедлительного расширения (:=). Это гарантирует, что BitBake будет вычислять THISDIR в момент обнаружения директивы, а не позднее, когда расширение может привести в каталог, где нет нужных файлов.

Включайте двоеточие в конце строки при её добавлении пути в начало. Это нужно для того, чтобы указать BitBake расширение пути путём вставки каталогов в начало пути поиска.

Ниже приведен ещё один вариант использования.

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
```

В этом примере система сборки расширяет переменную FILESPATH для включения файлов из того же каталога, что и соответствующий файл добавления. Выражение FILESEXTRAPATHS_prepend := "path_1:path_2:path_3:" добавляет 3 пути.

В другом примере показано, как можно расширить путь поиска и включить зависящее от MACHINE переопределение, полезное на уровне BSP - FILESEXTRAPATHS_prepend_intel-x86-common := "${THISDIR}/${PN}:"

Этот оператор присутствует в файле linux-yocto-dev.bbappend, который имеется в YP Source Repositories (раздел meta-intel/common/recipes-kernel/linux). Здесь в зависимости от машины изменяется специальное определение PACKAGE_ARCH для множества машин meta-intel.

Для уровней, поддерживающих один пакет BSP, переопределение может задаваться значением MACHINE.

Добавление пути в начало через файлы .bbappend позволяет расширять путь с помощью множества файлов дополнения, находящихся на разных уровнях но относящихся к одному заданию.

FILESOVERRIDES

Подмножество OVERRIDES, используемое системой сборки OE для создания FILESPATH. Переменная использует переопределение для автоматического расширения FILESPATH (пример работы приведен в описании FILESPATH, а более подробная информация о переопределениях - в разделе Conditional Syntax (Overrides) [4]). По умолчанию переменная задаётся в форме FILESOVERRIDES = "${TRANSLATED_TARGET_ARCH}:${MACHINEOVERRIDES}:${DISTROOVERRIDES}". Переменную не следует редактировать вручную. Значения совпадают с ожидаемыми переопределениями и используются системой сборки ожидаемым способом.

FILESPATH

Принятый по умолчанию набор каталогов, в котором система сборки OE ищет исправления и файлы. В процессе сборки BitBake просматривает каждый каталог из FILESPATH в указанном порядке для поиска файлов и исправления, заданных каждым URI file:// в операторах SRC_URI задания. Используемое по умолчанию значение переменной FILESPATH определено в файле base.bbclass каталога meta/classes в дереве исходных кодов.

```
FILESPATH = "${@base_set_filespath(["${FILE_DIRNAME}/${BP}", \
   "${FILE_DIRNAME}/${BPN}", "${FILE_DIRNAME}/files"], d)}"
```

Переменная FILESPATH преобразуется автоматически с использованием переопределений из переменной FILESOVERRIDES.

- Не следует редактировать переменную FILESPATH вручную. Если нужно задать поиск файлов в каталогах, отличающихся от принятых по умолчанию, расширьте FILESPATH с помощью FILESEXTRAPATHS.
- Принятые по умолчанию каталоги FILESPATH не отображаются на каталоги пользовательских уровней, где используются файлы добавления (.bbappend). Если нужно искать файлы или исправления, указанные в файлах добавления, следует расширить переменную FILESPATH с помощью FILESEXTRAPATHS.

Такое поведение системы поиска может быть полезно во многих случаях. В качестве примера может служить приведённая ниже структура каталогов с общей и машинозависимой конфигурацией.

```
files/defconfig
files/MACHINEA/defconfig
files/MACHINEB/defconfig
```

В этом примере оператор SRC_URI содержит file://defconfig. С учётом сказанного можно установить MACHINE = MACHINEA и заставить систему сборки использовать файлы из каталога files/MACHINEA, а установка MACHINE = MACHINEB приведёт к использованию файлов из files/MACHINEB. Для остальных машин система сборки будет использовать файлы из files/defconfig.

Дополнительная информации о внесении изменений в исходные коды приведена в разделах Patching [1] и Patching Code [2], а также в параграфе 7.1.19. do_patch.

FILESYSTEM_PERMS_TABLES

Позволяет задать свою таблицу настроек прав доступа к файлу как часть конфигурации процесса подготовки пакетов. Например, при потребности в согласованном наборе прав доступа для групп и пользователей в рамках проекта лучше устанавливать эти права в пакетах, но это возможно не всегда. По умолчанию система сборки OE

использует файл fs-perms.txt из каталога meta/files в дереве кодов. При создании своей таблицы разрешений следует поместить её на свой уровень или уровень дистрибутива.

Переменная определяется в файле conf/local.conf каталога сборки и указывает пользовательский файл fs-perms.txt. Можно задать несколько таблиц доступа. Пути к файлам должны быть определены в переменной BBPATH. Рекомендации по созданию таблицы прав даны в файле fs-perms.txt.

FONT_EXTRA_RDEPENDS

При наследовании класса fontcache задаёт зависимости при работе от шрифтовых пакетов. По умолчанию FONT_EXTRA_RDEPENDS имеет значение "fontconfig-utils".

FONT_PACKAGES

При наследовании класса fontcache указывает пакеты, содержащие файлы шрифтов, которые нужно кэшировать в Fontconfig. По умолчанию класс fontcache предполагает шрифты в основном задании пакета (${PN}). Эта переменная нужна для случаев, когда шрифты размещаются не в основном пакете.

FORCE_RO_REMOVE

Значение 1 форсирует удаление пакетов, указанных в ROOTFS_RO_UNNEEDED, при создании корневой файловой системы.

FULL_OPTIMIZATION

Опции, передаваемые в TARGET_CFLAGS и CFLAGS при компиляции оптимизированной системы (по умолчанию -O2 -pipe ${DEBUG_FLAGS}).

G

GCCPIE

Включает поддержку PIE[1] для компилятора GCC. Включение PIE в GCC осложняет организацию атак ROP[2]. По умолчанию в файле security_flags.inc указано GCCPIE ?= "--enable-default-pie".

GCCVERSION

Задаёт принятый по умолчанию компилятор GNU C (GCC). По умолчанию в файле meta/conf/distro/include/tcmode-default.inc задано GCCVERSION ?= "8.%". Можно указать другую версию в файле local.conf.

GDB

Сокращённая команда и аргументы для запуска отладчика GNU (gdb).

GITDIR

Каталог для хранения локальной копии репозитория Git при его клонировании.

GLIBC_GENERATE_LOCALES

Задаёт список языковых файлов GLIBC, если не нужны все такие файлы (занимает много времени). При удалении языка en_US.UTF-8 следует должным образом установить переменную IMAGE_LINGUAS. По умолчанию создаются файлы для всех языков. Значение GLIBC_GENERATE_LOCALES можно указать в файле local.conf как GLIBC_GENERATE_LOCALES = "en_GB.UTF-8 en_US.UTF-8".

GROUPADD_PARAM

При наследовании класса useradd эта переменная указывает, какие параметры для пакета следует передать команде groupadd, если нужно добавить группу при установке пакета. Например, для задания dbus указывается GROUPADD_PARAM_${PN} = "-r netdev"

Информация о стандартной команде Linux groupadd доступна по ссылке http://linux.die.net/man/8/groupadd.

GROUPMEMS_PARAM

При наследовании класса useradd эта переменная указывает, какие параметры для пакета следует передать команде groupmems, если нужно изменить состав группы при установке пакета.

Информация о стандартной команде Linux groupmems доступна по ссылке http://linux.die.net/man/8/groupmems.

GRUB_GFXSERIAL

Указывает режим работы загрузчика GRUB[3]. Установите для этой переменной значение 1 в файле local.conf или файле конфигурации дистрибутива для включения графического режима и консоли serial. Использование переменной рассмотрено в описании класса grub-efi.

GRUB_OPTS

Опции для добавления в конфигурацию загрузчика GRUB, разделённые точкой с запятой (;). Переменная не обязательна, её использование переменной рассмотрено в описании класса grub-efi.

GRUB_TIMEOUT

Задаёт время ожидания перед выбором заданной по умолчанию метки default LABEL загрузчике GRUB. Переменная не обязательна, её использование переменной рассмотрено в описании класса grub-efi.

GTKIMMODULES_PACKAGES

При наследовании класса gtk-immodules-cache эта переменная указывает пакеты, содержащие устанавливаемые модули ввода GTK+, когда они не находятся в основном пакете.

H

HOMEPAGE

Web-сайт, на котором можно найти дополнительную информацию о собираемой заданием программе.

HOST_ARCH

Имя целевой архитектуры, которое обычно совпадает с TARGET_ARCH. Система сборки OE поддерживает множество вариантов архитектуры, включая arm, i586, x86_64, powerpc, powerpc64, mips, mipsel.

HOST_CC_ARCH

Указывает зависящие от архитектуры флаги, передаваемые компилятору C. Принятая по умолчанию инициализация HOST_CC_ARCH зависит от цели сборки:

- TARGET_CC_ARCH при сборке для целевой платформы;
- BUILD_CC_ARCH при сборке для сборочного хоста (-native);
- BUILDSDK_CC_ARCH при сборке для SDK (nativesdk-).

HOST_OS

Задаёт имя целевой операционной системы, которое обычно совпадает с TARGET_OS. Для переменной можно установить значение linux в системах на основе glibc и linux-musl в системах на основе musl. Для платформ ARM/EABI возможны также значения linux-gnueabi и linux-musleabi.

[1]Position Independent Executables - независимые от расположения исполняемые файлы.
[2]Return Oriented Programming - ориентированное на возвращаемое значение программирование.
[3]GNU GRand Unified Bootloader.

HOST_PREFIX

Задаёт префикс для инструментов кросс-компиляции. HOST_PREFIX обычно совпадает с TARGET_PREFIX.

HOST_SYS

Задаёт систему, включая архитектуру и ОС, для которой выполняется сборка в контексте текущего задания. Система сборки OE автоматически устанавливает эту переменную на основе значений HOST_ARCH, HOST_VENDOR и HOST_OS. Например, для естественного задания на 32-битовой машине x86 с ОС Linux переменная будет иметь значение i686-linux, а для задания на платформе little-endian MIPS с ОС Linux - mipsel-linux. Менять переменную не нужно.

HOSTTOOLS

Список разделенных пробелами инструментов, которые можно вызывать на хосте сборки из сборочных задач. Использование этого фильтра помогает снизить вероятность нарушения работы сборочного хоста. Если заданный в переменной инструмент не найден, система сборки OE выдаёт ошибку и сборка не начинается.

HOSTTOOLS_NONFATAL

Список разделенных пробелами инструментов, которые можно вызывать на хосте сборки из сборочных задач. Использование этого фильтра помогает снизить вероятность нарушения работы сборочного хоста. В отличие от инструментов HOSTTOOLS система сборки OE не выдаёт ошибку при отсутствии указанного в HOSTTOOLS_NONFATAL. Таким образом, эта переменная указывает необязательные инструменты хоста.

HOST_VENDOR

Задаёт имя производителя целевого хоста. HOST_VENDOR обычно совпадает с TARGET_VENDOR.

I

ICECC_DISABLED

Управляет работой icecc (Icecream), рассмотренной в параграфе 6.49. icecc.bbclass. Установка в файле local.conf ICECC_DISABLED ??= "1" отключает функцию, ICECC_DISABLED = "" - включает.

ICECC_ENV_EXEC

Указывает пользовательский сценарий icecc-create-env, используется классом icecc и устанавливается в файле local.conf. Если сценарий не указан, система сборки OE использует принятый по умолчанию сценарий из задания icecc-create-env.bb, который является измененной версией сценария из состава icecc.

ICECC_PARALLEL_MAKE

Опции, передаваемые команде make в задаче do_compile для параллельной компиляции. Эта переменная обычно имеет форму -j x, где x указывает максимальное число параллельных потоков. Опция влияет на все машины в сети сборки, где работает демон iceccd. При поддержке машинами множества ядер определение максимального числа потоков может потребовать некоторых экспериментов для учёта скорости машин, задержек в сети, доступной памяти и текущей загрузки машин. Поэтому, в отличие от переменной PARALLEL_MAKE, для установки ICECC_PARALLEL_MAKE нет строгих правил.

При отсутствии ICECC_PARALLEL_MAKE система сборки не будет определять и задавать число используемых ядер, как это делается для PARALLEL_MAKE.

ICECC_PATH

Путь к исполняемому файлу icecc, задаваемый в файле local.conf. Если путь не задан, класс icecc пытается найти icecc с помощью команды which.

ICECC_USER_CLASS_BL

Указывает пользовательские классы, для который не следует использовать распределенную компиляцию Icecream. Переменная используется классом icecc и устанавливается в файле local.conf. Все указанные переменной классы будут компилироваться локально.

ICECC_USER_PACKAGE_BL

Указывает пользовательские задания, которые не нужно учитывать в распределенной компиляции Icecream. Переменная используется классом icecc и устанавливается в файле local.conf. Все указанные переменной пакеты будут компилироваться локально.

ICECC_USER_PACKAGE_WL

Указывает пользовательские задания с пустой переменной PARALLEL_MAKE, для которых нужно принудительно использовать параллельную удалённую компиляции с использованием системы распределенной компиляции Icecream. Переменная используется классом icecc и может указываться в файле local.conf.

IMAGE_BASENAME

Базовое имя выходных файлов образа, по умолчанию совпадающее с именем задания (${PN}).

IMAGE_BOOT_FILES

Список разделенных пробелами файлов, установленных в корневой раздел пр подготовке образа с использованием Wic и подключаемым модулем bootimg-partition. По умолчанию файлы устанавливаются с теми же именами, которые были у исходных файлов. Для изменения имён их следует отделять от исходных имён точкой с запятой (;). Исходные файлы должны размещаться в DEPLOY_DIR_IMAGE. Ниже приведены два примера.

```
IMAGE_BOOT_FILES = "u-boot.img uImage;kernel"
IMAGE_BOOT_FILES = "u-boot.${UBOOT_SUFFIX} ${KERNEL_IMAGETYPE}"
```

Кроме того, исходные файлы можно указать с помощью шаблона glob. В этом случае целевой файл должен иметь такое же имя, как и базовое имя в пути к исходному файлу. Для установки файлов в целевой системе их имена передаются после точки с запятой (;), как показано в приведённых ниже примерах.

```
IMAGE_BOOT_FILES = "bcm2835-bootfiles/*"
IMAGE_BOOT_FILES = "bcm2835-bootfiles/*;boot/"
```

Первый пример устанавливает все файлы из ${DEPLOY_DIR_IMAGE}/bcm2835-bootfiles в корневой раздел целевой системы, а второй устанавливает те же файлы в каталог boot внутри целевого раздела.

Информация о работе с Wic приведена в разделе Creating Partitioned Images Using Wic [2], а Глава 9. Справочник по OpenEmbedded Kickstart (.wks) содержит описание Wic.

IMAGE_CLASSES

Список классов, которые нужно наследовать всем образам. Обычно эта переменная служит для задания классов, регистрирующих разные типы образов, которые создает система сборки OE. По умолчанию используется IMAGE_CLASSES = image_types, но можно установить другое значение в файле local.conf или в конфигурационном файле дистрибутива (см. meta/classes/image_types.bbclass в каталоге исходных кодов).

IMAGE_CMD

Задаёт команду создания файла образа определённого типа, соответствующего значению IMAGE_FSTYPES (например, ext3, btrfs и т. п.). При установке переменной следует использовать переопределение для связанного типа, как показано ниже.

```
IMAGE_CMD_jffs2 = "mkfs.jffs2 --root=${IMAGE_ROOTFS} \
    --faketime --output=${DEPLOY_DIR_IMAGE}/${IMAGE_NAME}.rootfs.jffs2 \
    ${EXTRA_IMAGECMD}"
```

Обычно эту переменную не требуется устанавливать, если не добавляется поддержка нового типа образов. Примеры установки этой переменной можно найти в файле meta/classes/image_types.bbclass.

IMAGE_DEVICE_TABLES

Задаёт один или несколько файлов с пользовательскими таблицами устройств, которые передаются команде makedevs как часть создания образа. Эти файлы указывают базовые узлы устройства, которые следует создать в иерархии /dev внутри образа. Если переменная не указана, используется файл files/device_table-minimal.txt is used, найденный в пути BBPATH. Пример таблицы устройств имеется в файле meta/files/device_table-minimal.txt.

IMAGE_FEATURES

Основной список свойств (возможностей) для включения в образ. Обычно эта переменная указывается в задании для образа. Можно указать переменную в файле local.conf каталога сборки, но лучше так не поступать.

Для добавления возможностей за пределами задания для образа служит переменная EXTRA_IMAGE_FEATURES.

Список возможностей, включённых в выпуск YP, представлен в параграфе 12.3. Свойства образа. Пример настройки образа с помощью этой переменной приведен в разделе Customizing Images Using Custom IMAGE_FEATURES and EXTRA_IMAGE_FEATURES [2].

IMAGE_FSTYPES

Задаёт формат, применяемый системой сборки OE при создании корневой файловой системы. Например, для создания файлов систем, использующих форматы .ext3 и .tar.bz2 служит IMAGE_FSTYPES = "ext3 tar.bz2".

Полный список поддерживаемых форматов приведен в описании переменной IMAGE_TYPES.

- Если задание для образа использует строку inherit image и в нем установлена переменная IMAGE_FSTYPES, её значение должно быть установлено до строки inherit image.
- Способ обработки этой переменной в системе сборки OE не позволяет обновлять её содержимое с помощью _append или _prepend и требует использования оператора += для добавления опций в IMAGE_FSTYPES.

IMAGE_INSTALL

Используется в заданиях для указания пакетов, устанавливаемых в образ с помощью класса image. Переменную следует применять осторожно, чтобы не возникло проблем с порядком установки. Имеются вспомогательные классы (например, core-image), принимающие списки, используемые с IMAGE_FEATURES, и включающие элементы в автоматически создаваемые записи IMAGE_INSTALL в дополнение к имеющимся.

При использовании переменной лучше всего её задавать, как показано ниже.

```
IMAGE_INSTALL_append = " package-name"
```

Не забывайте указывать пробел перед именем пакета.

- При работе с образом core-image-minimal-initramfs не используйте переменную IMAGE_INSTALL для задания устанавливаемых пакетов. В этом случае нужна переменная PACKAGE_INSTALL, которая позволяет заданию initramfs использовать фиксированный набор приложений, на который не влияет переменная IMAGE_INSTALL. Информация о создании initramfs приведена в разделе Building an Initial RAM Filesystem (initramfs) Image [2].
- Использование переменной IMAGE_INSTALL с оператором BitBake += в файле /conf/local.conf file или иных файлах задания для образе не рекомендуется, поскольку применение этого оператора может приводить к проблемам с порядком установки. Класса core-image.bbclass устанавливает для IMAGE_INSTALL принятое по умолчанию значение с помощью оператора ?=, использование += для переменной IMAGE_INSTALL в файле conf/local.conf приводит к неожиданному поведению. Кроме того, та же операция из задания для образа может завершаться отказом в зависимости от ситуации. В обоих случаях поведение будет отличаться от ожидаемого пользователем для оператора +=.

IMAGE_LINGUAS

Задаёт список языков (locale) для включения в образ при создании корневой файловой системы. Система сборки OE автоматически разделяет языковые файлы по отдельным пакетам. Назначение переменной IMAGE_LINGUAS гарантирует включение языковых пакетов во все пакеты, выбранные для инсталляции в образ. Переменная задаётся в форме IMAGE_LINGUAS = "pt-br de-de". Пример показывает установку языковых файлов для бразильского португальского и немецкого языка для включённых в образ пакетов (т. е. *-locale-pt-br и *-locale-de-de, а для некоторых пакетов *-locale-pt и *-locale-de, поскольку часть пакетов обеспечивает файлы языков без учёта страны). Генерация языковых файлов GLIBC рассмотрена в описании переменной GLIBC_GENERATE_LOCALES.

IMAGE_MANIFEST

Файл манифеста для образа, перечисляющий все установленные пакеты (по одной строке на пакет) в форме packagename packagearch version. Класс image определяет файл манифеста как IMAGE_MANIFEST = "${DEPLOY_DIR_IMAGE}/${IMAGE_NAME}.rootfs.manifest". Местоположение выводится из переменных DEPLOY_DIR_IMAGE и IMAGE_NAME. Создание образов описано в разделе Image Generation [1].

IMAGE_NAME

Имя выходных файлов образа без расширений, задаваемое в форме IMAGE_NAME = "${IMAGE_BASENAME}-${MACHINE}-${DATETIME}".

IMAGE_OVERHEAD_FACTOR

Определяет коэффициент, который система сборки использует для начального размера образа в случаях, когда используемое в образе пространство (возврат команды du), умноженное на этот коэффициент, превышает сумму IMAGE_ROOTFS_SIZE и IMAGE_ROOTFS_EXTRA_SPACE. Результатом использования коэффициента для начального размера является увеличение свободного пространства в образе. По умолчанию система сборки использует для этой переменной значение 1,3, что приводит к добавлению 30% свободного пространства в финальный образ. Следует учитывать, что применяемые после инсталляции сценарии и система управления пакетами используют пространство из этой области. Поэтому коэффициент не задаёт точно свободное пространство, получаемое в результате. Определение общего размера образа зависит от переменной IMAGE_ROOTFS_SIZE.

Установленное по умолчанию свободное пространство в 30% обеспечивает достаточно места для загрузки и использования пост-установочных сценариев для большинства случаев. Если это значение не подходит, можно изменить размер свободного пространства. Например, для резервирования 50% свободно места можно задать

IMAGE_OVERHEAD_FACTOR = "1.5". Можно также обеспечить дополнительное свободное пространство на диске с помощью переменной IMAGE_ROOTFS_EXTRA_SPACE.

IMAGE_PKGTYPE

Определяет тип пакета (DEB, RPM, IPK или TAR), используемый системой сборки OE. Переменная определяется классом package_deb, package_rpm, package_ipk или package_tar. Класс package_tar больше не применяется и не поддерживается, поэтому не следует его использовать. Классы populate_sdk_* и image используют переменную IMAGE_PKGTYPE для упаковки образов и SDK.

Переменную IMAGE_PKGTYPE не следует устанавливать вручную, поскольку она задаётся опосредованно через подходящий класс package_* с использованием переменной PACKAGE_CLASSES. Система сборки OE применяет первый тип пакета (DEB, RPM или IPK) из этой переменной.

Архивы .tar никогда не применяются в качестве замены форматов DEB, RPM и IPK для образов и SDK.

IMAGE_POSTPROCESS_COMMAND

Задаёт список функций, однократно вызываемых системой сборки OE при создании окончательных выходных файлов. Функции разделяются точкой с запятой, например, IMAGE_POSTPROCESS_COMMAND += "function; ... "

Если нужно передать путь к корневой файловой системе при вызове функции, можно использовать переменную ${IMAGE_ROOTFS}, указывающую каталог, который станет корневой файловой системой образа.

IMAGE_PREPROCESS_COMMAND

Задаёт список функций, вызываемых системой сборки OE перед созданием окончательных выходных файлов. Функции разделяются точкой с запятой, например, IMAGE_PREPROCESS_COMMAND += "function; ... "

Если нужно передать путь к корневой файловой системе при вызове функции, можно использовать переменную ${IMAGE_ROOTFS}, указывающую каталог, который станет корневой файловой системой образа.

IMAGE_ROOTFS

Местоположение корневой файловой системы в процессе её создания (задача do_rootfs). Значение этой переменной не следует менять.

IMAGE_ROOTFS_ALIGNMENT

Задаёт выравнивание для выходного файла образа в килобайтах. Если размер образа не кратен этому значению, он округляется вверх до кратного значения. По умолчанию установлено значение 1 (см. IMAGE_ROOTFS_SIZE).

IMAGE_ROOTFS_EXTRA_SPACE

Задаёт дополнительное свободное пространства в создаваемом образе (в килобайтах). По умолчанию дополнительное пространство не выделяется (0). Заданное переменной пространство добавляется в образ после того, как система сборки определит размер образа в соответствии с переменной IMAGE_ROOTFS_SIZE.

Переменная полезна для резервирования свободного пространства в файловой системе после загрузки образа. Например, для обеспечения свободных 5 Гбайт можно указать IMAGE_ROOTFS_EXTRA_SPACE = "5242880". В частности, Yocto Project Build Appliance задаёт 40 Гбайт дополнительного места с помощью строки IMAGE_ROOTFS_EXTRA_SPACE = "41943040".

IMAGE_ROOTFS_SIZE

Определяет размер создаваемого образа к килобайтах. Система сборки OE определяет окончательный размер создаваемого образа с учётом начального диска, используемого для образа, запрошенного размера образа и запрошенного свободного пространства после загрузки образа. Псевдокод для определения размера можно представить в виде

```
if (image-du * overhead) < rootfs-size:
    internal-rootfs-size = rootfs-size + xspace
else:
    internal-rootfs-size = (image-du * overhead) + xspace
```

где

image-du	- значение, возвращаемое командой du для образа;
overhead	- IMAGE_OVERHEAD_FACTOR;
rootfs-size	- IMAGE_ROOTFS_SIZE;
internal-rootfs-size	- начальный размер корневой файловой системы до каких-либо изменений;
xspace	- IMAGE_ROOTFS_EXTRA_SPACE.

IMAGE_TYPEDEP

Указывает зависимость одного типа образа от другого, например, для образа из класса image-live IMAGE_TYPEDEP_live = "ext3". В этом примере переменная обеспечивает включение live в переменную IMAGE_FSTYPES и система сборки OE создает сначала образ ext3, поскольку одной из частей образа live является раздел формата ext3 с корневой файловой системой.

IMAGE_TYPES

Задаёт полный список поддерживаемых по умолчанию типов образов: btrfs, cpio, cpio.gz, cpio.lz4, cpio.lzma, cpio.xz, cramfs, elf, ext2, ext2.bz2, ext2.gz, ext2.lzma, ext3, ext3.gz, ext4, ext4.gz, hdddirect, hddimg, iso, jffs2, jffs2.sum, multiubi, squashfs, squashfs-lzo, squashfs-xz, tar, tar.bz2, tar.gz, tar.lz4, tar.xz, ubi, ubifs, wic, wic.bz2, wic.gz, wic.lzma.

Полная информация об этих типах приведена в файлах meta/classes/image_types*.bbclass дерева исходных кодов.

INC_PR

Помогает указать выпуск для заданий, использующих общий включаемый файл. Можно считать эту переменную частью варианта задания, устанавливаемой из включаемого файла.

Предположим, что имеется набор заданий, используемых в нескольких проектах, и для каждого из этих заданий задан выпуск (в переменной PR). В этом случае при изменении номера выпуска задания нужно найти все эти задания и убедиться в корректности обновления номера выпуска. Это может осложняться использованием заданий в разных местах, когда обновляются задания, обеспечивающие общую функциональность. Более эффективным способом в таких случаях является установка переменной INC_PR во включаемых файлах, совместно используемых заданиями, с последующим преобразованием переменной в заданиях для определения выпуска.

Ниже приведен пример использования переменной INC_PR с учётом общего включаемого файла, определяющего эту переменную. После её указания во включаемом файле переменную можно использовать для установки значений PR в каждом задании. При установке PR для задания можно более точно указать выпуск, добавляя значение в конце переменной INC_PR, как показано ниже.

```
recipes-graphics/xorg-font/xorg-font-common.inc:INC_PR = "r2"
recipes-graphics/xorg-font/encodings_1.0.4.bb:PR = "${INC_PR}.1"
recipes-graphics/xorg-font/font-util_1.3.0.bb:PR = "${INC_PR}.0"
recipes-graphics/xorg-font/font-alias_1.0.3.bb:PR = "${INC_PR}.3"
```

Первая строка в этом примере показывает базовый номер выпуска, используемый всеми заданиями, которые применяют включаемый файл. Остальные строки уточняют выпуски для отдельных заданий, устанавливая PR.

INCOMPATIBLE_LICENSE

Задаёт список разделенных пробелами имён лицензий (как в переменной LICENSE), которые следует исключить при сборке. Значения, не содержащие замены исключённых лицензий, собираться не будут, пакеты, с несовместимыми лицензиями будут удалены. Эта функциональность регулярно проверяется лишь для INCOMPATIBLE_LICENSE = "GPL-3.0 LGPL-3.0 AGPL-3.0". Ваши установки могут отличаться, но все равно может потребоваться указание несовместимых лицензий или замен для создания образа.

INHERIT

Задаёт глобальное наследование указанных классов. Анонимные функции классов не выполняются для базовой конфигурации и в каждом отдельном задании. Система сборки OE игнорирует изменение переменной INHERIT в отдельных заданиях. Дополнительная информация приведена в разделе INHERIT Configuration Directive [4].

INHERIT_DISTRO

Указывает классы, которые будут наследоваться на уровне дистрибутива. Обычно менять эту переменную не требуется. Используемое по умолчанию значение переменной задано в файле meta/conf/distro/defaultsetup.conf как INHERIT_DISTRO ?= "debian devshell sstate license".

INHIBIT_DEFAULT_DEPS

Предотвращает добавление заданных по умолчанию зависимостей (компилятор C и стандартная библиотека libc) в DEPENDS. Переменная обычно применяется в заданиях, которым не нужен компилятор C. Установка значения 1 предотвращает добавление принятых по умолчанию зависимостей.

INHIBIT_PACKAGE_DEBUG_SPLIT

Установка значения 1 в этой переменной предотвращает отделение отладочной информации при подготовке пакетов. По умолчанию система сборки отделяет отладочную информацию в процессе выполнения задачи do_package (см. описание PACKAGE_DEBUG_SPLIT_STYLE).

INHIBIT_PACKAGE_STRIP

Установка значения 1 отменяет удаление системой сборки символов из собранных пакетов и предотвращает включение исходных файлов в пакеты -dbg. По умолчанию система сборки OE вырезает символы из двоичных файлов и помещает отладочные символы в ${PN}-dbg. Поэтому не следует устанавливать переменную INHIBIT_PACKAGE_STRIP, если планируется отладка.

INHIBIT_SYSROOT_STRIP

Установка значения 1 отменяет удаление системой сборки символов из двоичных файлов sysroot, выполняемое по умолчанию. Для использования переменной в задании нужно включить класс staging, который применяет функцию sys_strip() для проверки значения переменной и выполнения нужных действий.

Переменная применяется редко. Примером может служить создание прошивки для пустой платформы (bare-metal) с использованием внешних инструментов GCC. Более того, даже при вырезании информации из двоичных файлов инструментария остаются другие файлы, которые нужны для сборки и не вырезаются.

INITRAMFS_FSTYPES

Задаёт формат выходного образа initramfs, используемого при загрузке. Поддерживаемые форматы совпадают с возможными форматами IMAGE_FSTYPES. По умолчанию в файле meta/conf/bitbake.conf для этой переменной задаётся значение cpio.gz. Механизм initramfs в Linux в отличие от initrd предполагает возможность сжатия образа.

INITRAMFS_IMAGE

Задаёт имя задания (PROVIDES) для образа, используемое при сборке образа initramfs. Иными словами, переменная вызывает дополнительное задание для сборки как зависимость от используемого для корневой файловой системы задания (например, core-image-sato). Представленное задание для образа initramfs должно устанавливать в IMAGE_FSTYPES значение INITRAMFS_FSTYPES.

Образ обеспечивает временную корневую файловую систему, используемую для начальной инициализации (например, загрузки модулей, требуемых для монтирования реальной корневой файловой системы). В файле meta/recipes-core/images/core-image-minimal-initramfs.bb дерева исходных кодов приведен пример задания initramfs. Для выбора этого образца в качестве задания для образа initramfs нужно установить в переменной INITRAMFS_IMAGE значение core-image-minimal-initramfs. Использование переменной INITRAMFS_IMAGE рассмотрено в файле meta-poky/conf/local.conf.sample.extended, а также в описании классов image и kernel.

Если значение переменной INITRAMFS_IMAGE пусто (принято по умолчанию), образ initramfs не создаётся.

Переменная INITRAMFS_IMAGE_BUNDLE позволяет связать генерируемый образ с образом ядра. Сведения о создании образов initramfs приведены в разделе Building an Initial RAM Filesystem (initramfs) Image [2].

INITRAMFS_IMAGE_BUNDLE

Управляет дополнительным проходом do_bundle_initramfs при компиляции ядра для образа, заданного переменной INITRAMFS_IMAGE, с целью создания одного двоичного файла с образами ядра и initramfs. Для этого используется свойство ядра CONFIG_INITRAMFS_SOURCE.

Дополнительный проход компиляции для связывания initramfs предотвращает зацикливание зависимостей между заданиями для ядра и initramfs, когда initramfs включает модули ядра. В таких случаях задание initramfs зависит от ядра для получения модулей, а ядро зависит от initramfs, поскольку initramfs встраивается в образ ядра.

Объединённый двоичный файл размещается в каталоге tmp/deploy внутри каталога сборки.

Установка INITRAMFS_IMAGE_BUNDLE = "1" в конфигурационном файле системы сборки OE ведёт к генерации образа ядра с initramfs, указанной в INITRAMFS_IMAGE. По умолчанию класс kernel устанавливает пустое значение INITRAMFS_IMAGE_BUNDLE ?= "".

Переменная должна указываться в конфигурационном файле, а не в файле задания. Дополнительная информация приведена в файле local.conf.sample.extended, а создание initramfs описано в разделе Building an Initial RAM Filesystem (initramfs) Image [2].

INITRAMFS_LINK_NAME

Имя ссылки для исходного образа файловой системы initramfs. Переменная задаётся в файле meta/classes/kernel-artifact-names.bbclass как INITRAMFS_LINK_NAME ?= "initramfs-${KERNEL_ARTIFACT_LINK_NAME}". Этот же файл задаёт переменную KERNEL_ARTIFACT_LINK_NAME ?= "${MACHINE}".

INITRAMFS_NAME

Базовое имя исходного образа файловой системы initramfs. Переменная задаётся в файле meta/classes/kernel-artifact-names.bbclass как INITRAMFS_NAME ?= "initramfs-${KERNEL_ARTIFACT_NAME}". Этот же файл задаёт KERNEL_ARTIFACT_NAME ?= "${PKGE}-${PKGV}-${PKGR}-${MACHINE}${IMAGE_VERSION_SUFFIX}".

INITRD

Задаёт список файловых систем для конкатенации и использования в качестве исходного RAM-диска (initrd). Переменная не обязательна и применяется классом image-live.

INITRD_IMAGE

При сборке загружаемых образов live (IMAGE_FSTYPES содержит live) переменная задаёт задание для образа, которое следует собрать для создания образа исходного RAM-диска. По умолчанию задано значение core-image-minimal-initramfs (см. параграф 6.53. image-live.bbclass).

INITSCRIPT_NAME

Имя файла со сценарием инициализации, установленное в ${sysconfdir}/init.d. Переменная применяется в заданиях, использующих класс update-rc.d.bbclass, и является обязательной.

INITSCRIPT_PACKAGES

Список пакетов, содержащих сценарии инициализации. При задании нескольких пакетов нужно добавлять имя пакета в конце к другим INITSCRIPT_* как переопределение. Переменная применяется в заданиях, использующих класс update-rc.d.bbclass. Переменная не обязательна и по умолчанию имеет значение переменной PN.

INITSCRIPT_PARAMS

Задаёт опции для передачи update-rc.d. Например, INITSCRIPT_PARAMS = "start 99 5 2 . stop 20 0 1 6 ." указывает сценарий с уровнем запуска (runlevel) 99, запускаемый на уровнях 2 и 5, останавливаемый на уровнях 0, 1 и 6.

По умолчанию переменная имеет значение defaults, устанавливаемое классом update-rc.d. Значение переменной передается через команду update-rc.d. Дополнительная информация доступна по ссылке.

INSANE_SKIP

Задаёт проверки QA, пропускаемые для конкретного пакета в задании. Например, для пропуска проверки символьных ссылок для файлов .so в основном пакете задания в это задание добавляется приведённая ниже переменная. Следует указывать переопределенное имя задания (в примере ${PN}).

```
INSANE_SKIP_${PN} += "dev-so"
```

Список проверок QA, которые можно указать в этой переменной, приведен в параграфе 6.56. insane.bbclass.

INSTALL_TIMEZONE_FILE

По умолчанию задание tzdata включает файл /etc/timezone. Установка INSTALL_TIMEZONE_FILE = 0 на уровне конфигурации отменяет это.

IPK_FEED_URIS

При использовании менеджера IPK и включённом на целевой платформе управлении пакетами эта переменная может служить для установки opkg в целевом образе с целью указания источника пакетов на означенном сервере. После организации источника можно устанавливать и обновлять пакеты при работе платформы.

K

KARCH

Указывает архитектуру ядра, применяемую для сборки конфигурации - powerpc, i386, x86_64, arm, qemu, mips. Переменная KARCH определяется в описаниях BSP [3].

KBRANCH

Регулярное выражение, применяемое процессом сборки для явного указания ветви ядра, которая проверяется, изменяется (patch) и настраивается в процессе сборки. Нужно задать эту переменную для точного выбора ветви ядра, используемой в процессе сборки.

Значение этой переменной устанавливается в наборе заданий и файле добавления для ядра. Например, для ядра linux-yocto_4.12 файлом задания является meta/recipes-kernel/linux/linux-yocto_4.12.bb. Переменная KBRANCH в этом файле задаётся в форме

```
KBRANCH ?= "standard/base"
```

Переменная применяется также из файла добавления для ядра, чтобы указать конкретную ветвь ядра для целевой платы или оборудования. В предыдущем примере файл добавления для ядра (linux-yocto_4.12.bbappend) размещается на уровне BSP для данной машины. Например, файл добавления для Beaglebone, EdgeRouter и базовых версий 32- и 64-битовых машин IA (meta-yocto-bsp) называется meta-yocto-bsp/recipes-kernel/linux/linux-yocto_4.12.bbappend. Ниже приведены операторы и этого файла.

```
KBRANCH_genericx86  = "standard/base"
KBRANCH_genericx86-64  = "standard/base"
KBRANCH_edgerouter = "standard/edgerouter"
KBRANCH_beaglebone = "standard/beaglebone"
KBRANCH_mpc8315e-rdb = "standard/fsl-mpc8315e-rdb"
```

Операторы KBRANCH указывают ветвь ядра, которая будет использоваться при сборке для каждого BSP.

KBUILD_DEFCONFIG

При использовании с классом kernel-yocto задаёт размещённый в дереве исходных кодов ядра конфигурационный файл для использования при сборке ядра. Обычно при использовании файла defconfig для настройки конфигурации ядра при сборке файл помещается в каталог вашего уровня как и файлы исправлений, а также файлы с фрагментами конфигурации. Однако если вы хотите использовать файл defconfig как часть дерева ядра, можно воспользоваться переменной KBUILD_DEFCONFIG и добавить переменную KMACHINE для указания файла defconfig.

Для использования переменной следует указать её в файле добавления для задания ядра в виде KBUILD_DEFCONFIG_KMACHINE ?= defconfig_file. Например, для сборки raspberrypi2 KMACHINE с файлом defconfig, названным bcm2709_defconfig, служит KBUILD_DEFCONFIG_raspberrypi2 = "bcm2709_defconfig".

Как вариант, можно указать в файле добавления KBUILD_DEFCONFIG_pn-linux-yocto ?= defconfig_file.

Информация об использовании KBUILD_DEFCONFIG приведена в разделе Using an "In-Tree" defconfig File [3].

KERNEL_ALT_IMAGETYPE

Задаёт другой тип образа ядра в дополнение к типу, указанному переменной KERNEL_IMAGETYPE.

KERNEL_ARTIFACT_NAME

Указывает имя для всех элементов сборки (artifact). Значение переменной задаётся в файле meta/classes/kernel-artifact-names.bbclass как KERNEL_ARTIFACT_NAME ?= "${PKGE}-${PKGV}-${PKGR}-${MACHINE}${IMAGE_VERSION_SUFFIX}". Дополнительная информация приведена в описаниях переменных PKGE, PKGV, PKGR и MACHINE. Для переменной IMAGE_VERSION_SUFFIX устанавливается значение DATETIME.

KERNEL_CLASSES

Список классов, определяющих типы образов ядра, которые должен наследовать класс kernel. Обычно эта переменная добавляется для включения расширенных типов образов. Примером является класс kernel-fitimage,

включающий поддержку fitImage и заданный в файле meta/classes/kernel-fitimage.bbclass. Можно зарегистрировать свои типы образов ядра с помощью этой переменной.

KERNEL_DEVICETREE

Задаёт имя создаваемого файла с деревом устройств ядра Linux (.dtb). Имеется устаревший способ указания полного пути к дереву устройств, однако предпочтительней использовать файл .dtb. Для использования переменной нужно указать включаемые файлы в задании для ядра в форме require recipes-kernel/linux/linux-dtb.inc или require recipes-kernel/linux/linux-yocto.inc.

KERNEL_DTB_LINK_NAME

Имя ссылки на двоичное дерево устройств (DTB[1]) в файле meta/classes/kernel-artifact-names.bbclass, указанное как KERNEL_DTB_LINK_NAME ?= "${KERNEL_ARTIFACT_LINK_NAME}". Переменная KERNEL_ARTIFACT_LINK_NAME в том же файле имеет форму KERNEL_ARTIFACT_LINK_NAME ?= "${MACHINE}".

KERNEL_DTB_NAME

Базовое имя DTB, задаваемое в файле meta/classes/kernel-artifact-names.bbclass как KERNEL_DTB_NAME ?= "${KERNEL_ARTIFACT_NAME}". Переменная KERNEL_ARTIFACT_NAME в том же файле имеет форму KERNEL_ARTIFACT_NAME ?= "${PKGE}-${PKGV}-${PKGR}-${MACHINE}${IMAGE_VERSION_SUFFIX}".

KERNEL_EXTRA_ARGS

Задаёт дополнительные аргументы команды make, которые система сборки OE использует при компиляции ядра.

KERNEL_FEATURES

Включает дополнительные метаданные ядра. В системе сборки OE принятые по умолчанию метаданные BSP обеспечиваются переменными KMACHINE и KBRANCH. Можно использовать переменную KERNEL_FEATURES из задания или файла добавления для ядра, чтобы добавить метаданные для всех или определённых BSP.

Добавляемые через эту переменную метаданные включают фрагменты конфигурации и описания функций, которые обычно включают исправления (patch), а также фрагменты конфигурации. Переменная KERNEL_FEATURES обычно переопределяется для конкретной машины. Таким способом обеспечивается проверяемый, но не обязательный набор функций и конфигурационных параметров ядра.

Например, можно в задание linux-yocto-rt_4.добавить функции netfilter и taskstats для всех BSP, а также конфигурации virtio для всех машин QEMU. Два последних оператора добавляют конкретные конфигурации к целевым типам машин.

```
KERNEL_EXTRA_FEATURES ?= "features/netfilter/netfilter.scc features/taskstats/taskstats.scc"
KERNEL_FEATURES_append = " ${KERNEL_EXTRA_FEATURES}"
KERNEL_FEATURES_append_qemuall = " cfg/virtio.scc"
KERNEL_FEATURES_append_qemux86 = " cfg/sound.scc cfg/paravirt_kvm.scc"
KERNEL_FEATURES_append_qemux86-64 = " cfg/sound.scc"
```

KERNEL_FIT_LINK_NAME

Имя ссылки на плоское дерево образа ядра (FIT), задаваемое в файле meta/classes/kernel-artifact-names.bbclass как KERNEL_FIT_LINK_NAME ?= "${KERNEL_ARTIFACT_LINK_NAME}". В том же файле указывается переменная KERNEL_ARTIFACT_LINK_NAME ?= "${MACHINE}".

KERNEL_FIT_NAME

Базовое имя плоского дерева образа ядра (FIT), задаваемое в файле meta/classes/kernel-artifact-names.bbclass как KERNEL_FIT_NAME ?= "${KERNEL_ARTIFACT_NAME}". В этом же файле устанавливается переменная KERNEL_ARTIFACT_NAME ?= "${PKGE}-${PKGV}-${PKGR}-${MACHINE}${IMAGE_VERSION_SUFFIX}".

KERNEL_IMAGE_LINK_NAME

Имя ссылки для образа ядра, устанавливаемое в файле meta/classes/kernel-artifact-names.bbclass как KERNEL_IMAGE_LINK_NAME ?= "${KERNEL_ARTIFACT_LINK_NAME}". В том же файле файле устанавливается переменная KERNEL_ARTIFACT_LINK_NAME ?= "${MACHINE}".

KERNEL_IMAGE_MAXSIZE

Задаёт максимальный размер образа ядра в килобайтах. При установке переменной размер ядра сравнивается с её значением при выполнении задачи do_sizecheck с возвратом ошибки в случае превышения заданного размера. Переменная полезна для систем с ограниченным пространством для хранения образа ядра. По умолчанию переменная не устанавливается и размер ядра не ограничен.

KERNEL_IMAGE_NAME

Базовое имя образа ядра. Переменная задаётся файлом meta/classes/kernel-artifact-names.bbclass в форме KERNEL_IMAGE_NAME ?= "${KERNEL_ARTIFACT_NAME}". Значение переменной KERNEL_ARTIFACT_NAME в том же файле имеет форму KERNEL_ARTIFACT_NAME ?= "${PKGE}-${PKGV}-${PKGR}-${MACHINE}${IMAGE_VERSION_SUFFIX}".

KERNEL_IMAGETYPE

Тип ядра, собираемого для устройства, который обычно задаётся конфигурационными файлами для машины. По умолчанию применяется zImage. Эта переменная используется при сборке ядра и передается команде make как цель сборки. Для сборки дополнительного типа образа используется переменная KERNEL_ALT_IMAGETYPE.

KERNEL_MODULE_AUTOLOAD

Указывает модули ядра, которые нужно автоматически загружать при загрузке системы. Переменную можно использовать в любом месте, доступном заданию для ядра или внешнего (out-of-tree) модуля (например, в файле конфигурации машины или дистрибутива, файле дополнения к заданию или самом задании).

```
KERNEL_MODULE_AUTOLOAD += "module_name1 module_name2 module_name3"
```

Включение KERNEL_MODULE_AUTOLOAD заставляет систему сборки OE указать список автоматически загружаемых модулей в файле /etc/modules-load.d/modname.conf. Модули указываются по одному в строке, например, KERNEL_MODULE_AUTOLOAD += "module_name".

Информация о заполнении файла modname.conf в синтаксисом modprobe.d приведена в описании переменной KERNEL_MODULE_PROBECONF.

KERNEL_MODULE_PROBECONF

Задаёт список модулей, для которых система сборки OE предполагает найти значения module_conf_modname, задающие конфигурацию каждого модуля. Информация о настройке модулей приведена в описании переменной module_conf.

KERNEL_PATH

Указывает местоположение исходных кодов ядра. Переменная получает значение STAGING_KERNEL_DIR в классе module. Работа с переменной описана в разделе Incorporating Out-of-Tree Modules [3].

[1]Device tree binary.

Для максимальной совместимости с внешними (out-of-tree) драйверами при сборке модулей система OE распознает и применяет переменную KERNEL_SRC, которая идентична KERNEL_PATH . Обе переменные используются внешними файлами Makefile для указания каталога с исходными кодами ядра.

KERNEL_SRC

Указывает местоположение исходных кодов ядра. Переменная получает значение STAGING_KERNEL_DIR в классе module. Работа с переменной описана в разделе Incorporating Out-of-Tree Modules [3].

Для максимальной совместимости с внешними (out-of-tree) драйверами при сборке модулей система OE распознает и применяет переменную KERNEL_PATH, которая идентична KERNEL_SRC. Обе переменные используются внешними файлами Makefile для указания каталога с исходными кодами ядра.

KERNEL_VERSION

Указывает версию ядра, извлечённую из файла version.h или utsrelease.h в дереве кода ядра. Установка этой переменной не оказывает влияния, пока не будет настроена конфигурация ядра, поэтому попытка обращения к ней до настройки конфигурации не приведёт к желаемому результату.

KERNELDEPMODDEPEND

Указывает, нужны ли данные, упомянутые в переменной PKGDATA_DIR. KERNELDEPMODDEPEND контролирует не наличие этих данных, а их использование. Если данные не нужны, укажите KERNELDEPMODDEPEND в задании initramfs, это позволит предотвратить возможное зацикливание зависимостей.

KFEATURE_DESCRIPTION

Даёт краткое описание фрагмента конфигурации. Эта переменная применяется в файлах .scc, описывающих файлы с фрагментами конфигурации. Например, в файле в файле smp.scc эта переменная включает опции SMP в форме define KFEATURE_DESCRIPTION "Enable SMP".

KMACHINE

Машина, известная ядру. Иногда используемое ядром имя машины не совпадает с именем в системе сборки OE. Например, имя машины, которое система OE воспринимает как core2-32-intel-common, будет иным в ядре Linux Yocto (intel-core2-32). Для таких случаев переменная KMACHINE сопоставляет имена машины в ядре и системе сборки OE.

Эти сопоставления выполняются в ветвях метаданных ядер Yocto Linux. В качестве примера рассмотрим файл common/recipes-kernel/linux/linux-yocto_3.19.bbappend.

```
LINUX_VERSION_core2-32-intel-common = "3.19.0"
COMPATIBLE_MACHINE_core2-32-intel-common = "${MACHINE}"
SRCREV_meta_core2-32-intel-common = "8897ef68b30e7426bc1d39895e71fb155d694974"
SRCREV_machine_core2-32-intel-common = "43b9eced9ba8a57add36af07736344dcc383f711"
KMACHINE_core2-32-intel-common = "intel-core2-32"
KBRANCH_core2-32-intel-common = "standard/base"
KERNEL_FEATURES_append_core2-32-intel-common = "${KERNEL_FEATURES_INTEL_COMMON}"
```

Оператор KMACHINE говорит, что ядро воспринимает имя машины как intel-core2-32, однако система сборки OE именует её как core2-32-intel-common.

KTYPE

Определяет тип ядра, используемый при сборке конфигурации. Задания linux-yocto определяют типы standard, tiny и preempt-rt (см. раздел Kernel Types [3]). Переменная KTYPE задаётся в описании BSP, а её значение должно соответствовать значению переменной LINUX_KERNEL_TYPE в задании для ядра.

L

LABELS

Указывает список целей для автоматической настройки конфигурации. Использование переменной рассмотрено в описании класса grub-efi.

LAYERDEPENDS

Список разделенных пробелами уровней, от которых зависит данный уровень. Можно также указывать конкретную версию уровня, добавляя её в конце имени. Например, LAYERDEPENDS_mylayer = "anotherlayer (=3)", где номер версии 3 уровня anotherlayer сравнивается с LAYERVERSION_anotherlayer. Переменная указывается в файле conf/layer.conf и должна иметь суффикс в виде имени конкретного уровня (например, LAYERDEPENDS_mylayer).

LAYERDIR

При использовании в файле layer.conf указывает путь к текущему уровню. Переменная не доступна за пределами layer.conf и ссылки разыменовываются сразу по завершении разбора файла.

LAYERRECOMMENDS

Список разделенных пробелами уровней, рекомендованных для использования с данным уровнем. Можно также указывать конкретную версию уровня, добавляя её в конце имени. Например, LAYERRECOMMENDS_mylayer = "anotherlayer (=3)", где номер версии 3 уровня anotherlayer сравнивается с LAYERVERSION_anotherlayer. Переменная указывается в файле conf/layer.conf и должна иметь суффикс в виде имени конкретного уровня (например, LAYERRECOMMENDS_mylayer).

LAYERSERIES_COMPAT

Перечисляет версии OpenEmbedded-Core, с которыми совместим уровень. Переменная LAYERSERIES_COMPAT позволяет указать, какие комбинации уровня и OE-Core предполагаются работоспособными. Переменная позволяет системе определить, когда уровень не будет проверяться с новыми выпусками OE-Core (например, не поддерживается).

Для указания версии OE-Core, с которой совместим уровень, используется эта переменная в файле conf/layer.conf. Для указания версии используйте имя выпуска YP (например, warrior). При указании нескольких версий OE-Core для уровня они разделяются пробелами, как показано ниже.

```
LAYERSERIES_COMPAT_layer_root_name = "warrior thud"
```

Установка LAYERSERIES_COMPAT требуется стандартом Yocto Project Compatible версии 2. Система сборки OE выдаёт предупреждения, если переменная не установлена для уровня.

LAYERVERSION

Опционально задаёт версию уровня одним числом. Можно использовать это значение в LAYERDEPENDS для другого уровня для указания зависимости от конкретной версии уровня. Переменная применяется в файле conf/layer.conf и должна иметь суффикс в виде имени конкретного уровня (например, LAYERVERSION_mylayer).

LD

Сокращённая команда и аргументы для запуска компоновщика.

LDFLAGS

Задаёт флаги, передаваемые компоновщику. Переменная экспортируется в окружение и доступна программам, собираемым на этапе компиляции. Принятое по умолчанию значение LDFLAGS зависит от собираемого объекта:

- TARGET_LDFLAGS при сборке для целевой платформы;
- BUILD_LDFLAGS при сборке для сборочного хоста (-native);
- BUILDSDK_LDFLAGS при сборке для SDK (nativesdk-).

LEAD_SONAME

Задаёт основной (первичный) файл скомпилированной библиотеки (.so), к которому класс debian применяет свои правила именования в образах с множеством библиотек. Переменная применяется с классам debian.

LIC_FILES_CHKSUM

Контрольная сумма текста лицензии в исходном коде задания для отслеживания изменений в тексте. Если текст лицензии будет изменён, это приведёт к отказу сборки, указывающему разработчику просмотреть изменение в тексте. Эта переменная должна указываться для всех заданий, кроме тех, где установлено LICENSE = CLOSED. Дополнительную информацию о лицензировании можно найти в разделе Tracking License Changes [2].

LICENSE

Список лицензий для задания в соответствии с приведёнными ниже правилами:

- не допускаются пробелы в именах лицензий;
- имена лицензий разделяются символом | при возможности выбора лицензии;
- имена лицензий разделяются символом & (амперсанд) при наличии нескольких лицензий для разных частей кода;
- между именами лицензий можно использовать символы пробела;
- для стандартных лицензий применяются имена из файла meta/files/common-licenses/ или имена флагов SPDXLICENSEMAP, заданные в файле meta/conf/licenses.conf.

Ниже приведено несколько примеров.

```
LICENSE = "LGPLv2.1 | GPLv3"
LICENSE = "MPL-1 & LGPLv2.1"
LICENSE = "GPLv2+"
```

Первый пример взят из заданий для Qt, распространяемых по лицензии LGPL версии 2.1 или GPL версии 3. Во втором примере представлено лицензирование Cairo, где применяются две лицензии для разных частей кода. Последний пример взят из sysstat, где используется одна лицензия.

Можно также задавать лицензии на уровне пакета для случаев использования компонентами разных лицензий. Например, если часть кода распространяется по лицензии GPLv2, но сопровождается документацией, которая использует лицензию GNU FDL[1] 1.2, можно указать

```
LICENSE = "GFDL-1.2 & GPLv2"
LICENSE_${PN} = "GPLv2"
LICENSE_${PN}-doc = "GFDL-1.2"
```

LICENSE_CREATE_PACKAGE

Установка LICENSE_CREATE_PACKAGE = 1 заставляет систему сборки OE создавать дополнительный пакет (${PN}-lic)ого задания и добавлять эти пакеты в переменную RRECOMMENDS_${PN}.

Пакет ${PN}-lic создает каталог /usr/share/licenses с именем ${PN} (базовое имя задания) и помещает в него файлы с лицензией и информацией об авторских правах (т. е. копии подходящих лицензий из meta/common-licenses, соответствующие лицензиям, указанным в переменной LICENSE метаданных задания, и копии файлов, указанных в LIC_FILES_CHKSUM как файлы с текстами лицензий).

Дополнительная информация о предоставлении текстов лицензий приведена в описаниях переменных COPY_LIC_DIRS и COPY_LIC_MANIFEST, а также в разделе Providing License Text [2].

LICENSE_FLAGS

перечисляет для задания дополнительные флаги, которые следует включить в LICENSE_FLAGS_WHITELIST, чтобы задание можно было собрать. Флаги в списке разделяются пробелами. Это значение не зависит от LICENSE и обычно применяется для маркировки заданий, которым могут требоваться дополнительные лицензии для использования в коммерческой продукции. Дополнительная информация о лицензировании приведена в разделе Enabling Commercially Licensed Recipes [2].

LICENSE_FLAGS_WHITELIST

Перечисляет флаги лицензий, которым при указании в LICENSE_FLAGS для задания, не следует препятствовать в сборке. Это иногда называют «белым списком» лицензий. Дополнительная информация о лицензировании приведена в разделе Enabling Commercially Licensed Recipes [2].

LICENSE_PATH

Путь к дополнительным лицензиям, используемым при сборке. По умолчанию система сборки OE применяет COMMON_LICENSE_DIR для определения каталога с общим текстом лицензии, используемой при сборке. Эта переменная позволяет добавить каталоги в форме LICENSE_PATH += "path-to-additional-common-licenses".

LINUX_KERNEL_TYPE

Определяет тип ядра, применяемый при сборке конфигурации. В заданиях linux-yocto определены типы standard, tiny и preempt-rt (см. раздел Kernel Types [3]). Если переменная LINUX_KERNEL_TYPE не задана, применяется тип standard. Вместе с переменной KMACHINE значение LINUX_KERNEL_TYPE определяет аргументы поиска, применяемые инструментами ядра при поиске подходящего описания в метаданных ядра для определения исходных файлов и конфигурации.

LINUX_VERSION

Версия Linux с сайта kernel.org, для которой будет собираться образ ядра с помощью системы OE. Эта переменная указывается в задании для ядра. Например, задание linux-yocto-3.4.bb из каталога meta/recipes-kernel/linux определяет переменную

```
LINUX_VERSION ?= "3.4.24"
```

Значение LINUX_VERSION используется при определении переменной PV для задания

```
PV = "${LINUX_VERSION}+git${SRCPV}"
```

LINUX_VERSION_EXTENSION

Строка расширения, компилируемая в строку версии ядра Linux, собираемого в системе OE. Эта переменная указывается в задании для ядра. Например, задание определяет её как

```
LINUX_VERSION_EXTENSION ?= "-yocto-${LINUX_KERNEL_TYPE}"
```

[1]Free Documentation License - свободное распространение документации.

Определение этой переменной по существу задаёт значение конфигурационного параметра ядра CONFIG_LOCALVERSION, которое доступно по команде uname. Ниже приведен пример вывода команды.

```
$ uname -r
3.7.0-rc8-custom
```

LOG_DIR

Задаёт каталог, в который система сборки OE записывает log-файлы (по умолчанию ${TMPDIR}/log). Каталоги с журналами для каждой задачи задаются с помощью переменной T.

M

MACHINE

Задаёт целевое устройство, для которого собирается ядро. Переменная указывается в файле local.conf каталога сборки. По умолчанию задана архитектура x86, эмулируемая с помощью QEMU, в форме MACHINE ?= "qemux86". Переменная соответствует одноимённому файлу конфигурации машины, в котором установлена конфигурация для неё. Таким образом, при установке qemux86 будет в наличии файл qemux86.conf, хранящийся в каталоге исходных кодов (подкаталог meta/conf/machine). Ниже приведен список машин, включённых в дистрибутив YP.

```
MACHINE ?= "qemuarm"
MACHINE ?= "qemuarm64"
MACHINE ?= "qemumips"
MACHINE ?= "qemumips64"
MACHINE ?= "qemuppc"
MACHINE ?= "qemux86"
MACHINE ?= "qemux86-64"
MACHINE ?= "genericx86"
MACHINE ?= "genericx86-64"
MACHINE ?= "beaglebone"
MACHINE ?= "mpc8315e-rdb"
MACHINE ?= "edgerouter"
```

Последние 5 ссылок относятся к эталонным платам YP уровня meta-yocto-bsp. Возможна установка в переменной значений, обеспечиваемых дополнительными уровнями BSP.

MACHINE_ARCH

Задаёт архитектуру конкретной машины и устанавливается автоматически из переменной MACHINE или TUNE_PKGARCH. Не следует менять значение MACHINE_ARCH вручную.

MACHINE_ESSENTIAL_EXTRA_RDEPENDS

Зависящий от машины список обязательных пакетов для установки в создаваемый образ. Процесс сборки зависит от наличия этих пакетов. Кроме того, поскольку эти пакеты важны для машины, без них она может не загрузиться. Переменная влияет на образы, основанные на packagegroup-core-boot, включая образ core-image-minimal.

Эта переменная похожа на MACHINE_ESSENTIAL_EXTRA_RRECOMMENDS, но сборка создаваемого образа зависит от сборки пакетов из списка. При отсутствии нужных пакетов сборка станет невозможной.

В качестве примера рассмотрим машину, для которой нужно во время загрузки запускать пакет example-init для инициализации оборудования. В этом случае файл конфигурации для машины (.conf) должен включать строку

```
MACHINE_ESSENTIAL_EXTRA_RDEPENDS += "example-init"
```

MACHINE_ESSENTIAL_EXTRA_RRECOMMENDS

Список зависимых от машины пакетов для включения в собираемый образ. Процесс сборки не зависит от наличия этих пакетов. Однако эти пакеты важны для загрузки машины. Переменная влияет на образы, основанные на packagegroup-core-boot, включая образ core-image-minimal.

Эта переменная похожа на MACHINE_ESSENTIAL_EXTRA_RDEPENDS, но сборка образа не зависит от сборки пакетов из списка, т. е. при отсутствии пакета образ все равно будет собран. Обычно эта переменная служит для управления важными компонентами ядра, которые могут встраиваться непосредственно в ядро (не модуль) и тогда пакет создаваться не будет.

Рассмотрим пример с пользовательским ядром, где нужен драйвер для определённого сенсорного экрана, чтобы использовать машину. Драйвер может быть встроен в ядро или собран как модуль в зависимости от конфигурации. При сборке драйвера в виде модуля его нужно установить, но хочется, чтобы сборка выполнялась и для случая встраивания драйвера в ядро. Данная переменная устанавливает отношение «рекомендации» и во втором случае сборка не будет прерываться в случае отсутствия пакета. Для модуля kernel-module-ab123 в файл конфигурации машины включается строка MACHINE_ESSENTIAL_EXTRA_RRECOMMENDS += "kernel-module-ab123".

В этом примере задание kernel-module-ab123 должно явно установить свою переменную PACKAGES, чтобы BitBake не использовал переменную PACKAGES_DYNAMIC из задания ядра для соблюдения зависимости.

Примерами таких устройств являются драйверы флэш-памяти, экрана, клавиатуры, мыши или сенсорного экрана.

MACHINE_EXTRA_RDEPENDS

Список машинозависимых пакетов для установки в собираемый образ, которые не имеют значения при загрузке машины, но сборка образа зависит от этих пакетов. Переменная влияет лишь на образы, основанные на packagegroup-base, которые не включают core-image-minimal и core-image-full-cmdline. Переменная похожа на MACHINE_EXTRA_RRECOMMENDS, но отличается тем, что сборка образа зависит от указанных в ней пакетов.

Примером может служить машина с поддержкой WiFi, которая не существенна для загрузки. Однако при сборке более функционального образа может потребоваться WiFi. В этом случае предполагается наличие пакета с микрокодом WiFi, поэтому логична зависимость процесса сборки от наличия этого пакета. Для решения этой задачи (в предположении, что пакет называется wifidriver-firmware) следует включить в файл .conf для машины строку MACHINE_EXTRA_RDEPENDS += "wifidriver-firmware".

MACHINE_EXTRA_RRECOMMENDS

Список машинозависимых пакетов для установки в собираемый образ, которые не имеют значения при загрузке машины и сборка образа не зависит от этих пакетов. Переменная влияет лишь на образы, основанные на packagegroup-base, которые не включают core-image-minimal и core-image-full-cmdline. Переменная похожа на MACHINE_EXTRA_RDEPENDS, но отличается тем, что сборка образа не зависит от указанных в ней пакетов.

Примером может служить машина с поддержкой WiFi, которая не существенна для загрузки. Однако при сборке более функционального образа может потребоваться WiFi. В этом случае пакет с модулем WiFi для ядра не будет создаваться, если драйвер встроен в ядро, но нужно выполнение сборки без ошибок, несмотря на отсутствие пакета. Для решения этой задачи (в предположении, что пакет называется kernel-module-examplewifi) следует включить в файл .conf для машины строку MACHINE_EXTRA_RRECOMMENDS += "kernel-module-examplewifi".

MACHINE_FEATURES

Задаёт список аппаратных функций, которые MACHINE может поддерживать. Включение дополнительных свойств управляется переменными DISTRO_FEATURES, COMBINED_FEATURES и IMAGE_FEATURES. Список аппаратных возможностей, поддерживаемых YP, приведен в разделе 12.1. Свойства машины.

MACHINE_FEATURES_BACKFILL

Свойства, добавляемые в MACHINE_FEATURES при их отсутствии в переменной MACHINE_FEATURES_BACKFILL_CONSIDERED. Эта переменная задана в файле meta/conf/bitbake.conf и не предназначена для настройки пользователем. О сослаться на переменную для просмотра функций, «засыпанных» в конфигурации всех машин (см. параграф 12.4. Отключение автоматически добавляемых свойств).

MACHINE_FEATURES_BACKFILL_CONSIDERED

Свойства из MACHINE_FEATURES_BACKFILL, которые не следует добавлять в MACHINE_FEATURES при сборке (см. параграф 12.4. Отключение автоматически добавляемых свойств).

MACHINEOVERRIDES

Разделённый двоеточиями список переопределений, применяемых для конкретной машины. По умолчанию этот список включает значение переменной MACHINE.

Переменную MACHINEOVERRIDES можно расширить добавляя переопределения, применяемые к машине. Например, все машины, эмулируемые в QEMU (qemuarm, qemux86 и т. п.), включают файл meta/conf/machine/include/qemu.inc, который добавляет значение в начало переменной MACHINEOVERRIDES =. "qemuall:". Это позволяет переопределять переменные для всех машин, эмулируемых в QEMU, как показано в примере из задания connman-conf.

```
SRC_URI_append_qemuall = "file://wired.config \
                          file://wired-setup \
                         "
```

Базовым механизмом для MACHINEOVERRIDES является включение в значение по умолчанию переменной OVERRIDES.

MAINTAINER

Адрес электронной почты для поддержки дистрибутива.

MIRRORS

Задаёт дополнительные пути для поиска исходных кодов системой сборки OE, которая сначала пытается найти коды в локальном каталоге загрузки. При отсутствии кодов в этом каталоге система просматривает репозитории, указанные переменной PREMIRRORS, затем «восходящий источник» и репозитории, указанные переменной MIRRORS. В дистрибутиве (DISTRO) poky принятое по умолчанию значение MIRRORS указано в файле conf/distro/poky.conf репозитория meta-poky.

MLPREFIX

Задаёт префикс, добавляемый к переменной PN для получения специальной версии задания или пакета (версии Multilib). Переменная применяется там, где нужно добавить или удалить префикс переменной (например, BPN). MLPREFIX устанавливается при наличии префикса у переменной PN.

ML в имени MLPREFIX означает MultiLib. Это возникло в то время, когда строка nativesdk служила суффиксом, а не префиксом имени задания. Префикс nativesdk учитывается в MLPREFIX.

Для разъяснения случаев применения MLPREFIX рассмотрим пример с использованием BBCLASSEXTEND для обеспечения версии задания nativesdk в дополнении к версии для целевой платформы. Если это задание заявляет зависимость при сборке от других заданий с помощью переменной DEPENDS, то зависимость от foo будет автоматически переопределяться в зависимость от nativesdk-foo. Однако зависимости вида do_foo[depends] += "recipe:do_foo" не будут переопределяться автоматически. Если нужно переопределение таких зависимостей, можно указать do_foo[depends] += "${MLPREFIX}recipe:do_foo".

module_autoload

Переменная заменена KERNEL_MODULE_AUTOLOAD, поэтому следует отредактировать все файлы, включающие её. Например, module_autoload_rfcomm = "rfcomm" заменить на KERNEL_MODULE_AUTOLOAD += "rfcomm".

module_conf

Задаёт строки с синтаксисом modprobe.d для включения в файл /etc/modprobe.d/modname.conf. Эту переменную можно использовать в любом месте, доступном заданию для ядра или внешнего (out-of-tree) модуля (например, в файле конфигурации машины или дистрибутива, файле дополнения к заданию или самом задании). При использовании переменной нужно также перечислить модули ядра в переменной KERNEL_MODULE_AUTOLOAD.

Базовый синтаксис переменной имеет вид module_conf_module_name = "modprobe.d-syntax". Нужно применять переопределение имён модулей ядра.

Включение module_conf заставляет систему сборки OE заполнять файл /etc/modprobe.d/modname.conf строками с синтаксисом modprobe.d (информацию о синтаксисе можно получить по команде man modprobe.d). Например, добавление опций arg1 и arg2 для модуля mymodule может иметь вид module_conf_mymodule = "options mymodule arg1=val1 arg2=val2".

Автоматическая загрузка модулей ядра управляется переменной KERNEL_MODULE_AUTOLOAD.

MODULE_TARBALL_DEPLOY

Управляет созданием файла modules-*.tgz с модулями ядра, созданными при сборке (0 отменяет запись архива).

MODULE_TARBALL_LINK_NAME

Имя ссылки для архива модулей ядра, устанавливаемое в файле meta/classes/kernel-artifact-names.bbclass как MODULE_TARBALL_LINK_NAME ?= "${KERNEL_ARTIFACT_LINK_NAME}". Для KERNEL_ARTIFACT_LINK_NAME в том же файле устанавливается значение KERNEL_ARTIFACT_LINK_NAME ?= "${MACHINE}"

MODULE_TARBALL_NAME

Базовое имя архива модулей ядра, устанавливаемое в файле meta/classes/kernel-artifact-names.bbclass как MODULE_TARBALL_NAME ?= "${KERNEL_ARTIFACT_NAME}". Переменная KERNEL_ARTIFACT_NAME в том же файле получает значение KERNEL_ARTIFACT_NAME ?= "${PKGE}-${PKGV}-${PKGR}-${MACHINE}${IMAGE_VERSION_SUFFIX}".

MULTIMACH_TARGET_SYS

Однозначно указывает тип целевой системы, для которой собираются пакеты. Переменная позволяет размещать вывод для разных целевых систем в разные подкаталоги одного выходного каталога. По умолчанию переменная имеет значение ${PACKAGE_ARCH}${TARGET_VENDOR}-${TARGET_OS}. Некоторые классы (например, cross-canadian) меняют значение MULTIMACH_TARGET_SYS. Дополнительная информация приведена в описаниях переменных STAMP и STAGING_DIR_TARGET.

N

NATIVELSBSTRING

Строка, указывающая дистрибутив хоста. Строка включает идентификатор дистрибутива, за которым следует номер выпуска, возвращаемый командой lsb_release или прочитанный в файле /etc/lsb-release. Например, при работе на хосте Ubuntu 12.10 переменная будет иметь значение Ubuntu-12.10. Если дистрибутив определить не удаётся, переменная получает значение Unknown.

Переменная используется по умолчанию для разделения естественных пакетов разных дистрибутивов (например, для предотвращения проблем, связанный с несовместимостью версий glibc). Кроме того, эта переменная проверяется по SANITY_TESTED_DISTROS, если та установлена.

NM

Сокращённая команда и аргументы для запуска nm (просмотр символов из объектных файлов).

NO_GENERIC_LICENSE

Предотвращает ошибки QA при использовании в задании необычной, незакрытой лицензии. Имеются пакеты, такие как linux-firmware, со множеством мало распространённых лицензий. Время от времени также появляются новые лицензии, чтобы избежать введения множества базовых лицензий, которые применимы к конкретному пакету. Переменная NO_GENERIC_LICENSE позволяет скопировать лицензию, которой нет в числе базовых. Например, в задание можно добавить лицензию NO_GENERIC_LICENSE to с помощью NO_GENERIC_LICENSE[license_name] = "license_file_in_fetched_source". В приведённом ниже примере используется файл LICENSE.Abilis.txt file в качестве лицензии для извлечённого исходного кода.

```
NO_GENERIC_LICENSE[Firmware-Abilis] = "LICENSE.Abilis.txt"
```

NO_RECOMMENDATIONS

Предотвращает установку пакетов, которые лишь рекомендованы (устанавливаются через переменную RRECOMMENDS). Например, NO_RECOMMENDATIONS = "1" включает эту функцию.

Переменную можно установить глобально в файле local.conf или присоединить к заданию для конкретного образа, переопределяя имя задания в виде NO_RECOMMENDATIONS_pn-target_image = "1".

Важно понимать, что при включении в эту переменную пакетов, от которых зависят другие пакеты (указаны в переменной RDEPENDS), система сборки OE будет игнорировать запрос и установит пакеты, чтобы не нарушать зависимостей.

Некоторые рекомендованные пакеты могут потребоваться для отдельных функций, таких как модули ядра. Можно указать такие пакеты в переменной IMAGE_INSTALL.

Переменная поддерживается только для менеджеров пакетов IPK и RPM, но не поддерживается для DEB.

См. также описания переменных BAD_RECOMMENDATIONS и PACKAGE_EXCLUDE.

NOAUTOPACKAGEDEBUG

Выключает автоматическое выделение для пакета файлов .debug. Если задание требует установки FILES_${PN}-dbg вручную, можно задать переменную NOAUTOPACKAGEDEBUG, позволяющую определить содержимое отладочного пакета. Например,

```
NOAUTOPACKAGEDEBUG = "1"
FILES_${PN}-dev = "${includedir}/${QT_DIR_NAME}/Qt/*"
FILES_${PN}-dbg = "/usr/src/debug/"
FILES_${QT_BASE_NAME}-demos-doc = "${docdir}/${QT_DIR_NAME}/qch/qt.qch"
```

O

OBJCOPY

Сокращённая команда и аргументы для запуска objcopy.

OBJDUMP

Сокращённая команда и аргументы для запуска objdump.

OE_BINCONFIG_EXTRA_MANGLE

При наследовании класса binconfig указывает дополнительные аргументы для команды sed, которая меняет пути в сценариях конфигурации, установленные во время компиляции. Наследование этого класса ведёт к изменению всех путей в сценариях конфигурации так, чтобы они указывали каталог sysroot, чтобы все сборки, использующие сценарий применяли корректные каталоги для кросс-компиляции. Детали применения класса приведены в файле meta/classes/binconfig.bbclass каталога исходных кодов, а базовая информация - в параграфе 6.7. binconfig.bbclass.

OE_IMPORTS

Внутренняя переменная, указывающая системе сборки OE модули Python, импортируемые для каждой функции Python, выполняемой системой. Не нужно устанавливать эту переменную.

OE_INIT_ENV_SCRIPT

Имя сценария настройки среды сборки для организации работы в расширяемом SDK (по умолчанию oe-init-build-env). Для пользовательских сценариев настройки следует указывать имя сценария в переменной.

OE_TERMINAL

Управляет вызовом интерактивных терминалов системой сборки OE на сборочном хосте (например, использование команды BitBake с опцией -c devshell). Работа с терминальными сессиями описана в разделе Using a Development Shell [2]. Переменная OE_TERMINAL может принимать значения auto, gnome, xfce, rxvt, screen, konsole, none.

OEROOT

Каталог, из которого запускается сценарий верхнего уровня для настройки окружения (oe-init-build-env). При запуске сценария переменная OEROOT преобразуется в имя каталога, где этот сценарий размещён.

OLDEST_KERNEL

Указывает минимальную версию ядра Linux, с которой будут работать собираемые двоичные файлы. Эта переменная передается в сборку встроенной библиотеки glibc. По умолчанию эта переменная принимает значение из файла meta/conf/bitbake.conf, которое можно переопределить ф файле конфигурации дистрибутива.

OVERRIDES

Список разделенных двоеточиями переопределений, которые будут применены. Переопределения в BitBake позволяют селективно изменять значения переменных при завершении анализа. Набор переопределений в переменной OVERRIDES представляет «состояние» в процессе сборки, которое включает собираемое задание, машину, для которой выполняется сборка и т. п.

Например, если строка an-override представлена как элемент списка в OVERRIDES, приведённое назначение FOO_an-override = "overridden" будет переопределять FOO значением overridden в конце анализа.

В разделе Conditional Syntax (Overrides) [4] механизм переопределения описан более подробно.

Принятое по умолчанию значение OVERRIDES включает значения переменных CLASSOVERRIDE, MACHINEOVERRIDES и DISTROOVERRIDES. Другим важным переопределением, включённым по умолчанию, является pn-${PN}. Это позволяет установить переменные для определённого задания в файлах .conf. Например, FOO_pn-myrecipe = "myrecipe-specific value".

Простым способом узнать используемые переопределения является поиск строки OVERRIDES в выводе команды bitbake -e (см. раздел Viewing Variable Values [2]).

P

P

Имя и версия задания в форме ${PN}-${PV}.

PACKAGE_ARCH

Архитектура собираемых пакетов. По умолчанию для этой переменной устанавливается значение TUNE_PKGARCH при сборке пакета для целевой платформы, BUILD_ARCH при сборке для сборочного хоста и ${SDK_ARCH}-${SDKPKGSUFFIX} при сборке для SDK (см. SDK_ARCH). Однако, если выходные пакеты задания связаны с конкретной целевой системой, а не с базовой архитектурой машины, следует устанавливать в задании для PACKAGE_ARCH значение переменной MACHINE_ARCH в форме PACKAGE_ARCH = "${MACHINE_ARCH}"

PACKAGE_ARCHS

Задаёт список архитектур, совместимых с целевой машиной. Эта переменная устанавливается автоматически и обычно не следует менять её вручную. Записи в списке разделяются пробелами и упорядочены по приоритету. По умолчанию переменная имеет значение ${PACKAGE_EXTRA_ARCHS} ${MACHINE_ARCH}.

PACKAGE_BEFORE_PN

Включает легко добавляемые пакеты в PACKAGES перед ${PN} так, что эти пакеты могут указывать файлы, которые обычно включаются в принятые по умолчанию пакеты.

PACKAGE_CLASSES

Эта переменная, устанавливаемая в файле conf/local.conf каталога сборки, задаёт менеджер пакетов, который система сборки OE использует для упаковки. Можно указать в переменной один или несколько менеджеров, например, PACKAGE_CLASSES ?= "package_rpm package_deb package_ipk package_tar". Хотя класс package_tar и поддерживается, его функциональность ограничена, поскольку для него не поддерживаются зависимости между пакетами в машине установки. Поэтому применять данный класс не рекомендуется.

Система сборки использует только первый элемент из списка в качестве менеджера пакетов при создании образа или SDK. Однако пакеты будут создаваться с использованием и других указанных менеджеров. Например, для использования менеджера пакетов IPK в файле local.conf задаётся строка PACKAGE_CLASSES ?= "package_ipk".

Дополнительная информация о влиянии менеджера пакетов на производительность упаковки и сборки рассмотрена в параграфе 6.88. package.bbclass.

PACKAGE_DEBUG_SPLIT_STYLE

Определяет расщепление двоичных файлов и отладочной информации при создании пакетов *-dbg для отладчика GDB[1]. Переменная позволяет управлять местом размещения отладочной информации, которая может включать исходные файлы.

- *.debug* задаёт размещение отладочных символов вместе с двоичными файлами в каталоге .debug на целевой платформе. Например, при установке двоичного файла в /bin соответствующие отладочные символы будут помещены в /bin/.debug. Исходные коды помещаются в /usr/src/debug.
- *debug-file-directory* задаёт размещение отладочных символов /usr/lib/debug на целевой платформе и их разделение по пути установки двоичного файла. Например, при установке двоичного файла в /bin соответствующие отладочные символы помещаются в /usr/lib/debug/bin. Исходные коды помещаются в /usr/src/debug.
- *debug-without-src* похоже на .debug, но файлы исходных кодов не устанавливаются.
- *debug-with-srcpkg* похоже на .debug, но исходные файлы помещаются в отдельный пакет *-src pkg. Этот вариант применяется по умолчанию.

Отладка с использованием GDB описана в разделе Debugging With the GNU Project Debugger (GDB) Remotely [2].

PACKAGE_EXCLUDE_COMPLEMENTARY

Блокирует установку указанных пакетов при установке дополнений. Например, при использовании IMAGE_FEATURES для установки пакетов dev- можно отказаться от инсталляции всех пакетов, кроме определённого multilib. В переменной PACKAGE_EXCLUDE_COMPLEMENTARY можно указать регулярное выражение для отбора исключаемых из установки пакетов.

PACKAGE_EXCLUDE

Указывает пакеты, которые не следует устанавливать в образ. Например, PACKAGE_EXCLUDE = "package_name package_name package_name …". Эту переменную можно установить глобально в файле local.conf или присоединить к заданию для конкретного образа путём переопределения имени задания, как PACKAGE_EXCLUDE_pn-target_image = "package_name".

При запросе отказа от установки пакета, от которого зависят другие пакеты (указаны в переменной RDEPENDS), система сборки OE будет давать критическую ошибку. Поскольку система сборки останавливает процесс, можно использовать переменную в итеративном процессе удаления из системы ненужных компонент.

Переменная поддерживается только для менеджеров пакетов IPK и RPM, но не поддерживается для DEB.

См. также описания переменных BAD_RECOMMENDATIONS и NO_RECOMMENDATIONS.

PACKAGE_EXTRA_ARCHS

Задаёт список архитектур, совместимых с процессором устройства. Переменная полезна при сборке для нескольких разных устройств, использующих смешанные процессоры, например, XScale и ARM926-EJS.

PACKAGE_FEED_ARCHS

Опционально задаёт архитектуру, используемую как часть URI хранилищ пакетов при сборке. При наличии переменной её значение добавляется к финальной части URI, созданных из переменных PACKAGE_FEED_URIS и PACKAGE_FEED_BASE_PATHS.

Можно использовать PACKAGE_FEEDS_ARCHS в качестве «белого списка» вариантов архитектуры пакетов. Если такой список не нужен (типичная ситуация), переменную можно опустить, что приведёт к включению всех вариантов архитектуры для текущей машины из удалённого хранилища пакетов.

[1] GNU Project Debugger - отладчик проекта GNU.

Рассмотрим пример, где PACKAGE_FEED_URIS, PACKAGE_FEED_BASE_PATHS и PACKAGE_FEED_ARCHS заданы в файле local.conf, как показано ниже.

```
PACKAGE_FEED_URIS = "https://example.com/packagerepos/release \
                     https://example.com/packagerepos/updates"
PACKAGE_FEED_BASE_PATHS = "rpm rpm-dev"
PACKAGE_FEED_ARCHS = "all core2-64"
```

С учётом этих определений идентификаторы хранилищ пакетов будут иметь вид, представленный ниже.

```
https://example.com/packagerepos/release/rpm/all
https://example.com/packagerepos/release/rpm/core2-64
https://example.com/packagerepos/release/rpm-dev/all
https://example.com/packagerepos/release/rpm-dev/core2-64
https://example.com/packagerepos/updates/rpm/all
https://example.com/packagerepos/updates/rpm/core2-64
https://example.com/packagerepos/updates/rpm-dev/all
https://example.com/packagerepos/updates/rpm-dev/core2-64
```

PACKAGE_FEED_BASE_PATHS

Задаёт базовый путь, используемый при создании URI хранилищ пакетов, образуя среднюю часть URI, используемых системой сборки OE. Базовый путь включается между переменными PACKAGE_FEED_URIS и PACKAGE_FEED_ARCHS.

Рассмотрим пример, где PACKAGE_FEED_URIS, PACKAGE_FEED_BASE_PATHS и PACKAGE_FEED_ARCHS определены в файле local.conf, как показано ниже.

```
PACKAGE_FEED_URIS = "https://example.com/packagerepos/release \
                     https://example.com/packagerepos/updates"
PACKAGE_FEED_BASE_PATHS = "rpm rpm-dev"
PACKAGE_FEED_ARCHS = "all core2-64"
```

В этом случае идентификаторы хранилищ пакетов будут иметь вид, представленный ниже.

```
https://example.com/packagerepos/release/rpm/all
https://example.com/packagerepos/release/rpm/core2-64
https://example.com/packagerepos/release/rpm-dev/all
https://example.com/packagerepos/release/rpm-dev/core2-64
https://example.com/packagerepos/updates/rpm/all
https://example.com/packagerepos/updates/rpm/core2-64
https://example.com/packagerepos/updates/rpm-dev/all
https://example.com/packagerepos/updates/rpm-dev/core2-64
```

PACKAGE_FEED_URIS

Задаёт начальную часть URI хранилища пакета, используемого системой сборки OE. Полный идентификатор URI включает переменные PACKAGE_FEED_URIS, PACKAGE_FEED_BASE_PATHS и PACKAGE_FEED_ARCHS.

В приведённом ниже примере переменные PACKAGE_FEED_URIS, PACKAGE_FEED_BASE_PATHS и PACKAGE_FEED_ARCHS заданы в файле local.conf.

```
PACKAGE_FEED_URIS = "https://example.com/packagerepos/release \
                     https://example.com/packagerepos/updates"
PACKAGE_FEED_BASE_PATHS = "rpm rpm-dev"
PACKAGE_FEED_ARCHS = "all core2-64"
```

С учётом этих установок идентификаторы хранилищ пакетов будут иметь вид, представленный ниже.

```
https://example.com/packagerepos/release/rpm/all
https://example.com/packagerepos/release/rpm/core2-64
https://example.com/packagerepos/release/rpm-dev/all
https://example.com/packagerepos/release/rpm-dev/core2-64
https://example.com/packagerepos/updates/rpm/all
https://example.com/packagerepos/updates/rpm/core2-64
https://example.com/packagerepos/updates/rpm-dev/all
https://example.com/packagerepos/updates/rpm-dev/core2-64
```

PACKAGE_GROUP

Переменная PACKAGE_GROUP была переименована в FEATURE_PACKAGES. Если вы продолжаете использовать PACKAGE_GROUP, система сборки OE будет выдавать предупреждение.

PACKAGE_INSTALL

Окончательный список пакетов, передаваемый менеджеру пакетов для установки в образ. Этот список не обязательно совпадает с полным списком устанавливаемых пакетов. Эта переменная является внутренней для кода создания образа, поэтому в общем случае следует использовать IMAGE_INSTALL для указания устанавливаемых пакетов. Исключением является работа с образом core-image-minimal-initramfs. Для образов initramfs применяется переменная PACKAGE_INSTALL (см. Building an Initial RAM Filesystem (initramfs) Image [2]).

PACKAGE_INSTALL_ATTEMPTONLY

Задаёт список пакетов, которые система сборки OE пытается установить при создании образа. Если установка пакета не удаётся, система сборки не генерирует ошибки. Эту переменную пользователь обычно не задаёт.

PACKAGE_PREPROCESS_FUNCS

Задаёт список функций предварительной обработки каталога PKGD при разделении файлов по разным пакетам.

PACKAGE_WRITE_DEPS

Задаёт список зависимостей для сценарией предустановки и пост-установки инструментов native/cross. Если сценарий может выполняться во время создания rootfs, а не на целевой платформе, но зависит от естественного инструмента при работе, нужно указать этот инструмент в PACKAGE_WRITE_DEPS. Работа сценариев пост-установки описана в разделе Post-Installation Scripts [2].

PACKAGECONFIG

Эта переменная обеспечивает способ управления свойствами заданий на уровне отдельного задания. Блок PACKAGECONFIG определяется в заданиях при выборе свойств и аргументов для них. Ниже показана базовая структура блока.

```
PACKAGECONFIG ??= "f1 f2 f3 ..."
PACKAGECONFIG[f1] = "--with-f1,--without-f1,build-deps-f1,rt-deps-f1,rt-recs-f1"
PACKAGECONFIG[f2] = "--with-f2,--without-f2,build-deps-f2,rt-deps-f2,rt-recs-f2"
PACKAGECONFIG[f3] = "--with-f3,--without-f3,build-deps-f3,rt-deps-f3,rt-recs-f3"
```

Сама переменная PACKAGECONFIG задаёт список разделенных пробелами свойств, которые нужно включить. Для каждого свойства можно задать поведение, указав до 5 упорядоченных аргументов, разделенных запятыми. Любой аргумент можно опустить, сохраняя запятые. Для аргументов применяется показанный ниже порядок.

1. Дополнительные аргументы, которые следует добавить в список аргументов сценария настройки (EXTRA_OECONF или PACKAGECONFIG_CONFARGS), если свойство включено.
2. Дополнительные аргументы для EXTRA_OECONF или PACKAGECONFIG_CONFARGS, если свойство выключено.
3. Дополнительные зависимости при сборке (DEPENDS), если свойство включено.
4. Дополнительные зависимости при работе (RDEPENDS), если свойство включено.
5. Дополнительные рекомендации при работе (RRECOMMENDS), если свойство включено.

Рассмотрим блок PACKAGECONFIG из задания librsvg. В этом примере свойство croco имеет 3 аргумента, определяющих его поведение.

```
PACKAGECONFIG ??= "croco"
PACKAGECONFIG[croco] = "--with-croco,--without-croco,libcroco"
```

Аргументы --with-croco и libcroco применяются лишь при включённом свойстве. В этом случае --with-croco добавляется к аргументам сценария настройки, а libcroco - в DEPENDS. Если свойство отключено (например, в файле .bbappend на другом уровне, к сценарию настройки добавляется аргумент --without-croco вместо --with-croco. Показанная выше базовая структура PACKAGECONFIG одинакова для создания и изменения блока. При создании используется структура внутри задания. Для изменения имеющегося блока PACKAGECONFIG есть два способа.

- *Файл добавления.* Создаётся файл .bbappend с именем задания на вашем уровне и в нем переопределяется значение PACKAGECONFIG. Можно полностью переопределить переменную PACKAGECONFIG = "f4 f5" или добавить значение в конце PACKAGECONFIG_append = " f4"
- *Файл конфигурации.* Этот метод идентичен изменению блока через файл добавления, но редактируется файл local.conf или файл конфигурации дистрибутива. Можно переопределить переменную PACKAGECONFIG_pn-recipename = "f4 f5" или добавить значение в конце PACKAGECONFIG_append_pn-recipename = " f4"

PACKAGECONFIG_CONFARGS

Список разделенных пробелами опций настройки, создаваемый из переменной PACKAGECONFIG. Классы autotools и cmake используют PACKAGECONFIG_CONFARGS для передачи опций PACKAGECONFIG командам configure и cmake, соответственно. Если вы применяете PACKAGECONFIG, но не класс, обслуживающий задачу do_configure, нужно соответствующим образом использовать PACKAGECONFIG_CONFARGS.

PACKAGEGROUP_DISABLE_COMPLEMENTARY

Для заданий, наследующих класс packagegroup, установка PACKAGEGROUP_DISABLE_COMPLEMENTARY = "1" указывает, что обычные дополнительные пакеты (-dev, -dbg и т. п.) не следует автоматически создавать в задании packagegroup, как это принято по умолчанию.

PACKAGES

Список пакетов, создаваемых заданием. По умолчанию это ${PN}-dbg ${PN}-staticdev ${PN}-dev ${PN}-doc ${PN}-locale ${PACKAGE_BEFORE_PN} ${PN}.

В процессе создания пакетов задача do_package просматривает переменную PACKAGES и использует переменную FILES для каждого пакета при назначении файлов пакета. Если файл соответствует переменной FILES для нескольких пакетов из PACKAGES, он будет назначаться первому (левому) пакеты в списке.

Пустые пакеты из списка (т. е. не соответствующие ни одному из шаблонов FILES_pkg, установленных задачей do_install) не создаются, если это явно не задано переменной ALLOW_EMPTY.

PACKAGES_DYNAMIC

Заявление соответствия задания зависимостям при работе, найденным в других заданиях. Переменная на деле не задаёт зависимости, а лишь говорит о том, что их следует выполнять. Например, если жёсткая зависимость при работе (RDEPENDS) для другого пакета выполнена во время сборки через переменную PACKAGES_DYNAMIC, но пакет с именем модуля на деле не создаётся, сборка другого пакета будет нарушена. Таким образом, при попытке включить этот пакет в образ возникнет отказ зависимостей при подготовке пакета во время задачи do_rootfs.

Обычно, если возможна такая ситуация и пакета, который не был создан, был бы действительным без выполнения зависимости, следует использовать RRECOMMENDS (мягкая зависимость при работе) вместо RDEPENDS.

Пример использования PACKAGES_DYNAMIC для разделения пакетов приведен в разделе Handling Optional Module Packaging [2].

PACKAGESPLITFUNCS

Задаёт список функций, применяемых для дополнительного разделения файлов по отдельным пакетам. Задания могут добавлять значения в начало этой переменной или функции populate_packages для дополнительного разделения пакета. Во всех случаях функции следует устанавливать PACKAGES, FILES, RDEPENDS и другие переменные управления пакетами соответствующим образом для желаемого разделения пакетов.

PARALLEL_MAKE

Опции, передаваемые команде make в процессе выполнения задачи do_compile для управления параллельной компиляцией на сборочном хосте. Эта переменная обычно имеет форму -j x, где x задаёт максимальное число параллельных потоков. Для эффективной работы PARALLEL_MAKE команда должна запускаться с ${EXTRA_OEMAKE}. Проще всего это обеспечить путём использования функции oe_runmake.

По умолчанию система сборки OE автоматически устанавливает в этой переменной число ядер сборочного хоста. Если при сборке в процессе выполнения задачи do_compile возникают проблемы с зависимостями, можно сбросить переменную PARALLEL_MAKE для задания (см. раздел Debugging Parallel Make Races [2]).

Для систем с одним процессором не следует переопределять эту переменную для использования параллельной сборки. Однако в больших системах с несколькими процессорами можно установить для переменной PARALLEL_MAKE значение -j 20. Вопросы ускорения сборки рассмотрены в разделе Speeding Up a Build [2].

PARALLEL_MAKEINST

Опции, передаваемые команде make install в задаче do_install для использования параллельной установки. По умолчанию переменная принимает значение PARALLEL_MAKE. Для учёта PARALLEL_MAKEINST команда make должна вызываться с ${EXTRA_OEMAKE}, для чего можно просто использовать функцию oe_runmake.

Если при сборке программы возникают проблемы с зависимостями в задаче do_install, можно просто сбросить переменную PARALLEL_MAKEINST в задании (см. раздел Debugging Parallel Make Races [2]).

PATCHRESOLVE

Задаёт действие при отказе наложения изменений (patch) и может принимать значение noop или user. Принятое по умолчанию значение noop просто останавливает сборку, когда система OE не может применить исправление.

Установка значения user заставляет систему сборку запустить консоль (shell) с переходом в нужный каталог для исправления конфликта вручную. Переменная задана в файле local.conf.

PATCHTOOL

Задаёт утилиту для применения исправлений (patch) во время выполнения задачи do_patch (patch, quilt или git). По умолчанию используется утилита quilt для всех заданий, кроме quilt-native, для которого применяется patch, поскольку инструмент quilt недоступен во время исправления quilt-native. Для использования определённой утилиты можно указать PATCHTOOL = "patch", PATCHTOOL = "quilt" или PATCHTOOL = "git".

PE

Эпоха для задания. По умолчанию эта переменная не устанавливается. Переменная служит для обеспечения возможности обновления в тех случаях, когда смена номера версии выполняется несовместимым с прежними версиями способом. PE служит принятым по умолчанию значением переменной PKGE.

PF

Задаёт имя задания или пакета с включением номера версии и выпуска (glibc-2.13-r20+svnr15508/ или bash-4.2-r1/) в форме ${PN}-${EXTENDPE}${PV}-${PR}.

PIXBUF_PACKAGES

При наследовании класса pixbufcache переменная указывает пакеты, содержащие загрузчики pixbuf, используемые gdk-pixbuf. По умолчанию класс pixbufcache предполагает загрузчики из основного пакета задания (${PN}), а эта переменная позволяет указать другие пакеты.

PKG

Имя результирующего пакета, создаваемого системой сборки OE. При использовании переменной PKG должно применяться переопределение имён пакетов. Например, при переименовании выходного пакета классом debian следует указывать PKG_packagename.

PKG_CONFIG_PATH

Путь к файлам pkg-config для текущего контекста сборки. Программа pkg-config читает переменную из окружения.

PKGD

Указывает целевой каталог для упаковываемых файлов, которые будут разделены на отдельные пакеты. По умолчанию используется значение ${WORKDIR}/package, которое не следует менять.

PKGDATA_DIR

Указывает общий каталог данных глобального состояния, генерируемый в процессе упаковки. В этом процессе задача do_packagedata упаковывает данные для каждого задания и помещает их в эту общую временную область. По умолчанию эта переменная имеет значение ${STAGING_DIR_HOST}/pkgdata, менять которое не следует. Примеры использования этих данных приведены в разделах Automatically Added Runtime Dependencies [1] и Viewing Package Information with oe-pkgdata-util [2]. Информация о каталоге глобальных состояний приведена в описании переменной STAGING_DIR_HOST.

PKGDEST

Указывает родительский каталог для упаковываемых файлов после разделения их по отдельным пакетам. По умолчанию это каталог ${WORKDIR}/packages-split, в котором система сборки создает каталог для каждого пакета из PACKAGES. Менять имя этого каталога не следует.

PKGDESTWORK

Указывает временную рабочую область, где задача do_package сохраняет метаданные пакета. По умолчанию переменная PKGDESTWORK имеет значение ${WORKDIR}/pkgdata, которое не следует менять. Задача do_packagedata task копирует метаданные из PKGDESTWORK в PKGDATA_DIR для глобальной доступности.

PKGE

Эпоха, для которой пакет(ы) собирается заданием. По умолчанию значением PKGE является значение PE.

PKGR

Выпуск (revision) пакета, собираемого заданием. По умолчанию значением PKGR является значение PR.

PKGV

Номер версии пакета (пакетов), собираемого заданием. По умолчанию PKGV принимает значение переменной PV.

PN

Назначение этой переменной может меняться в зависимости от контекста - имя задания или получаемого в результате пакета.

PN указывает имя задания в контексте файла, используемого системой сборки OE в качестве входного для создания пакета. Имя обычно извлекается из имени файла задания. Например для задания с именем expat_2.0.1.bb по умолчанию значением PN будет expat.

Переменная указывает имя пакета в контексте файла, созданного или произведённого системой сборки OE.

Когда это применимо, переменная PN включает специальный суффикс или префикс. Например, при использовании bash для сборки пакетов для естественной машины PN будет bash-native, а при использовании bash для сборки пакетов для цели и для Multilib значением PN будет bash и lib64-bash, соответственно.

PNBLACKLIST

Перечисляет задания, которые не нужно собирать в системе OE. Переменная работает вместе с классом blacklist, наследуемым глобально. Переменная PNBLACKLIST задаётся в файле local.conf. Например, для предотвращения сборки myrecipe служит PNBLACKLIST[myrecipe] = "Not supported by our organization.".

POPULATE_SDK_POST_HOST_COMMAND

Задаёт список функций, однократно вызываемых системой сборки OE при создании хостовой части SDK. Функции разделяются точкой с запятой, например, POPULATE_SDK_POST_HOST_COMMAND += "function; ... ".

Если нужно передать путь к SDK при вызове функции, можно применять переменную ${SDK_DIR}, указывающую каталог, который система сборки OE будет использовать для вывода SDK.

POPULATE_SDK_POST_TARGET_COMMAND

Задаёт список функций, однократно вызываемых системой сборки OE при создании целевой части SDK. Функции разделяются точкой с запятой, например, POPULATE_SDK_POST_TARGET_COMMAND += "function; ... ".

Если нужно передать путь к SDK при вызове функции, можно применять переменную ${SDK_DIR}, указывающую каталог, который система сборки OE будет использовать для вывода SDK.

PR

Номер пересмотра (изменения) задания. По умолчанию используется значение r0, а последующие варианты имеют номера r1, r2 и т. д. При увеличении переменной PV значение PR обычно сбрасывается в r0.

Системе сборки OE не нужно знать PR для принятия решения о пересборке пакета, для этого служат входные контрольные суммы задания [1] вместе с механизмами stamp (5.2.20. build/tmp/stamps/) и кэширования общих состояний [1].

Переменная PR приобретает важное значение, когда менеджер пакетов динамически устанавливает программы в уже созданном образе. В таких случаях PR, которая по умолчанию имеет значение переменной PKGR, помогает менеджеру определить, какой пакет является более свежим, если имеется несколько пакетов с совпадающими номерами PV (PKGV). Наличие множества пакетов с одним PV обычно говорит о том, что пакеты имеют одну исходную версию, но различаются номерами вариантов (PR), которые могут включать изменения.

Значение PR не требуется увеличивать при изменениях, не влияющих на содержимое или метаданные пакета.

Поскольку ручное управление PR громоздко и подвержено ошибкам, имеется автоматизированное решение (Working With a PR Service [2]).

PREFERRED_PROVIDER

Если один элемент обеспечивается несколькими заданиями, эта переменная позволяет указать предпочтительное задание, что обеспечивает выбор элемента (предпочтительного поставщика). В переменной всегда следует использовать суффикс в форме имени предоставляемого элемента и указывать предпочтительное задание (PN), например, PREFERRED_PROVIDER_virtual/kernel ?= "linux-yocto". Здесь элемент virtual/kernel предоставляется несколькими заданиями и переменная PREFERRED_PROVIDER указывает имя (PN), задания, предпочитаемого для обеспечения virtual/kernel.

Ниже приведены ещё два примера.

```
PREFERRED_PROVIDER_virtual/xserver = "xserver-xf86"
PREFERRED_PROVIDER_virtual/libgl ?= "mesa"
```

Дополнительные сведения о работе с виртуальными провайдерами приведены в разделе Using Virtual Providers [2].

При использовании элемента virtual/* с переменной PREFERRED_PROVIDER задание, которые включают этот элемент в PROVIDES, но не были выбраны переменной PREFERRED_PROVIDER, не будут собираться.

PREFERRED_VERSION

При наличии нескольких версий заданий эта переменная указывает предпочтительную версию. Переменная всегда указывается с PN задания и указанием предпочтительной версии PV. Переменная поддерживает ограниченное использование символа % в качестве шаблона. Этому шаблону может соответствовать любое число символов, что может быть полезно при указании версий с длинными номерами, которые могут изменяться. Ниже приведены два примера.

```
PREFERRED_VERSION_python = "3.4.0"
PREFERRED_VERSION_linux-yocto = "4.12%"
```

Символ % можно указывать лишь в конце строки.

Указанная версия сопоставляется со значением PV, которое не обязательно совпадает с версией в имени задания. Рассмотрим в качестве примера задания foo_1.2.bb и foo_git.bb, где foo_git.bb включает PV = "1.1+git${SRCPV}". В этом случае корректным способом выбора foo_git.bb будет указание PREFERRED_VERSION_foo = "1.1+git%". Если же указать PREFERRED_VERSION_foo = "git", это не будет работать.

Иногда переменная PREFERRED_VERSION может устанавливаться конфигурационными файлами так, что её сложно изменить. В таких случаях можно использовать переменную OVERRIDES для машинозависимого переопределения, например, PREFERRED_VERSION_linux-yocto_qemux86 = "4.12%"

Хотя это и не рекомендуется, в крайнем случае можно воспользоваться переопределением forcevariable, которое является наиболее сильным из переопределений. Например, можно задать PREFERRED_VERSION_linux-yocto_forcevariable = "4.12%". Переопределение _forcevariable не получает особой обработки и действует лишь потому, что значение OVERRIDES по умолчанию включает forcevariable.

PREMIRRORS

Задаёт дополнительные пути для поиска исходных кодов системой сборки OE, которая сначала пытается найти коды в локальном каталоге загрузки. При отсутствии кодов в этом каталоге система просматривает репозитории, указанные переменной PREMIRRORS, затем «восходящий источник» и репозитории, указанные переменной MIRRORS.

Предположим, что используется дистрибутив (DISTRO) poky, для которого принятое по умолчанию значение PREMIRRORS указано в файле conf/distro/poky.conf репозитория meta-poky. Можно добавить нужные серверы в начало списка, указав их в файле local.conf в каталоге сборки, как показано ниже.

```
PREMIRRORS_prepend = "\
git://.*/.* http://www.yoctoproject.org/sources/ \n \
ftp://.*/.* http://www.yoctoproject.org/sources/ \n \
http://.*/.* http://www.yoctoproject.org/sources/ \n \
https://.*/.* http://www.yoctoproject.org/sources/ \n"
```

Эти изменения заставят систему сборки перехватывать запросы Git, FTP, HTTP, HTTPS и направлять их на указанное зеркало http://. Можно использовать URL типа file:// для указания локальных или сетевых каталогов.

PRIORITY

Указывает уровень важности пакета. Переменная PRIORITY считается частью политики дистрибутива, поскольку важность любого задания зависит от целей дистрибутива. Поэтому PRIORITY обычно не указывается в заданиях. Переменная может принимать значения required (требуется), standard (стандартный уровень), extra (дополнение) и optional (не обязательно, используется по умолчанию).

PRIVATE_LIBS

Задаёт установленные в задании библиотеки, которые следует игнорировать распознавателю общих библиотек системы сборки OE. Переменная обычно применяется в тех случаях, когда создаваемые заданием программы имеют свои версии библиотек, обычно создаваемых другим заданием. В таких случаях следует исключить зависимость от пакета со своими библиотеками не связанных с ним пакетов, которые обычно зависят от стандартных версий библиотек. Библиотеки в этой переменной следует указывать именами файлов. Ниже приведен пример из задания Firefox в meta-browser.

```
PRIVATE_LIBS = "libmozjs.so \
                libxpcom.so \
                libnspr4.so \
                libxul.so \
                libmozalloc.so \
                libplc4.so \
                libplds4.so"
```

Дополнительная информация представлена в разделе Automatically Added Runtime Dependencies [1].

PROVIDES

Список псевдонимов, под которыми может быть известно определённое задание. По умолчанию значение PN для задания неявно уже присутствует в списке PROVIDES. Если задание использует PROVIDES, дополнительные псевдонимы являются синонимами для задания и могут быть полезны для выполнения зависимостей других заданий в процессе сборки, указанных в DEPENDS.

В задании libav_0.8.11.bb указано PROVIDES += "libpostproc", в результате чего libav имеет псевдоним libpostproc.

Кроме дополнительных имён для заданий механизм PROVIDES применяется также для реализации виртуальных целей, которые являются именами, соответствующими некой определённой функциональности (например, ядро Linux). Задания, обеспечивающие соответствующую функциональность, указывают виртуальную цель в PROVIDES. Задания, зависящие от рассматриваемой функциональности, могут включать виртуальную цель в DEPENDS, оставляя выбор поставщика открытым. Обычно виртуальные цели имеют имена вида virtual/function (например, virtual/kernel). Символ дробной черты (/) просто является частью имени и не имеет синтаксического значения. Переменная PREFERRED_PROVIDER служит для выбора конкретного задания, обеспечивающего виртуальную цель.

Имеется соответствующий механизм для виртуальных зависимостей (пакетов) во время выполнения. Однако этот механизм не зависит от какой-либо особой функциональности обычного назначения переменных. Например, VIRTUAL-RUNTIME_dev_manager указывает пакет, который управляет каталогом /dev. Установка «предпочтительного поставщика» для зависимостей во время исполнения так же проста, как назначение в конфигурационном файле VIRTUAL-RUNTIME_dev_manager = "udev".

PRSERV_HOST

Хост и порт сетевой службы PR. В файле conf/local.conf.sample.extended каталога исходных кодов переменная задана в виде PRSERV_HOST = "localhost:0". Эту переменную нужно устанавливать при необходимости автоматического запуска локальной службы PR. Можно указать также значения для удалённой службы PR.

PTEST_ENABLED

Управляет включением функциональности тестирования пакетов (ptest) при сборке задания. Переменную не следует задавать напрямую, поскольку управление тестированием пакетов при сборке задаётся добавлением (или удалением) ptest в переменную DISTRO_FEATURES.

PV

Номер версии задания. Обычно номер версии извлекается из имени файла задания. Например, для задания expat_2.0.1.bb переменная PV по умолчанию будет иметь значение 2.0.1. Значение PV обычно не переопределяется в задании, если сборка не является нестабильной (рабочей) из репозитория исходных кодов), например, Git или Subversion.

Значение PV используется по умолчанию для переменной PKGV.

PYTHON_ABI

При использовании в заданиях, наследующих класс distutils3, setuptools3, distutils или setuptools, указывает интерфейс ABI, используемый для Python (по умолчанию "m"). Переменную не нужно задавать, это делает OE.

Система сборки OE применяет ABI для задания имён каталогов, используемых при установке заголовков и библиотек Python в (например .../python3.3m/...).

Задания, наследующие класс distutils, при кросс-сборке также используют эту переменную для поиска заголовков и библиотек версии Python, для которой предназначено расширение.

PYTHON_PN

При использовании в заданиях, наследующих класс distutils3, setuptools3, distutils или setuptools, указывает старшую версию Python для сборки. Для Python 2.x переменная PYTHON_PN будет иметь значение python2, для Python 3.x - python3. Если переменная не установлена, система сборки OE сделает это автоматически.

Переменная позволяет заданиям использовать общую инфраструктуру, в форме DEPENDS += "${PYTHON_PN}-native", где PYTHON_PN указывает версию зависимости.

R

RANLIB

Сокращённая команда и аргументы для запуска ranlib.

RCONFLICTS

Список пакетов, конфликтующих с другими пакетами. Установка пакетов будет блокироваться до устранения конфликтов. Как и другие переменные управления пакетами, RCONFLICTS нужно всегда указывать с переопределением имени пакета, например, RCONFLICTS_${PN} = "another_conflicting_package_name".

BitBake поддерживает указание зависимостей с учётом версии. Синтаксис зависит от формата пакетов, но BitBake скрывает эти различия. Базовый синтаксис указания версии имеет вид RCONFLICTS_${PN} = "package (operator version)". Оператором может служить =, <, >, <= или >=.

Например, для указания зависимости от версии 1.2 или выше для пакета foo можно задать RCONFLICTS_${PN} = "foo (>= 1.2)".

RDEPENDS

Список зависимостей пакета во время выполнения. Эти зависимости указывают пакеты, которые должны быть установлены для корректной работы данного пакета. Например, приведённая ниже строка указывает зависимость работы пакета foo от наличия установленных пакетов bar и baz.

```
RDEPENDS_foo = "bar baz"
```

Наиболее распространённые зависимости пакетов во время исполнения обнаруживаются и добавляются автоматически, поэтому в большинстве задания переменная RDEPENDS не нужна. Зависимости более подробно рассмотрены в разделе Automatically Added Runtime Dependencies [1].

Практическое влияние приведённого выше примера RDEPENDS состоит в том, что пакеты bar и baz объявляются зависимостями в пакете foo, когда он будет записан одной из задач do_package_write_*. Способ выполнения этого зависит от используемого формата пакета, который определяется переменной PACKAGE_CLASSES. Когда соответствующий менеджер устанавливает пакет, он знает, что нужно установить пакеты, от которых тот зависит.

Чтобы обеспечить сборку пакетов bar и baz, предыдущий пример RDEPENDS также вызывает добавление зависимости задач. Это зависимость задачи do_build для задания (не путайте с do_compile) от задач do_package_write_* заданий сборки bar и baz.

Имена в переменной RDEPENDS должны указывать другие пакеты - это не могут быть имена заданий. Хотя имена пакетов и заданий могут совпадать, важно отметить, что в переменной RDEPENDS указываются именно пакеты. Подробный список пакетов, создаваемых заданием, можно узнать из переменной PACKAGES.

Поскольку переменная RDEPENDS применяется к собираемым пакетам, её следует использовать с именем пакета в конце (помните, что одно задание может собирать несколько пакетов). Предположим, например, сборку пакета для разработки, который зависит от perl. В этом случае используется оператор RDEPENDS вида

```
RDEPENDS_${PN}-dev += "perl"
```

В этом примере пакет для разработки зависит от пакета perl. Таким образом, RDEPENDS включает ${PN}-dev как часть переменной. RDEPENDS_${PN}-dev включает ${PN} по умолчанию, что устанавливается в конфигурационном файле BitBake (meta/conf/bitbake.conf). Соблюдайте осторожность, чтобы случайно не удалить ${PN} при изменении RDEPENDS_${PN}-dev. Следует также применять оператор +=, а не просто =.

Имена пакетов в переменной RDEPENDS должны указывать пакеты в том виде, как они представлены в PACKAGES. Переменная PKG разрешает использовать для финального пакета другое имя (например, класс debian использует её для переименования пакетов), но это окончательное имя пакета не может использоваться в RDEPENDS, поскольку эта переменная предполагает независимость от применяемого формата пакетов.

Программа BitBake, используемая системой сборки OE, поддерживает в зависимостях указание версий. Хотя синтаксис версии зависит от формата упаковки, BitBake скрывает эту зависимость. Базовый синтаксис указания версии имеет вид RDEPENDS_${PN} = "package (operator version)".

Оператором может служить =, <, >, <= или >=, версия указывается номером. Можно использовать переменную EXTENDPKGV для полного указания версии пакета. Например, для указания зависимости от версии 1.2 или выше для пакета foo можно задать RDEPENDS_${PN} = "foo (>= 1.2)".

Информация о зависимостях во время сборки задаётся переменной DEPENDS. Дополнительные сведения о задачах и зависимостях приведены в разделах Tasks и Dependencies [4].

REQUIRED_DISTRO_FEATURES

При наследовании класса distro_features_check эта переменная указывает свойства дистрибутива, которые должны присутствовать в текущей конфигурации сборки задания. Если эта переменная указывает свойства, не указанные в DISTRO_FEATURES для текущей конфигурации, это будет вызывать ошибку и прекращение сборки.

RM_WORK_EXCLUDE

При включённом классе rm_work переменная задаёт список заданий, рабочие каталоги которых не следует удалять (см. параграф 6.114. rm_work.bbclass).

ROOT_HOME

Задаёт корневой каталог root. По умолчанию каталог указывается в конфигурационном файле BitBake как ROOT_HOME ??= "/home/root". Заданное по умолчанию значение вероятно будет использоваться, поскольку некоторые встраиваемые системы предпочитают создавать корневую файловую систему, доступную лишь для чтения, и хранить перезаписываемые данные в отдельном месте. Переменную можно переопределить на любом уровне или в файле local.conf. Поскольку для заданного по умолчанию значения используется «мягкое» назначение (??"), для переопределения можно указать любой из приведённых ниже вариантов.

```
ROOT_HOME = "/root"
ROOT_HOME ?= "/root"
```

ROOTFS

Указывает образ файловой системы для включения в корневую файловую системы. Переменная является необязательной и применяется с классом image-live.

ROOTFS_POSTINSTALL_COMMAND

Задаёт список функций для вызова после установки пакетов системой сборки OE. Функции разделяются символом точки с запятой (;), например ROOTFS_POSTINSTALL_COMMAND += "function; ... ".

Если требуется передать путь к корневой файловой системе команде с функцией, можно использовать переменную ${IMAGE_ROOTFS}, указывающую каталог, который станет корневой файловой системой образа.

ROOTFS_POSTPROCESS_COMMAND

Задаёт список функций для однократного вызова системой сборки OE при создании корневой файловой системы. Функции разделяются символом точки с запятой (;), например, ROOTFS_POSTPROCESS_COMMAND += "function; ... ". Если нужно передать путь к корневой файловой системе команде с функцией, можно использовать переменную ${IMAGE_ROOTFS}, указывающую каталог, который станет корневой файловой системой образа.

ROOTFS_POSTUNINSTALL_COMMAND

Задаёт список функций, вызываемых после удаления системой сборки OE ненужных приложений. Когда менеджер пакетов при работе системы (runtime) отключён. в образе, удаляется несколько пакетов, включая base-passwd, shadow и update-alternatives. Функции разделяются символом точки с запятой (;), например, ROOTFS_POSTUNINSTALL_COMMAND += "function; ... ".

Если требуется передать путь к корневой файловой системе команде с функцией, можно использовать переменную ${IMAGE_ROOTFS}, указывающую каталог, который станет корневой файловой системой образа.

ROOTFS_PREPROCESS_COMMAND

Задаёт список функций, вызываемых перед созданием корневой файловой системы системой сборки OE. Функции разделяются точкой с запятой, например, ROOTFS_PREPROCESS_COMMAND += "function; ... ".

Если нужно передать путь к корневой файловой системе при вызове функции, можно использовать переменную ${IMAGE_ROOTFS}, указывающую каталог, который станет корневой файловой системой образа.

RPROVIDES

Список псевдонимов имён пакетов, которые полезны для выполнения зависимостей во время работы как при сборке, так и на целевой платформе (как указано переменной RDEPENDS). Реальное имя пакета неявно уже присутствует в списке RPROVIDES.

Как и другие переменных управления пакетами, RPROVIDES всегда нужно использовать в сочетании с переопределением имени пакета, например, RPROVIDES_${PN} = "widget-abi-2".

RRECOMMENDS

Список пакетов для расширения применимости собираемого пакета. Собираемый пакет не зависит от этого списка, а применяет их для расширения сферы использования. Для задания зависимостей при работе пакетов служит переменная RDEPENDS.

Менеджер пакетов будет автоматически устанавливать пакеты из списка RRECOMMENDS при установке собираемого пакета. Однако можно предотвратить установку пакетов из списка с помощью переменных BAD_RECOMMENDATIONS, NO_RECOMMENDATIONS и PACKAGE_EXCLUDE.

Пакеты, указанные в RRECOMMENDS не обязательно создавать в действительности, однако должно существовать задание для каждого из этих пакетов в переменной PACKAGES, PACKAGES_DYNAMIC или RPROVIDES, иначе возникнет ошибка. Если задание имеется, но пакет не создаётся, сборка продолжится без ошибки.

Поскольку переменная RRECOMMENDS применяется к собираемым пакетам, нужно прикреплять переопределение к переменной для указания конкретного пакета, применимость которого будет расширена. Предположим, например, сборку пакета разработки, которая расширяется для поддержки беспроводных сетей. С этом случае указывается

```
RRECOMMENDS_${PN}-dev += "wireless_package_name"
```

В этом примере имя пакета (${PN}-dev) должно представляться как в пространстве имён PACKAGES до какого-либо переименования выходного пакета классами, такими как debian.bbclass.

BitBake, использующий систему сборки OE, поддерживает указание нужной версии. Хотя синтаксис такого указания зависит от формата упаковки, BitBake скрывает эти различия от вас. Ниже показан базовый синтаксис задания версии в переменной RRECOMMENDS.

```
RRECOMMENDS_${PN} = "package (operator version)"
```

В качестве оператора можно использовать =, <, >, <=, >=.

Например, приведённая ниже строка рекомендует версии не ниже 1.2 для пакета foo.

```
RRECOMMENDS_${PN} = "foo (>= 1.2)"
```

RREPLACES

Список пакетов, заменяемых данным пакетом. Менеджер пакетов применяет эту переменную для определения пакетов, которые следует установить взамен других в процессе обновления. Для одновременного удаления других пакетов нужно добавить их имена в переменную RCONFLICTS. Как и другие переменные управления пакетами, RREPLACES требуется применять вместе с переопределением имени пакета, например, RREPLACES_${PN} = "other_package_being_replaced".

BitBake поддерживает указание версий при замене. Синтаксис зависит от формата пакетов, однако BitBake маскирует эти различия. Общий синтаксис указания версий имеет вид RREPLACES_${PN} = "package (operator version)". В качестве оператора можно использовать =, <, >, <=, >=.

Например, строка RREPLACES_${PN} = "foo (>= 1.2)" рекомендует версии не ниже 1.2 для пакета foo.

RSUGGESTS

Список дополнительных пакетов, которые можно предложить при установке менеджеру пакетов во время инсталляции данного пакета. Функциональность поддерживают не все менеджеры пакетов. Как и другие переменные управления пакетами, RSUGGESTS требуется применять вместе с переопределением имени пакета.

```
RSUGGESTS_${PN} = "useful_package another_package"
```

S

S

Место в каталоге сборки, где размещаются неупакованные исходные коды задания. По умолчанию это каталог ${WORKDIR}/${BPN}-${PV}, где ${BPN} - базовое имя задания, а ${PV} - версия задания. Если из архива исходных кодов файлы извлечены в каталог, имя которого отличается от ${BPN}-${PV}, или исходный код получен из SCM (например, Git или Subversion), нужно установить переменную S в задании так, чтобы система сборки OE знала, где искать неупакованные исходные файлы.

Предположим, например, что каталог верхнего уровня в дереве исходных кодов называется poky, а принятым по умолчанию каталогом сборки служит poky/build. В этом случае рабочим каталогом системы сборки, используемым для хранения распакованного задания для db, будет

poky/build/tmp/work/qemux86-poky-linux/db/5.1.19-r3/db-5.1.19

Распакованные исходные файлы размещаются в каталоге db-5.1.19.

В следующем примере предполагается репозиторий Git. По умолчанию такие репозитории клонируются в ${WORKDIR}/git задачей do_fetch. Поскольку этот путь отличается от принятого по умолчанию значения S, нужно указать его как место размещения исходных кодов

```
SRC_URI = "git://path/to/repo.git"
S = "${WORKDIR}/git"
```

SANITY_REQUIRED_UTILITIES

Задаёт список консольных утилит, которые следует проверять во время начального тестирования работоспособности при запуске BitBake. Если какая-либо из утилит не установлена на сборочном хосте, BitBake сразу прерывает работу с возвратом ошибки.

SANITY_TESTED_DISTROS

Список идентификаторов дистрибутивов, для которых система сборки была протестирована. Идентификатор состоит из идентификатора дистрибутива, за которым следует номер выпуска, указанный инструментом lsb_release или прочитанный из файла /etc/lsb-release. Элементы списка задаются по одному в строках, завершающихся символом перевода строки (\n). Если строка SANITY_TESTED_DISTROS не пуста, а текущее значение NATIVELSBSTRING не присутствует в списке, система сборки выдаёт предупреждение, указывающее, что текущий дистрибутив не был проверен как хост сборки.

SDK_ARCH

Целевая архитектура для SDK. Обычно эта переменная не устанавливается напрямую и взамен применяется SDKMACHINE.

SDK_DEPLOY

Созданный и используемый классом populate_sdk_base каталог, в котором разворачивается SDK. Класс populate_sdk_base определяет переменную как SDK_DEPLOY = "${TMPDIR}/deploy/sdk".

SDK_DIR

Родительский каталог, используемый системой сборки OE при выводе SDK. Класс populate_sdk_base определяет переменную как SDK_DIR = "${WORKDIR}/sdk". Этот каталог является временным как часть WORKDIR, окончательный вывод помещается в SDK_DEPLOY.

SDK_EXT_TYPE

Управляет копированием элементов общего состояния в eSDK. При заданном по умолчанию значении full копируются все нужные элементы общего состояния, значение minimal оставляет элементы за пределами SDK.

При установке значения minimal нужно убедиться в установке переменной SSTATE_MIRRORS в конфигурации SDK для выборки требуемых элементов.

SDK_HOST_MANIFEST

Файл манифеста для хостовой части SDK, содержащий все установленные пакеты, которые делают хост частью SDK. Пакеты указываются по одному в строке в форме packagename packagearch version. Класс populate_sdk_base определяет файл манифеста как SDK_HOST_MANIFEST = "${SDK_DEPLOY}/${TOOLCHAIN_OUTPUTNAME}.host.manifest". Расположение файла выводится из переменных SDK_DEPLOY и TOOLCHAIN_OUTPUTNAME.

SDK_INCLUDE_PKGDATA

Значение 1 задаёт включение packagedata для всех заданий цели world в расширяемом SDK. Это позволяет команде devtool search находить эти задания, а также позволяет команде devtool add более эффективно отображать зависимости.

Включение SDK_INCLUDE_PKGDATA существенно увеличивает время сборки, потому что требуется собирать все задания из world. Размер eSDK также возрастает.

SDK_INCLUDE_TOOLCHAIN

Значение 1 задаёт включение инструментария в расширяемый SDK. Это особенно полезно при установке в SDK_EXT_TYPE значения minimal для снижения размера SDK, когда есть также необходимость в инструментарии. Например, может быть нужно включение инструментария из IDE или другого набора без дополнительных действий по установке инструментов. По умолчанию переменная имеет значение 0, если для SDK_EXT_TYPE выбрано значение minimal, и 1 для SDK_EXT_TYPE = "full".

SDK_INHERIT_BLACKLIST

Список классов для глобального удаления из значения INHERIT в конфигурации расширяемых SDK. Принятое по умолчанию значение задаёт класс populate-sdk-ext в форме SDK_INHERIT_BLACKLIST ?= "buildhistory icecc". Некоторые классы обычно не применимы в контексте расширяемых SDK и из можно отключить этой переменной.

Дополнительные сведения о настройке расширяемых SDK приведена в разделе Configuring the Extensible SDK [7].

SDK_LOCAL_CONF_BLACKLIST

Список переменных, которые не разрешено передавать из системы сборки OE в конфигурацию расширяемых SDK. Обычно эти переменные связаны с машиной, на которой работает система сборки, и поэтому могут создавать проблемы в расширяемых SDK. По умолчанию переменная устанавливается классом populate-sdk-ext в форме

```
CONF_VERSION
BB_NUMBER_THREADS
BB_NUMBER_PARSE_THREADS
PARALLEL_MAKE
PRSERV_HOST
SSTATE_MIRRORS
DL_DIR
SSTATE_DIR
TMPDIR
BB_SERVER_TIMEOUT
```

Дополнительные сведения о настройке расширяемых SDK приведена в разделе Configuring the Extensible SDK [7].

SDK_LOCAL_CONF_WHITELIST

Список переменных, передаваемых из системы сборки OE в конфигурацию расширяемых SDK. По умолчанию пустой список устанавливается классом populate-sdk-ext. Этот список переопределяет переменные, заданные с помощью SDK_LOCAL_CONF_BLACKLIST, а также другие переменные, автоматически включаемые в «чёрный список» символом / в начале значения, который обычно указывает путь и поэтому не пригоден для системы, где будет устанавливаться SDK.

Дополнительные сведения о настройке расширяемых SDK приведена в разделе Configuring the Extensible SDK [7].

SDK_NAME

Базовое имя для выходных файлов SDK, которое выводится из переменных DISTRO, TCLIBC, SDK_ARCH, IMAGE_BASENAME и TUNE_PKGARCH, как показано ниже.

```
SDK_NAME = "${DISTRO}-${TCLIBC}-${SDK_ARCH}-${IMAGE_BASENAME}-${TUNE_PKGARCH}"
```

SDK_OS

Задаёт операционную систему, для которой собирается SDK (по умолчанию значение BUILD_OS).

SDK_OUTPUT

Место, используемое системой сборки OE для вывода SDK. Класс populate_sdk_base определяет переменную, как показано ниже.

```
SDK_DIR = "${WORKDIR}/sdk"
SDK_OUTPUT = "${SDK_DIR}/image"
SDK_DEPLOY = "${DEPLOY_DIR}/sdk"
```

Каталог SDK_OUTPUT служит временным каталогом, являясь частью WORKDIR через SDK_DIR. Финальный вывод записывается в SDK_DEPLOY.

SDK_PACKAGE_ARCHS

Задаёт список вариантов архитектуры, совместимых с машиной SDK. Переменная устанавливается автоматически и обычно её не нужно редактировать. Элементы разделяются пробелами и указаны по приоритету. По умолчанию установлено значение "all any noarch ${SDK_ARCH}-${SDKPKGSUFFIX}".

SDK_POSTPROCESS_COMMAND

Задаёт список функций, однократно вызываемых системой сборки OE при создании SDK. Функции разделяются точкой с запятой, например, SDK_POSTPROCESS_COMMAND += "function; ... ".

Если нужно передать путь к SDK при вызове функции, можно применять переменную ${SDK_DIR}, указывающую каталог, который система сборки OE будет использовать для вывода SDK.

SDK_PREFIX

Префикс двоичных файлов инструментария для заданий nativesdk (по умолчанию ${SDK_SYS}-). Система сборки OE использует значение SDK_PREFIX для установки TARGET_PREFIX при сборке заданий nativesdk.

SDK_RECRDEP_TASKS

Список задач общего состояния, добавленных в расширяемый SDK. По умолчанию добавляются задачи do_populate_lic, do_package_qa, do_populate_sysroot и do_deploy, несмотря на принятое по умолчанию значение "". Для добавления других задач нужно указать их в переменной (например, при создании задач, которые нужно включить в SDK_TARGETS).

SDK_SYS

Указывает систему, для которой будет собираться SDK, включая архитектуру и ОС. Система сборки OE автоматически устанавливает эту переменную на основе SDK_ARCH, SDK_VENDOR и SDK_OS.

SDK_TARGET_MANIFEST

Файл манифеста для целевой части SDK, где указаны все установленные пакеты, составляющие целевую часть SDK. Пакеты указываются по одному в строке в форме packagename packagearch version. Класс populate_sdk_base задаёт файл как SDK_TARGET_MANIFEST = "${SDK_DEPLOY}/${TOOLCHAIN_OUTPUTNAME}.target.manifest". Местоположение выводится из переменных SDK_DEPLOY и TOOLCHAIN_OUTPUTNAME.

SDK_TARGETS

Список целей для установки из общего состояния как части установки стандартного или расширяемого SDK. По умолчанию установлено значение "${PN}" (образ, из которого собран SDK). Переменная является внутренней и обычно не меняется.

SDK_TITLE

Заголовок, выводимый при запуске установщика SDK. По умолчанию этот заголовок основан на значении переменной DISTRO_NAME или DISTRO и задаётся классом populate_sdk_base в форме SDK_TITLE ??= "${@d.getVar('DISTRO_NAME') or d.getVar('DISTRO')} SDK". Для дистрибутива poky переменная SDK_TITLE имеет значение Poky (Yocto Project Reference Distro). Способы смены принятого по умолчанию значения описаны в разделе Changing the Extensible SDK Installer Title [7].

SDK_UPDATE_URL

Необязательный идентификатор URL сервера обновления для расширяемого SDK. Если переменная установлена, её значение используется в качестве принятого по умолчанию сервера обновления командой devtool sdk-update.

SDK_VENDOR

Задаёт имя производителя SDK.

SDK_VERSION

Задаёт версию SDK. Конфигурационные файлы дистрибутивов (например, /meta-poky/conf/distro/poky.conf) задают переменную в виде SDK_VERSION = "${@d.getVar('DISTRO_VERSION').replace('snapshot-${DATE}','snapshot')}".

SDKEXTPATH

Принятый по умолчанию каталог для установки расширяемого SDK. По умолчанию эта переменная базируется на переменной DISTRO и задаётся классом populate_sdk_base как SDKEXTPATH ??= "~/${@d.getVar('DISTRO')}_sdk". Для дистрибутива poky переменная имеет значение poky_sdk.

Смена принятого по умолчанию каталога описана в разделе Changing the Default SDK Installation Directory [7].

SDKIMAGE_FEATURES

Эквивалент IMAGE_FEATURES для SDK, созданных из образа по команде bitbake -c populate_sdk imagename.

SDKMACHINE

Машина, для которой собирается SDK. Иными словами, SDK собирается так, чтобы работать на целевой платформе, заданной значением SDKMACHINE, указывающим нужный файл .conf из conf/machine-sdk/.

Можно использовать i686 и x86_64 в качестве значения этой переменной. Используемое по умолчанию значение i686 задаётся в файле local.conf сборочного каталоге.

```
SDKMACHINE ?= "i686"
```

Не следует устанавливать переменную SDKMACHINE в конфигурационном файле дистрибутива, поскольку такая установка не будет работать.

SDKPATH

Указывает путь, предлагаемый пользователю для инсталляции SDK, созданного системой сборки OE. Путь представляется как принятое по умолчанию место для установки SDK при работе сценария инсталляции SDK. Предлагаемый путь можно переопределить при работе сценария.

SDKTARGETSYSROOT

Полный путь к каталогу sysroot, используемый для кросс-компиляции в SDK, как при инсталляции в принятый по умолчанию SDKPATH.

SECTION

Секция, в которой следует разделить пакеты по категориям. Переменную могут использовать менеджеры пакетов.

SELECTED_OPTIMIZATION

Задаёт флаги оптимизации, передаваемые компилятору C при сборке для цели. Флаги передаются в принятом по умолчанию значении TARGET_CFLAGS. Переменная SELECTED_OPTIMIZATION принимает значение FULL_OPTIMIZATION, если на задано DEBUG_BUILD = "1". В последнем случае используется значение DEBUG_OPTIMIZATION.

SERIAL_CONSOLE

Указывает последовательную консоль (TTY) для использования getty. В значении следует указать скорость, затем устройство TTY через пробел. Указать можно лишь один порт, например, SERIAL_CONSOLE = "115200 ttyS0". Переменная SERIAL_CONSOLE отменена и взамен следует использовать SERIAL_CONSOLES.

SERIAL_CONSOLES

Указывает последовательные консоли (TTY) для использования getty. В значении следует указать скорость, затем устройство TTY через точку с запятой. Порты разделяются пробелами, например, SERIAL_CONSOLES = "115200;ttyS0 115200;ttyS1".

SERIAL_CONSOLES_CHECK

Задаёт последовательные консоли, которые должны быть указаны в SERIAL_CONSOLES, для сравнения с /proc/console перед включением и использованием getty. Переменная поддерживает псевдонимы в формате <device>:<alias>. Если устройство указано как sclp_line0 в /dev/ и ttyS0 задана в /proc/console, нужно указать SERIAL_CONSOLES_CHECK = "slcp_line0:ttyS0". Переменная поддерживается только с SysVinit (не systemd).

SIGGEN_EXCLUDE_SAFE_RECIPE_DEPS

Список зависимостей задания, которые не должны учитываться при определении подписей задач из одного задания, когда они зависят от задач из другого задания. Например, SIGGEN_EXCLUDE_SAFE_RECIPE_DEPS += "intone->mplayer2", где intone зависит от mplayer2. Можно применять специальный маркер * слава от зависимости, которому будут соответствовать все задания, кроме указанного справа, например, SIGGEN_EXCLUDE_SAFE_RECIPE_DEPS += "*->quilt-native". Здесь все задания, кроме quilt-native игнорируют подписи задач из quilt-native при определении своих подписей. Использование этой переменной является одним из способов удаления зависимостей, влияющих на подписи задач и требующих повторной сборки при изменении заданий. При добавлении в этот список неподходящего задания при работе программ могут возникать ошибки, если интерфейс задания будет изменён после сборки другого задания.

SIGGEN_EXCLUDERECIPES_ABISAFE

Список заданий, которые стабильны и никогда не меняются. ABI для заданий из этого списка представляется выводом задач, работающих для сборки задания. Использование этой переменной является одним из способов удаления зависимостей, влияющих на подписи задач и требующих повторной сборки при изменении заданий. При добавлении в этот список неподходящего задания при работе программ могут возникать ошибки, если интерфейс задания будет изменён после сборки другого задания.

SITEINFO_BITS

Задаёт разрядность CPU в целевой системе (32 или 64).

SITEINFO_ENDIANNESS

Задаёт порядок байтов на целевой системе и может принимать значение le (little-endian) или be (big-endian).

SKIP_FILEDEPS

Включает удаление всех файлов из раздела Provides пакета RPM. Удаление этих файлов требуется для пакетов, включающих заранее собранные исполняемые файлы и библиотеки, такие как libstdc++ и glibc. Чтобы включить удаление, нужно в файле сборочного каталога conf/local.conf задать SKIP_FILEDEPS = "1".

SOC_FAMILY

Группирует машины на основе семейства SOC[1] и обычно задаётся в общем файле .inc, включаемом в конфигурационные файлы всех машин. Нужно включить файл conf/machine/include/soc-family.inc, чтобы эта переменная появилась в MACHINEOVERRIDES.

SOLIBS

Задаёт суффикс для общих библиотек на целевой платформе. Принятый по умолчанию суффикс .so для систем Linux определён в файле meta/conf/bitbake.conf. Эта переменная используется в принятых по умолчанию значениях FILES_${PN}.

SOLIBSDEV

Задаёт суффикс для символьной ссылки на общую библиотеку разработки для целевой платформы. Принятый по умолчанию суффикс .so для систем Linux определён в файле meta/conf/bitbake.conf. Эта переменная используется в принятых по умолчанию значениях FILES_${PN}-dev.

SOURCE_MIRROR_FETCH

При выборке файлов для создания зеркала исходных кодов установка SOURCE_MIRROR_FETCH = "1" в файле local.conf обеспечивает выбор исходных кодов всех заданий независимо от совместимости с конфигурацией. Задание считается не совместимым с настроенной в данный момент машиной, когда любая (или обе) из переменных COMPATIBLE_MACHINE и COMPATIBLE_HOST задают совместимость с машиной, отличной от выбранной и хоста. Не следует задавать переменную SOURCE_MIRROR_FETCH без наличия зеркала исходных кодов (т. е. при обычной сборке).

SOURCE_MIRROR_URL

Задаёт свою переменную PREMIRRORS, в соответствии с которой начинается выборка исходных кодов до обращения к SRC_URI. Для использования этой переменной требуется глобальное наследование класса own-mirrors и предоставление URL для зеркал. Ниже показан базовый синтаксис.

```
INHERIT += "own-mirrors"
SOURCE_MIRROR_URL = "http://example.com/my_source_mirror"
```

В переменной SOURCE_MIRROR_URL можно задать лишь один идентификатор URL.

SPDXLICENSEMAP

Отображает общепринятые имена лицензий на идентификаторы SPDX из meta/files/common-licenses/. Подробное описание отображений приведено в файле meta/conf/licenses.conf. См. также описание переменной LICENSE.

SPECIAL_PKGSUFFIX

Список префиксов для PN, используемых системой сборки OE при создании вариантов заданий или пакетов. Список задаёт префиксы, вырезаемые при некоторых обстоятельствах (например, создание переменной BPN).

SPL_BINARY

Тип файла для загрузчика SPL[2], используемого некоторыми устройствами (например, плата BeagleBone) для загрузки. В таких случаях можно объявить тип двоичного файла SPL в файле u-boot.inc, используемом в задании U-Boot. Тип файла SPL по началу имеет значение null, заданное в файле u-boot.inc, как показано ниже.

```
# Some versions of u-boot build an SPL (Second Program Loader) image that
# should be packaged along with the u-boot binary as well as placed in the
# deploy directory.  For those versions they can set the following variables
# to allow packaging the SPL.
SPL_BINARY ?= ""
SPL_BINARYNAME ?= "${@os.path.basename(d.getVar("SPL_BINARY"))}"
SPL_IMAGE ?= "${SPL_BINARYNAME}-${MACHINE}-${PV}-${PR}"
SPL_SYMLINK ?= "${SPL_BINARYNAME}-${MACHINE}"
```

Переменная SPL_BINARY помогает формировать переменные SPL_*, используемые системой сборки OE.

Пример конфигурации BeagleBone приведен в разделе Creating a new BSP Layer Using the bitbake-layers Script [6].

SRC_URI

Список исходных файлов (локальных или удалённых). Эта переменная говорит системе сборки OE, что и как нужно извлекать для сборки. Например, если заданию или файлу дополнения нужно лишь загрузить архив из Internet, это задание или файл дополнения использует одну запись SRC_URI. Если же нужно, например, извлечь архив, применить два исправления и включить пользовательский файл, задание или файл дополнения будут включать 4 экземпляра этой переменной.

Ниже приведен список поддерживаемых протоколов URI. Эти протоколы сильно зависят от конкретных субмодулей BitBake Fetcher. В зависимости от применяемого сборщика используются разные параметры URL. Подробное описание работы со сборщиками приведено в разделе Fetchers [4].

- file:// извлекает файлы, которые обычно поставляются с метаданными с локальной машины (например, patch-файлы). Путь задаётся относительно значения переменной FILESPATH. Таким образом, система сборки просматривает по порядку перечисленные ниже каталоги, которые предполагаются подкаталогами места размещения файла задания (.bb) или дополнения (.bbappend):
  - ${BPN} - базовое имя задания без специальных суффиксов и номера версии;
  - ${BP} - ${BPN}-${PV} - базовое имя задания и версия, но без специального суффикса (имя пакета);
  - *files* - файлы в каталоге files, который размещён. вместе с файлом задания или дополнения.

[1] System On Chip - система на кристалле.

[2] Secondary Program Loader - вторичный загрузчик программ.

Если вы хотите, чтобы система сборки забрала файлы, указанные оператором SRC_URI в вашем файле дополнения, нужно расширить переменную FILESPATH, используя значение FILESEXTRAPATHS из вашего файла дополнения.

- bzr:// извлекает файлы из репозитория управления версиями Bazaar.
- git:// извлекает файлы из репозитория управления версиями Git.
- osc:// извлекает файлы из репозитория управления версиями OSC (OpenSUSE Build service).
- repo:// извлекает файлы из репозитория repo (Git).
- ccrc:// извлекает файлы из репозитория ClearCase.
- http:// извлекает файлы из Internet по протоколу http.
- https:// извлекает файлы из Internet по протоколу https.
- ftp:// извлекает файлы из Internet по протоколу ftp.
- cvs:// извлекает файлы из репозитория управления версиями CVS.
- hg:// извлекает файлы из репозитория управления версиями (hg).
- p4:// извлекает файлы из репозитория управления версиями Perforce (p4).
- ssh:// извлекает файлы из защищённой среды (secure shell).
- svn:// - извлекает файлы из репозитория управления версиями Subversion (svn).

Имеются стандартные и специфические для заданий SRC_URI. Ниже перечислены стандартные.

- apply - нужно ли применять исправления (по умолчанию apply - нужно).
- striplevel - уровень разрешения (striplevel) при наложении исправлений (по умолчанию 1).
- patchdir - каталог, в котором следует применять исправления (по умолчанию ${S}).

Ниже приведены варианты для заданий по сборке кода из системы контроля версий.

- mindate - применять исправление лишь в случаях, когда SRCDATE не меньше mindate.
- maxdate - применять исправление лишь в случаях, когда SRCDATE не больше maxdate.
- minrev - применять исправление лишь в случаях, когда SRCREV не меньше minrev.
- maxrev - применять исправление лишь в случаях, когда SRCREV не больше maxrev.
- rev - применять исправление лишь в случаях, когда SRCREV = rev.
- notrev - применять исправление лишь в случаях, когда SRCREV отличается от rev.

Имеется также ряд дополнительных опций, перечисленных ниже.

- unpack - управляет распаковкой файла, если он является архивом (по умолчанию архивы распаковываются).
- destsuffix - помещать файл (или извлекать его содержимое) в указанный подкаталог WORKDIR при использовании сборщика Git.
- subdir - помещать файл (или извлекать его содержимое) в указанный подкаталог WORKDIR при использовании локального сборщика (file://).
- localdir - помещать файл (или извлекать его содержимое) в указанный подкаталог WORKDIR при использовании сборщика CVS.
- subpath - ограничивать извлечение конкретным путём в дереве при использовании сборщика Git.
- name - задаёт имя, применяемое для привязки к контрольным суммам SRC_URI при указании в SRC_URI нескольких файлов.
- downloadfilename - задаёт имя, используемое при записи загружаемого файла.

SRC_URI_OVERRIDES_PACKAGE_ARCH

По умолчанию система сборки OE автоматически обнаруживает наличие в SRC_URI машинозависимых файлов и в этом случае меняет значение PACKAGE_ARCH. Установка для переменной значения 0 отключает это.

SRCDATE

Дата исходного кода, применяемого для сборки пакета. Эта переменная применяется только для кодов, извлечённых из SCM.

SRCPV

Возвращает строку версии текущего пакета, используемую при определении значения переменной PV. Переменная SRCPV определена в конфигурационном файле meta/conf/bitbake.conf дерева исходных кодов, как SRCPV = "${@bb.fetch2.get_srcrev(d)}".

Задания, которым нужно определить PV, делают это с помощью SRCPV. Например, задание (ofono_git.bb) в каталоге meta/recipes-connectivity дерева исходных кодов задаёт PV в виде PV = "0.12-git${SRCPV}".

SRCREV

Выпуск исходных кодов, используемых для сборки пакета. Эта переменная применима лишь к исходным кодам Subversion, Git, Mercurial и Bazaar. Если нужно собрать фиксированную версию без запросов к удалённому репозиторию всякий раз, когда BitBake анализирует задание, следует указать в переменной SRCREV полный идентификатор версии, а не просто тег. Информация об ограничениях при наследовании последней версии программы из SRCREV приведена в описании переменной AUTOREV и разделе Automatically Incrementing a Binary Package Revision Number [2].

SSTATE_DIR

Каталог для кэша общих состояний.

SSTATE_MIRROR_ALLOW_NETWORK

Установка значения 1 разрешает выборку из «зеркал», указанных в переменной SSTATE_MIRRORS, даже в случаях, когда выборка из сети отключена установкой BB_NO_NETWORK = "1". применение переменной SSTATE_MIRROR_ALLOW_NETWORK полезно с тех случаях, когда нужно указать в SSTATE_MIRRORS внутренний сервер для общего кэше состояний, но требуется запретить другие выборки из сети.

SSTATE_MIRRORS

Настраивает систему сборки OE на поиск других зеркал с подготовленными объектами данных кэша перед сборкой. Эта переменная работает как сборщики MIRRORS и PREMIRRORS, указывая расположение кэша для поиска общих объектов состояния (sstate).

Можно указать каталог файловой системы или идентификатор URL, такой как HTTP или FTP. Заданное место должно содержать результаты кэширования общего состояния (sstate-cache) из предшествующей сборки. В качестве sstate-cache могут служить результаты сборки на другой машине.

При указании на данные sstate на другой машине, использующей другую версию GCC для естественной сборки, нужно указать в SSTATE_MIRROR регулярное выражение для отображения локального пути поиска в путь на сервере. В путях нужно учитывать значение NATIVELSBSTRING, установленное классом uninative. Например,

отображение локального пути поиска universal-4.9 на серверный путь server_url_sstate_path имеет вид SSTATE_MIRRORS ?= file://universal-4.9/(.*) http://server_url_sstate_path/universal-4.8/\1 \n.

Если зеркало использует такую же структуру, как SSTATE_DIR, нужно добавить PATH в конце, как показано в примере ниже. Система сборки подставит корректный путь поиска в структуре каталогов.

```
SSTATE_MIRRORS ?= "\
file://.* http://someserver.tld/share/sstate/PATH;downloadfilename=PATH \n \
file://.* file:///some-local-dir/sstate/PATH"
```

SSTATE_SCAN_FILES

Управляет списком файлов, сканируемых системой сборки OE для жёстко заданных путей установки. Переменная содержит список разделенных пробелами имён файлов (не путей) с возможностью использования стандартных символов-шаблонов. При сборке система OE создает объект общего состояния (sstate) на первом этапе подготовки sysroot. Объект сканируется на предмет жёстко заданных путей для исходных мест установки. Список файлов, для которых сканируются пути, определяет переменная SSTATE_SCAN_FILES. Обычно задания добавляют файлы для сканирования в переменную SSTATE_SCAN_FILES, а не в переменную, задающую полный набор. Принятый по умолчанию список файлов задаёт класс sstate. Детали процесса даны в описании класса staging.

STAGING_BASE_LIBDIR_NATIVE

Задаёт путь к каталогу /lib в sysroot для сборочного хоста.

STAGING_BASELIBDIR

Задаёт путь к каталогу /lib в sysroot для целевой платформы текущего задания (STAGING_DIR_HOST).

STAGING_BINDIR

Задаёт путь к каталогу /usr/bin внутри sysroot целевой системы, для которой собирается текущее задание (STAGING_DIR_HOST).

STAGING_BINDIR_CROSS

Задаёт путь к каталогу с двоичными сценариями настройки, содержащими конфигурационные данные для других программ, которым нужны библиотеки или включаемые файлы программы, связанной со сценарием. Этот стиль настройки конфигурации сборки в значительной степени заменён pkg-config, поэтому при поддержке pkg-config соответствующей библиотекой рекомендуется использовать pkg-config вместо сценария настройки.

STAGING_BINDIR_NATIVE

Задаёт путь к каталогу /usr/ bin в каталоге sysroot для хоста сборки.

STAGING_DATADIR

Задаёт путь к каталогу /usr/share в sysroot для целевой платформы текущего задания (STAGING_DIR_HOST).

STAGING_DATADIR_NATIVE

Задаёт путь к каталогу /usr/share в каталоге sysroot для хоста сборки.

STAGING_DIR

Помогает создавать для заданий каталоги sysroot, используемые при подготовке пакетов. Информация о подготовке sysroot для заданий приведена в параграфе 7.1.22. do_populate_sysroot, разделах Sharing Files Between Recipes [2] и Configuration, Compilation, and Staging [1], а также в описании переменной SYSROOT_DIRS. Заданиям не следует писать напрямую в каталог STAGING_DIR, поскольку система сборки OE поддерживает каталог автоматически. Задания должны устанавливаться в ${D} задачей do_install в задании, а система сборки OE будет помещать часть файлов в sysroot.

STAGING_DIR_HOST

Указывает путь к каталогу sysroot для системы, на которой выполняется сборка компонент (системы, где размещаются компоненты). Для большинства заданий sysroot - это один из каталогов, куда задача do_populate_sysroot копирует файлы. Исключениями являются задания -native, где задача do_populate_sysroot использует каталог STAGING_DIR_NATIVE. В зависимости от типа задания и цели сборки STAGING_DIR_HOST может принимать одно из двух значений:

- ${STAGING_DIR}/${MACHINE} для заданий, собираемых для целевой машины;
- при сборке заданий для сборочного хоста значение пусто, если используются каталоги сборочного хоста.

Задания -native не устанавливаются в естественные пути хоста, такие как /usr, а помещаются в STAGING_DIR_NATIVE. При сборке таких заданий стандартные переменные окружения (такие как CPPFLAGS и CFLAGS) устанавливаются так, что поиск библиотек и заголовочных файлов выполняется по путям хоста и STAGING_DIR_NATIVE с использованием, например опции GCC -isystem. Таким образом, делается акцент на то, что переменные STAGING_DIR* должны рассматриваться как входные такими задачами, как do_configure, do_compile и do_install. Наличие реальной корневой системы, соответствующей STAGING_DIR_HOST, имеет концептуальный смысл для заданий -native, поскольку они используют двоичные и заголовочные файлы хоста.

STAGING_DIR_NATIVE

Задаёт путь к каталогу sysroot, используемому при сборке компонент, работающих на самом сборочном хосте.

STAGING_DIR_TARGET

Задаёт путь к sysroot в системе, для которой генерируется код. Для компонент, не создающих код (их большинство), переменная STAGING_DIR_TARGET совпадает с STAGING_DIR_HOST. Некоторые задания создают код для использования в целевой системе, но эти двоичные файлы в свою очередь генерируют код для другой отличающейся системы (например, задания cross-canadian). В терминологии GNU первичная система называется хостом (HOST), а вторичная - целью (TARGET). Таким образом, двоичные файлы хоста создают двоичные файлы для целевой платформы (TARGET). Переменная STAGING_DIR_HOST указывает каталог sysroot, используемый хост-системой, а STAGING_DIR_TARGET указывает sysroot для TARGET.

STAGING_ETCDIR_NATIVE

Задаёт путь к каталогу /etc в sysroot сборочного хоста.

STAGING_EXECPREFIXDIR

Задаёт путь к каталогу /usr в sysroot целевой платформы, для которой собирается текущее задание (STAGING_DIR_HOST).

STAGING_INCDIR

Задаёт путь к каталогу /usr/include в sysroot целевой платформы, для которой собирается текущее задание (STAGING_DIR_HOST).

STAGING_INCDIR_NATIVE

Задаёт путь к каталогу /usr/include в sysroot сборочного хоста.

STAGING_KERNEL_BUILDDIR
Указывает каталог, содержащий элементы сборки (artifact) ядра. Задания, собирающие программы, которым нужен доступ к таким элементам (например, systemtap-uprobes) могут просматривать STAGING_KERNEL_BUILDDIR для поиска нужных элементов после сборки ядра.

STAGING_KERNEL_DIR
Каталог с заголовками ядра, которые нужны для сборки внешних модулей.

STAGING_LIBDIR
Задаёт путь к каталогу /usr/lib в sysroot целевой платформы, для которой собирается текущее задание (STAGING_DIR_HOST).

STAGING_LIBDIR_NATIVE
Задаёт путь к каталогу /usr/lib в sysroot сборочного хоста.

STAMP
Задаёт базовый путь для создания штампов задания. Фактический путь к штампам определяется путём преобразования этой строки и добавления в неё. дополнительной информации. В настоящее время переменная задаётся файлом meta/conf/bitbake.conf в виде STAMP = "${STAMPS_DIR}/${MULTIMACH_TARGET_SYS}/${PN}/${EXTENDPE}${PV}-${PR}". Дополнительная информация об использовании штампов для решения вопроса о повторном запуске задач приведена в разделе Stamp Files and the Rerunning of Tasks [1].

STAMPS_DIR
Задаёт каталог, в который система сборки OE помещает штампы сборки (по умолчанию ${TMPDIR}/stamps).

STRIP
Сокращённое имя и аргументы для команды strip, используемой для исключения символов из файлов.

SUMMARY
Короткое (до 72 символов) описание двоичного пакета для системы установки (например, opkg, rpm или dpkg). По умолчанию значение SUMMARY служит значением переменной DESCRIPTION, если она не указана в задании.

SVNDIR
Каталог, в котором выбираются и сохраняются файлы системы Subversion.

SYSLINUX_DEFAULT_CONSOLE
Задаёт консоль, используемую по умолчанию при загрузке ядра. Для использования другой консоли следует установить нужное значение в файле конфигурации задания. Например, SYSLINUX_DEFAULT_CONSOLE = "console=ttyX", где X указывает номер нужной консоли. Класс syslinux исходно устанавливает для переменной пустое значение, но затем проверяет его.

SYSLINUX_OPTS
Задаёт опции, добавляемые в файл syslinux. Переменная устанавливается в задании, опции разделяются точкой с запятой (;). Класс syslinux использует эту переменную для создания набора опций.

SYSLINUX_SERIAL
Задаёт дополнительный последовательный порт или отключает порт при пустом значении переменной в задании. Принятое по умолчанию значение переменной задаёт класс syslinux в виде SYSLINUX_SERIAL ?= "0 115200". Этот класс проверяет и при необходимости использует переменную.

SYSLINUX_SPLASH
Указывает файл .LSS, используемый в качестве фона загрузочного меню VGA. Переменная назначается в задании. Класс syslinux проверяет переменную и при наличии файла система сборки OE устанавливает заставку.

SYSLINUX_SERIAL_TTY
Задаёт дополнительную консоль (console=tty…) для загрузки ядра. По умолчанию для этой переменной класс syslinux устанавливает SYSLINUX_SERIAL_TTY ?= "console=ttyS0,115200", проверяя и используя затем эту переменную.

SYSROOT_DESTDIR
Указывает временный каталог внутри рабочего каталога (по умолчанию ${WORKDIR}/sysroot-destdir), где собираются файлы, помещаемые в sysroot, в процессе выполнения задачи do_populate_sysroot.

SYSROOT_DIRS
Каталоги, помещаемые в sysroot задачей do_populate_sysroot. Принятые по умолчанию каталоги приведены ниже.

```
SYSROOT_DIRS = " \
    ${includedir} \
    ${libdir} \
    ${base_libdir} \
    ${nonarch_base_libdir} \
    ${datadir} \
"
```

SYSROOT_DIRS_BLACKLIST
Каталоги, не помещаемые в sysroot задачей do_populate_sysroot. Переменная может служить для исключения некоторых каталогов, указанных в SYSROOT_DIRS. По умолчанию исключаются приведённые ниже каталоги.

```
SYSROOT_DIRS_BLACKLIST = " \
    ${mandir} \
    ${docdir} \
    ${infodir} \
    ${datadir}/locale \
    ${datadir}/applications \
    ${datadir}/fonts \
    ${datadir}/pixmaps \
"
```

SYSROOT_DIRS_NATIVE
Каталоги, помещаемые в sysroot задачей do_populate_sysroot для заданий -native, в дополнение к каталогам из SYSROOT_DIRS. Включаемые по умолчанию дополнительные каталоги приведены ниже.

```
SYSROOT_DIRS_NATIVE = " \
    ${bindir} \
    ${sbindir} \
    ${base_bindir} \
    ${base_sbindir} \
    ${libexecdir} \
    ${sysconfdir} \
    ${localstatedir} \
```

"

Программы, собирающие задания -native, запускаются напрямую из sysroot (STAGING_DIR_NATIVE), поэтому нужно подготовить дополнительные каталоги с исполняемыми файлами и файлами поддержки.

SYSROOT_PREPROCESS_FUNCS

Список функций, выполняемых после размещения файлов в sysroot, которые обычно служат для дополнительной обработки размещённых файлов или размещения дополнительных файлов.

SYSTEMD_AUTO_ENABLE

При наследовании класса systemd эта переменная определяет, нужно ли автоматически запускать службу, указанную в SYSTEMD_SERVICE. Ао умолчанию службы запускаются при загрузке автоматически, что задаётся классом systemd в форме SYSTEMD_AUTO_ENABLE ??= "enable". Эту настройку можно изменить, указав disable.

SYSTEMD_BOOT_CFG

При установке EFI_PROVIDER = "systemd-boot" эта переменная задаёт конфигурационный файл, который следует использовать. По умолчанию класс systemd-boot устанавливает SYSTEMD_BOOT_CFG ?= "${S}/loader.conf". Информация о Systemd-boot доступна по ссылке Systemd-boot.

SYSTEMD_BOOT_ENTRIES

При EFI_PROVIDER = "systemd-boot" переменная задаёт список файлов (*.conf) для установки, содержащих по одной загрузочной записи в строке. По умолчанию класс systemd-boot задаёт SYSTEMD_BOOT_ENTRIES ?= "" (см. документацию Systemd-boot).

SYSTEMD_BOOT_TIMEOUT

При установке EFI_PROVIDER = "systemd-boot" задаёт время ожидания меню загрузки в секундах. По умолчанию класс systemd-boot устанавливает SYSTEMD_BOOT_TIMEOUT ?= "10" (см. документацию Systemd-boot).

SYSTEMD_PACKAGES

При наследовании класса systemd указывает файлы блоков systemd, когда они не найдены в основном задании пакета. По умолчанию переменная устанавливается так, что файлы блоков systemd предполагаются в основном задании пакета, - SYSTEMD_PACKAGES ?= "${PN}". Если это не так, в переменной SYSTEMD_PACKAGES нужно указать список пакетов, в которых система сборки будет искать файлы блоков systemd.

SYSTEMD_SERVICE

При наследовании класса systemd задаёт имя службы systemd для пакета. При указании этого файла в задании нужно использовать переопределение имени пакета, к которому применяется значение, в форме SYSTEMD_SERVICE_${PN} = "connman.service".

SYSVINIT_ENABLED_GETTYS

При использовании SysVinit задаёт список разделенных пробелами виртуальных терминалов, которые должны запускать getty (разрешая login), если USE_VT отлична от 0. По умолчанию принято SYSVINIT_ENABLED_GETTYS = "1" (getty запускается лишь на первом виртуальном терминале).

T

T

Указывает каталог, где BitBake размещает временные файлы (в основном log-файлы задач и сценариев при сборке конкретных заданий). Для этой переменной обычно устанавливается значение T = "${WORKDIR}/temp"

Переменная WORKDIR указывает каталог, в который BitBake распаковывает и собирает задание. По умолчанию эта переменная задаётся в файле bitbake.conf.

Переменную T не следует путать с переменной TMPDIR, которая указывает корень дерева, куда BitBake размещает вывод всего процесса сборки.

TARGET_ARCH

Архитектура целевой платформы. Система сборки OE поддерживает множество архитектур, включая arm, i586, x86_64, powerpc, powerpc64, mips, mipsel. Дополнительная информация о вариантах архитектуры приведена в описании переменной TUNE_ARCH.

TARGET_AS_ARCH

Задаёт зависимые от архитектуры флаги ассемблера для целевой системы и по умолчанию инициализируется значением TUNE_ASARGS в конфигурационном файле BitBake (meta/conf/bitbake.conf) как TARGET_AS_ARCH = "${TUNE_ASARGS}"

TARGET_CC_ARCH

Задаёт зависимые от архитектуры флаги компилятора C для целевой системы и по умолчанию инициализируется значением TUNE_CCARGS. Это общий подход для добавления LDFLAGS в TARGET_CC_ARCH в заданиях, собирающих программы для целевой системы, которые иначе не учитывали бы экспортируемую переменную LDFLAGS.

TARGET_CC_KERNEL_ARCH

Это специальный флаг компилятора ядра для настройки CPU или ABI. Флаг применяется редко и лишь в случаях, где значение TUNE_CCARGS не совместимо с компиляцией ядра. Переменная позволяет ядру (и связанным с ним модулям) использовать другую конфигурацию. Примером может служить файл meta/conf/machine/include/arm/feature-arm-thumb.inc в дереве исходных кодов.

TARGET_CFLAGS

Задаёт флаги для передачи компилятору C с при сборке для целевой системы. При сборке в контексте целевой системы CFLAGS по умолчанию получает значение этой переменной. Сценарий настройки окружения SDK устанавливает для переменной CFLAGS в среде значение TARGET_CFLAGS, чтобы при сборке исполняемых файлов с использованием SDK также применялись эти флаги.

TARGET_CPPFLAGS

Задаёт флаги для передачи препроцессору C (т. е. компиляторам C и C++) при сборке для целевой системы. При сборке в контексте целевой системы CPPFLAGS по умолчанию получает значение этой переменной. Сценарий настройки окружения SDK устанавливает для переменной CPPFLAGS в среде значение TARGET_CPPFLAGS, чтобы при сборке исполняемых файлов с использованием SDK также применялись эти флаги.

TARGET_CXXFLAGS

Задаёт флаги для передачи компилятору C++ при сборке для целевой системы. При сборке в контексте целевой системы CXXFLAGS по умолчанию получает значение этой переменной. Сценарий настройки окружения SDK устанавливает для переменной CXXFLAGS в среде значение TARGET_CXXFLAGS, чтобы при сборке исполняемых файлов с использованием SDK также применялись эти флаги.

TARGET_FPU

Задаёт метод обработки кода FPU. Для платформ без FPU, к которым относится большинство ARM CPU, эта переменная должна иметь значение "soft". В противном случае применяется эмуляция ядра, что снижает производительность.

TARGET_LD_ARCH

Задаёт зависимые от архитектуры флаги компоновщика для целевой системы и по умолчанию инициализируется значением TUNE_LDARGS в конфигурационном файле BitBake (meta/conf/bitbake.conf) как TARGET_LD_ARCH = "${TUNE_LDARGS}".

TARGET_LDFLAGS

Задаёт флаги для передачи компоновщику при сборке для целевой платформы. При сборке в контексте целевой системы переменная LDFLAGS по умолчанию получает значение этой переменной. Сценарий настройки окружения SDK устанавливает для переменной LDFLAGS в среде значение TARGET_LDFLAGS, чтобы при сборке исполняемых файлов с использованием SDK также применялись эти флаги.

TARGET_OS

Задаёт операционную систему целевой платформы. Переменная получает значение linux для систем на основе glibc (библиотека GNU C) и linux-musl для систем с библиотекой musl. Для платформ ARM/EABI возможны значения linux-gnueabi и linux-musleabi.

TARGET_PREFIX

Задаёт префикс для двоичных файлов инструментария целевой платформы. В зависимости от типа задания и цели сборки переменная TARGET_PREFIX может устанавливаться по разному.

- Для заданий, собирающих целевую машину применяется "${TARGET_SYS}-".
- Для естественных заданий устанавливается значение BUILD_PREFIX.
- Для заданий nativesdk устанавливается значение SDK_PREFIX.

TARGET_SYS

Задаёт систему (включая архитектуру и ОС), для которой выполняется сборка в контексте текущего задания. Система сборки OE автоматически устанавливает эту переменную на основе TARGET_ARCH, TARGET_VENDOR, и TARGET_OS. Менять переменную самостоятельно не требуется.

Для естественного задания для 32-битовой машины x86 с Linux переменная будет иметь значение i686-linux, для платформы little-endian MIPS с Linux - mipsel-linux.

TARGET_VENDOR

Задаёт имя производителя целевой платформы.

TCLIBC

Задаёт вариант стандартной библиотеки GNU C (libc) для использования при сборке. Переменная заменила POKYLIBC, которая больше не поддерживается. Возможны варианты glibc, musl, newlib или baremetal.

TCLIBCAPPEND

Задаёт суффикс, добавляемый к значению TMPDIR и указывающий вариант libc для сборки. При сборке нескольких вариантов в одном сборочном каталоге этот механизм обеспечивает предотвращение конфликтов. В файле defaultsetup.conf задано принятое по умолчанию значение TCLIBCAPPEND = "-${TCLIBC}". Однако такие дистрибутивы, как poky, обычно поддерживают один вариант libc и устанавливают TCLIBCAPPEND = "" в своём файле конфигурации.

TCMODE

Задаёт селектор набора инструментов, управляя характеристиками создаваемых приложений и образов путём указания системе сборки OE применяемого профиля набора инструментов. По умолчанию система сборки OE применяет свой инструментарий и переменная имеет значение default.

Если для TCMODE установлено другое значение, нужно обеспечить совместимость указанного инструментария с принятым по умолчанию. Использование слишком старых или новых компонент может вызвать проблемы при сборке. Совместимость разных компонент описывается в замечанию к выпускам YP (Release Notes), доступным на странице Downloads по ссылкам RELEASE INFORMATION для соответствующих выпусков.

Переменная TCMODE похожа на TCLIBC, которая контролирует вариант стандартной библиотеки GNU C (libc), используемый при сборке (glibc или musl).

При использовании дополнительных уровней можно задать внешний инструментарий. Примером может служить Sourcery G++ Toolchain, поддержка которого задана в отдельном уровне Mentor Graphics® meta-sourcery, доступном по ссылке http://github.com/MentorEmbedded/meta-sourcery/. Файл README для этого уровня описывает использование Sourcery G++ Toolchain в качестве внешнего инструментария. Вам нужно добавить уровень в свой файл bblayers.conf перед мета-уровнем и установить переменную EXTERNAL_TOOLCHAIN в файле local.conf, указав место размещения инструментов.

Используемые в упомянутом примере принципы применимы к любому внешнему инструментарию. Можно использовать уровень meta-sourcery в качестве шаблона для добавления своего уровня внешних инструментов.

TEST_EXPORT_DIR

Место, используемое системой сборки OE для экспорта тестов при установке TEST_EXPORT_ONLY = "1" (по умолчанию ${TMPDIR}/testimage/${PN}).

TEST_EXPORT_ONLY

Задаёт лишь экспорт тестов. Установка значение 1 приведёт к тому, что тесты будут экспортированы, но не будут запускаться в системе сборки.

TEST_LOG_DIR

Указывает место хранения журналов загрузки и SSH для машин QEMU (по умолчанию ${WORKDIR}/testimage). Фактические результаты теста сохраняются в журнале задачи log.do_testimage в каталоге ${WORKDIR}/temp/.

TEST_POWERCONTROL_CMD

При автоматизированном тестировании оборудования задаёт команду, применяемую для управления питанием на тестируемой целевой машине. Обычно эта команда указывает сценарий, который выполняет нужные действия (например, взаимодействует с управляемым через web сетевым удлинителем). Указанная команда должна ожидать в качестве последнего аргумента значение "off", "on" или "cycle" для включения, выключения или цикла (выключение с последующим включением).

TEST_POWERCONTROL_EXTRA_ARGS

При автоматизированном тестировании оборудования задаёт дополнительные аргументы, передаваемые команде, указанной в TEST_POWERCONTROL_CMD. Установка переменной необязательна. Она может служить, например, для разделения зависимых и независимых от машины частей команды.

TEST_QEMUBOOT_TIMEOUT

Время в секундах с начала загрузки, по истечение которого начнётся автоматизированное тестирование образа. Используемый по умолчанию тайм-аут 500 секунд позволяет процессу загрузки дойти до приглашения на вход в систему (login). Можно указать иное время ожидания в файле local.conf. Дополнительные сведения о тестировании образов приведены в разделе Performing Automated Runtime Testing [2].

TEST_SERIALCONTROL_CMD

При автоматизированном тестировании оборудования задаёт команду, применяемую для подключения последовательной консоли на тестируемой платформе. Команда просто должна подключиться к консоли и перенаправить в это соединение стандартный-ввод-вывод, как это делает обычная терминальная программа. Например, для использования терминала Picocom с устройством /dev/ttyUSB0 на скорости 115200 бит/с можно указать TEST_SERIALCONTROL_CMD = "picocom /dev/ttyUSB0 -b 115200".

TEST_SERIALCONTROL_EXTRA_ARGS

При автоматизированном тестировании оборудования задаёт дополнительные аргументы, передаваемые команде, указанной в TEST_SERIALCONTROL_CMD. Установка переменной необязательна. Она может служить, например, для разделения зависимых и независимых от машины частей команды.

TEST_SERVER_IP

IP-адрес сборочной машины, который обычно определяется автоматически и при отказе детектирования нужно установить его вручную. Переменная применяют лишь немногими тестами, такими как dnf, которым нужно загружать пакеты из WORKDIR/oe-rootfs-repo.

TEST_TARGET

Указывает целевой контроллер для выполнения тестов образа (по умолчанию TEST_TARGET = "QemuTarget"). Целевым контроллером служит класс, определяющий развёртывание образа на целевой платформе и запуск платформы. Уровень может расширять контроллеры путём добавления модулей в свой каталог /lib/oeqa/controllers и наследования абстрактного класса BaseTarget, который не может служить значением TEST_TARGET.

Поддерживаемые TEST_TARGET значения описаны ниже.

- *"QemuTarget"* - загрузка образа QEMU и запуск тестов (см. раздел Enabling Runtime Tests on QEMU).
- *"SimpleRemoteTarget"* - запуск тестов на целевой платформе, которая уже работает. Оборудование может находиться в сети или эмулироваться в QEMU. Требуется установка переменной TEST_TARGET_IP. Этот параметр определён в файле meta/lib/oeqa/controllers/simpleremote.py.

Сведения о запуске тестов на оборудовании приведены в разделе Enabling Runtime Tests on Hardware [2].

TEST_TARGET_IP

IP-адрес тестируемого оборудования. TEST_TARGET_IP не работает при установке TEST_TARGET = "qemu".

Вместе с адресом IP можно указать порт, например, TEST_TARGET_IP = "192.168.1.4:2201". Указание порта полезно при работе SSH через нестандартный порт, например, при размещении тестируемого устройства за межсетевым экраном или транслятором адресов и портов.

TEST_SUITES

Упорядоченный список тестов (модулей) для запуска применительно к образу с целью автоматического тестирования при работе. Система сборки OE обеспечивает базовый набор тестов для проверки образов. В настоящее время эти тесты работают только в QEMU. Тесты включают ping, ssh, df и др. Можно расширить список тестов, добавляя их в переменную TEST_SUITES в виде TEST_SUITES_append = " mytest". Можно также указать TEST_SUITES_append = " auto" для применения всех доступных тестов.

Использование этой опции заставляет систему сборки автоматически запускать применимые к образу тесты. Порядок тестов имеет значение и зависящие от других тестов проверки должны выполняться после проверки от которой они зависят. Например, при добавлении в список тестов test_A и test_B, где test_B зависит от test_A следует указывать TEST_SUITES = " test_A test_B".

Дополнительные сведения о тестировании образов приведены в разделе Performing Automated Runtime Testing [2].

TESTIMAGE_AUTO

Управляет запуском серии автоматизированных тестов для образов после успешной сборки. TESTIMAGE_AUTO = "1" вызывает автоматическую загрузку созданного образа в QEMU. Использование переменной также добавляет зависимости, поэтому сначала автоматически собираются все SDK, для которых запрошено тестирование.

Тесты написаны на Python с использованием модуля unittest и основная часть тестов запускает команды на целевой системе по протоколу ssh. Можно установить для переменной значение 1 в файле local.conf каталога сборки, чтобы система сборки OE автоматически выполнила тесты после сборки образа. Дополнительные сведения о включении, запуске и создании тестов приведены в разделе Performing Automated Runtime Testing [2] и параграфе 6.132. testimage*.bbclass.

THISDIR

Каталог, в котором в данный момент выполняется разбор BitBake. Не меняйте эту переменную.

TIME

Время начала сборки в формате HMS (например, 140159 для 14 часов, одной минуты и 59 секунд).

TMPDIR

Эта переменная указывает базовый каталог, используемый системой сборки OE для всего вывода результатов и промежуточных файлов (кроме общего кэша состояний). По умолчанию переменная TMPDIR указывает каталог tmp в каталоге сборки. Если нужно разместить временный каталог в другом месте, можно убрать знак комментария и отредактировать показанную ниже строку файла conf/local.conf в дереве исходных кодов.

```
#TMPDIR = "${TOPDIR}/tmp"
```

Примером такого использования служит указание в переменной TMPDIR локального диска, который не использует NFS, при размещении каталога сборки на NFS.

Файловая система, указанная TMPDIR, должна поддерживать стандартную семантику (например, различать регистр символов в именах, поддерживать блокировки POSIX и сохраняющиеся значения inode). С учётом имеющихся в NFS проблем и наличия ошибок в некоторых реализациях, NFS не соответствует этим требованиям, поэтому TMPDIR не может указывать NFS.

TOOLCHAIN_HOST_TASK

Эта переменная указывает пакеты, которые система сборки OE использует при создании SDK со средой кросс-компиляции. Эти пакеты являются частью инструментария, работающего на SDKMACHINE, и обычно каждый пакет должен иметь префикс nativesdk-. Рассмотрим в качестве примера команду сборки SDK

```
$ bitbake -c populate_sdk imagename
```

В этом случае в переменной указан принятый по умолчанию список пакетов, но его можно расширить (см. раздел Adding Individual Packages to the Standard SDK [7]). Базовые сведения о кросс-разработке в среде YP приведены в разделе Cross-Development Toolchain Generation [1], а организация среды разработки описана в [7].

TOOLCHAIN_OUTPUTNAME

Задаёт имя, используемое для вывода инструментария. Класс populate_sdk_base устанавливает TOOLCHAIN_OUTPUTNAME ?= "${SDK_NAME}-toolchain-${SDK_VERSION}". Дополнительная информация приведена в описаниях переменных SDK_NAME и SDK_VERSION.

TOOLCHAIN_TARGET_TASK

Перечисляет пакеты, используемые системой сборки OE при создании целевой части SDK (т. е. части, собираемой для целевой платформы), включающей библиотеки и файлы заголовков. Дополнительная информация о добавлении пакетов в SDK приведена в разделе Adding Individual Packages to the Standard SDK [7]. Базовые сведения об инструментах кросс-разработки в среде YP можно найти в разделе Cross-Development Toolchain Generation [1], организация среды кросс-разработки описана в [7].

TOPDIR

Верхний уровень каталога сборки. BitBake автоматически инициализирует переменную при установке рабочей среды с помощью oe-init-build-env (параграф 5.1.9.1 oe-init-build-env).

TRANSLATED_TARGET_ARCH

Очищенный (sanitized) вариант TARGET_ARCH. Переменная применяется в тех случаях, где архитектура должна указываться в значениях, не поддерживающих символы подчёркивания (например, в именах пакетов). Эти символы в TARGET_ARCH заменяются символами дефиса. Не меняйте эту переменную.

TUNE_ARCH

Каноническая архитектура GNU (arm, armeb, mips, mips64 и т. п.), которую BitBake использует для настройки конфигурации. Определения TUNE_ARCH зависят от конкретной архитектуры и могут быть статическими или динамическими. Детали для конкретного семейства CPU можно найти в файле README для архитектуры. Например, файл meta/conf/machine/include/mips/README в дереве исходных кодов содержит информацию для TUNE_ARCH в архитектуре mips.

TUNE_ARCH тесно связана с переменной TARGET_ARCH, задающей архитектуру целевой машины. Конфигурационный файл BitBake (meta/conf/bitbake.conf) устанавливает переменную в форме TARGET_ARCH = "${TUNE_ARCH}".

Список поддерживаемых архитектур включает arm, i586, x86_64, powerpc, powerpc64, mips, mipsel и др.

TUNE_ASARGS

Задаёт зависящие от архитектуры флаги ассемблера для целевой системы. Набор флагов основывается на выбранных настройках. Переменная TUNE_ASARGS задаётся с использованием включаемых файлов настройки, которые размещаются обычно в meta/conf/machine/include/, и зависит от TUNE_FEATURES. Например, файл meta/conf/machine/include/x86/arch-x86.inc определяет флаги для архитектуры x86 в форме TUNE_ASARGS += "${@bb.utils.contains("TUNE_FEATURES", "mx32", "-x32", "", d)}". Пакеты BSP выбирают настройку (tune), которая, в свою очередь, влияет на сами переменные настройки, которые передаются набором флагов.

TUNE_CCARGS

Задаёт зависящие от архитектуры флаги компилятора C для целевой системы. Набор флагов основывается на выбранных настройках. Переменная TUNE_CCARGS задаётся с использованием включаемых файлов настройки, которые размещаются обычно в meta/conf/machine/include/, и зависит от TUNE_FEATURES. Пакеты BSP выбирают настройку (tune), которая, в свою очередь, влияет на сами переменные настройки, которые передаются набором флагов.

TUNE_LDARGS

Задаёт зависящие от архитектуры флаги компоновщика для целевой системы. Набор флагов основывается на выбранных настройках. Переменная TUNE_LDARGS задаётся с использованием включаемых файлов настройки, которые размещаются обычно в meta/conf/machine/include/, и зависит от TUNE_FEATURES. Например, файл meta/conf/machine/include/x86/arch-x86.inc определяет флаги для архитектуры x86 в форме TUNE_LDARGS += "${@bb.utils.contains("TUNE_FEATURES", "mx32", "-m elf32_x86_64", "", d)}". Пакеты BSP выбирают настройку (tune), которая, в свою очередь, влияет на сами переменные настройки, которые передаются набором флагов.

TUNE_FEATURES

Свойства, применяемые для «настройки» компилятора с целью оптимизации для конкретного процессора. Свойства определяются в файлах настройки и позволяют динамически создавать аргументы (TUNE_*ARGS) на основе свойств. Система сборки OE проверяет свойства на предмет их поддержки и отсутствия конфликтов.

Конфигурационный файл BitBake (meta/conf/bitbake.conf) задаёт TUNE_FEATURES ??= "${TUNE_FEATURES_tune-${DEFAULTTUNE}}". Дополнительная информация приведена в описании переменной DEFAULTTUNE.

TUNE_PKGARCH

Архитектура пакета, воспринимаемая системой подготовки пакетов для выбора архитектуры, ABI и настройки выходных пакетов. Конкретная настройка задаётся переопределением _tune в форме TUNE_PKGARCH_tune-tune = "tune". Эти зависящие от настройки варианты архитектуры определяются во включаемых файлах для машин. Например, файл meta/conf/machine/include/tune-core2.inc указывает архитектуру в виде TUNE_PKGARCH_tune-core2-32 = "core2-32".

TUNEABI

Базовый интерфейс ABI, применяемый конкретной настройкой для данного уровня инструментов (по умолчанию разрешены все настройки). Провайдеры, использующие готовые библиотеки могут применять переменные TUNEABI_WHITELIST, TUNEABI_OVERRIDE и TUNEABI для проверки совместимости настройки с выбором библиотек. Использование переменной рассмотрено в параграфе 6.116. sanity.bbclass.

TUNEABI_OVERRIDE

При установленной переменной система сборки игнорирует TUNEABI_WHITELIST. Провайдеры, использующие готовые библиотеки могут применять переменные TUNEABI_WHITELIST, TUNEABI_OVERRIDE и TUNEABI для проверки совместимости настройки с выбором библиотек. Использование переменной рассмотрено в параграфе 6.116. sanity.bbclass.

TUNEABI_WHITELIST

Список разрешённых значений TUNEABI (по умолчанию разрешены все настройки). Провайдеры, использующие готовые библиотеки могут применять переменные TUNEABI_WHITELIST, TUNEABI_OVERRIDE и TUNEABI для проверки совместимости настройки с выбором библиотек. Использование переменной рассмотрено в параграфе 6.116. sanity.bbclass.

TUNECONFLICTS[feature]

Задаёт функции тестирования CPU или ABI, конфликтующие со свойством. Известные конфликты указываются во включаемых файлах машин внутри дерева исходных кодов. Примером может служить файл meta/conf/machine/include/mips/arch-mips.inc, в котором указан конфликт o32 и n64 с n32 в виде TUNECONFLICTS[n32] = "o32 n64".

TUNEVALID[feature]

Задаёт действительную функцию настройки CPU или ABI, сохраняемую как флаг. Действительные функции указаны во включаемых файлах машин (например, meta/conf/machine/include/arm/arch-arm.inc) в дереве исходных кодов. Ниже приведен пример из такого файла.

```
TUNEVALID[bigendian] = "Enable big-endian mode."
```

U

UBOOT_CONFIG

Настраивает UBOOT_MACHINE и может также определять IMAGE_FSTYPES в отдельных случаях. Ниже приведен пример с уровня meta-fsl-arm.

```
UBOOT_CONFIG ??= "sd"
UBOOT_CONFIG[sd] = "mx6qsabreauto_config,sdcard"
UBOOT_CONFIG[eimnor] = "mx6qsabreauto_eimnor_config"
UBOOT_CONFIG[nand] = "mx6qsabreauto_nand_config,ubifs"
UBOOT_CONFIG[spinor] = "mx6qsabreauto_spinor_config"
```

В этом примере выбран вариант sd из 4 возможных для UBOOT_MACHINE. Конфигурация sd определяет "mx6qsabreauto_config" как значение UBOOT_MACHINE, а "sdcard" задаёт IMAGE_FSTYPES для образа U-boot.

Обработка UBOOT_CONFIG рассмотрена в описании класса uboot-config (параграф 6.139. uboot-config.bbclass).

UBOOT_ENTRYPOINT

Задаёт точку входа для образа U-Boot, при создании которого эта переменная передается в качестве параметра команде uboot-mkimage.

UBOOT_LOADADDRESS

Задаёт адрес загрузки для образа U-Boot, при создании которого эта переменная передается в качестве параметра команде uboot-mkimage.

UBOOT_LOCALVERSION

Добавляет строку к имени локальной версии образа U-Boot. Например, для образа U-Boot версии 2013.10 можно задать полное имя 2013.10-yocto с помощью UBOOT_LOCALVERSION = "-yocto".

UBOOT_MACHINE

Задаёт значение, передаваемое команде make при сборке образа U-Boot и указывающее конфигурацию целевой платформы. Обычно эта переменная задаётся в файле конфигурации машины conf/machine/machine_name.conf.

Возможные значения переменной приведены в разделе Selection of Processor Architecture and Board Type файла U-Boot README.

UBOOT_MAKE_TARGET

Задаёт цель, указанную в Makefile (по умолчанию all).

UBOOT_SUFFIX

Указывает создаваемое расширение U-Boot. Например, u-boot.sb имеет расширение .sb. По умолчанию U-Boot использует расширение .bin

UBOOT_TARGET

Задаёт цель для сборки U-Boot, которая передается напрямую как часть команды make (например, SPL и AIS). Если эта переменная не задана, система сборки OE передаёт и использует значение all при сборке U-Boot.

UNKNOWN_CONFIGURE_WHITELIST

Задаёт список опций, для которых не нужно создавать предупреждений в процессе работы задачи do_configure, если сценарий configure сочтёт опции не пригодными. Обычно недействительные опции просто не передаются сценарию configure (т. е. их следует удалять из EXTRA_OECONF и PACKAGECONFIG_CONFARGS). Однако имеются, например, базовые опции, которые могут быть не пригодны для некоторых сценариев configure. Предупреждать о таких опциях нет смысла, поэтому их добавляют в UNKNOWN_CONFIGURE_WHITELIST.

Проверка аргументов configure с использованием UNKNOWN_CONFIGURE_WHITELIST является частью класса insane и включается лишь в заданиях, наследующих класс autotools.

UPDATERCPN

Для задания, наследующих класс update-rc.d, переменная UPDATERCPN задаёт включаемый пакет initscript. По умолчанию используется значение ${PN}. С учётом того, что почти все задания, устанавливающие пакет initscripts, упаковывают его в основной пакет задания, эту переменную редко приходится устанавливать для заданий.

UPSTREAM_CHECK_GITTAGREGEX

Можно проверить использование каждым заданием последней версии разрабатываемого кода с помощью bitbake -c checkpkg. Если исходный код взят из репозитория Git, система сборки OE определяет последнюю версию, указывая самый новый тег для репозитория. Переменная UPSTREAM_CHECK_GITTAGREGEX позволяет задать регулярное выражение для фильтрации тегов, если принятый по умолчанию фильтр работает некорректно.

```
UPSTREAM_CHECK_GITTAGREGEX = "git_tag_regex"
```

UPSTREAM_CHECK_REGEX

Служит для задания регулярного выражения вместо принятого по умолчанию, когда система проверки пакетов анализирует страницу, найденную с помощью UPSTREAM_CHECK_URI., Например, UPSTREAM_CHECK_REGEX = "package_regex".

UPSTREAM_CHECK_URI

Можно проверить использование последней версии разрабатываемого кода для каждого задания с помощью команды bitbake -c checkpkg. Если исходные коды взяты из архива, последняя версия определяется просмотром списка в каталоге, где размещён. архив с попыткой найти самый новый. Когда такой подход не работает, можно использовать UPSTREAM_CHECK_URI для указания URI со ссылкой на последнюю версию архива в форме UPSTREAM_CHECK_URI = "recipe_url".

USE_DEVFS

Управляет использованием devtmpfs для заполнения /dev. По умолчанию применяется USE_DEVFS = "1", но обычно указывается USE_DEVFS = "0" для статически заполняемого каталога /dev (см. раздел Selecting a Device Manager [2]).

USE_VT

При использовании SysVinit определяет, нужно ли запускать программу getty на любом из виртуальных терминалов для записи журнала через эти терминалы. По умолчанию используется USE_VT = "0" в файле конфигурации машины для систем без подключённого графического дисплея, не поддерживающих функции виртуального терминала.

USER_CLASSES

Список классов для глобального наследования, которые используются системой сборки OE для включения дополнительных свойств (например, buildstats, image-mklibs и т. п.). Принятый по умолчанию список задаётся в файле local.conf как USER_CLASSES ?= "buildstats image-mklibs image-prelink". Примером может служить файл meta-poky/conf/local.conf.sample в каталоге исходных кодов.

USERADD_ERROR_DYNAMIC

При установке значения error заставляет систему сборки OE выдавать сообщение об ошибке, если идентификаторы пользователя (uid) и группы (gid) не заданы в файлах files/passwd и files/group. При установке значения warn будут выводиться предупреждения.

По умолчанию система сборки динамически применяет значения uid и gid, поэтому переменная USERADD_ERROR_DYNAMIC не устанавливается. Если нужны статические идентификаторы, следует задать в файле local.conf переменную USERADD_ERROR_DYNAMIC = "error". Переопределение принятого по умолчанию поведения предполагает установку статических значений uid и gid с помощью переменных USERADDEXTENSION, USERADD_UID_TABLES и USERADD_GID_TABLES.

USERADD_GID_TABLES

Задаёт файл паролей, используемый для получения статических идентификаторов групп (gid) при добавлении группы системой сборки OE в процессе установки пакета. При использовании статических значений gid система сборки OE ищет в пути BBPATH файл files/group и применяет значения идентификаторов. Переменная задаётся в файле local.conf как USERADD_GID_TABLES = "files/group". Для использования статических значений gid устанавливается USERADDEXTENSION = "useradd-staticids".

USERADD_PACKAGES

При наследовании класса useradd эта переменная задаёт отдельные пакеты задания, для которых нужно добавлять пользователей или группы. Например, для добавления пользователя в основном пакете задания служит строка USERADD_PACKAGES = "${PN}".

При использовании переменной USERADD_PACKAGES нужно задать одну или несколько переменных USERADD_PARAM, GROUPADD_PARAM или GROUPMEMS_PARAM.

USERADD_PARAM

При наследовании класса useradd эта переменная задаёт для пакета параметры, которые следует передать команде useradd для добавления пользователя при установке пакета. Ниже приведен пример из задания dbus.

```
USERADD_PARAM_${PN} = "--system --home ${localstatedir}/lib/dbus \
                       --no-create-home --shell /bin/false \
                       --user-group messagebus"
```

Информация о стандартной команде Linux useradd доступна по ссылке http://linux.die.net/man/8/useradd.

USERADD_UID_TABLES

Задаёт файл паролей, используемый для получения статических идентификаторов пользователей (uid) при добавлении системой сборки OE пользователя в процессе установки пакета. При использовании статических значений uid система сборки OE ищет в пути BBPATH файл files/passwd и применяет найденные значения uid. Переменная задаётся в файле local.conf как USERADD_UID_TABLES = "files/passwd". Для использования статических идентификаторов указывается USERADDEXTENSION = "useradd-staticids".

USERADDEXTENSION

Значение useradd-staticids заставляет систему сборки OE при добавлении пользователей и групп применять статические файлы passwd и group найденные в пути BBPATH. Для этого в файл local.conf включается строка USERADDEXTENSION = "useradd-staticids", в результате чего система сборки OE реализует класс useradd-staticids. При использовании статических значений uid и gid нужно указать файлы files/passwd и files/group в переменных USERADD_UID_TABLES и USERADD_GID_TABLES, а также задать переменную USERADD_ERROR_DYNAMIC.

V

VOLATILE_LOG_DIR

Задаёт постоянное наличие на целевой платформе каталога /var/log, используемого для записи журналов операций пост-установки. По умолчанию задано VOLATILE_LOG_DIR = "yes", что означает временные файлы журналов, а для постоянного их хранения следует установить значение "no".

W

WARN_QA

Задаёт проверки качества, отказы которых считаются предупреждениями системы сборки OE. Переменная указывается в файле конфигурации дистрибутива. Список возможных проверок указан в параграфе 6.56. insane.bbclass.

WKS_FILE_DEPENDS

При включении в задание, создающее образ, эта переменная указывает зависимости при сборке. Переменная используется только при активных образах Wic (IMAGE_FSTYPES содержит связанные с Wic записи).

WKS_FILE_DEPENDS похожа на переменную DEPENDS и может применяться в заданиях, собирающих образы Wic, добавляя зависимости к указанным в DEPENDS. Переменная позволяет задать список дополнительных зависимостей (например, естественные инструменты, загрузчики и т. п.), которые нужны для сборки образов Wic. Например, WKS_FILE_DEPENDS = "some-native-tool" указывает тот или иной естественный инструмент, от которого зависит сборка образа.

WKS_FILE

Задаёт расположение файла Wic kickstart, используемого системой сборки OE для создания образа с разделами (image.wic). Информация о работе с такими образами приведена в разделе Creating Partitioned Images Using Wic [2], а формат kickstart описывает Глава 9. Справочник по OpenEmbedded Kickstart (.wks).

WORKDIR

Имя рабочего каталога, в котором система OE собирает задание. Этот каталог размещается в структуре TMPDIR и относится к собираемому заданию. Каталог WORKDIR определяется выражением ${TMPDIR}/work/${MULTIMACH_TARGET_SYS}/${PN}/${EXTENDPE}${PV}-${PR}

Реальный каталог зависит от нескольких аспектов:

- TMPDIR - выходной каталог верхнего уровня для сборки;
- MULTIMACH_TARGET_SYS - идентификатор целевой системы;
- PN - имя задания;
- EXTENDPE - эпоха (если переменная PE не задана, как в большинстве заданий, значение EXTENDPE пусто);
- PV - версия задания;
- PR - выпуск (revision) задания.

В качестве примера рассмотрим Source Directory верхнего уровня с именем poky, заданный по умолчанию каталог сборки poky/build и целевую систему qemux86-poky-linux. Пусть задание называется foo_1.3.0-r0.bb. В этом случае рабочим каталогом для сборки будет poky/build/tmp/work/qemux86-poky-linux/foo/1.3.0-r0

X

XSERVER

Указывает пакеты, которые нужно установить для поддержки X-сервера и драйверов для текущей машины при условии непосредственного включения в образ packagegroup-core-x11-xserver или опосредованного включения x11-base в IMAGE_FEATURES. По умолчанию значением XSERVER будет xserver-xorg xf86-video-fbdev xf86-input-evdev, если в конфигурации машины не указано иное.

Глава 14. Контекст переменных

Хотя большинство переменных может применяться почти в любом контексте (файлы .conf, .bbclass, .inc, .bb), некоторые из них зачастую связаны с конкретным местом или контекстом. В этой главе описаны базовые привязки переменных к контексту.

14.1. Конфигурация

В последующий параграфах перечислены переменные конфигурации с контекстом distribution, machine и local.

14.1.1. Дистрибутив (Distro)

Переменные, контекстом которых служит конфигурация дистрибутива (distro), включают DISTRO, DISTRO_NAME, DISTRO_VERSION, MAINTAINER, PACKAGE_CLASSES, TARGET_OS, TARGET_FPU, TCMODE, TCLIBC.

14.1.2. Машина

Переменные, контекстом которых служит конфигурация машины, включают TARGET_ARCH, SERIAL_CONSOLES, PACKAGE_EXTRA_ARCHS, IMAGE_FSTYPES, MACHINE_FEATURES, MACHINE_EXTRA_RDEPENDS, MACHINE_EXTRA_RRECOMMENDS, MACHINE_ESSENTIAL_EXTRA_RDEPENDS, MACHINE_ESSENTIAL_EXTRA_RRECOMMENDS.

14.1.3. Локальные переменные

Переменные, контекстом которых служит локальная конфигурация в файле local.conf, включают DISTRO, MACHINE, DL_DIR, BBFILES, EXTRA_IMAGE_FEATURES, PACKAGE_CLASSES, BB_NUMBER_THREADS, BBINCLUDELOGS, ENABLE_BINARY_LOCALE_GENERATION.

14.2. Задания

В последующих параграфах указаны переменные из контекста заданий - требуемые и задающие зависимости, пути и дополнительные данные для сборки.

14.2.1. Требуемые переменные

В заданиях должны присутствовать переменные LICENSE, LIC_FILES_CHKSUM и SRC_URI (для заданий, извлекающих локальные или удалённые файлы).

14.2.2. Зависимости

Зависимости между заданиями указываются в переменных DEPENDS, RDEPENDS, RRECOMMENDS, RCONFLICTS и RREPLACES.

14.2.3. Пути

Переменные, определяющие пути в заданиях, включают WORKDIR, S, FILES.

14.2.4. Дополнительная информация для сборки

Переменные, определяющие дополнительные данные для сборки заданий, включают DEFAULT_PREFERENCE, EXTRA_OECMAKE, EXTRA_OECONF, EXTRA_OEMAKE, PACKAGECONFIG_CONFARGS, PACKAGES.

Глава 15. Ответы на вопросы

15.1. Различия между Poky и OpenEmbedded

Термин Poky относится к конкретной системе сборки, предоставляемой YP и основанной на OE-Core и BitBake. Таким образом, базовым термином для систем сборки является «система сборки OE». Работа в YP с использованием Poky

тесно связана с OE, при этом изменения сначала передаются в OE-Core или BitBake, а затем возвращаются в Poky. Это полезно для обоих проектов.

15.2. Выполнение требований к Git, tar и Python
Нужны версии программ для сборочного хоста можно получить разными способами, описанными в параграфе 1.3. Требуемые версии Git, tar и Python.

15.3. Стабильность работы Poky/OpenEmbedded-Core
- Команда YP поддерживает уровень OE-Core компактным и целенаправленным, включая в него около 830 заданий в отличие от тысяч других, доступных в сообществе OE. Это упрощает тестирование и поддержку.
- Команда YP проводит ручное и автоматизированное тестирование для небольшого набора эталонного оборудования и эмулируемых платформ.
- В YP применяется автоматический сборщик, который обеспечивает тестирование сборки и интеграции.

15.4. Включение поддержки платы в YP
Поддержка новой платы добавляется созданием уровня BSP для неё. Создание уровня BSP описано в разделах Understanding and Creating Layers [2] и Yocto Project Board Support Package (BSP) Developer's Guide [6]. Если плата не является совсем экзотической, добавление её поддержки в YP достаточно просто.

15.5. Продукция, использующая систему сборки OE
Программы, работающие на Vernier LabQuest, используют систему сборки OE (см. сайт Vernier LabQuest). Существует множество подготовленных к производству (pre-production) устройств, использующих систему сборки OE и команда YP анонсирует их по мере выпуска.

15.6. Вывод системы сборки OE
Поскольку один и тот же набор заданий может применяться для вывода в разных форматах, вывод системы сборки OE зависит от способа запуска сборки. Обычно выводом служат образы, пригодные для установки на целевой платформе.

15.7. Добавление своего пакета в YP
Для добавления пакета нужно создать задание BitBake, как описано в разделе Writing a New Recipe [2].

15.8. Обновление программ на целевой платформе
Система сборки OE может создавать пакеты в разных форматах, включая IPK для OPKG, Debian (.deb) и RPM. Можно обновлять пакеты с помощью менеджера пакетов на устройстве, как в обычных дистрибутивах Linux. Однако управление пакетами на целевой платформе реализуется не всегда.

15.9. Ошибки chmod: XXXXX new permissions are r-xrwxrwx, not r-xr-xr-x
Возможно вы запустили сборку на файловой системе NTFS, хотя следует применять ext2, ext3 или ext4.

15.10. Ошибка 404 при загрузке исходных кодов в систему сборки OE
Ничего страшного! Система сборки OE проверяет все указанные зеркала исходных кодов в поисках архивов и заранее выбранных версий управляемых SCM программ. Эта проверка помогает при крупных инсталляциях, поскольку может снизить нагрузку на серверы SCM. Указанный в ошибке адрес просто является одним из настроенных в системе сборки зеркал.

15.11. Машинозависимые данные в пакете
Установите SRC_URI_OVERRIDES_PACKAGE_ARCH = "0" в файле .bb и обеспечьте маркировку файла как машинозависимого вручную, если это нужно. Код обработки SRC_URI_OVERRIDES_PACKAGE_ARCH содержится в файле meta/classes/base.bbclass.

15.12. Работа через межсетевой экран
Большая часть выборок исходных кодов системой сборки OE выполняется с помощью утилиты wget, поэтому следует указать настройки прокси в файле .wgetrc, который может размещаться в домашнем каталоге или в /usr/local/etc/wgetrc.

Ниже приведен пример настройки типов прокси в файле .wgetrc. По умолчанию эти настройки отключены символами комментария, которые следует удалить для включения нужных вариантов.

```
# You can set the default proxies for Wget to use for http, https, and ftp.
# They will override the value in the environment.
#https_proxy = http://proxy.yoyodyne.com:18023/
#http_proxy = http://proxy.yoyodyne.com:18023/
#ftp_proxy = http://proxy.yoyodyne.com:18023/

# If you do not want to use proxy at all, set this to off.
#use_proxy = on
```

YP также включает файл meta-poky/conf/site.conf.sample, показывающий настройки прокси-серверов для CVS и Git. Дополнительная информация о настройке разных типов прокси приведена на странице Working Behind a Network Proxy.

15.13. Различие между target и target-native

Цели *-native предназначены для работы на системах, применяемых для сборки. Обычно это инструменты, требуемые для помощи при сборке (например, quilt-native служит для применения patch-файлов). Не относящиеся к естественные (non-native) задания работают на целевых платформах.

15.14. Непонятные отказы при сборке

Если при одной и той же сборки возникают разные отказы, это может объясняться двумя причинами:

- проблемы на сборочном хосте;
- сборка происходит на виртуальной машине, а система виртуализации выдаёт ошибки.

Система сборки OE обрабатывает огромный объем данных, что связано с большим числом операций, включающих сеть, диски и процессоры, поэтому могут проявляться ошибки в любой из этих подсистем. Случайные и необъяснимые отказы всегда связаны с проблемами в оборудовании или системе виртуализации.

15.15. Проблемы с iconv.h при сборке естественных заданий

При получении сообщения об ошибке, указывающего, что библиотека GNU libiconv не применяется, но файл iconv.h включён из libiconv, нужно проверить наличие ранее установленной версии в файлах заголовков /usr/local/include.

```
#error GNU libiconv not in use but included iconv.h is from libiconv
```

Если такой файл найден, следует удалить прежнюю установку (uninstall) или временно переименовать файл.

Эта проблема является проявлением «утечки системы», когда система сборки OE находит и использует установленные ранее файлы при сборке естественного кода. Такая ошибка может быть связана не только с iconv.h. Убедитесь, что утечка не происходит из каталогов /usr/local/include и /opt.

15.16. Лицензионные требования

Вопросы лицензирования могут потребовать консультаций с юристами. Следует помнить, что для соблюдения требований GPL нужно включать информацию, позволяющую другим заново собрать и воспроизвести предлагаемый вами вариант. Это означает предоставление исходного кода, всех внесённых в него изменений (patch), а также конфигурации настройки и сборки пакета.

Информация о лицензировании представлена в разделе Licensing [1] и Maintaining Open Source License Compliance During Your Product's Lifecycle [2].

15.17. Отключение курсора на сенсорном экране

Нужно добавить файл форм-фактора, как описано в разделе Miscellaneous BSP-Specific Recipe Files [6]. Для переменной HAVE_TOUCHSCREEN устанавливается значение 1.

15.18. Активизация сетевых интерфейсов

Используемый по умолчанию файл interfaces из задания netbase не активизирует сетевые интерфейсы автоматически. Поэтому нужно добавить зависящее от BSP задание netbase, которое включает файл interfaces (см. раздел Miscellaneous BSP-Specific Recipe Files [6]). Например, можно добавить на свой уровень файлы

```
meta-MACHINE/recipes-bsp/netbase/netbase/MACHINE/interfaces
meta-MACHINE/recipes-bsp/netbase/netbase_5.0.bbappend
```

15.19. Увеличение свободного пространства для образа

По умолчанию система сборки OE использует коэффициент 1,3 при расчёте размера корневой файловой системы. Для изменения размера нужно установить несколько параметров, описанных ниже.

- *Размер образа.* Система сборки OE использует переменную IMAGE_ROOTFS_SIZE для задания размера образа в килобайтах. Размер определяется с учётом начальной корневой файловой системы до внесения каких-либо изменений (таких как запрошенный размер и дополнительное свободное место).
- *Служебное пространство.* Переменная IMAGE_OVERHEAD_FACTOR задаёт коэффициент, применяемый при расчёте размера (по умолчанию 1,3).
- *Дополнительное свободное пространство.* Переменная IMAGE_ROOTFS_EXTRA_SPACE служит для добавления свободного пространства в образ после определения IMAGE_ROOTFS_SIZE.

15.20. Поддержка имён файлов и каталогов с пробелами

Команда YP пытается решить эту задачу, но объем работы для системы сборки OE слишком велик, поскольку она включает много программ (например, autoconf), не способных работать с именами, содержащими пробелы. Пока эта задача не будет решена, поддержка имён с пробелами невозможна.

15.21. Использование внешнего инструментария

Настройка инструментария обеспечивает достаточную гибкость и контролируется в основном переменной TCMODE, которая указывает файл tcmode-*.inc file для включения каталога meta/conf/distro/include в дереве исходных кодов. По умолчанию переменная TCMODE имеет значение default, которое задаёт системе сборки OE использование встроенного инструментария (файл tcmode-default.inc). Однако можно задать и другие инструменты. В частности, значения external-* позволяют применять внешний инструментарий. Примером может служить использование Sourcery G++ Toolchain, поддержка которого задана в отдельном уровне meta-sourcery, доступном по ссылке http://github.com/MentorEmbedded/meta-sourcery/.

В дополнение к настройке инструментов можно задать и файл задания для инструментария, который должен упаковать заранее собранные объекты, такие как libgcc, libstdcc++, файлы locale и libc.

15.22. Работа OE через межсетевой экран и прокси

Способ получения системой сборки исходных кодов поддерживает множество настроек. Вы можете обеспечить её доступ к исходным кодам в большинстве случаев, если доступен транспорт HTTP.

При поиске исходных кодов система сборки сначала просматривает локальный каталог загрузки, затем используется переменная PREMIRRORS, «восходящий» источник и переменная MIRRORS в указанном порядке.

Для дистрибутива poky система сборки OE использует источники YP из переменной PREMIRRORS по умолчанию в качестве источников SCM и обновлений для обычных архивов, а затем возвращается к другим зеркалам, если с зеркалом YP возникает отказ.

Например, можно добавить конкретный сервер, который система сборки будет проверять раньше других, путём включения в файл local.conf строк, подобных приведённым ниже.

```
PREMIRRORS_prepend = "\
git://.*/.* http://www.yoctoproject.org/sources/ \n \
ftp://.*/.* http://www.yoctoproject.org/sources/ \n \
http://.*/.* http://www.yoctoproject.org/sources/ \n \
https://.*/.* http://www.yoctoproject.org/sources/ \n"
```

Эти изменения заставят систему сборки перехватывать запросы Git, FTP, HTTP, HTTPS и направлять из зеркалу http://. Можно использовать URL file:// для указания локальных или сетевых каталогов.

Существует также другой вариант BB_NO_NETWORK = "1".

Этот оператор говорит BitBake о необходимости выдать ошибку вместо попытки доступа через Internet. Это полезно в тех случаях, когда нужно ограничиться сборкой из локальных источников.

Ещё одним решением служит BB_FETCH_PREMIRRORONLY = "1".

Этот оператор ограничивает систему сборки извлечением файлов лишь из источников, заданных переменной PREMIRRORS. Этот метод полезен для воспроизведения сборок.

Можно также использовать оператор BB_GENERATE_MIRROR_TARBALLS = "1".

Этот оператор заставляет систему сборки генерировать архивы зеркал. Такой подход полезен при необходимости создания сервера-зеркала. В остальных случаях он приведёт лишь к увеличению времени сборки.

В заключение рассмотрим пример работы через межсетевой экран, пропускающий только трафик HTTP. Можно внести приведённые ниже изменения в файл local.conf, если по умолчанию используется сервер из PREMIRRORS.

```
PREMIRRORS_prepend = "\
ftp://.*/.* http://www.yoctoproject.org/sources/ \n \
http://.*/.* http://www.yoctoproject.org/sources/ \n \
https://.*/.* http://www.yoctoproject.org/sources/ \n"
BB_FETCH_PREMIRRORONLY = "1"
```

Это заставит систему сборки извлекать все источники по протоколу HTTP, а все обращения к источникам, не указанным в PREMIRRORS будут приводить к отказу.

Система сборки также принимает во внимание стандартные переменные окружения http_proxy, ftp_proxy, https_proxy и all_proxy для перенаправления запросов на промежуточные серверы (proxy). Дополнительную информацию можно найти на странице Working Behind a Network Proxy.

15.23. Очистка результатов предыдущей сборки

Очистить результаты предыдущей сборки достаточно просто. При использовании BitBake для сборки образа весь вывод помещается в каталог, созданный при запуске сценария настройки окружения (например. oe-init-build-env). По умолчанию сборочный каталог называется build, но можно назвать его иначе. В сборочном каталоге имеется подкаталог tmp. Для удаления вывода сборки с сохранением исходных кодов и загруженных файлов достаточно удалить содержимое каталога tmp.

15.24. Странные значения ${bindir} и ${libdir} для заданий -native

Возможно придётся применять исполняемые файлы и библиотеки из каталога, отличающегося из того, в который они изначально установлены. Ситуацию осложняет то, что оной раз эти файлы и библиотеки компилируются с ожиданием запуска из заданного изначально каталога. В этом случае проблема решается переносом файлов.

Этот случай является фундаментальной проблемой для людей, поддерживающих пакеты основных дистрибутивов Linux, а также для системы сборки OE. Поэтому имеется устоявшееся решение проблемы. Предполагается, что файлы Makefile, инструменты автоматической настройки и другие системы сборки будут учитывать переменные окружения, такие как bindir, libdir и sysconfdir, указывающие расположение исполняемых файлов, библиотек и данных при фактическом выполнении программы. Также предполагается учёт переменной DESTDIR, которая добавляется перед всеми другими переменными при установке файлов системой сборки. Понятно, что на деле программа не запускается из DESTDIR.

Когда система OE использует задание для сборки программу под целевую архитектуру (т. е. одно из предназначенных для включения в создаваемый образ), эта программа в конечно итоге запускается из корневой файловой системы этого образа. Таким образом, система сборки подставляет значение /usr/bin для bindir, /usr/lib для libdir и т. д.

DESTDIR является путём в каталоге сборки. Однако при сборке заданием естественной (т. е. предназначенной для сборочного хоста) программы она не будет устанавливаться в корневую систему сборочного хоста. Поэтому система сборки использует пути внутри сборочного каталога для DESTDIR, bindir и связанных переменных. Чтобы лучше понять это, рассмотрим два пути, из которых первый представляется обычным, а второй - нет.

```
/home/maxtothemax/poky-bootchart2/build/tmp/work/i586-poky-linux/zlib/
   1.2.8-r0/sysroot-destdir/usr/bin

/home/maxtothemax/poky-bootchart2/build/tmp/work/x86_64-linux/
   zlib-native/1.2.8-r0/sysroot-destdir/home/maxtothemax/poky-bootchart2/
   build/tmp/sysroots/x86_64-linux/usr/bin
```

Пути эти кажутся необычными, но они корректны. Первый указывает цель, а второй предназначен для «естественного» задания. Эти пути являются результатом применения механизма DESTDIR и на практике весьма эффективны, несмотря на необычный вид.

15.25. Проблемы с заданиями *-native

Задание не доступно для других заданий. Файлы отсутствуют в естественном sysroot, задание устанавливается в некорректное место или возникают ошибки с правами доступа в задаче do_install.

Такая ситуация возникает, когда система сборки не распознает переменные окружения, предоставленные ей программой BitBake. Такая проблема возникала с файлом Makefile, в котором использовалась переменная окружения BINDIR вместо стандартной переменной bindir. Жёстко заданное значение /usr/bin работает в большинстве случаев, но не вариантом -native для задания. Ошибки с правами доступа могут быть вызваны файлом Makefile, который игнорирует DESTDIR или использует иное имя для этой переменной окружения.

Глава 16. Участие в разработке и дополнительная информация

16.1. Введение

Команда YP рада людям, экспериментирующим с YP. Имеется множество способов получения помощи при возникновении трудностей или ошибок. Здесь также представлена информация об участии в работе YP.

16.2. Вклад в разработку

YP с удовольствие принимает вклад других людей. Вы можете представить свои изменения в проект путём создания о и отправки запроса (pull) или прислав patch-файлы по электронной почте. Информация об обоих способах и определении ответственного за поддержку каждой области кода приведена в разделе Submitting a Change to the Yocto Project [2].

16.3. Yocto Project Bugzilla

YP использует свою реализацию Bugzilla для отслеживания дефектов (ошибок). Реализация of Bugzilla удобна для групповой работы, поскольку отслеживаются ошибки и изменения кода, которые могут использоваться для обмена информацией между разработчиками, представления изменений, обзора исправлений и контроля качества.

Иногда полезно зафиксировать, исследовать или отследить ошибку в самом YP (например, при возникновении проблем с поведением системы сборки, не соответствующим документации или ожиданиям).

Существуют базовые процедуры и рекомендации по представлению ошибок в Bugzilla. Информацию о фиксации ошибок в YP можно найти:

- в разделе Submitting a Defect Against the Yocto Project [2];
- на странице Bugzilla wiki.

Информация о системе Bugzilla в целом доступна по ссылке http://www.bugzilla.org/about/.

16.4. Списки рассылки

Имеется множество списков рассылки, поддерживаемых YP, а также списки OE для обсуждения, представления правок и анонсов. Подписаться на интересующую рассылку можно с помощью приведённых ниже ссылок.

- http://lists.yoctoproject.org/listinfo/yocto - общие вопросы YP.
- http://lists.openembedded.org/mailman/listinfo/openembedded-core - обсуждение OpenEmbedded-Core.
- http://lists.openembedded.org/mailman/listinfo/openembedded-devel - обсуждение OE.
- http://lists.openembedded.org/mailman/listinfo/bitbake-devel - обсуждение BitBake.
- http://lists.yoctoproject.org/listinfo/poky - обсуждение Poky.
- http://lists.yoctoproject.org/listinfo/yocto-announce - анонсы выпусков YP и другие важные сообщения.

Дополнительная информация о рассылках приведена на сайте Yocto Project.

16.5. Дополнительная информация

- *Сайт Yocto Project*
- *Страница Yocto Project Wiki*. Основная страница wiki для YP, содержащая сведения о планировании проекта, устройстве выпусков, контроле качества и автоматизации и т. п.
- *OpenEmbedded* - сборочная система YP, служащая основой для Poky.
- *BitBake* - инструмент для обработки метаданных.
- *Yocto Project Mega-Manual* - один файл HTML, включающий все руководства YP. Удобен для контекстного поиска.
- *FAQ* - ответы на часто задаваемые вопросы.

- *Release Notes.* Свойства, обновления и возможные проблемы для текущего выпуска YP. Документы Release Notes доступны на странице Downloads сайта YP по ссылке RELEASE INFORMATION для нужного выпуска.
- *Bugzilla* - система отслеживания ошибок в YP, позволяющая каждому сообщить о возникшей проблеме.
- *Bugzilla Configuration and Bug Tracking Wiki Page*. Информация об установке и использовании YP Bugzilla для записи и отслеживания дефектов YP.
- *Internet Relay Chat (IRC).* Два канала IRC для обсуждения YP (#yocto) и Poky (#poky).
- *Quick EMUlator (QEMU)*. Система эмуляции и виртуализации с открытым исходным кодом.

Литература

[1] Yocto Project Overview and Concepts Manual, https://www.yoctoproject.org/docs/2.7.1/overview-manual/overview-manual.html (перевод).

[2] Yocto Project Development Tasks Manual, http://www.yoctoproject.org/docs/2.7.1/dev-manual/dev-manual.html (перевод).

[3] Yocto Project Linux Kernel Development Manual, http://www.yoctoproject.org/docs/2.7.1/kernel-dev/kernel-dev.html (перевод).

[4] BitBake User Manual, https://www.yoctoproject.org/docs/2.7.1/bitbake-user-manual/bitbake-user-manual.html (перевод).

[5] Yocto Project Quick Build, https://www.yoctoproject.org/docs/2.7.1/brief-yoctoprojectqs/brief-yoctoprojectqs.html.

[6] Yocto Project Board Support Package (BSP) Developer's Guide, http://www.yoctoproject.org/docs/2.7.1/bsp-guide/bsp-guide.html (перевод).

[7] Yocto Project Application Development and the Extensible Software Development Kit (eSDK), http://www.yoctoproject.org/docs/2.7.1/sdk-manual/sdk-manual.html (перевод).

[8] Toaster User Manual, https://www.yoctoproject.org/docs/2.7.1/toaster-manual/toaster-manual.html.

[9] Yocto Project Profiling and Tracing Manual, https://www.yoctoproject.org/docs/2.7.1/profile-manual/profile-manual.html.

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru