Справочник по командному процессору Bash

Редакция 5.0 для Bash версии 5.0.

Декабрь 2018 г.

Chet Ramey, Case Western Reserve University.

Brian Fox, Free Software Foundation.

Оглавление

1. Введение

1.1. Что такое Bash?

Bash является «оболочкой» или интерпретатором команд для операционных систем GNU. Имя служит сокращением Bourne-Again Shell. Stephen Bourne является автором прямого предка современного интерпретатора Unix sh из седьмой редакции Bell Labs Research Unix.

Интерпретатор bash в значительной степени совместим с sh и включает полезные свойства интерпретаторов Korn (ksh) и C (csh). Предполагается, что это будет совместимая реализация части Shell and Tools спецификации IEEE POSIX (IEEE Standard 1003.1). Интерпретатор предлагает функциональные улучшения sh для интерактивного и программируемого применения. Хотя операционные системы GNU включают другие интерпретаторы, такие как csh, по умолчанию применяется bash. Как и другие программы GNU интерпретатор bash является переносимым. В настоящее время он работает практически на всех версиях Unix и некоторых других операционных систем. Имеются независимые реализации для платформ MS-DOS, OS/2 и Windows.

1.2. Что такое оболочка?

В своей основе оболочка является макропроцессором, выполняющим команды. Термина макропроцессор обозначает функциональность преобразования текста и символов в более крупные выражения.

Оболочки Unix являются одновременно интерпретаторами команд и языками программирования. В качестве интерпретатора команд оболочка служит интерфейсом между пользователем и обширным набором утилит GNU. Свойства языка программирования позволяют комбинировать эти утилиты. Могут создаваться файлы с командами, которые сами могут служить командой. Эти новые команды имеют такой же статус как системные команды в каталогах типа /bin, давая пользователям и группам возможность организации среды для автоматизации своих задач. Оболочки могут использоваться в интерактивном или командном режиме. В первом случае выполняются команды, введённые пользователем, во втором - считанные из файла.

Оболочка позволяет выполнять команды GNU в синхронном или асинхронном режиме. В синхронном режиме оболочка ждёт завершения команды перед восприятием последующего ввода, асинхронные команды могут выполняться параллельно с чтением и выполнением оболочкой других команд. Перенаправление позволяет контролировать ввод и вывод команд. Кроме того, оболочка позволяет контролировать содержимое командных сред.

Оболочки включают небольшой набор внутренних команд (builtin), реализующих функции, которые невозможно или неудобно реализовать в отдельных утилитах. Например, команды cd, break, continue и exec не могут быть реализованы вне оболочки, поскольку они напрямую манипулируют этой оболочкой. Команды history, getopts, kill или pwd, наряду с другими, могут быть реализованы в отдельных утилитах, но внутренние команды более удобны. Хотя интерпретация и выполнение команд важны, большая часть возможностей (и сложностей) оболочек связана со встроенными языками программирования. Подобно языкам высокого уровня они включают переменные, конструкции управления потоком, кавычки и функции.

Оболочки предоставляют также функции, предназначенные для интерактивного применения, а не для программирования. Эти интерактивные функции включают управление заданиями, редактирование команд, историю команд и псевдонимы.

2. Определения

POSIX
: Семейство стандартов для открытых систем на базе Unix. Bash относится в основном к части Shell and Utilities стандарта POSIX 1003.1.

blank
: Символ пробела или табуляции.

builtin - внутренняя команда
: Команда, реализованная внутри оболочки, а не во внешней программе на файловой системе.

control operator - оператор управления
: Маркер, выполняющий функцию управления. Это может быть перевод строки или одна из последовательностей ||, &&, &, ;, ;;, ;&, ;;&, |, |&, (,).

exit status - статус выхода
: Значение, возвращаемое командой по завершении (0 - 255).

field - поле
: Блок текста, являющийся результатом одного или нескольких shell-преобразований (expansion). После преобразования при выполнении команды результирующие поля применяются как имена команд и аргументы.

filename - имя файла
: Строка символов, служащая для идентификации файла.

job -задание
: Набор процессов, образующих конвейер (pipeline), и все процессы-потомки, относящиеся к одной группе.

job control - управление заданием
: Механизм, с помощью которого можно селективно останавливать (suspend) и возобновлять (resume) процессы.

metacharacter - метасимвол
: Символ, который при использовании без кавычек является разделителем слов. Метасимволы включают пробел, символ табуляции, перевод строки, а также символы |, &, ;, (,), <, и >.

name - имя
: Слово, состоящее лишь из букв, цифр и символов подчёркивания, начинающееся с буквы или символа подчёркивания. Имена используется для shell-переменных и функций, иногда их называют идентификаторами.

operator - оператор
: Оператор управления или перенаправления (см. параграф 3.6. Перенаправление). Оператор содержит хотя бы один метасимвол без кавычек.

process group - группа процессов
: Набор связанных процессов, имеющих общий идентификатор группы.

process group ID - идентификатор группы процессов
Уникальный идентификатор, представляющий группу процессов при их выполнении.

reserved word - зарезервированное слово
Слово, имеющее в оболочке специальное назначение. Большинство зарезервированных слов служит для создания конструкций управления потоком, например, for и while.

return status - статус возврата
Синоним статуса выхода (завершения).

signal - сигнал
Механизм, с помощью которого ядро может уведомить процесс о произошедшем в системе событии.

special builtin - специальная внутренняя команда
Внутренняя команда оболочки, указанная стандартном POSIX как специальная (особая).

token - маркер
Последовательность символов, воспринимаемая оболочкой как единый блок. Маркер - это слово или оператор.

word - слово
Последовательность символов, трактуемая как слово. Слова не могут включать метасимволы без кавычек.

3. Базовые свойства

Bash - это сокращение Bourne-Again SHell. Bourne shell - традиционная оболочка Unix, написанная изначально Stephen Bourne. Все внутренние функции Bourne shell доступны в bash, правила преобразования и работы с кавычками взяты из спецификации POSIX для «стандартной» оболочки Unix.

В этой главе кратко описаны основные компоненты оболочек - команды, управляющие структуры, функции, параметры, преобразования, перенаправления, обеспечивающие возможность направлять ввод и вывод в именованные файлы и из них, а также выполнение команд оболочки.

3.1. Синтаксис

При чтении входных данных выполняется последовательность операций. Если входные данные указывают начало комментария, командный процессор игнорирует символ # и оставшуюся часть строки.

В иных случаях, упрощенно говоря, оболочка считывает ввод и делит его на слова и операторы, используя правила преобразования кавычек для выбора трактовки слов и символов.

Затем процессор разбирает полученные маркеры на команды и другие конструкции, удаляя специальную трактовку некоторых слов и символов и преобразуя иные, перенаправляет ввод и вывод, выполняет специальные команды, ждёт результата завершения команды и делает код завершения доступным для дальнейшей проверки или обработки.

3.1.1. Операции

Ниже приведён краткий список операций командного процессора при чтении и выполнении команд

1. Чтение ввода из файла (3.8. Сценарии оболочки), из строки, представленной в качестве аргумента опции -c при вызове (6.1. Вызов Bash), или с пользовательского терминала.
2. Разбиение входных данных на слова и операторы в соответствии с правилами обработки кавычек из параграфа 3.1.2. Кавычки. Эти маркеры разделяются метасимволами. Здесь же выполняется преобразование псевдонимов (6.6. Псевдонимы).
3. Разбор маркеров на простые и составные команды (3.2. Команды).
4. Выполнение различных преобразований (3.5. Преобразования) с разбиением преобразованных маркеров на имена файлов (3.5.8. Преобразование имён файлов), команды и аргументы.
5. Выполнение требуемых перенаправлений (3.6. Перенаправление) с удалением операторов и операндов перенаправления из списка аргументов.
6. Выполнение команды (3.7. Выполнение команд).
7. Необязательное ожидание завершения команды и получения статуса выхода (3.7.5. Статус выхода).

3.1.2. Кавычки

Кавычки служат для обхода специальной трактовки некоторых символов и слов командным процессором. Это может использоваться для отключения специальной трактовки особых символов и зарезервированных слов, а также для отключения преобразований параметров.

Каждый из метасимволов (2. Определения) имеет для командного процессора особый смысл и должен помещаться в кавычки для обычной трактовки. При использовании средств преобразования истории команд (9.3. Преобразование истории) символ преобразования истории (обычно !) должен заключаться в кавычки, чтобы предотвратить преобразование. Более подробно преобразование истории рассмотрено в параграфе 9.1. Средства Bash History.

Имеется три механизма «кавычек» - escape-символы, одинарные и двойные кавычки.

3.1.2.1. Экранирование символов

Символ обратной дробной черты (\) без кавычек является в Bash символом экранирования (escape). Он сохраняет буквальное значение следующего за ним символа, за исключением перевода строки. При получении последовательности \newline без кавычек она трактуется как продолжение текста на следующей строке (т. е. просто удаляется из входного потока).

3.1.2.2. Одинарные кавычки

Заключение символов в одинарные кавычки (') обеспечивает буквальную трактовку каждого символа внутри кавычек. Одинарные кавычки не могут помещаться внутри другой пары одинарных кавычек даже с символом экранирования.

3.1.2.3. Двойные кавычки

Заключение символов в двойные кавычки (") обеспечивает буквальную трактовку каждого символа внутри кавычек, за исключением символов $, ‘, \, а при преобразовании истории ещё и символа !. При работе оболочки в режиме POSIX (6.11. Режим POSIX) символ ! не имеет особого значения внутри двойных кавычек даже при включённом преобразовании истории. Символы $ и ‘ сохраняют особое значение даже в двойных кавычках (3.5. Преобразования). Символ \ сохраняет особое значение лишь в тех случаях, когда за ним следует символ $, ‘, ", \ или newline (в таких случаях символ просто удаляется). Символ \ перед символами, не имеющими особого значения, сохраняется. Двойные кавычки могут применяться внутри двойных кавычек при использовании с символом экранирования \. При включённом преобразовании истории оно будет выполняться, пока символ ! в двойных кавычках не экранирован символом \ (не удаляется перед !). Специальные символы * и @ имеют особую трактовку в двойных кавычках (3.5.3. Преобразование параметров оболочки).

3.1.2.4. Кавычки ANSI-C

Слова в форме $’string’ обрабатываются особо. Слово преобразуется в строку с заменой экранированных \ символов в соответствии со стандартом ANSI C. Escape-последовательности декодируются, как описано ниже.

\a	Сигнал (звонок).
\b	«Забой» (backspace).
\e	Символ экранирования (не ANSI C).
\E	
\f	Перевод страницы (form feed).
\n	Новая строка.
\r	Возврат каретки.
\t	Горизонтальная табуляция.
\v	Вертикальная табуляция.
****	Обратная дробная черта (\).
\’	Одинарная кавычка.
\"	Двойная кавычка.
\?	Знак вопроса.
\nnn	Восьмибитовый символ с восьмеричным кодом nnn (1-3 восьмеричных цифры).
\xHH	Восьмибитовый символ с шестнадцатеричным кодом HH (1-2 шестнадцатеричных цифры).
\uHHHH	Символ Unicode (ISO/IEC 10646) с шестнадцатеричным кодом HHHH (1-4 шестнадцатеричных цифры)
\UHHHHHHHH	Символ Unicode (ISO/IEC 10646) с шестнадцатеричным кодом HHHHHHHH (1-8 шестнадцатеричных цифр)
\cx	Символ control-x.

Преобразованный результат помещается в одинарные кавычки как при отсутствии символа $.

3.1.2.5. Зависимая от языка трансляция

Строка в двойных кавычках с предшествующим символом $ будет преобразовываться в соответствии с текущим языком (locale). Если в качестве locale задан C или POSIX, символ $ игнорируется. Если строка транслируется и заменяется, результат помещается в двойные кавычки. Некоторый системы используют каталог сообщений, определяемый переменной LC_MESSAGES, другие создают имя каталога сообщений из значения переменной окружения TEXTDOMAIN, возможно с добавлением суффикса .mo. При использовании переменной TEXTDOMAIN может потребоваться установка в TEXTDOMAINDIR местоположения файлов каталога сообщений. Иногда используются обе переменные в стиле TEXTDOMAINDIR/LC_MESSAGES/LC MESSAGES/TEXTDOMAIN.mo.

3.1.3. Комментарии

В неинтерактивной оболочке или в интерактивной со включённой опцией interactive_comments внутренней функции shopt (4.3.2. Внутренняя команда shopt) слово, начинающееся с # и последующие символы до конца строки игнорируются. Интерактивная оболочка без включённой опции interactive_comments не поддерживает комментарии. Опция interactive_comments включена по умолчанию в интерактивных оболочках, описание которых приведено в разделе 6.3. Интерактивные оболочки.

3.2. Команды

Простые команды, такие как **`echo a b c`**, состоят из слова команды и разделённых пробелами аргументов. Более сложные команды состоят из простых команд, сгруппированных тем или иным способом - в конвейер, где вывод одной команды служит вводом для другой, цикл или конструкцию с условием.

3.2.1. Простые команды

Простые команды применяются наиболее часто и представляют собой последовательность слов, разделённых пробелами, завершающуюся одним из операторов управления (2. Определения). Первое слово обычно задаёт команду для выполнения, а последующие служат аргументами команды.

Статус возврата (3.7.5. Статус выхода) для простой команды является статусом выхода, возвращённым функцией POSIX 1003.1 waitpid, или 128+n, если команда была прервана сигналом n.

3.2.2. Конвейеры

Конвейер - это одна или несколько команд, разделённых оператором | или |&. Формат конвейера имеет вид

```
[time [-p]] [!] command1 [ | or |& command2 ] ...
```

Вывод каждой команды в конвейере соединяется через канал (pipe) со входом следующей команды, т. е. каждая команда читает результат предыдущей команды. Соединение организуется до перенаправлений, заданных командой.

При использовании оператора |& стандартный вывод ошибок command1 вместе со стандартным выводом соединяется со стандартным вводом command2 через канал. Это сокращённо задаётся в форме 2>&1 |. Неявное перенаправление стандартного вывода ошибок на стандартный вывод выполняется после перенаправлений, заданных командой.

Зарезервированное слово time активизирует вывод статистики для конвейера по завершении. Статистика включает прошедшее время, а также время выполнения команды в пользовательском и системном пространстве. Опция -p устанавливает формат вывода POSIX. При работе оболочки в режиме POSIX (6.11. Режим POSIX) слово time не считается зарезервированным, если за ним следует маркер, начинающийся с символа -. В переменной TIMEFORMAT может быть задана строка, управляющая выводом значений времени (5.2. Переменные Bash). Использование time в качестве зарезервированного слова разрешено во внутренних функциях, функциях оболочки и конвейерах.

При работе оболочки в режиме POSIX (6.11. Режим POSIX) слово time следовать за newline. В этом случае оболочка выводит время в пользовательском и системном пространстве, затраченное оболочкой и её потомками. Для задания формата вывода времени может применяться переменная TIMEFORMAT.

Если конвейер не выполняется асинхронно (3.2.3. Списки), оболочка ждёт завершения всех команд конвейера.

Каждая команд конвейера выполняется в своей субоболочке, которая является отдельным процессом (3.7.3. Среда выполнения команды). Если включена опция lastpipe при использовании внутренней команды shopt (4.3.2. Внутренняя команда shopt), последний элемент конвейера может запускаться процессом оболочки.

Статусом выход из конвейера служит статус завершения последней команды, пока не включена опция pipefail (4.3.1. Внутренняя команда set). При включённой опции pipefail статусом завершения конвейера будет последний отличный от нуля код завершения команды, или 0, если все команды конвейера завершились успешно. Если перед конвейером указан символ !, статусом завершения будет логическое обращение описанного выше статуса. Оболочка ждёт завершения всех команд конвейера перед возвратом значения.

3.2.3. Списки

Список включает один или несколько конвейеров, разделённых операторами ;, &, && или ||, которые могут завершаться символом ;, &, или newline. Операторы списка && и || имеют одинаковый приоритет, за ними следуют ; и &, также имеющие одинаковый приоритет. В списке может присутствовать один или несколько символов newlinesдля разделения команд (эквивалент ;).

Если команда завершается оператором управления &, она выполняется асинхронно в субоболочке. Это называется выполнением команды в фоновом режиме (background), а здесь такие команды называются асинхронными. Оболочка не ждёт завершения команды и возвращает статус 0 (true). При активном управлении заданиями (7. Управление заданиями) стандартным вводом асинхронных команд при отсутствии явного перенаправления служит /dev/null.

Команды, разделённые символом ;, выполняются последовательно и оболочка дожидается выполнения каждой команды по очереди. Статусом возврата служит код завершения последней выполненной команды. Списки && и || являются последовательностями из одного или нескольких конвейеров, разделённых символами && или || и выполняемых слева направо.

В списке

```
command1 && command2
```

command2 выполняется тогда и только тогда, когда команда command1 вернула статус 0 (успех). А в списке

```
command1 || command2
```

command2 выполняется тогда и только тогда, когда команда command1 вернула отличный от 0 статус. Статусом возврата этих списков является статус выхода последней выполненной команды.

3.2.4. Составные команды

Составные команды являются конструкциями языка программирования оболочки. Каждая конструкция начинается зарезервированным словом или оператором и заканчивается соответствующим зарезервированным словом или оператором. Все перенаправления (3.6. Перенаправление) связанные с составной командой, применяются к каждой команде внутри конструкции, если явно не задано иное.

В большинстве случаев список команд в описании составной команды может быть отделен от остальной команды одним или несколькими символами новой строки и может сопровождаться в конце символом newline вместо ;.

Bash поддерживает конструкции с циклами, условиями и группировкой для выполнения команд как единого блока.

3.2.4.1. Конструкции с циклами

Ниже описаны конструкции с циклами, поддерживаемые bash. Отметим, что символ ; в циклах можно заменить одним или несколькими символами новой строки.

unit

Синтаксис цикла unit имеет вид

```
unit test-commands; do consequent-commands; done
```

Выполнение consequent-commands продолжается до тех пор, пока test-commands не возвратит отличный от 0 статус выхода. Статусом выхода из цикла является статус выхода последней выполненной команды из consequent-commands или 0, если ничего не выполнялось.

while

Синтаксис цикла while имеет вид

```
while test-commands; do consequent-commands; done
```

Выполнение consequent-commands продолжается до тех пор, пока test-commands не возвратит статус выхода 0. Статусом выхода из цикла является статус выхода последней выполненной команды из consequent-commands или 0, если ничего не выполнялось.

for

Синтаксис цикла for имеет вид

```
for name [ [in [words ...] ] ; ] do commands; done
```

Происходит преобразование words (3.5. Преобразования) и выполняются команды один раз для каждого элемента в полученном списке с именем, привязанным к текущему элементу. При отсутствии in words цикл for выполняет команды один раз для каждого позиционного параметра, который установлен, как при указании in "$@" (3.4.2.

Специальные параметры). Статусом возврата цикла является статус выхода последней выполненной команды. Если нет элементов при преобразовании words, никакие команды не выполняются и возвращается статус 0.

Поддерживается также другая форма цикла for в виде

```
for (( expr1 ; expr2 ; expr3 )) ; do commands ; done
```

Сначала вычисляется арифметическое выражение expr1 в соответствии с правилами, описанными ниже (6.5. Арифметика командного процессора). Затем в цикле вычисляется арифметическое выражение expr2, пока оно не даст результат 0 и в каждом цикле с отличным от 0 значением выполняются команды commands и вычисляется арифметическое выражение expr3. Если любое из выражений опущено, оно считается имеющим значение 1. Статусом возврата служит статус выхода последней выполненной команды или false, если какое-либо из выражений недействительно.

Внутренние операторы break и continue (4.1. Внутренние элементы Bourne Shell) могут служить для управления циклом.

3.2.4.2. Конструкции с условием

if

Синтаксис конструкции if показан ниже.

```
if test-commands; then
        consequent-commands;
[elif more-test-commands; then
        more-consequents;]
[else alternate-consequents;]
fi
```

Выполняется список test-commands и, если он возвращает 0, выполняется список consequent-commands. Если test-commands возвращает ненулевой статус, выполняется список elif и при нулевом статусе возврата выполняется список more-consequents. При наличии else alternate-consequents и ненулевом статусе возврата в финальном операторе if или elif, выполняется список команд alternate-consequents. Статусом возврата будет статус выхода последней выполненной команды или 0, если ни одно из условий не было выполнено.

case

Синтаксис конструкции case показан ниже.

```
case word in
        [ [(] pattern [| pattern]...) command-list ;;]...
esac
```

В конструкции case будет селективно выполняться список command-list, соответствующий первому совпадению. Сопоставление выполняется в соответствии с правилами параграфа 3.5.8.1. Сопоставление с шаблоном. Если включена опция nocasematch (4.3.2. Внутренняя команда shopt), при сопоставлении регистр символов не принимается во внимание. Символ | служит для разделения шаблонов сопоставления, а) завершает список шаблонов, который вместе со связанным command-list называют clause (условие).

Каждое условие должно завершаться последовательностью ;;, ;& или ;;&. Для word до сравнения выполняется преобразование тильды и параметров, подстановка команд, арифметическое преобразование и удаление кавычек (3.5.3. Преобразование параметров оболочки). Для каждого шаблона pattern выполняется преобразование тильды и параметров, подстановка команд и арифметическое преобразование

В конструкции может использоваться произвольной число условий (clause), каждое из которых завершается последовательностью ;;, ;& или ;;&. Первый соответствующий шаблон pattern определяет выполняемый список command-list. Часто в качестве последнего варианта применяется шаблон *, которому соответствует все.

Ниже приведён пример использования case в сценарии для описания одной интересной особенности животных.

```
echo -n "Enter the name of an animal: "
read ANIMAL
echo -n "The $ANIMAL has "
case $ANIMAL in
        horse | dog | cat) echo -n "four";;
        man | kangaroo ) echo -n "two";;
        *) echo -n "an unknown number of";;
esac
echo " legs."
```

При использовании в конце условия оператора ;; после первого совпадения дополнительных сопоставлений не проводится. При использовании ;& вместо ;; продолжается выполнение списка command-list, связанного со следующим условием. Использование ;;& вместо ;; заставляет оболочку проверять шаблоны следующего условия и выполнять связанный с ним список command-list при совпадении.

Если ни один шаблон не подошёл, возвращается статус 0, в остальных случаях возвращается статус выхода последнего command-list.

select

С помощью конструкции select легко создавать меню. Синтаксис конструкции похож на синтаксис for.

```
select name [in words ...]; do commands; done
```

Список words после in преобразуется, создавая список элементов. Набор преобразованных слов выводится в стандартный выходной поток ошибок с указанием номера каждого слова. При отсутствии in words выводятся позиционные параметры, как будто указано in "$@". Затем выводится приглашение PS3 и считывается строка со стандартного ввода. Если эта строка содержит число, соответствующее одному из выведенных слов, для переменной name устанавливается значение этого слова. Если строка пуста, слова и приглашения выводятся вновь. Если прочитан символ EOF, выполнение select завершается. При всех прочих значениях ввода для name устанавливается пустое значение. Считанная строка сохраняется в переменной REPLY.

Команды commands выполняются после каждого выбора, пока не будет выполнена команда break, после чего работа select завершается.

Ниже приведён пример, позволяющий выбрать имя файла в текущем каталоге и вывести имя и индекс файла.

```
select fname in *;
do
echo you picked $fname \($REPLY\)
break;
done
```

((...))

```
(( expression ))
```

Арифметическое выражение, вычисляемое в соответствии с правилами параграфа 6.5. Арифметика командного процессора. Если значение отлично от нуля, возвращается статус 0, в противном случае - 1. Это полный эквивалент выражения let "expression" (см. раздел 4.2. Внутренние команды bash).

[[...]]

```
[[ expression ]]
```

Возвращает 0 или 1 в зависимости от вычисления условного выражения expression. Выражения состоят из примитивов, описанных в разделе 6.4. Условные выражения bash. Расщепление слов и преобразование имён файлов не выполняется между скобками [[ ]], но выполняется преобразование тильды, параметров и переменных, преобразование арифметических выражений, подстановка команд и процессов, а также удаление кавычек. Условные операторы, такие как -f должны указываться без кавычек.

При использовании с [[операторы < и > сортируются лексикографически в соответствии с языковыми настройками. При использовании операторов == и != строка справа от оператора считается шаблоном и сопоставление выполняется по правилам, описанным в параграфе 3.5.8.1. Сопоставление с шаблоном, как при включённой опции extglob. Оператор = идентичен ==. При включённой опции nocasematch (4.3.2. Внутренняя команда shopt) сопоставление выполняется без учёта регистра символов. Возвращается значение 0 при соответствии строк для == и несоответствии для !=, в остальных случаях возвращается значение 1. Любая часть шаблона может быть заключена в кавычки для её сопоставления как строки.

При доступности дополнительного бинарного оператора =~ он имеет такой же порядок применения как == и !=. При его использовании строка справа считается регулярным выражением POSIX и сопоставляется соответствующим образом (как в regex 3). Возвращается 0 при соответствии строки шаблону и 1 в противном случае. Если регулярное выражение синтаксически некорректно, условное выражение возвращает 2. При включённой опции nocasematch (4.3.2. Внутренняя команда shopt) сопоставление выполняется без учёта регистра символов. Любая часть шаблона может быть заключена в кавычки для её сопоставления как строки. Заключённые в скобки части регулярных выражений должны обрабатываться осторожно, поскольку обычные символы кавычек утрачивают свой смысл между скобками. Если шаблон хранится в переменной оболочки, преобразование переменной в кавычках ведёт к сопоставлению всего шаблона как строки. Подстроки, соответствующие заключённым в скобки субвыражениям в регулярном выражении сохраняются в массиве переменной BASH_REMATCH. Элемент BASH_REMATCH с индексом 0 является частью строки, соответствующей всему регулярному выражению. Элемент с индексом n является частью строки, соответствующей n-му субвыражению в скобках.

Например, приведённое ниже выражение будет соответствовать строке line (сохранённой в переменной оболочки line), если в значении есть последовательность символов, содержащая любое число (включая 0) пробелов или экземпляров символа a, за которым следует b.

```
[[ $line =~ [[:space:]]*?(a)b ]]
```

В результате такие значения, как ‘aab’ и ‘ aaaaaab’ дадут совпадение, как и строки, содержащие лишь символ b.

Сохранение регулярного выражения в переменной оболочки зачастую полезно для предотвращения проблем с кавычками для символов, имеющих в оболочке особое значение. Иногда сложно задать регулярное выражение буквально без использования кавычек или сохранить след применения кавычек регулярным выражением с учётом удаления кавычек оболочкой. Использование переменной для сохранения шаблона снижает вероятность таких проблем. Ниже приведён аналог предыдущего примера с использованием такой переменной.

```
pattern='[[:space:]]*?(a)b'
[[ $line =~ $pattern ]]
```

Если нужно сопоставление с символом, имеющим особое значение в регулярных выражениях, шаблон следует поместить в кавычки. Например в шаблоне xxx.txt точка (.) соответствует любому символу в строке (обычная трактовка в регулярном выражении), а шаблон "xxx.txt" будет требовать точного совпадения. Программистам следует внимательно относиться к символам \, поскольку в оболочке и регулярных выражениях они задают специальную трактовку следующего символа. Например,

```
pattern='\.'
[[ . =~ $pattern ]]
[[ . =~ \. ]]
[[ . =~ "$pattern" ]]
[[ . =~ '\.' ]]
```

Первые два сопоставления дадут совпадение, остальные - нет, поскольку в них символ \ является частью шаблона (кавычки). В двух первых примерах символ \ отменяет специальную трактовку точки, поэтому символ точки будет давать совпадение. Если бы строка в двух первых примерах отличалась от ‘.’ (например, ‘a’), совпадения бы не было, поскольку специальная трактовка точки отменена символом \.

Выражения можно комбинировать с помощью приведённых ниже операторов, указанных в порядке применения.

(expression)	Возвращает значение expression и может применяться для изменения порядка применения операторов.
! expression	Возвращает true, если expression имеет значение false.
expression1 && expression2	Возвращает true, если expression1 и expression2 имеют значение true.
expression1 || expression2	Возвращает true, expression1 или expression2 имеет значение true.

С операторами && и || выражение expression2 не будет вычисляться, если expression1 уже определяет результат.

3.2.4.3 Команды группировки

Bash обеспечивает два способа группировки команд с целью выполнения единым блоком. При группировке команд перенаправление может применяться ко всему набору команд. Например, вывод команд может перенаправляться в один поток.

()

```
( list )
```

Указание списка команд в круглых скобках вызывает создание среды субоболочки (3.7.3. Среда выполнения команды) и выполнение каждой команды из списка. Выполнение команд в субоболочке ведёт к тому, что назначения переменных, выполненные командами, не сохраняются по завершении работы.

{}

```
{ list; }
```

Указание списка команд в фигурных скобках вызывает выполнение этих команд в контексте текущей оболочки без создания субоболочек. Список должен завершаться символом ; или newline.

Кроме создания субоболочки эти варианты имеют различие, обусловленное историческими причинами. Символы фигурных скобок являются зарезервированными, поэтому между скобками и списком команд должен присутствовать

пробел или другой метасимвол. Круглые скобки являются оператором и воспринимаются как отдельные маркеры даже при отсутствии пробелов между ними и списком. Для обоих вариантов статусом завершения является статус выхода списка команд.

3.2.5. Копроцессы

Копроцесс (coprocess) - это команда оболочки, которой предшествует зарезервированное слово coproc. Копроцесс выполняется асинхронно в субоболочке, как будто команда содержала в конце управляющий оператор &, с двунаправленным каналом между оболочкой и копроцессом. Формат запуска копроцесса показан ниже.

```
coproc [NAME] command [redirections]
```

Эта команда создаёт копроцесс с именем NAME. Если параметр NAME не задан, применяется имя COPROC. Параметр NAME недопустимо применять с простой командой command (3.2.1. Простые команды), поскольку он будет считаться первым словом простой команды.

При выполнении копроцесса оболочка создаёт переменную массива (6.7. Массивы) с именем NAME в контексте выполняемой оболочки. Стандартный вывод команды соединяется через канал с дескриптором файла в исполняющей оболочке и этот дескриптор назначается переменной NAME[0]. Стандартный ввод команды соединён каналом с дескриптором файла в исполняющей оболочке и этот дескриптор назначен переменной NAME[1]. Этот канал организуется до перенаправлений, заданных командой (3.6. Перенаправление). Дескрипторы файлов могут служить аргументами команд оболочки и перенаправлений с использованием стандартного преобразования слов. Кроме дескрипторов, созданных специально для выполнения подстановок команд и процессов, в субоболочке нет доступных дескрипторов файлов.

Идентификатор процесса вызванной для выполнения копроцесса оболочки доступен как значение переменной NAME_PID. Можно использовать внутреннюю команду wait для ожидания завершения копроцесса.

Поскольку копроцесс создаётся как асинхронная команда, coproc всегда возвращает успешный результат. Статусом возврата копроцесса является статус выхода команды command.

3.2.6. GNU Parallel

Имеются способы параллельного выполнения команд, не встроенные в Bash и одним из них является GNU Parallel. Как ясно из имени, GNU Parallel может использоваться для сборки и запуска команд в параллель. Можно запустить одну команду с разными аргументами, будь то имена файлов, пользователей и хостов или строки, прочитанные из файлов. GNU Parallel обеспечивает сокращения для большинства базовых операций (строки ввода и их части, способы задания источника ввода и т. п.). Parallel может заменять команды xargs или feed из различных входов в нескольких разных экземплярах Bash. Полное описание пакета можно найти в документации GNU Parallel, а ниже приведено несколько кратких примеров использования.

Легко заменить xargs для упаковки gzip всех файлов html в текущем каталоге и его подкаталогах, как показано ниже.

```
find . -type f -name '*.html' -print | parallel gzip
```

Если нужно «защитить» в именах специальные символы, такие как newline, можно использовать опцию find -print0 и опцию parallel -0.

Можно применить Parallel для переноса файлов из текущего каталога, когда число файлов слишком велико для обработки командой mv.

```
ls | parallel mv {} destdir
```

В этом случае {} заменяется каждой строкой со стандартного ввода. Хотя команда ls будет работать в большинстве случаев, её недостаточно для работы со всеми возможными именами. Если нужно обрабатывать файлы со специальными символами в именах, можно применить команду вида

```
find . -depth 1 \! -name '.*' -print0 | parallel -0 mv {} destdir
```

Эта команда запустит столько команд mv, сколько имеется файлов в текущем каталоге. Можно эмулировать параллельное использование xargs с помощью опции -X, как показано ниже.

```
find . -depth 1 \! -name '.*' -print0 | parallel -0 -X mv {} destdir
```

GNU Parallel может заменить некоторые распространённые выражения для работы со строками, считываемыми из файла. В приведённом ниже примере это имена файлов, указанные по одному в строке. Обычный вариант имеет вид

```
while IFS= read -r x; do
do-something1 "$x" "config-$x"
do-something2 < "$x"
done < file | process-output
```

Но это можно выразить более компактно в форме

```
cat list | parallel "do-something1 {} config-{} ; do-something2 < {}" |
           process-output
```

Parallel обеспечивает встроенный механизм удаления расширений в именах файлов, пригодный для массового преобразования или переименования файлов. Например,

```
ls *.gz | parallel -j+0 "zcat {} | bzip2 >{.}.bz2 && rm {}"
```

Эта команда будет распаковывать все файлы в текущем каталоге, имеющие расширение .gz, используя программу bzip2 с запуском одного задания на процессор (-j+0) в параллель. В примере для краткости использована команда ls, но можно использовать find для обработки файлов со специальными символами в именах. Parallel может принимать аргументы из командной строки и предыдущий пример можно выразить в форме

```
parallel "zcat {} | bzip2 >{.}.bz2 && rm {}" ::: *.gz
```

Если команда создаёт вывод, может оказаться желанным связать порядок ввода с порядком вывода. Например,

```
{
        echo foss.org.my ;
        echo debian.org ;
        echo freenetproject.org ;
```

```
} | parallel traceroute
```

будет выводить сначала результат traceroute для закончившейся первой трассировки. Если добавить опцию -k

```
{
        echo foss.org.my ;
        echo debian.org ;
        echo freenetproject.org ;
} | parallel -k traceroute
```

первым будет выведен результат трассировки foss.org.my.

Кроме того, parallel можно применять для запуска последовательности команд оболочки в параллель, подобно cat file | bash. Нет ничего необычного в том, чтобы взять список имён файлов, создать серию команд для работы с ними и передать этот список команд в оболочку. Parallel позволяет ускорить обработку. В предположении, что file содержит нужный список команд (по одной в строке) это может иметь вид

```
parallel -j 10 < file
```

Команды будут выполняться параллельно после их преобразования (поскольку команды не заданы явно) блоками по 10 одновременных заданий.

3.3. Функции оболочки

Shell-функции обеспечивают способ группировки команд для последующего выполнения с использованием одного имени для группы. Функции выполняются подобно «обычным» командам. Когда имя функции применяется как имя простой команды, выполняется список команд, связанных с именем функции. Shell-функции выполняются в контексте текущей оболочки без создания нового процесса.

Синтаксис объявления функции имеет форму

```
name () compound-command [ redirections ]
```

или

```
function name [()] compound-command [ redirections ]
```

Это определяет функцию с именем name. Зарезервированное слово function является необязательным, а при его указании круглые скобки становятся необязательными. Тело функции составляет набор команд compound-command (3.2.4. Составные команды). Обычно составная команда представляет собой список команд заключенный в фигурные скобки { }, но это может быть любая составная команда с одним исключением - при использовании зарезервированного слова function без круглых скобок фигурные скобки являются обязательными. В режиме POSIX (6.11. Режим POSIX), имя name не может совпадать с именем какой-либо из специальных внутренних функций (4.4. Специальные внутренние команды). Все перенаправления (3.6. Перенаправление), связанные с функцией, применяются при выполнении функции.

Определение функции можно удалить с помощью опции -f внутренней функции unset (4.1. Внутренние элементы Bourne Shell).

Статус выхода при определении функции имеет значение 0, если не возникло синтаксической ошибки и нет уже функции с таким именем, доступной лишь для чтения. При выполнении функции статус определяется статусом выхода последней выполненной в теле функции команды.

Отметим, что в силу исторических причин фигурные скобки вокруг тела функции в большинстве случаев отделены от тела пробелом или символом новой строки. Это связано с тем, что фигурные скобки являются зарезервированными словами и для распознания в таком качестве должны быть отделены пробелом или иным метасимволом. Кроме того, при использовании фигурных скобок список команд должна завершаться символом ;, & или newline.

При выполнении функции её аргументы становятся позиционными параметрами (3.4.1. Позиционные параметры). Специальный параметр #, который преобразуется в число позиционных параметров, обновляется с учётом изменений. Специальный параметр 0 не меняется. Первому элементу переменной FUNCNAME присваивается имя функции при её выполнении.

Все остальные аспекты среды выполнения оболочки идентичны для функции и вызвавшего её процесса, за исключением того, что прерывания DEBUG и RETURN не наследуются, если для функции не задан атрибут трассировки trace с помощью внутренней функции declare или опции -o внутренней функции set, а прерывание ERR не наследуется, пока не задана опция -o errtrace. Внутренняя функция trap описана в разделе 4.1. Внутренние элементы Bourne Shell.

При установке в переменной FUNCNEST численного значения больше 0 она определяет максимальный уровень вложенности функций. Превышение этого уровня ведёт к прерыванию команды.

Если в функции выполняется внутренняя команда return, работа функции завершается и выполняется команда, следующая за вызовом функции. Любые команды, связанные с прерыванием RETURN выполняются перед возвратом управления. Когда функция завершается, позиционные параметры и специальный параметр # восстанавливают значения, которые были перед вызовом функции. Если функции return передано числовое значение, оно будет служить статусом возврата. В остальных случаях функция возвращает статус выхода последней выполненной команды.

Локальные переменные функции можно объявить с помощью встроенного оператора local. Эти переменные будут видны только функции и вызываемым ею командам.

Локальные переменные «затеняют» одноимённые переменные, объявленные до вызова функции (вне её). Например, локальная переменная, определённая внутри функции, будет скрывать одноимённую глобальную переменную. Ссылки и назначения будут относиться к локальной переменной, не затрагивая глобальную. При возврате из функции глобальная переменная вновь станет видимой.

Оболочка применяет динамическую область действия для управления видимостью переменных и их значения являются результатом последовательности вызовов функций, которые приведи к текущей функции. Значение переменной, которое видит функция, зависит от её значения у вызывающей стороны. Например, если переменная var

объявлена как локальная в func1 и func1 вызывает другую функцию func2, ссылки на var внутри func2 будет относиться к локальной переменной var из func1, затеняя любую глобальную переменную var, если она имеется. Например,

```
In func2, var = func1 local
func1()
{
        local var='func1 local'
        func2
}
func2()
{
        echo "In func2, var = $var"
}
var=global
func1
```

Внутренняя функция unset также имеет динамическую область действия. Если переменная определена локально, unset будет сбрасывать её, а в противном случае искать эту переменную ы вызывающей области. Если переменная в локальной области действия сброшена, это будет сохраняться до её установки в этой области или возврата из функции. После возврата из функции все экземпляры переменных вызвавшей функцию области становятся доступными. Если команда unset была применена к переменной вызывающей области, становится доступным экземпляр, затенённый этой переменной (если он есть).

Имена функций и определения могут быть перечислены с помощью опции -f внутренней команды declare (typeset) (4.2. Внутренние команды bash). Опция -F в команде declare или typeset будет выводить только имена функций (может также выводить имя файла и номер строки, если включена опция extdebug). Функции можно экспортировать, чтобы субоболочки автоматически получали их определения при указании опции -f в команде export (4.1. Внутренние элементы Bourne Shell).

Функции могут быть рекурсивными, для ограничения уровня рекурсии может применяться переменная FUNCNEST. По умолчанию глубина рекурсии не ограничена.

3.4. Параметры оболочки

Параметр - это элемент, сохраняющий значения. Параметр может быть именем, номером или одним из специальных символов, указанных ниже. Переменные являются именованными параметрами. Переменная имеет значение и может иметь атрибуты, присваиваемые с помощью внутренней команды declare (4.2. Внутренние команды bash).

Параметр установлен, если ему присвоено значение (включая null). Установленная переменная может быть сброшена внутренней командой unset и установлена выражением вида

```
name=[value]
```

Если аргумент value не задан, переменной назначается пустая строка (null). Все значения подвергаются преобразованию тильды, параметров и переменных, подстановке команд, арифметическим преобразованиям и удалению кавычек. Если переменная имеет атрибут integer, её значение определяется арифметическим преобразованием (3.5.5. Арифметическое преобразование) даже при отсутствии выражения $((...)). Расщепление слов не применяется, за исключением описанного ниже случая $@, преобразование имён файлов также не выполняется. Операторы назначения могут присутствовать как аргументы внутренних команд alias, declare, typeset, export, readonly и local. В режиме POSIX (6.11. Режим POSIX) эти встроенные переменные могут присутствовать в команде после одного или нескольких экземпляров внутренней команды command и сохранять свойства операторов присваивания.

В контексте, где оператор присваивание задаёт значение переменной оболочки или индексу массива (6.7. Массивы), оператор += может служить для добавления в конец или сложения с прежним значением переменной. Это включает аргументы внутренних команд, таких как declare, которые принимают операторы присваивания. При использовании += с переменной, для которой установлен атрибут integer, значение вычисляется путём арифметического преобразования и складывается с текущим значением, которое тоже преобразуется. Когда += применяется к переменной массива с помощью композитного присваивания (6.7. Массивы), переменная не сбрасывается, как при использовании = и новое значение добавляется в конец массива с индексом на 1 больше текущего максимального индекса (массив с индексом) или добавляется новая пара ключ-значение в ассоциативный массив. Для строковой переменной этот оператор просто добавляет строку в конец имеющейся.

Переменной может быть назначен атрибут nameref с помощью опции -n внутренней команды declare или local (4.2. Внутренние команды bash) для создания nameref или ссылки на иную переменную. Это позволяет манипулировать переменными опосредованно. Всякий раз, когда переменная nameref указывается, назначается, отменяется или изменяются её атрибуты (иные, нежели nameref), реальная операция выполняется над переменной, заданной значением nameref. Обычно nameref применяется в функциях оболочки для ссылки на переменную, имя которой передаётся как аргумент функции. Например, если имя переменной передаётся shell-функции как первый аргумент, выполнение declare -n ref=$1 внутри функции создаёт nameref-переменную ref, значением которой является имя переменной, переданное первым аргументом. Ссылки и назначение переменной ref, а также изменение её атрибутов будут трактоваться как ссылки, назначение и изменение атрибутов переменной, имя которой передано как $1.

Если переменная управления циклом for имеет атрибут nameref, список слов может содержать переменные оболочки и ссылка на имя будет создаваться для каждого слова списка при выполнении цикла. Для переменных-массивов атрибут nameref не поддерживается, однако переменные nameref могут указывать переменные массива и индексы. Nameref можно отменить с помощью опции -n внутренней команды (4.1. Внутренние элементы Bourne Shell). Если же unset выполняется с именем переменной nameref в качестве аргумента, сброшена будет переменная, указанная nameref.

3.4.1. Позиционные параметры

Позиционные параметры обозначаются одной или несколькими цифрами, обозначение 0 не используется. Позиционные параметры присваиваются из значений аргументов оболочки при вызове и могут переназначаться с помощью внутренней команды set. Параметр N можно указывать как ${N} или $N (если N содержит одну цифру). Позиционные параметры не могут назначаться операторами присваивания. Для установки и сброса позиционных

параметров используются внутренние команды set и unset (4. Внутренние команды оболочки). Позиционные параметры временно заменяются при выполнении shell-функций (3.3. Функции оболочки).

Если номер позиционного параметра содержит более одной цифры, параметр должен заключаться в фигурные скобки.

3.4.2. Специальные параметры

Некоторые параметры оболочки имеют особое значение и их можно лишь указывать без возможности назначения.

`* ($*)` Преобразуется в позиционные параметры начиная с первого. Если преобразуемый параметр не заключён в двойные кавычки, каждый позиционный параметр становится отдельным словом. В контексте преобразования эти слова подвергаются расщеплению и преобразованию путей. Если преобразуемый параметр заключён в двойные кавычки, он преобразуется в одно слово, где значения параметров разделены первым символом специальной переменной IFS. Таким образом, "$*" преобразуется в "$1c$2c...", где c - первый символ значения переменной IFS. Если IFS не задана, разделителем служит пробел, а при IFS = null параметры не разделяются.

`@ ($@)` Преобразуется в позиционные параметры начиная с первого. В контексте, где выполняется расщепление слов, каждый позиционный параметр преобразуется в отдельное слово, которое при отсутствии двойных кавычек подвергается расщеплению слов. В контексте без расщепления слов параметр преобразуется в одно слово с разделением позиционных параметров пробелами. Параметр, заключенный в двойные кавычки, преобразуется с расщеплением слов и выделением каждого параметра в отдельное слово. Таким образом, "$@" является эквивалентом "$1" "$2" ... Если раскрытие двойных кавычек выполняется внутри слова преобразованное первое слово объединяется с начальной частью исходного слова, а преобразование последнего - с последней частью исходного слова. При отсутствии позиционных параметров преобразования "$@" и $@ не происходит (т. е. параметр удаляется).

`# ($#)` Преобразуется в число позиционных параметров в десятичное значение.

`? ($?)` Преобразуется в статус выполненного последним не фонового (foreground) конвейера.

`- ($-)` Преобразуется в текущие флаги опций, заданные при вызове внутренней командой set или установленные самой оболочкой (например, опция -i).

`$ ($$)` Преобразуется в идентификатор процесса оболочки. В субоболочке () указывается идентификатор процесса вызвавшей оболочки, а не субоболочки.

`! ($!)` Преобразуется в идентификатор процесса для задания, помещённого последним в фоновый режим и выполняемого как асинхронная команда или с помощью внутренней команды bg (7.2. Внутренние средства управления заданиями).

`0 ($0)` Преобразуется в имя оболочки или сценария оболочки. Значение устанавливается при инициализации оболочки. Если Bash вызывается с файлом команд (3.8. Сценарии оболочки), в $0 задаётся имя этого файла. При запуске Bash с опцией -c (6.1. Вызов Bash), в $0 помещается первый аргумент после выполняемой строки, если он имеется. В остальных случаях параметр получает имя файла, использованного для вызова Bash, указанное аргументом 0.

`_ ($_)` При запуске оболочки для параметра устанавливается абсолютный путь, заданный при вызове оболочки или сценария и переданный в окружении или списке аргументов. Затем параметр преобразуется в последний предыдущей простой команды, выполняемой в режиме foreground, после преобразования. Устанавливается также полный путь, используемый для вызова каждой выполняемой команды, и помещается в окружение экспортируемое этой команде. При проверке почты в этом параметре содержится имя почтового файла.

3.5. Преобразования

Преобразование командной строки выполняется после её разбиения на маркеры (компоненты). Имеется несколько типов преобразований, указанных в порядке их применения:

1. преобразование скобок;
2. преобразование тильды;
3. преобразование параметров и переменных;
4. арифметическое преобразование и подстановка команд (слева направо);
5. расщепление слов;
6. преобразование имён файлов.

На системах, которые это поддерживают, возможна также подстановка процессов. Она выполняется одновременно с преобразованием тильды, параметров, переменных, арифметическими преобразованиями и подстановкой команд.

Преобразование скобок, расщепление слов и имён файлов могут увеличивать число слов, а в остальных преобразованиях число слов сохраняется. Единственным исключением является преобразование "$@" и $* (3.4.2. Специальные параметры), а также "${name[@]}" и ${name[*]} (6.7. Массивы).

По завершении этих преобразований выполняется удаление кавычек (3.5.9. Удаление кавычек), если сами они на находятся в других кавычках.

3.5.1. Раскрытие скобок

Раскрытие скобок может генерировать произвольные строки символов. Этот механизм похож на преобразование имён файлов (3.5.8. Преобразование имён файлов), но создаваемые имена могут не существовать. Преобразуемая структура может включать преамбулу, за которой следует цепочка разделённых запятыми строк или выражение между парой скобок, затем может следовать дополнение (постскриптум). Преамбула добавляется перед каждой строкой, содержащейся внутри скобок, а постскриптум добавляется в конец её. Преобразование выполняется слева направо.

Выражения в скобках могут быть вложенными. Полученные в результате преобразования строки не сортируются. Например,

```
bash$ echo a{d,c,b}e
ade ace abe
```

Последовательность имеет форму {x..y[..incr]}, где x и y - целые числа и одиночные символы, а необязательный инкремент incr - целое число. При указании целых числе выражение преобразуется в числа всего диапазона x - y, включая эти значения. Представленные значения целых чисел могут использовать префикс 0 для выравнивания числа цифр каждого элемента. Если x или y начинается с 0, оболочка пытается генерировать элементы с одинаковым числом цифр, добавляя при необходимости 0. При указании символов выражение преобразуется в последовательность символов, лексикографически входящих в диапазон x - y (включительно) в соответствии с принятой по умолчанию языковой настройкой C (locale). Отметим, что оба значения x и y должны быть одного типа. При указании инкремента он задаёт разницу (интервал) между элементами. По умолчанию используется инкремент 1 или -1 (что подходит).

Раскрытие скобок выполняется до всех других преобразований и поэтому все символы, имеющие особое значение в других преобразованиях, сохраняются. Преобразование является строго текстовым и Bash не пытается синтаксически интерпретировать содержимое выражения или текста между скобками.

Для корректного раскрытия скобок выражение должно содержать открывающую и закрывающую скобку без кавычек, а также хотя бы одну запятую без кавычек или корректное выражение последовательности. Некорректное выражение в скобках остаётся не преобразованным. Символу { или , может предшествовать экранирующий символ \. Для предотвращения конфликтов при преобразовании параметров строка ${ не считается допустимой при раскрытии скобок и блокирует преобразование до закрывающей скобки.

Приведённая ниже конструкция обычно применяется как сокращение в случаях, когда генерируемый общий префикс строки длиннее, чем в приведённом выше примере

```
mkdir /usr/local/src/bash/{old,new,dist,bugs}
```

или

```
chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}
```

3.5.2. Преобразование тильды

Если слово начинается с символа ~ (тильда) без кавычек, все символы до первого символа / без кавычек (или просто все, если / без кавычек нет) считаются символами с тильда-префиксом. Если ни один из символов последовательности с тильдой не заключён в кавычки, символы после тильды трактуются как возможное имя пользователя (login name). Если это имя является пустой строкой, символ тильды заменяется значением переменной оболочки HOME. Если эта переменная не установлена, используется имя домашнего каталога пользователя, запустившего оболочку. В остальных случаях строка с тильда-префиксом заменяется домашним каталогом, связанным с указанным именем пользователя.

Если строка с префиксом имеет значение ~+, оно заменяется значением переменной оболочки PWD, а строка ~- заменяется значением переменной OLDPWD, если она задана.

Если после тильды следует число N (возможно с префиксом + или -), строка заменяется соответствующим элементом из стека каталогов, который будет выводится внутренней командой dirs при вызове с символами, следующими за тильдой (6.8. Стек каталогов). При отсутствии префикса между тильдой и числом предполагается префикс +.

Если имя пользователя (login) не действительно или преобразование тильды привело к отказу, строка остаётся прежней.

В каждом назначении переменной проверяется наличие тильда-префиксов без кавычек сразу за : или первым =. В таких случаях также выполняется преобразование тильды. Это позволяет применять имена файлов с тильдой в переменных PATH, MAILPATH, CDPATH и оболочка будет назначать преобразованные значения.

Ниже приведены трактовки в Bash тильда-префиксов без кавычек.

~	Значение переменной $HOME
~/foo	$HOME/foo
~fred/foo	Каталог foo в домашнем каталоге пользователя fred
~+/foo	$PWD/foo
~-/foo	${OLDPWD-'~-'}/foo
~N	Строка, которая будет выведена командой dirs +N
~+N	Строка, которая будет выведена командой dirs +N
~-N	Строка, которая будет выведена командой dirs -N

Bash также выполняет преобразование тильды в словах, удовлетворяющих условиям назначения переменных (3.4. Параметры оболочки), когда они указываются в качестве аргументов простых команд. Bash не делает этого, за исключением перечисленных выше команд объявления при работе в режиме POSIX.

3.5.3. Преобразование параметров оболочки

Символ $ вводит преобразование параметров, подстановку команд или арифметическое преобразование. Преобразуемое имя параметра или символ могут быть заключены в фигурные скобки, которые не обязательны, но позволяют отделить преобразуемую переменную от следующих непосредственно за ней символов, которые могут быть сочтены частью имени. При использовании скобок закрывающей скобкой служит первый символ }, не экранированный с помощью \, не заключенный в кавычки и не входящий во вложенное арифметическое преобразование, подстановку команд или преобразование параметров.

Базовая форма для преобразования параметров имеет вид ${parameter} и выполняется подстановка значения parameter, который является параметром оболочки (3.4. Параметры оболочки) или ссылкой на массив (6.7. Массивы). Скобки требуются для позиционных параметров с несколькими цифрами, а также параметров, за которыми следуют символы, не являющиеся частью имени.

Если первым символом параметра является !, а параметр не является nameref, это задаёт уровень косвенности (indirection). Bash использует значение, полученное преобразованием остальной части параметра, в качестве нового параметра. Затем этот параметр преобразуется и значение используется в дальнейшем преобразовании вместо преобразования исходного параметра. Для значения выполняется преобразование тильды и параметров, подстановка команд и арифметическое преобразование. Если параметр является nameref, он преобразуется в имя переменной,

указанной параметром, вместо полного косвенного преобразования. Исключением являются преобразования ${!prefix*} и ${!name[@]}, описанные ниже. Для введения косвенности восклицательный знак должен быть указан сразу после левой скобки.

В каждом из описанных ниже случаев слово подвергается преобразованию тильды и параметров, подстановке команд и арифметическому преобразованию.

Когда не выполняется преобразование подстрок с использованием описанных ниже форм (например, ':-'), Bash проверяет параметры, которые не сброшены (unset) или имеют значение null. Пропуск двоеточия ведёт к проверке лишь сброшенных (unset) параметров. Т. е. при наличии двоеточия оператор проверяет наличие параметра и его «непустоту» (not null), а без двоеточия проверяется лишь наличие параметра.

${parameter:−word}
Если parameter не установлен или имеет значение null, подставляется преобразование word. В остальных случаях подставляется значение parameter.

${parameter:=word}
Если parameter не установлен или имеет значение null, ему назначается преобразование word, после чего выполняется подстановка параметра. Такое назначение не применяется для позиционных и специальных параметров.

${parameter:?word}
Если parameter не установлен или имеет значение null, преобразование word (или об отсутствии word) записывается на стандартное устройство вывода ошибок и оболочка, если она не является интерактивной, завершает работу. В остальных случаях подставляется значение parameter.

${parameter:+word}
Если parameter не установлен или имеет значение null, не подставляется ничего. П остальных случаях подставляется преобразование word.

${parameter:offset}
${parameter:offset:length}
Это называется преобразованием подстроки (Substring Expansion). Преобразуется до length символов parameter, начиная с символа, заданного значением offset. Если parameter имеет значение @, для массива применяется индекс @, * или имя ассоциативного массива (результаты будут разными, как описано ниже). Если параметр length не задан, преобразуется подстрока значения parameter, начиная с offset и до конца значения. Параметры length и offset являются арифметическими выражениями (6.5. Арифметика командного процессора).
Если offset преобразуется в значение меньше 0, это значение используется для отсчёта от конца значения parameter. Если length преобразуется в значение меньше 0, это значение считается смещением от конца значения parameter, а не числом символов и преобразование выполняется для символов между offset и result. Отметим, что отрицательное значение offset должно отделяться от двоеточия хотя бы одним пробелом, для предотвращения путаницы с преобразованием :-.
Ниже приведено несколько примеров, демонстрирующих преобразование подстрок для параметров и индексированных массивов.

```
$ string=01234567890abcdefgh
$ echo ${string:7}
7890abcdefgh
$ echo ${string:7:0}
$ echo ${string:7:2}
78
$ echo ${string:7:-2}
7890abcdef
$ echo ${string: -7}
bcdefgh
$ echo ${string: -7:0}
$ echo ${string: -7:2}
bc
$ echo ${string: -7:-2}
bcdef
$ set -- 01234567890abcdefgh
$ echo ${1:7}
7890abcdefgh
$ echo ${1:7:0}
$ echo ${1:7:2}
78
$ echo ${1:7:-2}
7890abcdef
$ echo ${1: -7}
bcdefgh
$ echo ${1: -7:0}
$ echo ${1: -7:2}
bc
$ echo ${1: -7:-2}
bcdef
$ array[0]=01234567890abcdefgh
$ echo ${array[0]:7}
7890abcdefgh
$ echo ${array[0]:7:0}
$ echo ${array[0]:7:2}
78
$ echo ${array[0]:7:-2}
7890abcdef
$ echo ${array[0]: -7}
bcdefgh
$ echo ${array[0]: -7:0}
$ echo ${array[0]: -7:2}
bc
```

```
$ echo ${array[0]: -7:-2}
bcdef
```

Если parameter имеет значение @, результатом будут length позиционных параметров, начиная с offset. Отрицательное значение offset задаёт отсчёт от последнего позиционного параметра (-1). Если length преобразуется в отрицательное значение, это вызывает ошибку. Ниже приведены примеры преобразования подстрок с использованием позиционных параметров.

```
$ set -- 1 2 3 4 5 6 7 8 9 0 a b c d e f g h
$ echo ${@:7}
7 8 9 0 a b c d e f g h
$ echo ${@:7:0}
$ echo ${@:7:2}
7 8
$ echo ${@:7:-2}
bash: -2: substring expression < 0
$ echo ${@: -7:2}
b c
$ echo ${@:0}
./bash 1 2 3 4 5 6 7 8 9 0 a b c d e f g h
$ echo ${@:0:2}
./bash 1
$ echo ${@: -7:0}
```

Если parameter является индексированным массивом с индексом @ или *, результатом будет length элементов массива, начиная с ${parameter[offset]}. Отрицательное значение offset задаёт отсчёт от максимального индекса указанного массива назад. Если length преобразуется в отрицательное значение, это вызывает ошибку. Ниже приведены примеры преобразования подстрок с индексированным массивом.

```
$ array=(0 1 2 3 4 5 6 7 8 9 0 a b c d e f g h)
$ echo ${array[@]:7}
7 8 9 0 a b c d e f g h
$ echo ${array[@]:7:2}
7 8
$ echo ${array[@]: -7:2}
b c
$ echo ${array[@]: -7:-2}
bash: -2: substring expression < 0
$ echo ${array[@]:0}
0 1 2 3 4 5 6 7 8 9 0 a b c d e f g h
$ echo ${array[@]:0:2}
0 1
$ echo ${array[@]: -7:0}
```

Преобразование подстрок в ассоциативных массивах даёт неопределённый результат.

Индексирование подстрок начинается с 0, если не используются позиционные параметры, для которых отсчёт по умолчанию начинается с 1. Если для позиционных параметров задано offset = 0, в начале списка выводится $@.

${!prefix*}
${!prefix@}

Преобразуются в имена переменных, начинающиеся с prefix, разделённые первым символом специальной переменной IFS. При использовании @ и наличии двойных кавычек имя каждой переменной преобразуется в отдельное слово.

${!name[@]}
${!name[*]}

Если name является переменной массива, преобразуется в список индексов (ключей), назначенных в name. Если name не является массивом, преобразуется в 0 при установленной переменной name и в null для остальных случаев. При использовании @ и наличии двойных кавычек каждый ключ преобразуется в отдельное слово.

${#parameter}

Подставляется число символов преобразованного значения parameter. Если parameter имеет значение * или @, подставляется число позиционных параметров. Если parameter является именем массива с индексом * или @, подставляется число элементов массива. Если parameter является именем массива с отрицательным индексом, отсчёт начинается с конца массива (-1 указывает последний элемент).

${parameter#word}
${parameter##word}

Аргумент word преобразуется для создания шаблона и сопоставления по описанным ниже правилам (3.5.8.1. Сопоставление с шаблоном). Если шаблон соответствует началу преобразованного значения parameter, результатом преобразования будет преобразованное значение с удалением самого короткого (вариант #) или самого длинного (вариант ##) совпадения. Если parameter имеет значение * или @, операция удаления шаблона применяется к каждому позиционному параметру по очереди и результатом преобразования является полученный список. Если parameter является именем массива с индексом * или @, операция удаления шаблона применяется к каждому элементу массива поочерёдно и результатом преобразования является полученный список.

${parameter%word}
${parameter%%word}

Аргумент word преобразуется для создания шаблона и сопоставления по описанным ниже правилам (3.5.8.1. Сопоставление с шаблоном). Если шаблон соответствует концу преобразованного значения parameter, результатом преобразования будет преобразованное значение с удалением самого короткого (вариант ;) или самого длинного (вариант %%) совпадения. Если parameter имеет значение * или @, операция удаления шаблона применяется к каждому позиционному параметру по очереди и результатом преобразования является полученный список. Если parameter является именем массива с индексом * или @, операция удаления шаблона применяется к каждому элементу массива поочерёдно и результатом преобразования является полученный список.

${parameter/pattern/string}

Аргумент pattern преобразуется для создания шаблона как при преобразовании имён, parameter преобразуется и наибольшее совпадение с шаблоном заменяется значением string. Сопоставление выполняется по правилам параграфа (3.5.8.1. Сопоставление с шаблоном). Если pattern начинается с /, все совпадения заменяются на string (обычно замена выполняется лишь для первого совпадения). Если pattern начинается с #, ему соответствует

начало преобразованного значения parameter, если pattern начинается с %', ему соответствует конец преобразованного значения parameter. Если string = null, совпадение с шаблоном удаляется и символ / после pattern может не указываться. При включённой опции оболочки nocasematch (4.3.2. Внутренняя команда shopt) сопоставление выполняется без учёта регистра символов. Если parameter имеет значение * или @, подстановка выполняется поочерёдно для каждого позиционного параметра и результатом преобразования является полученный список. Если parameter является именем массива с индексом * или @, подстановка применяется к каждому элементу массива поочерёдно и результатом преобразования является полученный список.

${parameter^pattern}
${parameter^^pattern}
${parameter,pattern}
${parameter,,pattern}

Преобразование меняет регистр символов parameter. Аргумент pattern преобразуется для создания шаблона как в преобразовании имён файлов. Каждый символ преобразованного значения parameter сравнивается с шаблоном и при совпадении регистр символа меняется. Не следует сопоставлять шаблон с несколькими символами сразу. Оператор ^ задаёт преобразование нижнего регистра в верхний, а ',' - обратное преобразование. Преобразования ^^ и ,, конвертируют каждый соответствующий символ в преобразованном значении, ^ и ',' сопоставляют и преобразуют только первый символ преобразованного значения. Если pattern отсутствует, это трактуется как шаблон ?, которому соответствует каждый символ. Если parameter имеет значение * или @, преобразование регистра выполняется поочерёдно для всех позиционных параметров и результатом преобразования является полученный список. Если parameter является именем массива с индексом * или @, преобразование регистра применяется к каждому элементу массива поочерёдно и результатом преобразования является полученный список.

${parameter@operator}

Преобразование является изменением значения parameter или информации о parameter в зависимости от оператора operator, задаваемого одной буквой.

QQ Преобразование является строкой заключённого в кавычки значения parameter в пригодном для ввода формате.
E Преобразование является строкой значения parameter с \-экранированием как в механизме $'...'.
P Строка, полученная в результате преобразования parameter как будто это строка приглашения (6.9. Управление формой приглашения).
A Преобразование строки в форме оператора присваивания или объявления, которое при вычислении воссоздаёт parameter с его атрибутами и значением.
a Строка значений флагов, представляющих атрибуты parameter.

Если parameter имеет значение * или @, операция поочерёдно применяется к каждому позиционному параметру и результатом служит полученный список. Если parameter является именем массива с индексом * или @, операция применяется к каждому элементу массива поочерёдно и результатом является полученный список.
Результат подвергается дальнейшему преобразованию в форме расщепления слов и преобразования путей.

3.5.4. Подстановка команд

Подстановка позволяет заменить команду её выводом и указывается в форме $(command) или 'command'. Bash выполняет преобразование путём выполнения команды command в субоболочке и подстановки её стандартного вывода в основную оболочку с удалением завершающих символов newline. Символы newline внутри вывода не удаляются, но могут быть исключены при расщеплении слов. Подстановку $(cat file) можно заменить более быстрым эквивалентом $(< file).

При использовании старой формы подстановки с обратными кавычками символ \ сохраняет своё буквальное значение, кроме тех случаев, когда за ним следует $, ', или \. Первая обратная кавычка без предшествующего символа \ завершает выражение для подстановки команды. При использовании формы $(command) все символы между скобками являются частью команды без особой трактовки.

Подстановки команд могут быть вложенными. В случае использования формы с обратными кавычками внутренние кавычки экранируются символом \. Если подстановка указана в двойных кавычках, для результата не выполняется расщепление слов и преобразование имён файлов.

3.5.5. Арифметическое преобразование

Арифметическое преобразование позволяет вычислить результат арифметического выражения и подставить вместо него результат. Формат арифметического выражения имеет вид $((expression)). Здесь expression трактуется как при указании в двойных кавычках, но такие кавычки внутри круглых скобок не имеют специального значения. Все маркеры в expression подвергаются преобразованию параметров и переменных, подстановке команд и удалению кавычек. Результат считается арифметическим выражением для расчёта значения. Арифметические выражения могут быть вложенными.

Расчёт выполняется в соответствии с правилами раздела 6.5. Арифметика командного процессора. Если выражение не действительно, Bash печатает сообщение об ошибке на стандартном устройстве вывода и не делает подстановки.

3.5.6. Подстановка процессов

Подстановка процессов позволяет указывать ввод или вывод процесса по имени файла. Подстановка имеет вид <(list) или >(list). Список процессов запускается асинхронного и его вход или выход отображается как имя файла. Это имя передаётся в качестве аргумента текущей команде как результат преобразования. При использовании формы >(list) запись в файл будет служить вводом списка. Если применяется форма <(list), переданный в качестве аргумента файл должен считываться для получения списка. Отметим, что между символом < или > и левой скобкой не может быть пробела, поскольку в этом случае выражение будет задавать перенаправление. Подстановка процессов доступна в системах, поддерживающих именованные каналы (fifo) и метод именования открытых файлов /dev/fd.

При доступности подстановки процессов она выполняется одновременно с преобразованием параметров и переменных, подстановкой команд и арифметическим преобразованием.

3.5.7. Расщепление слов

Оболочка сканирует результат преобразования параметров, подстановки команд и арифметического преобразования, которые выполнялись вне двойных кавычек, на предмет расщепления слов. Оболочка трактует каждый символ переменной $IFS как разделитель слов в результатах предшествующих преобразований. Если переменная IFS не задана или имеет принятое по умолчанию значение <space><tab><newline>, последовательности <space>, <tab>, <newline> в начале и в конце результата игнорируются, а последующие символы IFS считаются разделителями слов. Если значение IFS отличается от принятого по умолчанию, последовательности пробельных символов (<space><tab><newline>), а также пробельные символы IFS в начале и в конце игнорируются. Любой непробельный символ IFS вместе со смежными пробельными символами IFS считается разделителем слов, равно как и последовательность пробельных символов IFS, Если переменная IFS имеет пустое значение (null), слова не разделяются.

Явно пустые аргументы ("" или ’’) сохраняются и передаются командам в виде пустых строк. Неявные null-аргументы без кавычек, полученные в результате преобразования параметров, не имеющих значений, удаляются. Если преобразуемый параметр в двойных кавычках имеет пустое значение, полученный в результате null-аргумент сохраняется и передаётся команде как пустая строка. Когда заключенный в кавычки null-аргумент является частью слова, которое преобразуется в непустое значение, null-аргумент удаляется. Например, слово -d’’ преобразуется в -d.

При отсутствии преобразований расщепления слов не производится.

3.5.8. Преобразование имён файлов

После расщепления слов, если не задана опция -f внутренней команды set (4.3.1. Внутренняя команда set), Bash сканирует каждое слово в поиске символов *, ? и [. При наличии любого из этих символов слово считается шаблоном и заменяется отсортированным по алфавиту списком имён файлов, соответствующих шаблону (3.5.8.1. Сопоставление с шаблоном). Если соответствующих шаблону имён не найдено, а опция оболочки nullglob отключена, слово сохраняется без изменений. При включённой опции nullglob и отсутствии совпадений слово удаляется, выводится сообщение об ошибке и команда не выполняется. Если включена опция оболочки nocaseglob, сопоставление выполняется без учёта регистра символов.

При выполнении преобразования, символ ‘.’ в начале имени или сразу после / должен сопоставляться буквально, если не задана опция оболочки dotglob. Имена ‘.’ и ‘..’ всегда должны сопоставляться буквально, даже если включена опция dotglob. В остальных случаях символ ‘.’ не имеет специального значения.

При сопоставлении имён файлов символ / всегда должен точно соответствовать такому же символу в шаблоне, но в ином контексте сопоставления он может соответствовать иным символам, как описано в параграфе 3.5.8.1. Сопоставление с шаблоном.

Опции nocaseglob, nullglob, failglob и dotglob описаны в параграфе 4.3.2. Внутренняя команда shopt.

Переменную оболочки GLOBIGNORE можно применять для ограничения набора имён, соответствующих шаблону. Если эта переменная установлена, каждое совпадающее имя файла сопоставляется также с одним из шаблонов в GLOBIGNORE и удаляется из списка при совпадении. Если установлена опция nocaseglob, сопоставление с GLOBIGNORE выполняется без учёта регистра. Имена . и .. игнорируются при установке непустой переменной GLOBIGNORE. Однако установка непустого значения GLOBIGNORE будет включать опцию dotglob, поэтому все имена, начинающиеся с ‘.’, будут совпадать с шаблоном. Для поддержки прежнего поведения с игнорированием имён, начинающихся с точки, следует добавить шаблон ‘.*’ в переменную GLOBIGNORE. Опция dotglob отключена, если переменная GLOBIGNORE не установлена.

3.5.8.1. Сопоставление с шаблоном

Все символы шаблона, за исключением описанных ниже специальных символов, должны совпадать при сопоставлении. Символ nul не может включаться в шаблоны, символ \ экранирует следующий символ, а сам при сопоставлении не учитывается. Специальные символы для буквальной трактовки в шаблоне должны заключаться в кавычки.

Специальные символы шаблонов и их трактовка описаны ниже.

- ***** Соответствует любой строке, включая пустую. При включённой опции globstar и использовании символа * в контексте преобразования имён файлов, два смежных символа * в одном шаблоне будут соответствовать всем файлам и имеющимся каталогам и подкаталогам. Если перед парой символов * имеется /, соответствовать шаблону будут лишь каталоги и подкаталоги.
- **?** Соответствует одному (любому) символу.

`[…]` Соответствует одному из указанных в скобках символов. Пара символов, разделённых дефисом указывает диапазон, включающий граничные символы, с использованием порядка и набора символов текущего языка. Если после открывающей скобки [указан символ ! или ^, соответствовать шаблону будет любой символ, не входящий в диапазон. Символ − может указываться в качестве первого или последнего в диапазоне,] - в качестве первого. Порядок сортировки символов в диапазоне определяется текущим значением locale и переменными LC_COLLATE и LC_ALL, если они установлены.

Например, с используемым по умолчанию C диапазон [a-dx-z] будет эквивалентен набору [abcdxyz]. Многие языки (locale) сортируют символы по словарю и для них [a-dx-z] обычно не является эквивалентом [abcdxyz] и может быть, например, эквивалентом [aBbCcDdxXyYz]. Для получения традиционной интерпретации диапазонов в квадратных скобках можно использовать C locale путём установки в переменной LC_COLLATE или LC_ALL значения C или включения опции globasciiranges.

Внутри скобок [] можно указать класс символов в форме [:class:], где class - одно из значений, определённых стандартом POSIX: alnum, alpha, ascii, blank, cntrl, digit, graph, lower, print, punct, space, upper, word, xdigit.

Классу соответствует любой символ указанного класса, например, классу word соответствуют буквы, цифры и _.

Внутри скобок [] можно указать класс эквивалентности с использованием синтаксиса [=c=], которому будут соответствовать все символы с таким весом сравнения (collation) в соответствии с настройкой, как у символа c.

Внутри скобок [] синтаксис [.symbol.] соответствует символу symbol.

Если опция оболочки extglob включена с использованием внутренней команды shopt, применяется несколько расширенных сопоставлений с шаблоном. В последующих описаниях pattern-list - это список, включающий по меньшей мере один шаблон с разделением шаблонов символом |. Могут создаваться композитные шаблоны с использованием одного или нескольких субшаблонов, описанных ниже.

`?(pattern-list)`	Соответствует не более, чем одному вхождению указанных шаблонов.
`*(pattern-list)`	Соответствует любому числу вхождений указанных шаблонов.
`+(pattern-list)`	Соответствует не менее, чем одному вхождению указанных шаблонов.
`@(pattern-list)`	Соответствует одному вхождению указанных шаблонов.
`!(pattern-list)`	Соответствует всему, за исключением любого из указанных шаблонов.

Сопоставление расширенных шаблонов с длинными строками является медленным, особенно при наличии в шаблонах чередований a в строках - многочисленных совпадений. Использование отдельных сопоставлений для более коротких строк или применение массива строк вместо одной длинной строки может ускорить процесс.

3.5.9. Удаление кавычек

После выполнения описанных выше преобразований удаляются все не заключённые в кавычки символы \, ’ и ", не являющиеся результатом предшествующих преобразований.

3.6. Перенаправление

Перед выполнением команды её ввод и вывод могут быть перенаправлены с использованием специальной нотации командного процессора. Перенаправление позволяет дублировать, открывать, закрывать, ссылаться на другие файлы и менять файлы, которые команда читает или записывает. Описанные ниже перенаправления могут предшествовать команде, а также указываться внутри или после команды. Перенаправления обрабатываются в порядке их указания, слева направо.

Для каждого перенаправления, которому может номер дескриптора файла, вместо такого номера может быть указано слово вида {varname}. В этом случае для каждого оператора перенаправления, кроме >&- и <&-, командный процессор может назначить дескриптор файла больше 10 и присвоить его {varname }. Если слову {varname } предшествует >&- или <&-, значение varname определяет дескриптор файла для закрытия. Если указано слово {varname}, перенаправление остаётся за рамками команды, что позволяет программировать управление дескриптором.

В последующих описаниях при опущенном номере дескриптора файла первый символ перенаправления < задаёт стандартный ввод (дескриптор 0), а первый символ > - стандартный вывод (дескриптор 1). Для слова за оператором перенаправления в последующих описаниях (если явно не указано иное) выполняется преобразование фигурных скобок, тильды и параметров, подстановка команд, арифметические преобразования, удаление кавычек, преобразование имени файла и расщепление слов. Если в результате получается более одного слова, bash возвращает ошибку.

Порядок перенаправления имеет значение. Например, команда **`ls > dirlist 2>&1`** направляет стандартный вывод (дескриптор 1) и стандартный вывод ошибок (дескриптор 2) в dirlist, а команда **`ls 2>&1 > dirlist`** направляет в файл dirlist лишь стандартный вывод, поскольку стандартный вывод ошибок выполняет копирование стандартного вывода до его перенаправления в dirlist.

Bash обрабатывает некоторые имена файлов при перенаправлениях в соответствии с приведённым ниже описанием. Если операционная система, где работает bash, использует такие файлы, bash будет применять их. В остальных случаях будет выполняться внутренняя эмуляция, описанная ниже.

/dev/fd/fd
Если fd является пригодным целым числом, файловый дескриптор fd дублируется.

/dev/stdin
Файловый дескриптор 0 дублируется.

/dev/stdout
Файловый дескриптор 1 дублируется.

/dev/stderr
Файловый дескриптор 2 дублируется.

/dev/tcp/host/port
Если host является действительным именем хоста или адресом Internet, a port - целочисленным номером порта или именем службы bash пытается открыть соответствующий сокет TCP.

/dev/udp/host/port

Если host является действительным именем хоста или адресом Internet, а port - целочисленным номером порта или именем службы bash пытается открыть соответствующий сокет UDP.

Отказ при открытии или создании файла ведёт к отказу перенаправления.

Перенаправление с файловыми дескрипторами больше 9 следует применять с осторожностью, поскольку это может вызывать конфликты с дескрипторами, используемыми внутри командного процессора.

3.6.1. Перенаправление ввода

Перенаправление ввода ведёт к тому, что открывается для чтения файл, имя которого получено из слова команды, для дескриптора n или стандартного ввода (дескриптор 0), если n отсутствует. Базовый формат перенаправления ввода имеет вид

```
[n]<word
```

3.6.2. Перенаправление вывода

Перенаправление ввода ведёт к тому, что открывается для записи файл, имя которого получено из слова команды, для дескриптора n или стандартного вывода (дескриптор 1), если n отсутствует. Файл при открытии усекается до нулевого размера. Базовый формат перенаправления вывода имеет вид

```
[n]>[|]word
```

Если используется оператор перенаправления > и включена опция noclobber для внутренней команды set, перенаправление будет приводить к отказу, если полученное после преобразования имя указывает реально существующий обычный файл. Если указан оператор перенаправления >| или оператор > с выключенной опцией noclobber, будет выполнена попытка перенаправления даже при наличии указанного словом файла.

3.6.3. Добавление перенаправленного вывода

Перенаправление вывода таким способом ведёт к открытию файла, имя которого получено из преобразования word, в режиме добавления в конец для файлового дескриптора n или стандартного вывода (дескриптор 1), если n отсутствует. Если нужного файла нет, он создаётся. Базовый формат перенаправления в конец файла имеет вид

```
[n]>>word
```

3.6.4. Перенаправление стандартного вывода и стандартного вывода ошибок

Эта конструкция позволяет перенаправить стандартный вывод (дескриптор 1) и стандартный вывод ошибок (дескриптор 2) в файл, имя которого определяется преобразованием параметра word. Применяются 2 варианта - &>word и >&word. Первая форма семантически эквивалентна >word 2>&1. При использовании второй формы параметр word может быть преобразован не в число или -. В таких случаях применяются другие операторы перенаправления (3.6.8. Дублирование файловых дескрипторов) для обеспечения совместимости.

3.6.5. Добавление стандартного вывода и стандартного вывода ошибок

Эта конструкция позволяет перенаправить стандартный вывод (дескриптор 1) и стандартный вывод ошибок (дескриптор 2) в конец имеющегося файла, имя которого указывается преобразованием word. Формат добавления вывода в файл имеет вид &>>word, что семантически эквивалентно выражению >>word 2>&1 (см. 3.6.8. Дублирование файловых дескрипторов).

3.6.6. Перенаправление из документа

Этот тип перенаправления задаёт оболочке чтение файла до строки, содержащей лишь word (без пробелов в конце). Все прочитанные из файла строки затем служат стандартным вводом (или файлом с дескриптором n, если он задан) для команды.

Формат here-document имеет вид

```
[n]<<[-]word
        here-document
delimiter
```

Для word не выполняется преобразования параметров и переменных, подстановки команд, арифметического преобразования и преобразования имён файлов. Если какая-либо часть word заключена в кавычки, разделитель delimiter является результатом удаления кавычек из word, а строки here-document не раскрываются. Если word не содержит кавычек, все строки here-document подвергаются преобразованию параметров, подстановке команд и арифметическому преобразованию. Последовательность \newline игнорируется, а символ \ должен применяться для экранирования \, $ и ‘.

Если задан оператор перенаправления <<-, все предшествующие символы табуляции игнорируются во входных строках и строке, содержащей delimiter. Это позволяет работать с shell-сценариями, использующими табуляцию.

3.6.7. Перенаправление из строки

Вариант here-document, имеющий формат [n]<<< word. Параметр word подвергается преобразованию тильды, параметров и переменных, подстановке команд, арифметическому преобразованию и удалению кавычек. Преобразование путей и расщепление слов не применяются. Результат представляется как одна строка с символом newline в конце на стандартный ввод (или файл с дескриптором n, если он задан.

3.6.8. Дублирование файловых дескрипторов

Оператор перенаправления [n]<&word служит для дублирования дескрипторов входных файлов. Если word преобразуется в одну или несколько цифр, дескриптор файла, обозначенный n, становится соответствующим преобразованному числу. Если word преобразуется в -, файл с дескриптором n закрывается. Если n не задано, используется стандартный ввод (дескриптор 0).

Оператор [n]>&word используется аналогичным способом для дублирования дескрипторов выходных файлов. Если n не задано, используется стандартный вывод (дескриптор 1). Если цифры word не задают дескриптора файла, открытого для вывода, возникает ошибка перенаправления. Если word преобразуется в -, файл с дескриптором n закрывается. Особым случаем является отсутствие n и преобразование word, отличное от цифр и -. В этом случае выполняется перенаправление стандартного вывода и стандартного вывода ошибок, описанное выше.

3.6.9. Перемещение файловых дескрипторов

Оператор перенаправления [n]<&digit- перемещает файловый дескриптор digit в файловый дескриптор n или стандартный ввод (дескриптор 0), если параметр n не задан. Файл с дескриптором digit закрывается после дублирования.

Оператор [n]>&digit- перемещает файловый дескриптор digit в файловый дескриптор n или стандартный вывод (дескриптор 1), если параметр n не задан.

3.6.10. Дескрипторы открытия файлов для чтения и записи

Оператор перенаправления [n]<>word вызывает открытие файла, имя которого задано преобразованием word, для чтения и записи с файловым дескриптором n (или 0, n не задано). Если такого файла нет, он создаётся.

3.7. Выполнение команд

3.7.1. Преобразование простых команд

При вводе простой команды оболочка выполняет перечисленные ниже преобразования, назначения и перенаправления.

1. Слова, которые анализатор отметил как назначения переменных (до команды), и перенаправления сохраняются для последующей обработки.
2. Слова, не относящиеся к назначению переменных и перенаправлению преобразуются (3.5. Преобразования). Если после преобразования остаются какие-либо слова, первое из них считается именем команды, а остальные - её аргументами.
3. Выполняется перенаправление (3.6. Перенаправление).
4. Текст после символа = в каждом назначении переменной подвергается преобразованию тильды и параметров, подстановке команд, арифметическому преобразованию и удалению кавычек до назначения его переменной.

Если результат не содержит имени команды, назначения применяются к переменным текущей оболочки. В остальных случаях переменные назначаются в среде выполняемой команды и не влияют на окружение текущей оболочки. При попытке задать значение переменной, доступной лишь для чтения, возникает ошибка и команда возвращает отличных от 0 статус.

Если результат не содержит имени команды, перенаправления выполняются, но не оказывают влияния на текущую среду оболочки. Ошибка перенаправления вызывает завершение с отличным от 0 статусом возврата.

Если после преобразований осталось имя команды, выполнение продолжается, как описано ниже. В противном случае команда завершается. Если какое-либо преобразование включает подстановку команд, статусом возврата команды будет статус выхода последней подстановки команды. Если подстановки команд не было, возвращается статус 0.

3.7.2. Поиск и выполнение команд

Если результатом расщепления команды на слова является простая команда с необязательным списком аргументов, выполняются перечисленные ниже действия.

1. Если имя команды не содержит символов /, оболочка пытается найти нужную команду. При наличии функции оболочки с нужным именем, эта функция вызывается, как описано в разделе 3.3. Функции оболочки.
2. Если имя не соответствует функции, оболочка просматривает список внутренних команд и при нахождении нужной команды выполняет её.
3. Если имени нет в числе функций и внутренних команд и оно не включает символов /, Bash выполняет поиск исполняемого файла в каталогах переменной $PATH. Bash использует хэш-таблицу для запоминания полных путей к исполняемым файлам для снижения числа обращений к переменной PATH (см. описание hash в параграфе 4.1. Внутренние элементы Bourne Shell). Полный поиск в переменной $PATH выполняется лишь при отсутствии команды в хэш-таблице. Если поиск не дал результата, оболочка пытается найти функцию с именем command_not_found_handle. При наличии такой функции она вызывается в отдельной среде выполнения с использованием исходной команды и её параметров в качестве аргументов, а статус выхода этой функции служит статусом завершения субоболочки. Если функция не найдена, оболочка выводит сообщение об ошибке и возвращает статус 127.
4. Если поиск дал результат или имя команды содержит один или несколько символов /, оболочка выполняет указанную именем команду в отдельной среде. Аргументу 0 присваивается имя команды, а остальные аргументы служат аргументами выполняемой команды.
5. Если при выполнении команды возникает отказ по причине того, что файл не является исполняемым и этот файл не является каталогом, предполагается, что это сценарий оболочки и выполняется процедура, описанная в параграфе 3.8. Сценарии оболочки.
6. Если команда не была запущена асинхронно, оболочка ждёт завершения команды и определяет его статус.

3.7.3. Среда выполнения команды

Среда выполнения оболочки включает ряд элементов, перечисленных ниже.

- Открытые файлы, наследуемые оболочкой при вызове и изменённые путём перенаправления, предоставляемых внутренней функции exec.
- Текущий рабочий каталог, установленный командой cd, pushd, popd или унаследованный при вызове.
- Режим создания файлов, установленный umask или унаследованный от родителя оболочки.
- Текущие прерывания (trap) установленные функцией trap.
- Параметры оболочки, заданные путём назначения переменных, с помощью команды set или унаследованные от родителя оболочки.
- Функции оболочки, определённые в процессе работы или унаследованные от родителя.
- Опции, включённые при вызове (принятые по умолчанию или заданные в команде) или установленные с помощью set.
- Опции, включённые shopt (4.3.2. Внутренняя команда shopt).
- Псевдонимы, определённые с помощью alias (6.6. Псевдонимы).
- Идентификаторы процессов, включая идентификаторы фоновых заданий (3.2.3. Списки), значения $$ и $PPID.

При выполнении простой команды, не являющейся внутренней, или shell-функции она вызывается в отдельной среде исполнения, включающей указанные ниже элементы (если не указано иное, значения наследуются из оболочки).

- Открытые файлы оболочки с учётом изменений и дополнений, заданных перенаправлениями команды.
- Текущий рабочий каталог.
- Маска режима создания файлов.
- Переменные и функции оболочки, указанные для экспорта, вместе с переменными экспортированными для команды, которые были переданы в окружении (3.7.4. Окружение)
- Прерывания, захваченные оболочкой, сбрасываются к значениям, унаследованным от родителя оболочки, а игнорируемые оболочкой прерывания не учитываются.

Команда, вызываемая в этой отдельной среде, не может оказывать влияния на среду оболочки.

Подстановки команд, команды, объединённые круглыми скобками, и асинхронные команды вызываются в среде субоболочки, которая дублирует окружение основной оболочки за исключением того, что установленные оболочкой прерывания сбрасываются к значениям, унаследованным оболочкой при вызове от родителя. Изменения в среде субоболочки не могут влиять на окружение оболочки.

Субоболочки, порождённые для выполнения подстановки команд, наследуют значение опции -e от родительской оболочки. При работе не в режиме POSIX оболочка Bash сбрасывает опцию -e в таких субоболочках.

Если за командой следует символ &, а управление заданиями не активизировано, принятым по умолчанию стандартным вводом для команды служит /dev/null. В остальных случаях вызываемая команда наследует дескрипторы файлов от вызывающей оболочки с учётом перенаправлений.

3.7.4. Окружение

При вызове программы она получает массив строк, называемый окружением или средой. Этот список включает пары «имя-значение» в формате name=value.

Bash обеспечивает несколько способов управления средой. При вызове оболочка сканирует своё окружение и создаёт параметр для каждого найденного имени, автоматически помечая его для экспорта в дочерние процессы. Выполняемые команды наследуют окружение. Команды export и declare -x позволяют добавлять параметры и функции в среду или удалять их. При изменении параметра в окружении новое значение становится частью среды взамен прежнего. Среда, наследуемая каждой выполняемой командой, состоит из начального окружения оболочки, значения которого можно изменять, за исключением пар, удаляемых командой unset или export -n, а также дополнений, вносимых командами export и declare -x.

Среду для простой команды или функции можно временно дополнить с помощью префиксов с назначением параметров, как описано в параграфе 3.4. Параметры оболочки. Эти назначения влияют лишь на среду, видимую данной командой. Если установлена опция -k (4.3.1. Внутренняя команда set), все назначения параметров помещаются в среду для команды в дополнение к предшествующим имени команды.

Когда Bash вызывает внешнюю команду, в переменной $_ устанавливается полный путь к этой команде и он передаётся в среду данной команды.

3.7.5. Статус выхода

Статусом завершения выполняемой команды является число, возвращаемое системным вызовом waitpid или эквивалентной функцией. Статус может принимать значения от 0 до 255, а значения более 125 могут иметь особый смысл. Статус вывода внутренних функций и составных команд также ограничен значением 125. Специальные значения используются оболочкой при некоторых обстоятельствах для указания особых вариантов отказа.

В оболочке возврат командой статуса 0 означает успешное выполнение, а отличное от 0 значение говорит об отказе. Эта, на первый взгляд нелогичная, схема обеспечивает однозначную индикацию успеха и различные способы указания отказов. При завершении команды со статусом N оболочка Bash возвращает в качестве статуса значение 128+N.

Если команда не найдена, дочерний процесс, созданный для её выполнения, возвращает статус 127. Если найденная команда не является исполняемой, статус возврата будет иметь значение 126. При отказе команды в результате ошибки при преобразовании или перенаправлении возвращается статус больше 0.

Статус выхода используется условными командами Bash (3.2.4.2. Конструкции с условием) и некоторыми списками (3.2.3. Списки).

Все внутренние команды Bash возвращают статус выхода 0 при успешном выполнении и отличных от 0 статус при отказе, что может применяться в условных выражениях и списках. Все внутренние команды возвращают статус 2 для индикации некорректного использования, что обычно связано с непригодными опциями или отсутствием аргументов.

3.7.6. Сигналы
При работе Bash в интерактивном режиме в отсутствие каких-либо прерываний оболочка игнорирует сигнал SIGTERM (поэтому kill 0 не прерывает активную оболочку), а SIGINT перехватывается и обрабатывается (с прерыванием функции ожидания). Когда Bash получает SIGINT, выполнение всех имеющихся циклов прерывается. Сигнал SIGQUIT игнорируется во всех случаях. При включённом управлении заданиями (7. Управление заданиями) Bash игнорирует SIGTTIN, SIGTTOU и SIGTSTP.

Для внешних команд, запускаемых Bash, значения обработчиков сигналов наследуются оболочкой от родителя. Когда управления заданиями нет, асинхронные команды игнорируют сигналы SIGINT и SIGQUIT в дополнение к унаследованным обработчикам. Команды, выполняемые в результате подстановки, игнорируют полученные от клавиатуры сигналы управления заданиями SIGTTIN, SIGTTOU, SIGTSTP.

По умолчанию оболочка завершает работу по сигналу SIGHUP. Перед выходом интерактивная оболочка повторяет SIGHUP для всех заданий (работающих и остановленных). Остановленные задания передают SIGCONT, подтверждающий получение SIGHUP. Чтобы оболочка не передавала SIGHUP конкретному заданию, следует удалить его из таблицы заданий с помощью внутренней команды disown (7.2. Внутренние средства управления заданиями) или пометить его как не получающее сигнал SIGHUP с помощью disown -h.

Если была установлена опция huponexit с помощью внутренней команды shopt (4.3.2. Внутренняя команда shopt), Bash передаёт SIGHUP всем заданиям при завершении интерактивной оболочки (login shell).

Если Bash ожидает завершения команды и получает сигнал, для которого установлена ловушка, прерывание не будет выполняться до завершения команды. Когда Bash ждёт завершения асинхронной команды с помощью внутренней команды wait, получение сигнала, для которого установлена ловушка, будет приводить к незамедлительному возврату из внутренней команды wait со статусом 128, после чего сразу выполняется прерывание.

3.8. Сценарии оболочки
Сценарий оболочки (shell script) представляет собой текстовый файл с командами оболочки. При использовании такого файла в качестве первого аргумента, не являющегося опцией в вызове Bash и отсутствии опций -c и -s (6.1. Вызов Bash) Bash считывает и выполняет команды из этого файла, затем возвращает управление. Такой режим работы называется неинтерактивным. Файл ищется сначала в текущем каталоге, затем в каталогах $PATH.

При выполнении сценария Bash устанавливает в специальном параметре 0 имя файла вместо имени оболочки, а позиционными параметрами становятся оставшиеся аргументы. Если их нет, позиционные параметры не устанавливаются.

Сценарий можно сделать исполняемым с помощью команды chmod. Когда Bash такой файл в $PATH, порождается субоболочка для его выполнения. Т. е. команда filename arguments эквивалентна bash filename arguments, если filename является исполняемым файлом сценария. Эта субоболочка инициализирует себя, как будто для сценария вызвана новая оболочка, за исключением того, что расположение команд, запомненных родителем (см. описание hash в параграфе 4.1. Внутренние элементы Bourne Shell), сохраняется потомком.

Большинство версий Unix делают это частью механизма выполнения команд. Если первая строка сценария начинается символами #!, остальная часть этой строки задаёт интерпретатор команд для сценария. Это позволяет выбрать интерпретатор Bash, awk, Perl и т. п., а оставшуюся часть сценария написать на соответствующем языке.

Аргументы интерпретатора состоят из одного необязательного аргумента в первой строке файла сценария, следующего за именем интерпретатора, за которым может следовать имя файла сценария и остальные аргументы. Bash выполнит нужные действия в операционной систем, которая сама не справляется с этим. Отметим, что некоторые старые версии Unix ограничивают размер строки с именем интерпретатора и аргументами 32 символами.

Сценарии Bash часто начинаются строкой #! /bin/bash (в предположении установки Bash в каталоге /bin), поскольку это обеспечивает выполнение команд интерпретатором Bash даже при запуске сценария из иной оболочки.

4. Внутренние команды оболочки
Встроенные команды реализованы в самой оболочке. При использовании внутренней команды в качестве первого слова простой команды (3.2.1. Простые команды) оболочка напрямую выполняет команду без вызова других программ. Внутренние команды нужны для реализации функций, которые невозможно или неудобно выполнять внешними утилитами. В этой главе кратко описаны встроенные функции Bash, унаследованные от Bourne Shell, а также внутренние команды, присущие Bash или расширенные в этой оболочке.

Некоторые внутренние команды описаны в других параграфах - команды интерфейса Bash для управления заданиями (7.2. Внутренние средства управления заданиями), стек каталогов (6.8.1. Внутренние команды стека каталогов), команды истории (9.2. Внутренние команды истории Bash) и программируемые инструменты завершения команд (8.7. Внутренние функции программируемого дополнения). Многие из этих команд были расширены в POSIX или Bash.

Если явно не указано иное, каждая из внутренних команд, описанная как воспринимающая опции с префиксом -, принимает -- для обозначения конца опций. Внутренние элементы :, true, false и test/[не принимают опций и не применяют специальной трактовки --. Внутренние функции exit, logout, return, break, continue, let и shift воспринимают и обрабатывают аргументы с префиксом -, не требуя --. Остальные встроенные функции, которые принимают аргументы, но не указаны как принимающие опции, трактуют аргументы с префиксом - как ошибки и требуют -- для предотвращения такой интерпретации.

4.1. Внутренние элементы Bourne Shell

Приведённые ниже внутренние операторы и команды унаследованы от Bourne Shell и реализованы в соответствии со стандартом POSIX.

: (двоеточие)

```
: [arguments]
```

Не делает ничего, кроме преобразования аргументов и выполнения перенаправлений. Статус возврата 0.

. (точка)

```
. filename [arguments]
```

Считывает и выполняет команды из файла filename в контексте текущей оболочки. Если filename не начинается с символа /, для поиска файла применяется переменная PATH. Когда bash работает не в режиме POSIX, просматривается также текущий каталог, если файл не найден в $PATH. Если представлены какие-либо аргументы, они становятся позиционными параметрами при выполнении filename. В остальных случаях позиционные параметры остаются неизменными. При включённой опции -T source наследует все ловушки (прерывания) DEBUG, а если это не так, любая строка DEBUG trap сохраняется перед вызовом source и восстанавливается после него, а source отменяет DEBUG trap при выполнении. Если опция -T не задана и выполняемый файл меняет DEBUG, по завершении source сохраняется новое значение. Статусом возврата является статус выхода последней выполненной команды или 0, если команды не выполнялись. Если файл filename не найден или его не удалось причитать, статус возврата будет отличен от 0. Этот оператор эквивалентен команде source.

break

```
break [n]
```

Выход из цикла for, while, unit или select loop. При наличии параметра n выполняется выход из n-го вложенного цикла. Значение n должно быть не меньше 1. Возвращает статус 0, пока n не меньше 1.

cd

```
cd [-L|[-P [-e]] [-@] [directory]
```

Меняет текущий рабочий каталог на directory. Если параметр directory не указан, применяется значение переменной HOME. Все аргументы после directory игнорируются. При наличии переменной оболочки CDPATH она используется как путь поиска - каждое имя в CDPATH просматривается на предмет directory. Имена каталогов в CDPATH разделяются символом :. Если параметр directory начинается с символа /, CDPATH не используется.

Опция -P отключает следование по символьным ссылкам. Символьные ссылки преобразуются, когда cd проходит directory и до обработки экземпляра .. в directory. По умолчанию или при задании опции -L символьные ссылки в directory преобразуются после того, как cd обработает экземпляр .. в directory. Если .. присутствует в directory, это значение обрабатывается путём удаления непосредственно предшествующего элемента пути в направлении / или начала directory.

При наличии опций -e и -P и невозможности определения текущего каталога после смены cd будет возвращать статус отказа.

В системах, поддерживающих опцию -@, она задаёт расширенные атрибуты, связанные с файлом в качестве directory.

Если directory имеет значение -, оно преобразуется в $OLDPWD до попытки смены каталога.

При использовании непустого имени каталога из CDPATH или первом аргументе - и успешной смене каталога абсолютное имя нового рабочего каталога передаётся на стандартный вывод.

При смене каталога возвращается статус 0, при неудаче - отличное от нуля значение.

continue

```
continue [n]
```

Возобновляет следующую итерацию цикла for, while, unit или select. Если указан параметр n, возобновляется n-й цикл. Значение n должно быть не меньше 1. Возвращает статус 0, если не было задано значение n меньше 1.

eval

```
eval [arguments]
```

Аргументы (arguments) объединяются в одну команду, которая считывается и выполняется, Статус завершения команды будет статусом выхода eval. При отсутствии arguments или пустых аргументах возвращается статус 0.

exec

```
exec [-cl] [-a name] [command [arguments]]
```

При наличии параметра command команда заменяет оболочку без создания нового процесса. Если задана опция -l, оболочка помещает тире в начале нулевого аргумента, передаваемого команде (это то, что делает программа login). Опция -c задаёт выполнение команды с пустым окружением. При наличии опции -a оболочка передаёт name в качестве нулевого аргумента для command. Если команда не может быть выполнена по какой-либо причине, неинтерактивная оболочка завершается, пока не включена опция execfail и возвращается статус отказа. Интерактивная оболочка возвращает отказ при невозможности выполнить файл (команду). Субоболочка завершается безусловно при отказе exec. Если команда не задана, могут использоваться перенаправления для воздействия на среду текущей оболочки. Если перенаправления не приводят к ошибке, возвращается статус 0, в случае ошибки статус будет ненулевым.

exit

```
exit [n]
```

Выход из оболочки с возвратом родителю статуса n. Если параметр n не задан, возвращается статус завершения последней команды. Все прерывания EXIT выполняются до завершения оболочки.

export

```
export [-fn] [-p] [name[=value]]
```

Помечает каждое имя name для передачи дочерним процессам в среде. Наличие опции -f указывает, что имена относятся к функциям оболочки, без этой опции они указывают переменные оболочки. Опция -n отменяет пометки для экспорта name. Если имена не указаны или задана опция -p, выводится список всех экспортируемых переменных в форме, пригодной для ввода. Если после имени переменной следует значение (=value), это значение устанавливается для переменной.

Возвращается статус 0, если не было задано непригодных опций или имён переменных (name), а также опция -f не была указана с недействительным именем функции оболочки.

getopts

```
getopts optstring name [args]
```

Команда getopts используется сценариями оболочки для разбора позиционных параметров. Аргумент optstring содержит символы опции для распознавания. Если за символом следует двоеточие, предполагается наличие у опции аргумента, который должен быть отделен пробелом. Двоеточие и знак вопроса не могут использоваться как символы опций. При каждом вызове getopts помещает следующую опцию в переменную оболочки name, инициализируя переменную при её отсутствии, а индекс следующего обрабатываемого элемента - в переменную OPTIND. Переменная OPTIND инициализируется значением 1 при каждом вызове оболочки или shell-сценария. Когда опции нужен аргумент, getopts помещает его в переменную OPTARG. Оболочка не сбрасывает OPTIND автоматически и переменную нужно сбрасывать вручную между вызовами getopts в рамках одного вызова оболочки, если будет применяться новый набор параметров.

При достижении конца опций getopts завершает работу с кодом возврата, отличным от 0. В OPTIND указывается индекс первого аргумента, не являющегося опцией, а для name устанавливается значение '?'.

Команда getopts обычно анализирует позиционные параметры, но при наличии дополнительных аргументов в args разбираются эти аргументы.

Ошибки getopts могут возвращаться двумя способами. Если первым символом optstring является двоеточие, используется «тихое» уведомление об ошибках, а при обычной работе выводится диагностическое сообщение с указанием непригодных или пропущенных опций. Если OPTERR = 0, сообщения об ошибках не выводятся даже при отсутствии двоеточия в первом символе optstring.

При обнаружении недействительной опции getopts помещает ? в name и в обычном (не «тихом») режиме выводит сообщение об ошибке и сбрасывает OPTARG. В «тихом» режиме символ опции помещается в OPTARG без вывода диагностического сообщения.

Если нужный аргумент не найден и для getopts не задан «тихий» режим, в name помещается ?, OPTARG сбрасывается и выводится диагностическое сообщение. Если getopts работает в «тихом» режиме, в name помещается :, а в OPTARG - найденный символ опции.

hash

```
hash [-r] [-p filename] [-dt] [name]
```

При каждом вызове hash запоминаются полные пути к командам, указанным в качестве аргументов, поэтому их уже не нужно искать при последующих вызовах. Поиск команд выполняется путём просмотра каталогов, указанных в переменной $PATH. Все ранее запомненные пути отбрасываются. Опция -p отключает поиск пути a filename служит для указания местоположения name. Опция -r заставляет оболочку забыть все запомненные местоположения, опция -d заставляет забыть запомненное местоположение каждого значения name. При наличии опции -t выводится полный путь, которому соответствует каждое значение name. При указании множества аргументов name с опцией -t, значения name выводятся перед хэшированными путями. Опция -l обеспечивает вывод в формате, пригодном для ввода. Если опция -l задана без каких-либо других аргументов команды, выводится информация о запомненных командах.

Статус возврата имеет значение 0, если команда не содержит недействительной опции и name найдено.

pwd

```
pwd [-LP]
```

Выводит абсолютный (полный) путь к текущему каталогу. Опция -P исключает вывод символьных ссылок в пути, опция -L включает. Статусом возврата будет 0, если не возникло ошибок при определении пути к текущему каталогу или не была задана недействительная опция.

readonly

```
readonly [-aAf] [-p] [name[=value]] ...
```

Помечает каждую переменную name как доступную лишь для чтения. Значения этой переменной невозможно будет изменить путём назначения. При наличии опции -f параметр name относится к функциям оболочки. Опция -a связывает name с переменными индексированных массивов, -A - с переменными ассоциативных массивов. При наличии обеих опций будет применяться -A. Если аргументов name не задано или указана опция -p, выводится список переменных, доступных только для чтения, в формате пригодном для ввода. Могут применяться другие опции для ограничения выводимого множества имён readonly. Если имя задано в форме name=value, для переменной устанавливается значение value. Команда возвращает статус 0, если не указано недействительных опций, параметр name не задаёт недействительную переменную или функцию оболочки и опция -f не указана с параметром name, не соответствующим функции оболочки.

return

```
return [n]
```

Вызывает остановку функции оболочки с возвратом статуса n. Если параметр n не указан, статусом возврата будет статус выхода последней команды, выполненной в функции. Если команда return вызывается обработчиком прерываний (trap), для определения статуса используется последняя команда, вызванная перед обработчиком прерывания. При вызове return в процессе обработки прерывания DEBUG статус возврата определяется последней командой, выполненной обработчиком прерывания перед вызовом return. Команду return можно применять для прерывания сценариев, выполняемых с помощью внутренней команды . (source), возвращая n или статус выхода последней выполненной команды сценария. При указанном значении n возвращаются 8 младших битов этого значения. Любая команда, связанная с прерыванием RETURN, выполняется до возобновления исполнения после функции или сценария. Статус возврата отличается от 0, если команда return вызвана с аргументом, не являющимся числом, использована за пределами функции или не в процессе выполнения сценария с помощью внутренней команды . или source.

shift

```
shift [n]
```

Сдвигает позиционные параметры влево на n позиций, в результате чего параметры $n+1 - $# становятся $1 - $#-n, а параметры $# - $#-n+1 сбрасываются. Значение n должно быть неотрицательным целым числом или $#. Если n=0 или больше $#, позиционные параметры не меняются. По умолчанию предполагается n=1. Возвращается статус 0, если n не превышает $# и не является отрицательным.

test

[

```
test expr
```

Вычисляется условное выражение expr и возвращается статус 0 (true) или 1 (false). Каждый оператор и операнд должен быть отдельным аргументом. Выражения состоят из примитивов, описанных в разделе 6.4. Условные выражения bash. Команда test не принимает опций, а также не воспринимает (игнорирует) аргумент --, указывающий завершение опций.

При использовании формы [последним аргументом команды должна быть закрывающая скобка].

Выражения могут комбинироваться с использованием перечисленных ниже операторов (в порядке снижения предпочтения). Результат зависит от числа аргументов, как описано ниже. При 5 и более аргументах используется порядок применения операторов.

! expr True, если expr даёт false.

(expr) Возвращает значение expr и может использоваться для переопределения порядка применения операторов.

expr1 -a expr2 True, если оба выражения expr1 и expr2 дают true.

expr1 -o expr2 True, если любое их выражений expr1 и expr2 даёт true.

Внутренние функции test и [оценивают условные выражения на основе правил, учитывающих число аргументов.

0 Выражение даёт false.

1 Выражение даёт true тогда и только тогда, когда аргумент отличен от null.

2 Если первым аргументом является !, выражение даёт true тогда и только тогда, когда второй аргумент имеет значение null. Если первым аргументом является один из унарных условных операторов (6.4. Условные выражения bash), выражение даёт true, если унарная проверка даёт true. Если первый аргумент не является действительным унарным оператором, выражение даёт false.

3 Применяются приведённые ниже правила в указанном порядке.

1. Если вторым аргументом является один из бинарных условных операторов (6.4. Условные выражения bash), результатом выражения является результат применения бинарного оператора к первому и третьему аргументам. При наличии у команды 3 аргументов операторы -a и -o считаются бинарными.
2. Если первым оператором является !, значение является отрицанием теста с использованием второго и третьего аргументов.
3. Если первым аргументом является (, а третьим -), результатом будет тест для второго аргумента (внутри скобок).
4. В остальных случаях выражение даёт false.

4 Если первым оператором является !, результатом будет отрицание трехаргументного выражения, образуемого оставшимися аргументами. В остальных случаях выражение анализируется и оценивается с использованием порядка применения операторов и приведённых выше правил.

5 и более Выражение анализируется и оценивается с использованием порядка применения операторов и приведённых выше правил.

При использовании с test или [операторы < и > работают с лексикографической сортировкой на базе ASCII.

times

```
times
```

Выводит пользовательское и системное время, применяемое оболочкой и её потомками. Статус возврата 0.

trap

```
trap [-lp] [arg] [sigspec ...]
```

Команды, указанные параметром arg, будут считываться и выполняться при получении оболочкой сигнала sigspec. Если параметр arg не задан (и есть один сигнал sigspec) или имеет значение -, каждое заданное размещение сигнала сбрасывается к значению перед запуском оболочки. Если arg является пустой строкой (null), сигнал, заданный каждым параметром sigspec, игнорируется оболочкой и вызываемыми ею командами. Если arg отсутствует и задана опция -p, оболочка выводит команды прерывания, связанные с каждым sigspec. Если аргументы не заданы или указана лишь опция -p, команда trap выводит список команд, связанных с каждым номером сигнала в форме, которая может служить вводом для оболочки. Опция -l заставляет оболочку выводить список имён сигналов с их номерами. Каждый параметр sigspec является именем или номером сигнала. Регистр символов в именах сигналов не принимается во внимание, а префикс SIG не обязателен.

Если sigspec имеет значение 0 или EXIT, команда arg выполняется при выходе из оболочки. Если sigspec = DEBUG, команда arg выполняется перед каждой простой командой, for, case, select, каждым арифметическим преобразованием для команды и перед выполнением первой команды в функции оболочки. В описании опции extdebug внутренней команды shopt (4.3.2. Внутренняя команда shopt) рассмотрено влияние на прерывание DEBUG. Если sigspec = RETURN, команда arg выполняется каждый раз, когда завершается shell-функция или сценарий, выполняемые с использованием внутренних функций . или source.

Если sigspec = ERR, команда arg выполняется всякий раз, когда конвейер (может состоять из одной простой команды), список или составная команда возвращает отличный от нуля статус завершения при соблюдении перечисленных далее условий. Прерывание ERR не выполняется, если отказавшая команда является частью списка, следующего сразу за ключевым словом unit или while, частью проверки, следующей сразу за ключевым словом if или elif, частью команды, выполняемой в списке && или ||, кроме команды, следующей за последним && или ||, не последней командой конвейера или командой, статус возврата которой инвертируется символом !. Этим же условия действуют для опции errexit (-e).

Сигналы, игнорируемые на входе в оболочку, не могут быть перехвачены или сброшены. Перехваченные сигналы, которые не игнорируются, возвращаются к исходным значениям субоболочки или среды, где они были созданы.

Статус возврата равен 0, пока sigspec не задаёт действительный сигнал.

umask

```
umask [-p] [-S] [mode]
```

Задаёт для процессов создания файлов маску mode. Если значение mode начинается с цифры, оно считается восьмеричным числом, в противном случае - символьной маской, аналогичной используемой в команде chmod. Если параметр mode не указан, выводится текущее значение маски. Если задана опция -S без аргумента, маска выводится в символьном формате. Если задана опция -p, а параметр mode не указан, маска выводится в форме, пригодной для ввода. При изменении маски или отсутствии параметра mode команда возвращает 0.

Отметим, что при задании маски восьмеричным числом каждая цифра маски вычитается из 7, т. е. маска 022 задаёт режим доступа к файлу 755.

unset

```
unset [-fnv] [name]
```

Удаляет каждую переменную или функцию с именем name. При наличии опции -v удаляются переменные оболочки с соответствующим именем, опция -f задаёт удаление определений функций оболочки. Если задана опция -n, а name является переменной с атрибутом nameref, будет сбрасываться name, а не переменная, которую указывает

name. Опция -n не работает вместе с -f. Если опций не задано, каждое значение name указывает переменную, а если такой переменной нет, сбрасывается функция с именем name. Доступные лишь для чтения переменные и функции не могут быть сброшены командой unset. Команда возвращает 0, если не было попытки сбросить доступный лишь для чтения элемент.

4.2. Внутренние команды bash

В этом разделе описаны внутренние команды, которые уникальны для Bash или были расширены в этой оболочке. Некоторые из этих команд относятся к стандарту POSIX.

alias

```
alias [-p] [name[=value] ...]
```

Без аргументов или с опцией -p команда alias печатает список псевдонимов на стандартном устройстве вывода в форме, подходящей для ввода в другие команды. При наличии аргументов определяется псевдоним для каждого параметра name, значение которого указано. Если значение не задано, выводится name и value имеющегося псевдонима. Псевдонимы описаны в разделе 6.6. Псевдонимы

bind

```
bind [-m keymap] [-lpsvPSVX]
bind [-m keymap] [-q function] [-u function] [-r keyseq]
bind [-m keymap] -f filename
bind [-m keymap] -x keyseq:shell-command
bind [-m keymap] keyseq:function-name
bind [-m keymap] keyseq:readline-command
```

Выводит текущие привязки ключей и функций Readline (8. Редактирование командной строки), привязывает последовательность-ключ к функции или макросу Readline или устанавливает переменную Readline. Каждый аргумент, не являющийся опцией, служит командой в том виде, как она указана в файле инициализации Readline (8.3. Файл инициализации Readline), но каждая привязка или команда должна передаваться как отдельный аргумент, например, '"\C-x\C-r":re-read-init-file'. Значение опций описано ниже.

-m keymap	Задаёт использование keymap в качестве раскладки, на которую будут влиять последующие привязки. Пригодными именами привязок являются emacs, emacs-standard, emacs-meta, emacs-ctlx, vi, vi-move, vi-command, vi-insert. Привязка vi эквивалентна vi-command (vi-move также является синонимом), emacs эквивалентна emacs-standard.
-l	Список имён всех функций Readline.
-p	Выводит имена функций Readline и привязки в форме, пригодной для ввода или файла инициализации Readline.
-P	Выводит имена функций Readline и привязки.
-v	Выводит имена и значения переменных Readline в форме, пригодной для ввода или файла инициализации Readline.
-V	Выводит имена и значения переменных Readline.
-s	Выводит последовательности нажатия клавиш Readline, привязанные к макросам и строкам вывода в форме, пригодной для ввода или файла инициализации Readline.
-S	Выводит последовательности нажатия клавиш Readline, привязанные к макросам и строкам вывода.
-f filename	Считывает привязки клавиш из файла.
-q function	Запрос клавиш вызова указанной функции.
-u function	Отвязка всех клавиш для указанной функции.
-r keyseq	Удаление всех текущих привязок для keyseq.
-x keyseq:shell-command	Вызывает выполнение shell-command при вводе keyseq. При выполнении команды оболочка устанавливает в переменной READLINE_LINE текущее содержимое буфера строк Readline, а в READLINE_POINT - текущую точку вставки. Если выполненная команда изменила значение READLINE_LINE или READLINE_POINT, новые значения будут отражены в состоянии редактирования.
-X	Выводит все последовательности клавиш, привязанные к командам оболочки в пригодном для ввода формате.

Команда возвращает статус 0, если не было ошибок или непригодных опций.

builtin

```
builtin [shell-builtin [args]]
```

Запускает внутреннюю команду оболочки, передавая ей аргументы и возвращая статус завершения. Это полезно при определении функции оболочки, имя которой совпадает со внутренней командой, для сохранения функциональности команды внутри функции. Статус возврата будет ненулевым, если параметр shell-builtin не задаёт внутреннюю команду оболочки.

caller

```
caller [expr]
```

Возвращает контекст любого активного вызова подпрограммы (функции оболочки или сценария, выполняемого со внутренней командой . или source).

При запуске без параметра expr вызывающая сторона выводит номер строки и имя файла для текущего вызова подпрограммы. Если expr содержит неотрицательное целое число, выводится номер строки, имя подпрограммы и файл, соответствующие данной позиции в стеке вызовов. Эта информация может применяться, например, для вывода трассировки стека. Текущим является кадр 0.

Команда возвращает статус отличный от 0, если оболочка не выполняет вызов подпрограммы или expr не соответствует действительной позиции в стеке вызовов.

command

```
command [-pVv] command [arguments ...]
```

Запускает команду command с аргументами arguments, игнорируя функцию оболочки с именем command. Выполняются лишь внутренние команды и команды, найденные по пути PATH. Т. е. при наличии функции ls команда command ls внутри этой функции будет выполнять внешнюю команду ls, а не вызывать функцию рекурсивно. Опция -p задаёт использование принятого по умолчанию значения PATH, обеспечивающего

нахождение всех стандартных утилит. Статус возврата в этом случае будет 127, если command не удаётся найти или возникает ошибка. В остальных случаях возвращается статус выхода command.

При наличии опции -V или -v выводится описание command. Опция -v обеспечивает вывод одного слова, указывающего команду, а -V даёт более подробный вывод. В этом случае статус возврата будет 0, если команда найдена.

declare

```
declare [-aAfFgilnrtux] [-p] [name[=value] ...]
```

Объявляет переменные и задаёт их атрибуты. Если параметров name не задано, команда выводит значения переменных.

Опция -p задаёт вывод значений и атрибутов для каждого параметра name. При использовании -p с аргументами name все другие опции, за исключением -f и -F, игнорируются. При указании опции -p без name будут выводиться значения и атрибуты переменных, имеющих атрибуты, заданные другими опциями. Если других опций нет, -p обеспечивает вывод значений и атрибутов всех переменных оболочки. Опция -f ограничивает вывод shell-функциями.

Опция -F предотвращает вывод определений функций и печатаются лишь имена и атрибуты. Если включена опция extdebug с помощью внутренней команды shopt (4.3.2. Внутренняя команда shopt), будут также выводиться имена файлов и номера строк, где определена каждая переменная name. Опция -F подразумевает наличие -f. Опция -g задаёт создание и изменение переменных в глобальном масштабе даже при выполнении declare из функции оболочки. В других случаях опция игнорируется.

Ниже приведён список опций, которые могут служить для ограничения числа выводимых переменных.

- **-a** Каждое имя, являющееся переменной индексированного массива (6.7. Массивы).
- **-A** Каждое имя, являющееся переменной ассоциативного массива (6.7. Массивы).
- **-f** Только имена функций.
- **-i** Переменная трактуется как целое число, при установке значения выполняются арифметические преобразования (6.5. Арифметика командного процессора).
- **-l** При установке значения символы верхнего регистра переводятся в нижний. Атрибут upper-case отключён.
- **-n** Задаёт для каждой переменной name атрибут nameref, делая её ссылкой на другую переменную. Эта переменная указывается значение value для name. Все ссылки, назначения и изменения атрибутов name, кроме использующих или меняющих сам атрибут -n, выполняются для переменной, указанной value. Атрибут nameref не применим к переменным массивов.
- **-r** Делает переменные name доступными лишь для чтения. Этим переменным не могут быть присвоены или сброшены значения в последующих операторах или командах unset.
- **-t** Задаёт для каждого name атрибут trace. Трассируемые функции наследуют ловушки DEBUG и RETURN из вызывающей оболочки. Атрибут trace не имеет специального значения для переменных.
- **-u** При установке значения символы нижнего регистра переводятся в верхний. Атрибут lower-case отключён.
- **-x** Помечает каждую переменную name для экспорта в последующие команды через окружение.

Использование + вместо - отключает атрибут, за исключением того, что +a и +A не могут использоваться для уничтожения переменных массива, а +r не будет удалять атрибут readonly. При использовании в функции команда declare делает каждое имя name локальным как при использовании команды local, за исключением случаев наличия опции -g. Если за name следует =value, для переменной устанавливается значение value.

При использовании -a или -A и синтаксиса составного назначения для создания переменных массивов дополнительные атрибуты не будут работать до последующих назначений. Статусом возврата будет 0, если нет непригодных опций, попыток определить функцию с помощью -f foo=bar, попыток задать значение доступной лишь для чтения переменной, попыток назначить значения переменной массива без использования синтаксиса составного назначения (6.7. Массивы), указания в name недействительной переменной оболочки, попыток отключить статус readonly для переменной readonly, попыток отключить статус массива для переменной массива или попыток вывести несуществующую функцию с помощью опции -f.

echo

```
echo [-neE] [arg ...]
```

Выводит аргументы (arg), разделённые пробелами и завершающиеся символом новой строки. Статус возврата имеет значение 0, если не возникло ошибок. При наличии опции -n символ newline в конце строки подавляется. Опция -e включает \-экранирование последующих символов. Опция -E отключает экранирование символов даже в системах, где оно включено по умолчанию. Опция оболочки xpg_echo может служить для динамического управления преобразованием экранируемых символов в эхо-строке. Команда echo не интерпретирует последовательность — (конец опций).

Интерпретируемые командой echo escape-последовательности приведены ниже.

- **\a** Сигнал (звонок).
- **\b** «Забой» (backspace).
- **\c** Подавление последующего вывода.
- **\e** Символ экранирования (не ANSI C).
- **\E**
- **\f** Перевод страницы (form feed).
- **\n** Новая строка.
- **\r** Возврат каретки.
- **\t** Горизонтальная табуляция.
- **\v** Вертикальная табуляция.
- **** Обратная дробная черта (\).
- **\0nnn** Восьмибитовый символ с восьмеричным кодом nnn (1-3 восьмеричных цифры).
- **\xHH** Восьмибитовый символ с шестнадцатеричным кодом HH (1-2 шестнадцатеричных цифры).
- **\uHHHH** Символ Unicode (ISO/IEC 10646) с шестнадцатеричным кодом HHHH (1-4 шестнадцатеричных цифры)
- **\UHHHHHHHH** Символ Unicode (ISO/IEC 10646) с шестнадцатеричным кодом HHHHHHHH (1-8 шестнадцатеричных цифр)

enable

```
enable [-a] [-dnps] [-f filename] [name ...]
```

Включает и отключает внутренние команды оболочки. Запрет внутренней команды позволяет использовать взамен одноимённую внешнюю команду (файл) без указания полного пути, хотя обычно оболочка выбирает сначала внутреннюю команду. При указании опции -n заданные аргументами name команды отключаются. В противном

случае команды name включаются. Например, для использования двоичного исполняемого файла test, доступного в пути $PATH, вместо внутренней функции можно использовать команду enable -n test.

Если задана опция -p или нет аргумента name, выводится список внутренних команд оболочки. При отсутствии других аргументов список будет включать все включённые внутренние команды оболочки. С опцией -a указывается включена команда или выключена.

Опция -f задаёт загрузку новой внутренней команды name из общего объекта filename в системе, поддерживающей динамическую загрузку. Опция -d удаляет внутреннюю команду, загруженную с -f.

Если опций не задано, выводится список внутренней команд оболочки. Опция -s ограничивает включение специальными внутренними командами POSIX. Если опции -s и -f указаны вместе, новая внутренняя команда становится специальной (4.4. Специальные внутренние команды).

Команда возвращает статус 0, пока name задаёт внутреннюю команду оболочки и при загрузке новой внутренней команды из общего объекта не возникает ошибки.

help

```
help [-dms] [pattern]
```

Выводит информацию о встроенных командах. Если задан аргумент pattern, будет выводиться подробная информация о всех командах , соответствующих шаблону. В остальных случаях выводится список внутренних команд. Опции команды описан ниже.

-d Выводить краткое описание для каждого значения pattern.

-m Выводить описание для каждого значения pattern в формате, похожем на вывод man.

-s Выводить лишь краткий синтаксис применения для каждого значения pattern.

Команда возвращает статус 0, если хотя бы одна команда соответствует pattern.

let

```
let expression [expression ...]
```

Внутренняя команда let позволяет выполнять арифметические операции с переменными оболочки. Каждый аргумент expression преобразуется в соответствии с правилами раздела 6.5. Арифметика командного процессора. Если последнее преобразование даёт 0, команда let возвращает статус 1, в остальных случаях возвращается 0.

local

```
local [option] name[=value] ...
```

Для каждого аргумента создаётся локальная переменная с именем name и значением value. Аргумент option может задавать любые опции, пригодные для команды declare. Команда local может применяться лишь в функциях, она делает переменную name видимой только в этой функции и её потомках. Если указано имя -, набор параметров оболочки делается локальным для функции, где вызвана команда local. Опции оболочки, изменённые внутри функции, восстанавливают свои значения при возврате из функции. Команда возвращает статус 0, пока она применяется внутри функции, значение name действительно и переменная name доступна для записи.

logout

```
logout [n]
```

Выход из оболочки login с возвратом родителю статуса n.

mapfile

```
mapfile [-d delim] [-n count] [-O origin] [-s count]
    [-t] [-u fd] [-C callback] [-c quantum] [array]
```

Читает строки в переменную индексированного массива со стандартного ввода или из файла с дескриптором fd, если задана опция -u. По умолчанию массив задан переменной MAPFILE. Опции команды описаны ниже.

-d Первый символ аргумента delim служит разграничителем строк вместо символа newline. Если delim является пустой строкой, mapfile будет завершать строку при считывании символа NUL.

-n Максимальное число копируемых строк (count). По умолчанию число строк не ограничено (0).

-O Начинать заполнение массива с индекса origin (по умолчанию 0).

-s Отбрасывать первые count прочитанных строк.

-t Удалять символ delim (по умолчанию newline) в конце каждой строки.

-u Читать строки из файла с дескриптором fd вместо стандартного ввода.

-C Оценивать обратный вызов (callback) при считывании каждых quantum строк. Значение задаёт -c quantum.

-c Задаёт число строк, считываемых между вызовами callback.

Если опция -C задана без указания -c, по умолчанию применяется quantum = 5000. При оценке обратного вызова ему представляется индекс назначаемого следующим элемента массива и строка для присвоения этому элементу как добавочные аргументы. Оценка callback выполняется после чтения строки но до назначения элемента массива. Если в команде явно не указано значение origin, команда mapfile очищает массив перед присвоением значений.

Команда mapfile возвращает 0, если не заданы непригодные опции или их аргументы, массив корректен и допускает назначение, а также является индексированным.

printf

```
printf [-v var] format [arguments]
```

Записывает форматированные аргументы на стандартный вывод в соответствии с аргументом format. Опция -v задаёт присваивание значений переменной var вместо печати на стандартном устройстве вывода.

Аргумент format - строка символов, содержащая 3 типа объектов: обычные символы, которые просто копируются на стандартный вывод, escape-последовательности, которые преобразуются и копируются на стандартный вывод, а также спецификации формата, каждая из которых вызывает печать следующего аргумента. В дополнение к стандартным форматам printf(1) команда printf интерпретирует описанные ниже расширения.

%b Заставляет printf преобразовывать escape-последовательности с \ в соответствующем аргументе, как это делается в echo -e (4.2. Внутренние команды bash).

%q Заставляет printf выводить соответствующий аргумент в формате, пригодном для ввода.

%(datefmt)T Заставляет printf выводить строку даты и времени, полученную с использованием datefmt как строки формата для strftime(3). Соответствующий аргумент является целым числом секунд с начала эпохи. Могут применяться два специальных значения аргумента: -1 представляет текущее время, -2 - время вызова оболочки. Если аргумент не задан, предполагается -1. Это расширяет обычное поведение printf.

Аргументы для нестроковых спецификаторов формата трактуются как константы языка C, за исключением того, что допускаются начальные знаки + и -, а в случае первого символа одинарных или двойных кавычек значением является ASCII-код следующего символа.

Аргумент format при необходимости применяется многократно для всех arguments. Если format требует больше arguments, чем представлено, дополнительные спецификации формата ведёт себя как при представлении

значения 0 или пустой строки. Команда возвращает статус 0 при успешном выполнении и отличное от нуля значение при отказе.

read

```
read [-ers] [-a aname] [-d delim] [-i text] [-n nchars]
    [-N nchars] [-p prompt] [-t timeout] [-u fd] [name ...]
```

Считывает одну строку со стандартного ввода или из файла с дескриптором fd, указанного в опции -u, выделяет слова, как описано в параграфе 3.5.7. Расщепление слов и назначает первое слово первой переменной name, второе - следующей и т. д. Если слов больше, чем аргументов name, оставшиеся слова и разделители между ними назначаются последнему аргументу name. Если слов меньше, чем аргументов name, соответствующим переменным name присваиваются пустые значения. Для разделения слов используются символы значения переменной IFS как при расщеплении слов (3.5.7. Расщепление слов). Символ \ может служить для отмены специальной трактовки следующего символа или для продолжения строки. Если аргументы name не указаны, считанная строка назначается переменной REPLY. Команда возвращает статус 0, если не было получен символ конца файла, не исчерпано время ожидания (в этом случае возвращается статус больше 128), не было ошибок при назначении переменных (например, попытка записи в переменную, доступную лишь для чтения) и не был указан непригодный дескриптор файла в опции -u.

Опции команды описаны ниже.

-a aname Слова присваиваются последовательным элементам массива aname, начиная с индекса 0. Все элементы aname удаляются перед назначением. Аргументы name игнорируются.

-d delim Первый символ delim используется для завершения строки ввода вместо newline. Если delim является пустым, чтение строки будет прерываться NUL-символом.

-e Для получения строки используется команда Readline (8. Редактирование командной строки) с текущими (или принятыми по умолчанию, если редактирование строк ещё не применялось) настройками, и принятого в Readline по умолчанию дополнения имён файлов.

-i text Если для редактирования строк будет применяться Readline, текст помещается в буфер редактирования.

-n nchars Команда read возвращает управление после считывания nchars символов, не ожидая ввода всей строки, но учитывает разделитель, если он встретился до считывания nchars символов.

-N nchars Команда read возвращает управление после считывания в точности nchars символов, не ожидая ввода всей строки, если раньше не был получен символ EOF или истекло время ожидания. Ограничители не трактуются в качестве таковых и не приводят к возврату управления, пока не будет прочитано nchars символов. Результат не расщепляется по символам IFS, это делается для того, чтобы переменной назначался весь набор прочитанных символов (кроме \, как указано ниже).

-p prompt Выводит приглашение без завершающего символа newline перед попыткой чтения ввода. Приглашение отображается лишь при вводе с терминала.

-r Эта опция отменяет действие \ в качестве символа экранирования и он считается частью строки. В частности, \newline не используется как продолжение строки.

-s «Тихий» режим. При вводе символов с терминала они не выводятся на экране.

-t timeout Задаёт считывание по времени и возвращает отказ, если строка (или заданное число символов) не была считана за timeout секунд. Аргумент timeout может быть десятичным числом с дробной частью, отделённой точкой. Эта опция действует лишь при чтении с терминала, из канала или иного специального файла, но не действует при чтении из обычного файла. При возникновении тайм-аута считанные данные сохраняются в переменной, заданной параметром name. Если timeout = 0, время команда завершает работу сразу, без попытки считывания данных. Команда возвращает статус 0, если данные доступны в указанном дескрипторе файла. При тайм-ауте возвращается значение больше 128.

-u fd Чтение данных из файла с дескриптором fd.

readarray

```
readarray [-d delim] [-n count] [-O origin] [-s count]
    [-t] [-u fd] [-C callback] [-c quantum] [array]
```

Читает строки в переменную индексированного массива со стандартного ввода или из файла с дескриптором fd, если задана опция -u.

Синоним mapfile.

source

```
source filename
```

Синоним команды . (4.1. Внутренние элементы Bourne Shell).

type

```
type [-afptP] [name ...]
```

Для каждого аргумента name указывает способ интерпретации при использовании как имени команды.

При указании опции -t команда type выводит одно слово, которое задаёт псевдоним, функцию, внутреннюю команду, файл или ключевое слово, соответствующее аргументу name. Если соответствующего name элемента не найдено, ничего не выводится и возвращается статус отказа.

Опция -p option задаёт вывод имени дискового файла, который будет выполнен, или ничего, если -t не будет возвращать файл. Опция -P задаёт поиск в пути для каждого name, даже если -t не будет возвращать файл. Если команда хэширована, опции -p и -P будут выводить хэшированное имя, которое не обязательно соответствует первому файлу, найденному в $PATH.

При указании опции -a команда type возвращает все полные пути к исполняемым файлам name. При отсутствии опции -p будут выведены также псевдонимы и функции.

При указании опции -f команда type не пытается найти функции оболочки.

Статусом возврата будет 0, если найдены все name.

typeset

```
typeset [-afFgrxilnrtux] [-p] [name[=value] ...]
```

Команда typeset поддерживается для совместимости в оболочкой Korn и является синонимом внутренней команды declare.

ulimit

```
ulimit [-HSabcdefiklmnpqrstuvxPT] [limit]
```

Команда ulimit обеспечивает контроль ресурсов, доступных запускаемому оболочкой процессу, в системах, поддерживающих такой контроль. Опции команды описаны ниже.

-S Изменение и отчёт о мягком ограничении, связанном с ресурсом.
-H Изменение и отчёт о жёстком ограничении, связанном с ресурсом.
-a Вывод всех установленных ограничений.
-b Максимальный размер буфера сокетов.
-c Максимальный размер создаваемых core-файлов.
-d Максимальный размер сегмента данных процесса.
-e Максимальный приоритет при планировании (nice).
-f Максимальный размер файлов, записываемых оболочкой и её потомками.
-i Максимальное число ожидающих сигналов.
-k Максимальное число выделяемых очередей kqueue.
-l Максимальный размер блокируемой памяти.
-m Максимальный размер резидентного блока (многие системы не соблюдают это ограничение).
-n Максимальное число дескрипторов открытых файлов (большинство систем не поддерживает это ограничение).
-p Размер буфера канала (pipe).
-q Максимальное число байтов в очереди сообщений POSIX.
-r Максимальный приоритет при планировании в реальном масштабе времени.
-s Максимальный размер стека.
-t Максимальное время процессора в секундах.
-u Максимальное число процессов для одного пользователя.
-v Максимальный объем виртуальной памяти, доступной оболочке (на некоторых системах также потомкам).
-x Максимальное число файловых блокировок.
-P Максимальное число псевдотерминалов.
-T Максимальное число потоков (thread).

Если аргумент limit задан и опция -a не используется, команда устанавливает новое ограничение для указанного ресурса. Специальные значения hard, soft и unlimited указывают текущие значения жёстких и мягких ограничений или отсутствие ограничений. Жёсткий предел не может быть увеличен пользователем (кроме root), а мягкий предел может быть увеличен до значения не более жёсткого предела.

В остальных случаях выводится текущее мягкое ограничение для ресурса, если не задана опция -H. При установке новых ограничений и отсутствии опции -H или -S задаётся сразу мягкий и жёсткий предел. Если опции не заданы, предполагается -f. Значения увеличиваются с шагом 1024 байта, за исключением -t (секунды), -p (блоки по 512 байтов), -P, -T, -b, -k, -n и -u (произвольные), а в режиме POSIX (6.11. Режим POSIX) -c и -f (блоки по 512 байтов).

Команда возвращает отличный от 0 статус, если указана непригодная опция или аргумент, а также при возникновении ошибки в случае установки ограничения.

unalias

```
unalias [-a] [name ... ]
```

Удаляет name из списка псевдонимов. Опция -a задаёт удаление всех псевдонимов.

4.3. Изменение поведения оболочки

4.3.1. Внутренняя команда set

Эта команда весьма сложна и заслуживает отдельного рассмотрения. Команда позволяет менять значения параметров оболочки, устанавливать позиционные параметры, выводить имена и значения переменных среды.

```
set [--abefhkmnptuvxBCEHPT] [-o option-name] [argument ...]
set [+abefhkmnptuvxBCEHPT] [+o option-name] [argument ...]
```

При отсутствии опций и аргументов команда set выводит имена и значения всех переменных оболочки и функций с сортировкой в соответствии с текущим языком (locale) в формате, пригодном в качестве ввода для установки или сброса уже установленных переменных. Переменные с доступом только для чтения не могут быть сброшены. В режиме POSIX выводятся лишь переменные оболочки.

Опции командной строки служат для установки или сброса атрибутов оболочки. Описание опций приведено ниже.

-a

Каждой переменной или функции, которая создаётся или изменяется, назначается атрибут экспорта и она помечается для экспорта в среду последующих команд.

-b

Задаёт незамедлительный возврат статуса прерванных фоновых заданий вместо вывода перед следующим основным приглашением.

-e

Немедленно завершить работу, если конвейер (3.2.2. Конвейеры), который может включать одну простую команду (3.2.1. Простые команды), список (3.2.3. Списки) или составную команду (3.2.4. Составные команды), возвращает отличный от 0 статус. Работа оболочки не завершается, если вызвавшая отказ команда является частью списка, следующего сразу после ключевого слова while или unit, частью проверочного выражения в операторе if, частью любой команды, выполняемой в списке && или || (кроме последней в списке), любой командой в конвейере (кроме последней) или статус возврата команды инвертируется оператором !. Если составная команда, не являющаяся субоболочкой, возвращает ненулевой статус в результате отказа при игнорировании опции -e, работа оболочки не завершается. Если установлено прерывание ERR, оно выполняется перед завершением работы оболочки. Эта опция применяется раздельно к среде оболочки и каждой субоболочки (3.7.3. Среда выполнения команды) и может приводить к завершению работы субоболочки до выполнения в ней всех команд.

Если составная команда или shell-функция выполняется в контексте, где опция -e не принимается во внимание, ни одна из команд, выполняемых внутри составной команды или в теле функции, не будет зависеть от опции -e даже при её наличии и возврате командой отказа. Если составная команда или функция оболочки задаёт опцию -e в контексте, где эта опция игнорируется, эта установка не будет оказывать влияния, пока не завершится составная команда или команда, включающая функцию.

-f

Отключает преобразование имён файлов (globbing).

-h

Поиск и запоминание (хэш) команд, которые отыскивались для выполнения (включено по умолчанию).

-k

Все параметры в форме операторов присваивания помещаются в среду для команды в дополнение к параметрам перед именем команды.

-m

Управление заданиями включено (7. Управление заданиями). Все процессы работают в отдельной группе. По завершении фонового задания оболочка выводит строку со статусом завершения.

-n

задаёт чтение команд без их выполнения. Это можно применять для проверки сценариев на предмет ошибок. Опция игнорируется в интерактивной оболочке.

-o option-name

Задаёт опцию, соответствующую option-name.

allexport	Совпадает с -a.
braceexpand	Совпадает с -B.
emacs	Задаёт использование интерфейса редактирования командной строки в стиле emacs (8. Редактирование командной строки). Влияет также на интерфейс редактирования для read -e.
errexit	Совпадает с -e.
errtrace	Совпадает с -E.
functrace	Совпадает с -T.
hashall	Совпадает с -h.
histexpand	Совпадает с -H.
history	Включает историю команд, как описано в разделе 9.1. Средства Bash History. Опция включена по умолчанию в интерактивных оболочках.
ignoreeof	Интерактивная оболочка не будет завершать работу при считывании EOF.
Keyword	Совпадает с -k.
monitor	Совпадает с -m.
noclobber	Совпадает с -C.
noexec	Совпадает с -n.
noglob	Совпадает с -f.
nolog	А настоящее время игнорируется.
notify	Совпадает с -b.
nounset	Совпадает с -u.
onecmd	Совпадает с -t.
physical	Совпадает с -P.
pipefail	При установке возвращаемым значением конвейера будет значение статуса последней (самой правой) команды или 0, если все команды выполнены успешно. Опция по умолчанию отключена.
posix	Меняет поведение Bash, если принятые по умолчанию установки отличаются от стандарта POSIX, в соответствии с этим стандартом (6.11. Режим POSIX). Опция предназначена для обеспечения поведения Bash в строгом соответствии с надмножеством этого стандарта.
privileged	Совпадает с -p.
verbose	Совпадает с -v.
vi	Задаёт использование стиля vi при редактировании строк. Влияет также на интерфейс редактирования команды read -e.
xtrace	Совпадает с -x.

-p

Включает привилегированный режим, в котором файлы $BASH_ENV и $ENV не обрабатываются, функции оболочки не наследуются из среды, а переменные SHELLOPTS, BASHOPTS, CDPATH и GLOBIGNORE, при наличии их в окружении, игнорируются. Если оболочка запущена с действующим идентификатором пользователя (группы), отличным от идентификатора реального пользователя (группы), а опция -p не задана, эти действия выполняются и для действующего идентификатора пользователя устанавливается идентификатор реального пользователя. Если опция -p указана при запуске, идентификатор пользователя не меняется. Выключение этой опции ведёт к установке действующих идентификаторов пользователя и группы в соответствии с реальными.

-t

Выход после считывания и выполнения одной команды.

-u

Задаёт трактовку неустановленных переменных и параметров (кроме @ и *) как ошибку при преобразовании параметров. Сообщение об ошибке отображается на стандартном устройстве вывода и работа неинтерактивной оболочки завершается.

-v

Выводит входные строки оболочки по мере их чтения.

-x

Выводит трассировку простых команд для команд, команд case и select, а также арифметики для команд и аргументов или связанных списков слов после преобразования, но до выполнения. Значение переменной PS4 преобразуется и выводится результат перед командой и её преобразованными аргументами.

-B

Оболочка выполняет раскрытие скобок (3.5.1. Раскрытие скобок). Опция включена по умолчанию.

-C

Предотвращает переписывание имеющихся файлов при перенаправлении вывода с использованием >, >&, и <>.

-E

При установке опции все прерывания (trap) в ERR наследуются функциями оболочки, подстановками команд и командами, выполняемыми в среде субоболочки. Обычно прерывания ERR в таких случаях не наследуется.

-H

Включает стиль ! в подстановке истории (9.3. Преобразование истории). Опция включена по умолчанию для интерактивных оболочек.

-P

Установленная опция отключает преобразование символьных ссылок при выполнении команд, меняющих текущий каталог (таких как cd). Вместо этого будет использоваться реальный каталог. По умолчанию Bash следует логической цепочке каталогов при выполнении команд, меняющих текущий каталог.

Например, если /usr/sys является символьной ссылкой на /usr/local/sys

```
$ cd /usr/sys; echo $PWD
/usr/sys
$ cd ..; pwd
/usr
```

При установке опции -P

```
$ cd /usr/sys; echo $PWD
/usr/local/sys
$ cd ..; pwd
/usr/local
```

-T

При установленной опции любые прерывания в DEBUG и RETURN наследуются функциями оболочки, подстановками команд и командами, выполняемыми в среде субоболочки. Обычно прерывания DEBUG и RETURN в таких случаях не наследуется.

--

Если за этой опцией не следует аргумент, позиционные параметры не устанавливаются. В противном случае позиционные параметры устанавливаются в соответствии со значениями аргументов, даже если некоторые из них начинаются с символа -.

-

Указывает завершение опции и назначение оставшихся аргументов позиционным параметрам. Опции -x и -v выключаются. Если аргументов нет, позиционные параметры не меняются.

Использование символа + вместо - отключает соответствующую опцию. Опции могут также применяться при вызове оболочки. Текущий набор опций хранится в переменной $-.

Оставшиеся N аргументов являются позиционными параметрами $1, $2, . . . $N. Для специального параметра # устанавливается значение N.

Команда всегда возвращает статус 0, если не задана недействительная опция.

4.3.2. Внутренняя команда shopt

Эта команда позволяет изменить необязательное поведение оболочки.

```
shopt [-pqsu] [-o] [optname ...]
```

Команда меняет значения, контролирующие поведение дополнительной оболочки. Это могут быть перечисленные ниже параметры или (при использовании опции -o) параметры, доступные с опцией -o внутренней команде set (4.3.1. Внутренняя команда set). При вызове без опций или с опцией -p команда выводит список всех доступных для установки опций с индикацией уже установленных. Если при этом задан параметр optname, вывод ограничивается указанными опциями. Опция -p обеспечивает вывод в форме, которая может использоваться в качестве входных данных.

-s Включает (set) каждую опцию из optname.
-u Выключает (unset) каждую опцию из optname.
-q Подавляет обычный вывод. Статус возврата указывает были опция optname установлена или сброшена. При наличии нескольких аргументов optname с опцией -q статус возврата будет 0, если установлены все optname.
-o Ограничивает значения optname заданными для опции -o внутренней команды set (4.3.1. Внутренняя команда set).

Если опция -s или -u используется без аргумента optname, команда shopt показывает лишь установленные или сброшенные опции, соответственно.

Если явно не указано иное, опции shopt по умолчанию отключены.

Статусом возврата при просмотре опций является 0, если все опции из optname включены и отличное от 0 значение в противном случае. При установке или сбросе опций возвращается статус 0, если optname указывает действительную опцию оболочки.

assoc_expand_once

При установке опции оболочка подавляет многократное вычисление индексов ассоциативных массивов в процессе вычисления арифметических выражений при выполнении внутренних команд, которые могут назначать переменные и разыменовывать массивы.

autocd

При установке этой опции команда, являющаяся именем каталога, будет восприниматься как аргумент команды cd. Опция может применяться лишь в интерактивных оболочках.

cdable_vars

При установленной опции аргумент внутренней команды cd, не являющийся каталогом, считается именем переменной, значение которой задаёт каталог для перехода.

cdspell

При включённой опции незначительные ошибки при указании каталога в команде cd будут исправляться. Проверяется перестановка символов, пропуски и избыточные символы. Если исправление найдено, выводится корректный путь и команда выполняется. Опция применяется только в интерактивных оболочках.

checkhash

При установке опции Bash проверяет наличие команды, найденной в хэш-таблице, перед попыткой её выполнения. Если хэшированной команды больше не существует, выполняется обычный поиск в заданных путях.

checkjobs

При установке опции Bash выводит состояния всех остановленных и работающих заданий, имеющихся в интерактивной оболочке. Если какие-то задания выполняются, это откладывает выход до второй попытки выхода без промежуточной команды (7. Управление заданиями). Оболочка всегда откладывает завершение при наличии остановленных заданий.

checkwinsize

При установленной опции Bash проверяет размер терминального окна после каждой внешней команды и при необходимости обновляет значения LINES и COLUMNS. По умолчанию опция включена.

cmdhist

При установленной опции Bash пытается сохранить все строки многострочной команды в одной записи истории. Это позволяет впоследствии редактировать такие команды. Опция включена по умолчанию, но работает лишь при включённой истории команд (9.1. Средства Bash History).

compat31

При установленной опции Bash меняет своё поведение в соответствии с версией 3.1 применительно к аргументам в кавычках для оператора условных команд =~ и зависимого от языка сравнения строк при использовании команды [[с операторами < и >. Версии до bash-4.1 используют сортировку ASCII и strcmp(3), а bash-4.1 и более поздние версии используют сортировку установленного языка и strcoll(3).

compat32

При установленной опции Bash меняет своё поведение в соответствии с версией 3.2 применительно к сравнению строк на основе языка (locale) в операторах < и > условных команд [[и прерыванию списка команд. В Bash версий 3.2 и меньше выполнялся переход к следующей команде в списке после завершения команды прерыванием.

compat40

При установленной опции Bash меняет своё поведение в соответствии с версией 4.0 применительно к сравнению строк на основе языка (locale) в операторах < и > условных команд [[и прерыванию списка команд. В Bash версий 4.0 и выше список прерывается как при получении прерывания оболочкой, а в более ранних продолжается со следующей командой из списка.

compat41

При установленной опции Bash в режиме POSIX трактует одинарные кавычки при преобразовании параметров в двойных кавычках как специальный символ. Одинарные кавычки должны быть парными, а символы между ними считаются заключёнными в кавычки. Это поведение относится к режиму POSIX до версии 4.1. По умолчанию Bash ведёт себя как в предшествующих версиях.

compat42

При установленной опции Bash не обрабатывает строку замены в преобразовании слова подстановки шаблона с использованием удаления кавычек.

compat43

При установленной опции Bash не выводит предупреждения при попытке использовать в качестве аргумента заключённое в кавычки назначение составного массива, считает ошибки преобразования слов не критическими (по умолчанию они являются критическими и ведут к завершению работы оболочки) и не сбрасывает состояние цикла при выполнении функции оболочки (это позволяет прервать или продолжить цикл в функции оболочки для воздействия на цикл в контексте вызывающего элемента).

compat44

При установленной опции Bash сохраняет позиционные параметры в переменных BASH_ARGV и BASH_ARGC до их использования, независимо от включения режима расширенной отладки.

complete_fullquote

При установленной опции Bash заключает в кавычки все метасимволы в именах файлов и каталогов при дополнении. Если опция не задана, Bash удаляет метасимволы , такие как $, из числа помещаемых в кавычки в дополненных именах файлов, когда эти метасимволы появляются в ссылках на переменные оболочки в словах, которые нужно дополнить. Это означает, что символ $ в именах переменных, которые преобразуются в каталоги, не будет заключаться в кавычки, однако он не будет заключаться в кавычки и в именах файлов. Это действует лишь в тех случаях, когда bash использует символ \ для дополненных имён файлов. Опция установлена по умолчанию, что соответствует принятому по умолчанию поведению Bash до версии 4.2.

direxpand

При установленной опции Bash заменяет имена каталогов результатами преобразования слов при дополнении имён файлов. Это меняет содержимое буфера редактирования readline. Если опция не задана, Bash пытается сохранить пользовательский ввод.

dirspell

При установленной опции Bash пытается корректировать имена каталогов при дополнении, если заданного пользователем каталога не существует.

dotglob

При установленной опции Bash включает имена, начинающиеся с ., в результат преобразования имён. Имена . и .. должны совпадать явно даже при установке dotglob.

execfail

При установленной опции неинтерактивная оболочка не будет завершаться при невозможности выполнить файл, указанный как аргумент внутренней команды exec. Интерактивная оболочка не завершается при отказах exec.

expand_aliases

При установленной опции псевдонимы преобразуются в соответствии с разделом 6.6. Псевдонимы. Эта опция по умолчанию включена для интерактивной оболочки.

extdebug

Если опция установлена при вызове оболочки, организуется выполнение профиля отладчика до запуска оболочки как при указании опции --debugger. При установке опции после вызова оболочки, включается поведение, предназначенное для отладчиков, как описано ниже.

1. Опция -F внутренней команды declare (4.2. Внутренние команды bash) выводит имя файла с исходным кодом и номер строки, соответствующие каждому имени функции, заданному в качестве аргумента.
2. Если команда, запускаемая прерыванием DEBUG, возвращает отличный от 0 статус, следующая команда пропускается.
3. Если команда, запускаемая прерыванием DEBUG, возвращает 2 и оболочка выполняется в подпрограмме (функция оболочки или сценарий, запущенный с помощью . или source), имитируется вызов return.
4. Переменные BASH_ARGC и BASH_ARGV обновляются, как указано в параграфе 5.2. Переменные Bash.
5. Трассировка функций включена, подстановка команд, функции оболочки и субоболочки, вызываемые с помощью (command), наследуют ловушки DEBUG и RETURN.
6. Трассировка ошибок включена, подстановка команд, функции оболочки и субоболочки, вызываемые с помощью (command), наследуют ловушку ERR.

extglob

Опция включает функции расширенного сопоставления с шаблонами (3.5.8.1. Сопоставление с шаблоном).

extquote
Включает поддержку кавычек $'string' и $"string" в преобразовании выражений ${parameter}. По умолчанию включена.

failglob
При включённой опции отсутствие совпадений с шаблоном при преобразовании имён файлов ведёт к ошибке.

force_fignore
При включённой опции суффиксы, заданные переменной оболочки FIGNORE (5.2. Переменные Bash), ведут к игнорированию слов при дополнении, даже когда игнорируемое слово является единственным вариантом дополнения. Опция включена по умолчанию.

globasciiranges
При включённой опции выражения диапазонов, используемые в выражениях сопоставления с шаблоном, содержащих скобки (3.5.8.1. Сопоставление с шаблоном), ведут себя как при сравнении с традиционной языковой установкой C. Т. е. порядок символов текущей языковой установки не принимается во внимание и символы верхнего и нижнего регистра ASCII будут сравниваться вместе.

globstar
При включённой опции шаблон ** в контексте преобразования имён файлов будет соответствовать всем файлам, а также каталогам и подкаталогам. Если за шаблоном следует символ /, соответствовать будут лишь (под)каталоги.

gnu_errfmt
При включённой опции сообщения оболочки об ошибках выводятся в стандартном формате сообщений gnu.

histappend
При включённой опции список истории добавляется в конец файла, указанного значением переменной HISTFILE, вместо переписывания файла при выходе из оболочки.

histreedit
При включённой опции и использовании Readline пользователь может повторно редактировать неудачную подстановку истории.

histverify
При включённой опции и использовании Readline результат подстановки истории не передаётся сразу же анализатору оболочки, а полученная в результате строка загружается в буфер редактирования Readline, что обеспечивает возможность дополнительного изменения.

hostcomplete
При включённой опции и использовании Readline интерпретатор Bash будет пытаться дополнять имя хоста, когда при дополняется слово с символом @ (8.4.6. Завершение строк из Readline). Опция включена по умолчанию.

huponexit
При включённой опции Bash будет передавать сигнал SIGHUP всем заданиям при выходе из интерактивной (login) оболочки (3.7.6. Сигналы).

inherit_errexit
При включённой опции подстановка команд наследует опцию errexit вместо её сброса в среде субоболочки. Опция включена в режиме POSIX.

interactive_comments
Позволяет интерактивной оболочке игнорировать содержимое строки, начиная с символа # и до конца строки (включена по умолчанию).

lastpipe
При включённой опции и выключенном управлении заданиями оболочка запускает последнюю команду конвейера, которая не была выполнена в фоновом режиме в среде текущей оболочки.

lithist
Если опция включена вместе с опцией cmdhist, многострочные команды сохраняются в списке истории с символами новой строки вместо ;.

localvar_inherit
При включённой опции локальные переменные наследуют значения и атрибуты одноимённых переменных из предыдущей области действия до присвоения новых значений. Атрибут nameref не наследуется.

localvar_unset
При включённой опции вызов unset для локальных переменных в областях действия предыдущих функций маркирует их так, что последующий поиск видит их неустановленными, пока не произойдёт возврат из функции. Это идентично поведению при отмене локальных переменных в области действия текущей функции.

login_shell
Оболочка устанавливает эту опцию при запуске в режиме входа в систему - login (6.1. Вызов Bash). Изменить значение невозможно.

mailwarn
При включённой опции выводится сообщение "The mail in mailfile has been read", если к файлу, который Bash проверяет на предмет почты, было обращение с момента последней проверки.

no_empty_cmd_completion
При включённой опции и использовании Readline оболочка Bash не пытается искать в PATH возможные дополнения при попытке дополнения пустой строки.

nocaseglob
При включённой опции Bash сопоставляет имена файлов без учёта регистра символов для преобразования имён.

nocasematch
При включённой опции Bash сравнивает шаблон без учёта регистра символов при сопоставлении во время выполнения условных команд case или [[conditional, при подстановке шаблонов в преобразовании слов и при фильтрации возможных дополнений как части программируемого дополнения.

nullglob
При включённой опции Bash позволяет преобразование шаблонов имён, которым ничего не соответствует, в пустую строку, а не в себя.

progcomp
При включённой опции разрешены средства программируемого дополнения (8.6. Программируемое дополнение). Опция включена по умолчанию.

progcomp_alias
При включённой опции и включённом программируемом дополнении Bash трактует имя команды, не имеющее дополнения, как возможный псевдоним и пытается его преобразовать. Если это псевдоним, Bash пытается выполнить программируемое дополнение, используя слово команды из преобразованного псевдонима.

promptvars
При включённой опции строки приглашения подвергаются преобразованию параметров, подстановке команд, арифметическому преобразованию и удалению кавычек, как описано ниже (6.9. Управление формой приглашения). Опция включена по умолчанию.

restricted_shell
Оболочка устанавливает эту опцию при запуске в ограниченном режиме (6.10. Ограниченная оболочка). Изменить значение нельзя и оно не сбрасывается при выполнении стартовых файлов, что позволяет им обнаруживать ограниченный режим оболочки.

shift_verbose
При включённой опции внутренняя команда используется source использует переменную PATH для поиска каталога, содержащего файл, указанный как аргумент. Опция включена по умолчанию.

xpg_echo
При включённой опции внутренняя команда echo по умолчанию преобразует \-экранирование.

Статус возврата при перечислении параметров имеет значение 0, если все все имена опций (optname) включены и отличен от 0 в противном случае. При установке или сбросе опций статус возврата имеет значение 0, если опция действительна для оболочки.

4.4. Специальные внутренние команды

В силу исторических причин стандарт POSIX выделяет некоторые внутренние команды оболочки как специальные. При работе Bash в режиме POSIX специальные команды отличаются от прочих в 3 аспектах:

1. при поиске специальные команды отыскиваются до функций оболочки;
2. при возврате специальной внутренней функцией отличного от 0 статуса интерактивная оболочка завершается;
3. предшествующие команде операторы назначения продолжают действовать в среде оболочки после выполнения команды.

При работе Bash не в режиме POSIX специальные внутренние команды ведут себя как обычные внутренние команды Bash. Режим Bash POSIX описан в разделе 6.11. Режим POSIX.

Специальные команды POSIX включают break, :, ., continue, eval, exec, exit, export, readonly, return, set, shift, trap, unset.

5. Переменные оболочки

В этом разделе описаны параметры оболочки, используемые Bash. Многим переменным Bash автоматически присваивает используемые по умолчанию значения.

5.1. Переменные Bourne Shell

Bash использует некоторые переменные оболочки так же, как это делает Bourne shell. В некоторых случаях Bash поддерживает для переменных заданные по умолчанию значения.

CDPATH
Разделённый двоеточиями список каталогов, используемых в качестве пути поиска внутренней команды cd.

HOME
Домашний каталог текущего пользователя. применяется по умолчанию внутренней командой cd без аргументов. Значение переменной также используется при преобразовании тильды (3.5.2. Преобразование тильды).

IFS
Список символов, служащих разделителями полей. Применяется при расщеплении слов в преобразованиях.

MAIL
Если этот параметр указывает файл или каталог и переменная MAILPATH не задана, Bash информирует пользователя о приходе почты в указанный файл или каталог в формате Maildir.

MAILPATH
Разделённый двоеточиями список файлов, которые оболочка периодически проверяет на предмет новых почтовых сообщений. Каждый элемент списка может задавать сообщение, выводимое при появлении новой почты в файле, которое указывается через ? после имени файла. При использовании в тексте сообщения символы $_ преобразуются в имя текущего файла почты.

OPTARG
Значение последнего аргумента опции, обработанного внутренней командой getopts.

OPTIND
Индекс последнего аргумента опции, обработанного внутренней командой getopts.

PATH
Разделённый двоеточиями список каталогов, которые оболочка просматривает в поиске команд. Пустое (null) имя в переменной PATH указывает текущий каталог и указывается двумя двоеточиями подряд или двоеточием в начале или в конце списка каталогов.

PS1
Строка основного приглашения (по умолчанию (‘\s-\v\$ ’). Полный список escape-последовательностей, преобразуемых перед выводом PS1, приведён в разделе 6.9. Управление формой приглашения.

PS2
Строка вторичного приглашения (по умолчанию ‘> ’), преобразуемого так же, как PS1 перед выводом.

5.2. Переменные Bash

Эти переменные устанавливаются и используются в Bash, но обычно не имеют значения в других оболочках. Некоторые переменные, используемые Bash описаны в разделе 7.3. Переменные управления заданиями.

BASH

Полный путь, использованный для запуска данного экземпляра Bash.

BASHOPTS

Список разделённых двоеточиями включённых опций оболочки. Каждое слово в этом списке является допустимым аргументом опции -s внутренней команды shopt (4.3.2. Внутренняя команда shopt). Опциями в BASHOPTS являются те, для которых shopt показывает значение on. Если эта переменная присутствовала в окружении при запуске Bash, каждая из опций списка будет включена до считывания каких-либо стартовых файлов. Переменная доступна лишь для чтения.

BASHPID

Выводит идентификатор текущего процесса Bash. Это отличается от $$ в некоторых случаях, таких как субоболочки, которым не требуется повторная инициализация Bash. Назначение BASHPID эффекта не даёт. Если переменная сброшена, она теряет особые свойства даже в случае последующей переустановки.

BASH_ALIASES

Переменная ассоциативного массива, элементы которого соответствуют внутреннему списку псевдонимов, поддерживаемому внутренней командой alias (4.1. Внутренние элементы Bourne Shell). Добавляемые в массив элементы появляются в списке псевдонимов, однако их сброс (unset) в настоящее время не ведёт к удалению из списка. Если переменная сброшена, она теряет особые свойства даже в случае последующей переустановки.

BASH_ARGC

Переменная массива, значения которого указывают число параметров в каждом кадре текущего стека вызовов при выполнении bash. На вершине стека находится число параметров текущей подпрограммы (shell-функции или сценария, выполняемого с . или source). Когда подпрограмма выполняется, число параметров передаётся в BASH_ARGC. Оболочка устанавливает BASH_ARGC только в режиме расширенной отладки (4.3.2. Внутренняя команда shopt). Установка extdebug после запуска оболочки для выполнения сценария или ссылка на эту переменную, когда extdebug не установлена, может давать несогласованный результат.

BASH_ARGV

Переменная массива, содержащая все параметры в текущем стеке вызовов исполнения bash. На вершине стека находится последний параметр последней подпрограммы, а первый параметр начального вызова находится на дне. При выполнении подпрограммы переданные ей параметры вталкиваются в BASH_ARGV. Оболочка устанавливает BASH_ARGV только в режиме расширенной отладки (4.3.2. Внутренняя команда shopt). Установка extdebug после запуска оболочки для выполнения сценария или ссылка на эту переменную, когда extdebug не установлена, может давать несогласованный результат.

BASH_ARGV0

При ссылке на эту переменную она преобразуется в имя оболочки или сценария оболочки (идентично $0, как описано в 3.4.2. Специальные параметры). Назначение BASH_ARGV0 вызывает присвоение этого же значения переменной $0. При сбросе BASH_ARGV0 переменная теряет свои особые свойства даже в случае переустановки.

BASH_CMDS

Переменная ассоциативного массива, элементы которого соответствуют внутренней хэш-таблице команд, поддерживаемой внутренней командой hash (4.1. Внутренние элементы Bourne Shell). Добавляемые в массив элементы появляются в хэш-таблице, однако сброс (unset) элемента массива не ведёт к удалению имени команды из хэш-таблицы. При сбросе BASH_CMDS переменная теряет свои особые свойства даже в случае переустановки.

BASH_COMMAND

Команда, выполняемая или готовящаяся к выполнению в данный момент, если только оболочка не выполняет команду в результате прерывания (trap), когда это будет команда, выполнявшаяся в момент прерывания.

BASH_COMPAT

Значение, используемое для установки уровня совместимости оболочки (4.3.2. Внутренняя команда shopt). Значение может быть десятичным (например, 4.2) или целым (например, 42) числом, соответствующим желаемому уровню совместимости. Если переменная BASH_COMPAT сброшена или установлена пустая строка, для уровня совместимости устанавливается текущая версия. Если в BASH_COMPAT установлено значение, не являющееся одним из действительных уровней совместимости, оболочка выводит сообщение об ошибке и устанавливает для уровня совместимости текущую версию. Действительные уровни совместимости соответствуют опциям совместимости, воспринимаемым внутренней командой shopt (например, compat42 означает, что значения 4.2 и 42 являются действительными). Текущая версия также является пригодным значением.

BASH_ENV

Если эта переменная установлена при вызове Bash для выполнения shell-сценария, она преобразуется и используется как имя стартового файла, считываемого до выполнения сценария (6.2. Стартовые файлы Bash).

BASH_EXECUTION_STRING

Аргумент команды для опции вызова -c.

BASH_LINENO

Переменная массива, элементами которого являются номера строк в исходных файлах, где вызывается каждый соответствующий элемент FUNCNAME. ${BASH_LINENO[$i]} - номер строки в исходном файле (${BASH_SOURCE[$i+1]}), где была вызвана ${FUNCNAME[$i]} (или ${BASH_LINENO[$i-1]} при указании с другой shell-функцией). Для получения номера текущей строки используется LINENO.

BASH_LOADABLES_PATH

Список разделённых двоеточиями каталогов, которые оболочка просматривает в поиске динамически загружаемых внутренних команд, указанных командой enable.

BASH_REMATCH

Переменная массива, элементы которого назначены бинарным оператором =~ условной команды [[(3.2.4.2. Конструкции с условием). Элемент с индексом 0 является частью строки, соответствующей всему регулярному выражению. Элемент с индексом n является частью строки, соответствующей n-му параметризованному субвыражению. Переменная доступна лишь для чтения.

BASH_SOURCE

Переменная массива, элементы которого являются именами исходных файлов, где определена соответствующая shell-функция, названная в массиве FUNCNAME. Функция оболочки ${FUNCNAME[$i]} определена в файле ${BASH_SOURCE[$i]} и вызывается из ${BASH_SOURCE[$i+1]}

BASH_SUBSHELL

Инкрементируется на 1 в каждой субоболочке или среде субоболочки, когда оболочка начинает выполняться в этой среде. Начальное значение равно 0.

BASH_VERSINFO

Доступный лишь для чтения массив (6.7. Массивы), элементы которого содержат данные о версии этого экземпляра Bash.

BASH_VERSINFO[0] Старший номер версии (выпуск - release).
BASH_VERSINFO[1] Младший номер версии (версия).
BASH_VERSINFO[2] Уровень исправлений (patch level).
BASH_VERSINFO[3] Версия сборки.
BASH_VERSINFO[4] Статус выпуска (например, beta1).
BASH_VERSINFO[5] Значение MACHTYPE.

BASH_VERSION

Номер версии текущего экземпляра Bash.

BASH_XTRACEFD

При установке целого значения, соответствующего действительному дескриптору файла, Bash будет записывать трассировку вывода, генерируемую при включении set -x для этого дескриптора. Это позволяет отделить вывод трассировки от диагностики и сообщений об ошибках. Дескриптор файла закрывается при сбросе или установке нового значения BASH_XTRACEFD. Сброс или назначение пустой строки BASH_XTRACEFD вызывают передачу трассировки на стандартный вывод ошибок. Установка BASH_XTRACEFD = 2 (стандартный вывод ошибок) и последующий сброс приводят к закрытию стандартного вывода ошибок.

CHILD_MAX

Задаёт число выходных дочерних состояний, запоминаемых оболочкой. Bash не позволяет снижать это значение меньше описанного ниже минимума, заданного POSIX, а также задан максимум (в настоящее время 8192). Минимальное значение зависит от системы.

COLUMNS

Используется командой select для определения ширины терминала при выводе списков выбора. При включённой опции checkwinsize (4.3.2. Внутренняя команда shopt) и в интерактивной оболочке при получении сигнала SIGWINCH устанавливается автоматически.

COMP_CWORD

Индекс в ${COMP_WORDS} слова, содержащего текущую позицию курсора. Переменная доступна лишь в функциях оболочки, вызываемых средствами программируемого дополнения (8.6. Программируемое дополнение).

COMP_LINE

Текущая строка команды, доступная лишь в функциях оболочки и внешних командах, вызываемых средствами программируемого дополнения (8.6. Программируемое дополнение).

COMP_POINT

Индекс текущей позиции курсора относительно начала текущей команды. Если курсор находится в конце команды, переменная имеет значение ${#COMP_LINE}. Переменная доступна лишь в функциях оболочки, вызываемых средствами программируемого дополнения (8.6. Программируемое дополнение).

COMP_TYPE

Целое число, соответствующее типу предпринятого дополнения, которое вызвало функцию завершения, - TAB для обычного завершения, ? Для списка завершений после последовательных нажатий TAB, ! Для списка вариантов при частичном завершении слова, @ для списка завершений если слово не изменено, % для завершения меню. Переменная доступна лишь в функциях оболочки, вызываемых средствами программируемого дополнения (8.6. Программируемое дополнение).

COMP_KEY

Клавиша (финальная клавиша или комбинация), используемая для вызова текущей функции завершения.

COMP_WORDBREAKS

Набор символов, трактуемых библиотекой Readline как разделители слов при дополнении слова. При сбросе переменной она теряет свои особые свойства даже после переустановки.

COMP_WORDS

Переменная массива, состоящая из отдельных слов текущей строки команды. Строка делится на слова, как её будет расщеплять Readline, с использованием COMP_WORDBREAKS, как описано выше. Переменная доступна лишь в функциях оболочки, вызываемых средствами программируемого дополнения (8.6. Программируемое дополнение).

COMPREPLY

Переменная массива, из которой Bash считывает возможные дополнения, созданные shell-функцией, вызванной средством программируемого дополнения (8.6. Программируемое дополнение). Каждый элемент массива содержит одно возможное дополнение.

COPROC

Переменная массива, созданная для хранения дескрипторов файлов для ввода и вывода безымянных процессов (3.2.5. Копроцессы).

DIRSTACK

Переменная массива, содержащая текущий стек каталогов в порядке их отображения внутренней функцией dirs. Назначение элементов массива может служить для изменения включённых в стек каталогов, но для добавления и удаления должны применяться внутренние команды pushd и popd. Назначение этой переменной не меняет текущего каталога. При сбросе переменной она не восстанавливает особые свойства даже после переустановки.

EMACS

Если Bash находит эту переменную в среде при запуске оболочки со значением t, предполагается работа оболочки в буфере Emacs с запретом редактирования строк.

ENV

Похожа на BASH_ENV и применяется при вызове оболочки в режиме POSIX (6.11. Режим POSIX).

EPOCHREALTIME

При каждой ссылке на параметр он преобразуется в число секунд эпохи Unix, выраженное действительным значением с микросекундной точностью (см. описание библиотеки C). Назначение EPOCHREALTIME игнорируется. При сбросе переменной её свойства не восстанавливаются даже после переустановки.

EPOCHSECONDS

При каждой ссылке на этот параметр он преобразуется в число секунд эпохи Unix (см. описание библиотеки C). Назначение EPOCHSECONDS игнорируется. При сбросе переменной она не восстанавливает особые свойства даже после переустановки.

EUID

Эффективный числовой идентификатор текущего пользователя, доступный лишь для чтения.

EXECIGNORE

Список разделённых двоеточиями шаблонов (3.5.8.1. Сопоставление с шаблоном), определяющий имена файлов при поиске с помощью команды search в PATH. Файлы, имена путей для которых полностью соответствуют одному из шаблонов, не считаются исполняемыми для дополнения и запуска команд через поиск в PATH. Это не влияет на поведение команд [, test и [[. Полные имена в хэш-таблице команд не учитываются EXECIGNORE. Переменную следует использовать для игнорирования файлов общих библиотек, у которых установлен бит выполнения, но они не являются исполняемыми файлами. При сопоставлении учитывается опция оболочки extglob.

FCEDIT

Редактор, используемый по умолчанию внутренней командой fc с опцией -e .

FIGNORE

Разделённый двоеточиями список суффиксов, которые игнорируются при дополнении имён файлов. Имя файла, в котором суффикс совпадает с одним из элементов FIGNORE, исключается из числа совпадений (например, .o:~)

FUNCNAME

Переменная массива, содержащая имена всех shell-функций в стеке исполняемых вызовов. С индексом 0 в массиве размещается имя выполняемой в данный момент функции, нижним элементом (наибольший индекс) является main. Переменная существует только при выполнении shell-функции. Назначение FUNCNAME эффекта не даёт. При сбросе переменной она не восстанавливает особые свойства даже после переустановки.

Переменная может использоваться с BASH_LINENO и BASH_SOURCE, каждый элемент FUNCNAME имеет соответствующие элементы в BASH_LINENO и BASH_SOURCE для описания стека вызовов. Например, ${FUNCNAME[$i]} была вызвана из файла ${BASH_SOURCE[$i+1]}, строка ${BASH_LINENO[$i]}. Внутренняя функция caller выводит текущий стек вызовов, используя эту информацию.

FUNCNEST

При установке положительного числа определяет максимальный уровень вложенности функций. В случае превышения заданного уровня текущая команда прерывается.

GLOBIGNORE

Список разделённых двоеточиями шаблонов, определяющий набор имён файлов, игнорируемых при преобразовании имён. Если имя, соответствующее шаблону, соответствует также одному из шаблонов GLOBIGNORE, оно удаляется из списка совпадений. При сопоставлении учитывается опция оболочки extglob.

GROUPS

Переменная массива, содержащая список групп, в которые входит текущий пользователь. Назначение переменной не даёт эффекта. При сбросе переменной она не восстанавливает особые свойства даже после переустановки.

histchars

До трёх символов, управляющих преобразованием истории, быстрой постановкой и маркировкой (9.3. Преобразование истории). Первым служит символ преобразования истории (обычно !), который указывает начало преобразования. Второй символ указывает «быструю подстановку», когда этот символ является первым в строке (обычно ^). Необязательный третий символ (обычно #) указывает, что остальная часть строки является комментарием и пропускается в процессе подстановки. Это не обязательно ведёт к трактовке оставшейся части строки как комментария синтаксическим анализатором оболочки.

HISTCMD

Номер в истории или индекс списка истории для текущей команды. При сбросе переменной она не восстанавливает особые свойства даже после переустановки.

HISTCONTROL

Список разделённых двоеточиями значений, которые управляют способом хранения в списке истории. Если список включает ignorespace, начинающиеся с пробела строки не включаются в список. Значение ignoredups отключает сохранение строк, совпадающих с предшествующими записями. Значение ignoreboth является сокращением для комбинации ignorespace и ignoredups. Значение erasedups задаёт удаление из списка записи, совпадающей с текущей строкой, перед сохранением этой строки. Все прочие значения игнорируются. Если переменная сброшена (unset) или не содержит действительных значений, все строки, прочитанные анализатором оболочки, сохраняются в списке истории в зависимости от значения HISTIGNORE. Вторая и последующие строки многострочной команды не проверяются и включаются в список истории независимо от значения HISTCONTROL.

HISTFILE

Имя файла для записи списка истории. По умолчанию ~/.bash_history.

HISTFILESIZE

Максимальное число строк в файле истории, при достижении которого наиболее старая запись удаляется. Отсечка файла по этому значению происходит также при выходе из оболочки. Значение 0 задаёт пустой файл. Нечисловые и отрицательные значения блокируют отсечку. Оболочка устанавливает используемое по умолчанию значение HISTSIZE после считывания стартовых файлов.

HISTIGNORE

Список разделённых двоеточиями, служащих для выбора команд, сохраняемых в списке истории. Каждый шаблон привязывается к началу строки и должен соответствовать всей строке (без добавления неявного символа *). Каждый шаблон сопоставляет со строкой после проверок, заданных HISTCONTROL. В дополнение к обычным шаблонам оболочки символу & соответствует предыдущая строка списка истории. Символ & может экранироваться символом \, который исключается перед сопоставлением. Вторая и последующие строки многострочной команды не проверяются и добавляются в список истории независимо от значения HISTIGNORE. При сопоставлении учитываются настройки опции extglob.

HISTIGNORE включает функции переменной HISTCONTROL. Шаблон & идентичен ignoredups, а [ ]* - ignorespace. Комбинация этих шаблонов, разделённых двоеточием, обеспечивает функциональность ignoreboth.

HISTSIZE

Максимальное число команд, запоминаемых в списке истории. При значении 0 команды не сохраняются в списке. Отрицательные значения задают сохранение каждой команды (нет ограничения). Оболочка устанавливает по умолчанию значение 500 после считывания всех стартовых файлов.

HISTTIMEFORMAT

Непустое значение переменной задаёт строку формата для strftime при выводе временных меток, связанных с записями, выводимыми по команде history. При заданной переменной метки записываются в файл истории и будут доступны в другой сессии оболочки. При записи меток применяется символ комментария для обозначения меток.

HOSTFILE
Имя файла в формате /etc/hosts, которые оболочке следует считывать при дополнении имён хостов. Список возможных дополнений имён можно менять во время работы оболочки и при следующем дополнении имени Bash добавит содержимое нового файла к имеющемуся списку. Если переменная пуста или не указывает читаемый файл, Bash пытается использовать /etc/hosts. При сброшенной переменной список имён хостов очищается.

HOSTNAME
Имя текущего хоста.

HOSTTYPE
Строка описания машины, на которой работает Bash.

IGNOREEOF
Управляет реакцией оболочки на получение символа EOF как единственного ввода. Установленное значение указывает число последовательных символов EOF, которые могут быть прочитаны в качестве первых символов строк ввода до завершения работы оболочки. Если переменная содержит нечисловое значение или не имеет значения, используется число 10. Если переменной не существует, символ EOF обозначает конец ввода в оболочку. Переменная действует только для интерактивных оболочек.

INPUTRC
Имя файла инициализации Readline, используемого вместо принятого по умолчанию ~/.inputrc.

INSIDE_EMACS
Если Bash находит эту переменную в среде при запуске оболочки, предполагается работа в shell-буфере Emacs с возможностью отключения редактирования строк в зависимости от значения TERM.

LANG
Служит для определения категории языковой установки, не заданной явно переменной, начинающейся с LC_.

LC_ALL
Переопределяет значение LANG и любой другой переменной LC_, задающей категорию языковой установки.

LC_COLLATE
Определяет порядок сортировки, используемой для результатов преобразования имён файлов, а также поведение преобразования диапазонов, классов эквивалентности и порядка сортировки при преобразовании имён файлов и сопоставлении с шаблонами (3.5.8. Преобразование имён файлов).

LC_CTYPE
Задаёт интерпретацию символов и поведение классов символов при преобразовании имён файлов и сопоставлении с шаблонами (3.5.8. Преобразование имён файлов).

LC_MESSAGES
Задаёт языковую установку, используемую для трансляции строк в двойных кавычках с префиксом $ (3.1.2.5. Зависимая от языка трансляция).

LC_NUMERIC
Задаёт категорию языковой установки, применяемую для форматирования чисел.

LC_TIME
Задаёт категорию языковой установки, применяемую для форматирования даты и времени.

LINENO
Номер строки в выполняемом сценарии или функции оболочки.

LINES
Используется командой select для определения размера строки при выводе списков для выбора. Устанавливается автоматически при включённой опции checkwinsize (4.3.2. Внутренняя команда shopt) или в интерактивной оболочке при получении SIGWINCH.

MACHTYPE
Строка, полностью описывающая тип системы, где выполняется Bash, в формате gnu cpu-company-system.

MAILCHECK
Интервал (в секундах, по умолчанию 60) проверки оболочкой новых сообщений в файлах, заданных переменными MAILPATH и MAIL. Оболочка проверяет почту перед выводом текущего основного приглашения. Если переменная не установлена или не содержит неотрицательного целого числа, проверка почты не происходит.

MAPFILE
Переменная массива, создаваемая для хранения текста, считываемого внутренней командой mapfile, когда имя переменной не задано.

OLDPWD
Предыдущий рабочий каталог, установленный командой cd.

OPTERR
При установке значения 1 Bash будет выводить сообщения об ошибках, создаваемые командой getopts.

OSTYPE
Строка, описывающая операционную систему, в которой работает Bash.

PIPESTATUS
Переменная массива (6.7. Массивы), содержащая список состояний выхода (exit) для процессов в недавнем (последнем) foreground-конвейере (который мог содержать единственную команду).

POSIXLY_CORRECT
Если Bash находит эту переменную в среде при запуске оболочки, оболочка переходит в режим POSIX mode (6.11. Режим POSIX) до считывания стартовых файлов, как будто была задана опция --POSIX. Если переменная установлена при работе оболочки, Bash включает режим POSIX как при выполнении команды set -o POSIX. При переходе оболочки в режим POSIX переменная устанавливается, если она ещё не задана.

PPID
Идентификатор родительского процесса оболочки, доступный лишь для чтения.

PROMPT_COMMAND
Установленное значение считается командой, выполняемой перед выводом основного приглашения ($PS1).

PROMPT_DIRTRIM
Положительное значение считается числом компонент каталога, сохраняемых при преобразовании \w и \W в приглашениях (6.9. Управление формой приглашения). Удалённые символы заменяются многоточием.

PS0
Значение параметра преобразуется подобно PS1 и выводится интерактивной оболочкой после считывания команды и перед её выполнением.

PS3
Значение переменной используется в качестве приглашения для команды select (по умолчанию '#? ').

PS4
Значение параметра (по умолчанию '+ ') преобразуется подобно PS1 и преобразованное значение служит приглашением, выводимым перед эхо-выводом команды, когда установлена опция -x (4.3.1. Внутренняя команда set). Первый символ преобразованного значения может повторяться для указания множества уровней косвенности.

PWD
Текущий рабочий каталог, установленный командой cd.

RANDOM
При каждой ссылке на параметр генерируется случайное число от 0 до 32767, которое служит «затравкой» (seed) для генератора случайных чисел.

READLINE_LINE
Содержимое буфера строки Readline для использования с bind -x (4.2. Внутренние команды bash).

READLINE_POINT
Положение точки вставки в буфер строки Readline для использования с bind -x (4.2. Внутренние команды bash).

REPLY
Переменная, используемая по умолчанию внутренней командой read.

SECONDS
Преобразуется в число секунд с момент запуска оболочки. Назначение переменной меняет прошлое значение на указанное и далее будет выводиться сумма заданного значения с числом секунд после установки.

SHELL
Полный путь к оболочке. Если при запуске оболочки переменная не задана, Bash назначает ей полное имя файла текущей пользовательской оболочки login.

SHELLOPTS
Список разделённых двоеточиями включённых опций оболочки. Каждое слово в списке является действительным аргументом опции -o внутренней команды set (4.3.1. Внутренняя команда set). Опции из списка SHELLOPTS выводятся с пометкой on по команде set -o. Если эта переменная была установлена в среде при запуске Bash, каждая из опций списка включается до прочтения стартовых файлов. Переменная доступна лишь для чтения.

SHLVL
Инкрементируется при запуске каждого нового экземпляра Bash для учёта глубины вложенности оболочки.

TIMEFORMAT
Значение этого параметра служит строкой формата при выводе времени для конвейеров с зарезервированным словом time. Символ % указывает escape-последовательность, преобразуемую в значение времени или иную информацию, как показано ниже (в квадратных скобках указаны необязательные параметры).

`%%`	Литерал %.
`%[p][l]R`	Прошедшее время в секундах.
`%[p][l]U`	Затраченное процессором время в пользовательском режиме.
`%[p][l]S`	Затраченное процессором время в системном режиме.
`%P`	Процент загрузки CPU, рассчитанный как (%U + %S) / %R.

Параметр p указывает размер дробной части. Значение 0 задаёт вывод только целой части. Можно вывести до 3 знаков после точки и при задании большего числа будет использовано 3. Необязательный параметр l задаёт «длинный» формат с указанием минут в виде MM mSS.FFs. Значение p управляет дробной частью. Если переменная не задана, Bash использует значение $'\nreal\t%3lR\nuser\t%3lU\nsys\t%3lS'. При пустой переменной данные о времени не выводятся. При выводе времени в конце строки формата добавляется символ новой строки.

TMOUT
Положительное значение трактуется как принятый по умолчанию тайм-аут внутренней команды read (4.2. Внутренние команды bash). Команда select (3.2.4.2. Конструкции с условием) прерывается, если не получен ввод в течение TMOUT секунд при ожидании данных с терминала. В интерактивной оболочке значение считается числом секунд ожидания ввода строки после вывода основного приглашения. Если полная строка не введена за это время, работа Bash прерывается.

TMPDIR
Установленное значение Bash использует как имя каталога для создания временных файлов оболочки.

UID
Числовой идентификатор текущего реального пользователя, доступный лишь для чтения.

6. Свойства Bash

В этой главе описаны свойства, присущие только Bash.

6.1. Вызов Bash

```
bash [long-opt] [-ir] [-abefhkmnptuvxdBCDHP] [-o option]
        [-O shopt_option] [argument ...]
bash [long-opt] [-abefhkmnptuvxdBCDHP] [-o option]
        [-O shopt_option] -c string [argument ...]
bash [long-opt] -s [-abefhkmnptuvxdBCDHP] [-o option]
        [-O shopt_option] [argument ...]
```

Все односимвольные опции внутренней команды set (4.3.1. Внутренняя команда set) могут служить опциями при вызове оболочки. Кроме того, может применяться часть многосимвольных опций, которые должны указываться в командной строке до односимвольных опций.

--debugger
Задаёт профиль отладчика, запускаемого перед вызовом оболочки, включая режим расширенной отладки (см. описание опции extdebug внутренней команды shopt в параграфе 4.3.2. Внутренняя команда shopt).

--dump-po-strings
Выводит список всех заключённых в двойные кавычки строк с префиксом $ на стандартное устройство вывода в формате gnu gettext PO[1]. Эквивалент опции -D для формата вывода.

[1]Portable object - переносимый объект.

--dump-strings
Эквивалент -D.
--help
Выдаёт сообщение об использовании на стандартный вывод и завершает работу.
--init-file filename
--rcfile filename
Выполняет команды из filename (вместо ~/.bashrc) для интерактивной оболочки.
--login
Эквивалент -l.
--noediting
Отключает использование библиотеки gnu Readline (8. Редактирование командной строки) для чтения строк команд в интерактивной оболочке.
--noprofile
Отменяет загрузку системного стартового файла /etc/profile и всех персональных файлов инициализации (~/.bash_profile, ~/.bash_login, ~/.profile) при вызове Bash в качестве оболочки для регистрации в системе (login).
--norc
Отменяет чтение файла инициализации ~/.bashrc интерактивной оболочки. Принято по умолчанию для вызова sh.
--POSIX
Меняет поведение Bash в тех аспектах, где принятые по умолчанию операции отличаются от стандарта POSIX. Это сделано для того, чтобы оболочка вела себя как строгое надмножество стандарта (6.11. Режим POSIX).
--restricted
Делает оболочку ограниченной (6.10. Ограниченная оболочка).
--verbose
Эквивалент -v. Выводит прочитанные строки ввода оболочки.
--version
Выводит информацию о версии данного экземпляра Bash на стандартный вывод и завершает работу.
Имеются односимвольные опции, которые могут быть указаны при вызове, но не доступны внутренней команде set.

-c
Считывание и выполнение команд с первого аргумента, не являющегося опцией, и завершение работы. При наличии аргументов после строки команды первый из них назначается переменной $0, а остальные - позиционным параметрам. Назначение $0 задаёт имя оболочки, используемое в предупреждениях и сообщениях об ошибках.
-i
Переводит оболочку в интерактивный режим (6.3. Интерактивные оболочки).
-l
Заставляет оболочку действовать как при прямом вызове для входа в систему (login). Для интерактивной оболочки это эквивалентно запуску login-оболочки с помощью exec -l bash, в остальных случаях будут выполняться стартовые файлы login-оболочки. Команды exec bash -l и exec bash --login будут менять текущую оболочку на Bash login (см. 6.2. Стартовые файлы Bash).
-r
Делает оболочку ограниченной (6.10. Ограниченная оболочка).
-s
При наличии этой опции или отсутствии аргументов после обработки опций команда считывается со стандартного ввода, что позволяет задать позиционные параметры для интерактивной оболочки или при вводе из конвейера.
-D
Выводит список всей заключённых в двойные кавычки строк с префиксом $ на стандартное устройство вывода. К этим строкам применяются языковые преобразования, когда текущая настройка языка отличается от C и POSIX (3.1.2.5. Зависимая от языка трансляция). Подразумевается опция -n, команды не выполняются.
[-+]O [shopt_option]
Опция shopt является одной из опций оболочки, воспринимаемых командной shopt (4.3.2. Внутренняя команда shopt). При наличии такой опции -O устанавливает значение, +O отменяет. Если опция shopt не задана, выводятся имена и значения опций оболочки, воспринимаемые shopt, на стандартное устройство вывода. При использовании опции вызова +O формат вывода пригоден для использования в качестве ввода.
--
Символы -- указывают завершение опций и отключают дальнейшую их обработку. Все аргументы после -- считаются именами файлов и аргументами.
Оболочкой входа (login) является оболочка с первым символом аргумента «-» или вызванная с опцией --login.

Интерактивной является оболочка, запущенная без аргументов, не являющихся опциями (если не задана опция -s), без опции -c и соединённая с терминалами по входу и выходу (как определено isatty(3)), или запущенная с опцией -i (см. 6.3. Интерактивные оболочки).

Если после обработки опций остаются аргументы и не было задано ни одной из опций -c и -s, первый аргумент считается именем файла с командами оболочки (3.8. Сценарии оболочки). При таком вызове Bash параметру $0 назначается имя этого файла, а остальные аргументы передаются в позиционные параметры. Bash читает и выполняет команды из файла, а затем завершает работу. Статусом завершения Bash будет статус выхода последней выполненной в сценарии команды. При отсутствии команд статус выхода будет 0.

6.2. Стартовые файлы Bash

В этом разделе описано использование стартовых файлов в Bash. Если файл имеется, но не может быть прочитан, Bash сообщает об ошибке. Символ ~ преобразуется, как указано в параграфе 3.5.2. Преобразование тильды.

Интерактивные оболочки описаны в параграфе 6.3. Интерактивные оболочки.

Вызов как интерактивной входной оболочки или с опцией --login

При вызове Bash как интерактивной оболочки входа в систему (login) или неинтерактивной оболочки с опцией --login сначала считываются и выполняются команды из файла /etc/profile, если он имеется. После этого отыскиваются в

указанном порядке файлы ~/.bash_profile, ~/.bash_login, ~/.profile и первый найденный и читаемый файл считывается, а команды из него выполняются. Для блокировки такого поведения можно задать при запуске опцию --noprofile.

При завершении интерактивной login-оболочки или при выполнении в неинтерактивной login-оболочке внутренней команды exit интерпретатор Bash считывает файл ~/.bash_logout (при наличии) и выполняет команды из него.

Вызов как неинтерактивной и невходной оболочки

При запуске интерактивной оболочки, не являющейся входом в систему (login) Bash считывает файл ~/.bashrc (при его наличии) и выполняет команды из него. Это поведение можно блокировать опцией --norc, а опция --rcfile заставляет Bash читать и выполнять команды из файла ~/.bashrc.

Обычно файл ~/.bash_profile содержит строку

```
if [ -f ~/.bashrc ]; then . ~/.bashrc; fi
```

после (или до) связанной со входом в систему инициализации.

Вызов как неинтерактивной оболочки

При запуске Bash в неинтерактивном режиме (например, для выполнения сценария) отыскивается переменная среды BASH_ENV, её значение преобразуется и результат используется в качестве имени файла для считывания и выполнения. Bash ведёт себя как при выполнении команды

```
if [ -n "$BASH_ENV" ]; then . "$BASH_ENV"; fi
```

При этом значение переменной PATH не применяется для поиска файла. Как отмечено выше, при вызове неинтерактивной оболочки с опцией --login интерпретатор Bash пытается читать и выполнять команды из стартовых файлов login-оболочки.

Вызов с именем sh

При вызове Bash с именем sh предпринимается попытка поведения при старте в соответствии с исторической оболочкой sh и стандартом POSIX. При запуске в качестве интерактивной оболочки входа (login) или неинтерактивной оболочки с опцией --login сначала предпринимается попытка выполнить команды из файлов /etc/profile и ~/.profile в указанном порядке. Это можно блокировать опцией --noprofile.

При вызове в качестве интерактивной оболочки с именем sh интерпретатор Bash отыскивает переменную ENV, преобразует её (при наличии) и использует результат как имя файла для считывания и выполнения. Поскольку при вызове по имени sh оболочка не пытается считывать и выполнять команды из других стартовых файлов, опция --rcfile не будет работать. При вызове неинтерактивной оболочки по имени sh не предпринимается попыток чтения и выполнения других стартовых файлов. При вызове по имени sh интерпретатор Bash переходит в режим POSIX после чтения стартовых файлов.

Вызов в режиме POSIX

При запуске Bash в режиме POSIX (с опцией --POSIX) применяются стартовые файлы в соответствии со стандартом POSIX. В этом режиме интерактивная оболочка преобразует переменную ENV, считывая и выполняя команды из файла, имя которого получено в результате преобразования. Другие стартовые файлы не считываются.

Вызов удаленным демоном shell

Bash пытается определить запуск со стандартным вводом, подключённым к сетевому соединению, как при выполнении демоном удалённой оболочки (обычно rshd) или sshd. Если Bash идентифицирует такой режим, считываются и выполняются команды из файла ~/.bashrc при его наличии и доступности для чтения (это не будет происходить при вызове оболочки по имени sh). Можно блокировать считывание и выполнение команд опцией --norc или указать --rcfile для чтения и выполнения другого файла, но демоны rshd и sshd обычно не используют и не поддерживают эти опции.

Вызов с неэквивалентными эффективными и реальными uid/gid

При запуске Bash с эффективным идентификатором пользователя (группы), не совпадающим с реальным, и без опции -p стартовые файлы не будут использоваться, функции оболочки не наследуются из среды, переменные SHELLOPTS, BASHOPTS, CDPATH и GLOBIGNORE (при их наличии в среде) игнорируются, а эффективный идентификатор пользователя устанавливается в соответствии с реальным. При наличии опции -p эффективный идентификатор пользователя не будет меняться.

6.3. Интерактивные оболочки

6.3.1. Что такое интерактивная оболочка?

Интерактивной считается оболочка, запущенная без аргументов, не являющихся опциями (если не задана опция -s), без опции -c, со вводом и выводом (ошибок), подключёнными к терминалу (в соответствии с isatty(3)), или запущенная с опцией -i. Интерактивные оболочки обычно применяют для ввода и вывода терминал пользователя. Можно использовать вызов с опцией -s для задания позиционных параметров при старте оболочки.

6.3.2. Является ли оболочка интерактивной?

Для определения в сценарии интерактивного режима Bash можно проверять значения специального параметра ‘-’, который в интерактивном режиме содержит i. Например,

```
case "$-" in
*i*) echo This shell is interactive ;;
*) echo This shell is not interactive ;;
esac
```

Можно также проверять переменную PS1, которая устанавливается только в интерактивной оболочке. Например,

```
if [ -z "$PS1" ]; then
        echo This shell is not interactive
else
```

```
        echo This shell is interactive
    fi
```

6.3.3. Поведение интерактивной оболочки

При работе оболочки в интерактивном режиме меняется ряд аспектов поведения, описанных ниже.

1. Стартовые файлы считываются и выполняются как описано в параграфе 6.2. Стартовые файлы Bash.
2. Управление заданиями по умолчанию включено (7. Управление заданиями) и Bash игнорирует сигналы управления заданиями с клавиатуры (SIGTTIN, SIGTTOU, SIGTSTP).
3. Bash преобразует и выводит переменную PS1 перед считыванием первой строки команды, а также преобразует и выводит PS2 перед считыванием второй и последующих строк многострочной команды. Bash преобразует и выводит переменную PS0 после чтения команды, но до её выполнения. Полное описание escape-последовательностей строк приглашения приведено в разделе 6.9. Управление формой приглашения.
4. Bash использует значение переменной PROMPT_COMMAND как команду, выполняемую перед выводом основного приглашения $PS1 (5.2. Переменные Bash).
5. Для чтения команд с пользовательского терминала служит Readline (8. Редактирование командной строки).
6. Bash проверяет значение опции ignoreeof для установки -o вместо незамедлительного выхода при получении символа EOF со стандартного ввода во время чтения команд (4.3.1. Внутренняя команда set).
7. История команд (9.1. Средства Bash History) и её преобразование (9.3. Преобразование истории) включены по умолчанию. Bash сохраняет историю команд в файле, указанном $HISTFILE, при завершении работы со включённой историей.
8. Преобразование псевдонимов (6.6. Псевдонимы) выполняется по умолчанию.
9. При отсутствии ловушек (trap) Bash игнорирует SIGTERM (3.7.6. Сигналы).
10. При отсутствии ловушек сигнал SIGINT считывается и обрабатывается (3.7.6. Сигналы), прерывая некоторые внутренние функции оболочки.
11. Интерактивная оболочка входа (login) передаёт сигнал SIGHUP всем заданиям при выходе, если включена опция huponexit (3.7.6. Сигналы).
12. Опция вызова -n игнорируется, а set -n не оказывает влияния (4.3.1. Внутренняя команда set).
13. Bash периодически проверяет почту в соответствии со значениями переменных MAIL, MAILPATH и MAILCHECK (5.2. Переменные Bash).
14. Ошибки прерывания, обусловленные ссылками на несвязанные переменные оболочки после включения set -u не будут приводить к завершению работы (4.3.1. Внутренняя команда set).
15. Оболочка не будет завершать работу при ошибках преобразования, вызванных отменой или пустым значением var в ${var:?word} (3.5.3. Преобразование параметров оболочки).
16. Ошибки перенаправления для внутренних команд оболочки не будут приводить к завершению работы.
17. В режиме POSIX специальная встроенная функция возврата статуса ошибок не будет вызывать прекращение работы (6.11. Режим POSIX).
18. Отказ exec не будет вызывать прекращение работы (4.1. Внутренние элементы Bourne Shell).
19. Синтаксические ошибки при анализе не будут вызывать прекращение работы.
20. Простая корректировка ввода каталогов для внутренней команды cd включена по умолчанию (см. описание опции cdspell в параграфе 4.3.2. Внутренняя команда shopt).
21. Оболочка проверяет значение переменной TMOUT и будет завершать работу, если команда не считана в течение заданного числа секунд после вывода $PS1 (5.2. Переменные Bash).

6.4. Условные выражения bash

Условные выражения используются составной командой [[, а также внутренними командами test и [, поведение которых зависит от числа аргументов. Выражения могут быть унарными или бинарными и состоят из примитивов. Унарные выражения часто применяются для проверки статуса файлов. Имеются операторы сравнения строк и числовых выражений. Bash применяет особую обработку для некоторых имён файлов в выражениях. Если операционная система, где работает Bash, поддерживает такие файлы, Bash будет использовать их. В остальных случаях применяется внутренняя эмуляция - если аргумент file имеет форму /dev/fd/N, проверяется файл с дескриптором N, для /dev/stdin, /dev/stdout, /dev/stderr используются дескрипторы 0, 1, 2.

При использовании с [[операторы < и > работают на основе лексикографической сортировки для установленного языка (locale). Команда test использует сортировку ASCII.

Если явно не указано иное, работающие с файлами примитивы следуют по символьным ссылкам и действуют на целевой файл, а не на саму ссылку.

-a file	True, если file существует.
-b file	True, если file существует и является специальным блочным файлом.
-c file	True, если file существует и является специальным символьным файлом.
-d file	True, если file существует и является каталогом.
-e file	True, если file существует.
-f file	True, если file существует и является обычным файлом.
-g file	True, если file существует и для него установлен бит set-group-id.

`-h file`	True, если file существует и является символьной ссылкой.
`-k file`	True, если file существует и для него установлен бит sticky.
`-p file`	True, если file существует и является именованным каналом (FIFO).
`-r file`	True, если file существует и доступен для чтения.
`-s file`	True, если file существует и его размер больше 0.
`-t fd`	True, если дескриптор fd открыт и указывает на терминал.
`-u file`	True, если file существует и для него установлен бит set-user-id.
`-w file`	True, если file существует и доступен для записи.
`-x file`	True, если file существует и является исполняемым.
`-G file`	True, если file существует и принадлежит действующей группе.
`-L file`	True, если file существует и является символьной ссылкой.
`-N file`	True, если file существует и был изменён с момента последнего чтения.
`-O file`	True, если file существует и принадлежит действующему пользователю.
`-S file`	True, если file существует и является сокетом.
`file1 -ef file2`	True, если file1 и file2 указывают одно устройство и номер inode.
`file1 -nt file2`	True, если file1 новее (по времени изменения) чем file2 или file1 существует, а file2 - нет.
`file1 -ot file2`	True, если file1 старше чем file2 или file2 существует, а file1 - нет.
`-o optname`	True, если включена опция оболочки optname (4.3.1. Внутренняя команда set).
`-v varname`	True, если переменная оболочки varname установлена (имеет значение).
`-R varname`	True, если переменная оболочки varname установлена и ссылается на имя.
`-z string`	True, если размер string равен 0.
`-n string`	True, если размер string не равен 0.
`string`	
`string1 == string2` `string1 = string2`	True, если строки совпадают. При использовании с командой [[выполняется сопоставление с шаблоном, как описано в параграфе 3.2.4.2. Конструкции с условием. Оператор = следует использовать с командой test для соответствия POSIX.
`string1 != string2`	True, если строки не совпадают.
`string1 < string2`	True, если string1 лексикографически предшествует строке string2.
`string1 > string2`	True, если string1 лексикографически следует после строки string2.
`arg1 OP arg2`	OP может быть -eq, -ne, -lt, -le, -gt или -ge. Операторы возвращают true, если arg1 равен, не равен, меньше, не больше, больше, не меньше arg2. Arg1 и arg2 могут быть положительными или отрицательными целыми числами. При использовании с командой [[аргументы arg1 и arg2 считаются арифметическими выражениями (6.5. Арифметика командного процессора).

6.5. Арифметика командного процессора

Командный процессор позволяет вычислять арифметические выражения как один из видов преобразования или с использованием композитной команды ((, внутренней команды let или опции -i для внутренней команды declare.

Вычисление производится с целыми числами фиксированного размера без проверки переполнения, хотя деление на 0 отслеживается и помечается как ошибка. Операторы, порядок их применения, ассоциативность и значения совпадают с принятыми в языке C. Ниже приведён список операторов, сгруппированных по уровням предпочтения (порядку применения) в порядке снижения.

`id++ id--`	Пост-инкрементирование и пост-декрементирование переменной
`++id --id`	Предварительное инкрементирование и декрементирование переменной
`+ -`	Унарный + и -
`! ~`	Логическое и побитовое отрицание
`**`	Возведение в степень
`*/%`	Умножение, деление, остаток
`+ -`	Сложение, вычитание
`<< >>`	Побитовый сдвиг влево и вправо
`<= >= < >`	Сравнение
`== !=`	Равенство и неравенство
`&`	побитовая операция AND
`^`	Побитовая операция XOR (исключительное ИЛИ)
`|`	Побитовая операция OR (ИЛИ)
`&&`	Логическая операция AND (И)
`||`	Логическая операция OR (ИЛИ)
`expr ? expr : expr`	Условный оператор
`= *= /= %= += -= <<= >>= &= ^= |=`	Назначение
`expr1 , expr2`	Запятая

Переменные оболочки могут служить операндами, преобразование параметров выполняется до расчёта выражения. В выражении переменные оболочки могут задаваться именами без применения синтаксиса преобразования параметров. Переменная оболочки, имеющая значение null или не установленная, преобразуется в 0 при указании именем без использования синтаксиса преобразования. Значение переменной вычисляется как арифметическое выражения, когда на неё ссылаются или присвоено значение переменной, объявленной с атрибутом integer, с помощью declare -i. Значение null трактуется как 0. Переменная не обязана иметь атрибут integer для использования в выражении.

Константы, начинающиеся с 0, считаются восьмеричными значениями, а 0x или 0X в начале указывает шестнадцатеричное значение. В остальных случаях числа имеют форму [base#]n, где необязательный элемент base является десятичным числом от 2 до 64, представляющим основание, а n указывает значение по этому основанию. Если base# отсутствует, предполагается основание 10. При указании n цифры больше 9 указываются заглавными буквами, буквами нижнего регистра, а также символами @ и _ в указанном порядке. Если основание не превышает 36, в качестве цифр 10 - 35 могут применяться буквы в верхнем и нижнем регистре без его учёта.

Операторы при расчёте применяются в соответствии с порядком действий, субвыражения в скобках вычисляются заранее и могут менять порядок действий.

6.6. Псевдонимы

Псевдонимы позволяют заменить строку словом для использования в качестве первого слова простой команды. Оболочка поддерживает список псевдонимов, которые можно задавать и отменять с помощью внутренних команд alias и unalias.

Первое слово простой команды, если оно не заключено в кавычки, проверяется на предмет наличия псевдонима. Если псевдоним найден, слово заменяется текстом соответствующей строки. Символы /, $, ‘, = и все метасимволы оболочки, а также символы кавычек, указанные выше, не могут включаться в имена псевдонимов. Текст, заменяющий псевдоним, может содержать любой действительный ввод оболочки, включая метасимволы . Первое слово текста проверяется на предмет наличия псевдонима, но идентичное преобразуемому псевдониму слово второй раз не преобразуется. Это значит, например, что в псевдониме ls для текста "ls -F" Bash не будет пытаться рекурсивно подставлять замену ls. Если последний символ псевдонима является пробельным (blank), следующее слово команды также проверяется на предмет наличия и преобразования псевдонима.

Псевдонимы создаются и просматриваются с помощью команды alias, а удаляются командой unalias.

Механизма использования аргументов в тексте подстановки (как в csh) не поддерживается. Если аргументы нужны, следует использовать не псевдоним, а функцию оболочки (3.3. Функции оболочки).

Псевдонимы не преобразуются в неинтерактивной оболочке, если для неё не установлена опция expand_aliases с помощью внутренней команды shopt (4.3.2. Внутренняя команда shopt).

Правила определения и использования псевдонимов несколько запутаны. Bash всегда читает по меньшей мере одну полную строку ввода и все строки, образующие композитную команду, перед выполнением любой команды в строке или составной команды. Псевдонимы преобразуются при считывании команды, а не её выполнении. Поэтому определение псевдонима в одной строке с командой не будет работать до ввода следующей строки. Команда, следующая за определением псевдонима, в той же строке не сможет использовать псевдоним. Это также может вызывать проблемы при выполнении функций. Псевдонимы преобразуются при считывании определения функции, а не при её выполнении, поскольку определение функции само является командой. В результате псевдонимы, определённые в функции, недоступны, пока функция не будет выполнена. В целях безопасности следует помещать определения псевдонимов в отдельную строку и не применять псевдонимы в составных командах.

Практически во всех случаях функции оболочки предпочтительней, нежели псевдонимы.

6.7. Массивы

Bash поддерживает одномерные массивы с индексом и ассоциативные массивы. Внутренний оператор declare служит для явного объявления массива. Размер массивов не ограничен и не задаётся требований к непрерывности индексов. Индексированные массивы указываются с использованием целочисленных индексов, которые могут быть арифметическими выражениями (6.5. Арифметика командного процессора). Отсчёт индексов начинается с 0. Если явно не указано иное, индексы массива должны быть неотрицательными целыми числами. Ассоциативные массивы используют произвольные строки ключей.

Индексированный массив создаётся автоматически, если любая переменная задана с использованием синтаксиса

```
name[subscript]=value
```

Элемент subscript считается арифметическим выражением, которое должно преобразовываться в число. Для явного объявления массива служит форма, приведённая ниже.

```
declare -a name
```

Приемлем также синтаксис

```
declare -a name[subscript]
```

где значение subscript игнорируется.

Ассоциативные массивы создаются в форме

```
declare -A name.
```

Атрибуты могут быть заданы для переменной массива с использованием внутренней команды declare или readonly. Каждый атрибут применяется ко всем элементам массива.

Значения массивов указываются с помощью компонентного оператора присваивания в форме

```
name=(value1 value2 ... )
```

где каждый аргумент value имеет форму [subscript]=string. Индексированные массивы не требуют ничего, кроме строки. При задании значений таких массивов с указанием необязательного индекса значение присваивается с этим индексом, иначе назначается элемент, следующий за последним с увеличением индекса на 1. Индексы начинаются с 0.

При задании значения в ассоциативном массиве требуется указание индекса. Такой синтаксис принимается также внутренней командой declare. Отдельные элементы массива могут задаваться в форме name[subscript]=value.

Если при назначении в индексированный массив имя задано отрицательным индексом, значение интерпретируется относительно максимального индекса (вычитается), т. е. отрицательные индексы отсчитываются от конца массива и -1 указывает последний элемент.

Любой элемент массива можно указать в форме ${name[subscript]}. Скобки нужны для предотвращения конфликтов с операторами преобразования имён файлов. Если индекс subscript имеет значение @ или *, слово преобразуется во все элементы массива name. Эти индексы различаются лишь для слов в двойных кавычках. В этом случае ${name[*]} преобразуется в одно слово с разделением элементов массива первым символом переменной IFS, а ${name[@]} преобразуется в отдельные слова для каждого элемента массива. Если в массиве нет элементов, ${name[@]} преобразуется в пустое значение. Если происходит преобразование двойных кавычек внутри слова, первый преобразованный символ объединяется с началом исходного слова, а последний - с концом этого слова. Это аналогично преобразованию специальных символов @ и *. ${#name[subscript]} преобразуется в размер ${name[subscript]}. Если subscript имеет значение @ или *, результатом преобразования будет число элементов

массива. Если индекс массива преобразуется в число меньше 0, он интерпретируется относительно максимального индекса в массиве (т. е. -1 указывает последний элемент массива).

Ссылка на переменную массива без указания индекса эквивалентна ссылке с индексом 0. Любая ссылка на переменную с действительным индексом корректна и bash при необходимости будет создавать массив. Переменная массива считается установленной, если назначено значение индекса. Пустая строка является допустимым значением.

Можно получить ключи (индексы) массива, а также значения. ${!name[@]} и ${!name[*]} преобразуются в индексы, назначенные в переменной массива name. Трактовка значений в двойных кавычках похожа на преобразование специальных символов @ и * внутри двойных кавычек.

Для уничтожения массивов служит внутренняя команда unset. Для уничтожения элемента в индексном массиве служит unset name[subscript]. Интерпретация отрицательных значений индекса описана выше. Уничтожение последнего (единственного) элемента в переменной массива не уничтожает саму переменную, а команда unset name, где name указывает имя массива, удаляет весь массив. Индекс * или @ также позволяет удалить массив целиком.

При указании имени переменной с индексом в качестве аргумента команды (например, unset) без использования синтаксиса преобразования слов, описанного выше, для аргумента будет выполняться принятое в оболочке преобразование имён файлов. Если такое преобразование не желательно, аргумент следует поместить в кавычки.

Внутренние команды declare, local, readonly воспринимают опцию -a для указания индексированных массивов и -A для ассоциативных массивов. При наличии обеих опций приоритет отдаётся -A. Внутренняя команда read принимает опцию -a для назначения списка слов со стандартного ввода элементам массива. Внутренние команды set и declare выводят значения массивов в форме, пригодной для ввода.

6.8. Стек каталогов

Стек каталогов представляет собой список недавно посещённых каталогов. Внутренняя команда pushd добавляет каталоги в стек при изменении текущего каталога, а внутренняя команда popd удаляет каталоги из стека и меняет текущий каталог при удалении каталога. Внутренняя команда dirs выводит содержимое стека каталогов, указывая текущий каталог на вершине стека. Содержимое стека каталогов доступно также в переменной DIRSTACK.

6.8.1. Внутренние команды стека каталогов

dirs

```
dirs [-clpv] [+N | -N]
```

Выводит текущий список запомненных каталогов. Каталоги добавляются в список командой pushd и удаляются из него командой popd. Текущий каталог всегда является первым в стеке.

-c Очищает стек каталогов, удаляя из него все элементы.
-l Выводит в списке каталогов полные пути. По умолчанию домашний каталог заменяется тильдой (~).
-p Вывод стека каталогов по одной записи в строке.
-v Вывод стека каталогов по одной записи в строке с номером записи в начале каждой строки.
+N Выводит N-й каталог из списка (отсчёт с 0, слева направо).
-N Выводит N-й каталог из списка (отсчёт с 0, справа налево).

popd

```
popd [-n] [+N | -N]
```

При отсутствии аргументов popd удаляет верхний каталог из стека и выполняет команду cd для нового верхнего каталога. Элементы нумеруются с 0, начиная с первого каталога в списке dirs, т. е. popd эквивалентно popd +0.

-n Подавляет обычную смену каталога при удалении каталогов из стека.
+N Удаляет N-й каталог (слева направо в списке, выводимом dirs, считая с 0).
-N Удаляет N-й каталог (справа налево в списке, выводимом dirs, считая с 0).

pushd

```
pushd [-n] [+N | -N | dir]
```

Сохраняет текущий каталог на вершине стека, затем выполняет команду cd dir. При отсутствии аргументов меняет местами два верхних каталога в стеке и делает новый каталог текущим.

-n Подавляет обычную смену каталога при ротации или добавлении каталогов в стек.
+N Помещает N-й каталог (слева направо в списке, выводимом dirs, считая с 0) на вершину путём ротации стека.
-N Помещает N-й каталог (справа налево в списке, выводимом dirs, считая с 0) на вершину путём ротации стека.

dir

Делает dir вершиной стека и новым текущим каталогом, как при указании его в качестве аргумента команды cd.

6.9. Управление формой приглашения

Значение переменной PROMPT_COMMAND проверяется непосредственно перед каждым выводом основного приглашения Bash. Если переменная установлена и значение отличается от null, оно выводится, как будто было набрано в командной строке. Символы, имеющие специальное значение в переменных приглашения PS0, PS1, PS2 и PS4 описаны ниже.

\a	«Звонок»
\d	Дата в формате «день недели, месяц, число(например, Tue May 26).
\D{format}	Значение format передаётся strftime(3) и результат помещается в строку приглашения. Пустое значение format задаёт представление времени в соответствии с языком (разделителями служат пробелы).
\e	Символ экранирования (escape).
\h	Имя хоста до первой точки.
\H	Полное имя хоста.
\j	Число заданий, обслуживаемых в данный момент оболочкой.
\l	Базовое имя терминального устройства оболочки.
\n	Новая строка.
\r	Возврат каретки.
\s	Имя оболочки, базовое имя $0 (часть после финального символа /).
\t	Время в 24-часовом формате HH:MM:SS.

\T	Время в 12-часовом формате HH:MM:SS.
\@	Время в 12-часовом формате am/pm.
\A	Время в 24-часовом формате HH:MM.
\u	Имя текущего пользователя.
\v	Версия Bash (например, 2.00)
\V	Выпуск Bash, версия + patchlevel (например, 2.00.0)
\w	Текущий рабочий каталог с заменой значения $HOME тильдой (используется переменная $PROMPT_DIRTRIM).
\W	Базовое имя $PWD с заменой значения $HOME тильдой.
\!	Номер текущей команды в истории.
\#	Номер этой команды.
\$	Задаёт символ # для пользователя с uid =0, и $ для остальных.
\nnn	Символ с восьмеричным кодом ASCII nnn.
****	\.
\[	Начало последовательности непечатаемых символов. Может использоваться для задания символов управления терминалом в приглашении.
\]	Конец последовательности непечатаемых символов.

Номер команды и номер в истории обычно различаются - номер в истории указывает позицию в списке истории, который может включать восстановленные из файла истории команды (9.1. Средства Bash History), а номер команды указывает позицию в последовательности команд, выполненных в данном сеансе оболочки.

После декодирования строки выполняется преобразование параметров, подстановка команд, арифметические преобразования и удаление кавычек в соответствии со значением опции promptvars (4.3.2. Внутренняя команда shopt).

6.10. Ограниченная оболочка

Если Bash запускается по имени rbash или при вызове задана опция --restricted или -r, оболочка становится ограниченной для организации среды, более управляемой по сравнению со стандартной оболочкой. Поведение ограниченной оболочки отличается от bash в перечисленных ниже аспектах, которые запрещены или не выполняются.

- Смена каталогов внутренней командой cd.
- Установка и отмена значений переменных SHELL, PATH, ENV, BASH_ENV.
- Ввод команд, имена которых содержат символ /.
- Указание в качестве аргументов внутренней команды . имён файлов с символом /.
- Указание имён файлов с символом / в качестве аргументов опции -p внутренней команды hash.
- Импорт определений функций из среды оболочки при старте.
- Разбор значения SHELLOPTS из среды оболочки при старте.
- Перенаправление вывода с помощью операторов >, >|, <>, >&, &> и >>.
- Использование внутренней функции exec для замены оболочки другой командой.
- Добавление или удаление внутренних команд с помощью опций -f и -d внутренней команды enable.
- Использование внутренней команды enable для включения отключённых внутренних функций оболочки.
- Задание опции -p для внутренней команды command.
- Отключение ограниченного режима с помощью set +r или set +o restricted.

Эти ограничения применяются после чтения всех стартовых файлов.

При выполнении команд, которые считаются сценариями оболочки (3.8. Сценарии оболочки), rbash отключает все ограничения в оболочке, вызванной для выполнения сценария.

6.11. Режим POSIX

Запуск Bash с опцией --POSIX или выполнение команды set -o POSIX в процессе работы Bash будет переводить оболочку в режим более точного соответствия стандарту POSIX путём смены поведения для соответствия POSIX в тех областях, где принятое по умолчанию поведение Bash отличается от стандарта.

При вызове по имени sh оболочка Bash входит в режим POSIX после считывания стартовых файлов.

Ниже приведён список изменений, обусловленных переходом в режим POSIX.

1. Bash обеспечивает установку переменной POSIXLY_CORRECT.
2. Когда команды больше нет в хэш-таблице, Bash будет заново просматривать $PATH для поиска нового местоположения. Это доступно также по команде shopt -s checkhash.
3. Сообщение, выводимое кодом управления заданиями и внутренними функциями при завершении задания с ненулевым статусом, имеет вид Done(статус).
4. Сообщение, выводимое кодом управления заданиями и внутренними функциями при остановке задания, имеет вид Stopped(signame), где signame указывает имя сигнала (например, SIGTSTP).
5. Преобразование псевдонимов выполняется всегда (даже в неинтерактивном режиме).
6. Зарезервированные слова в контексте, где они распознаются, не подвергаются преобразованию псевдонимов.
7. В POSIX преобразование в PS1 и PS2 символа ! в номер истории и !! в ! включено и преобразование параметров для PS1 и PS2 не зависит от опции promptvars.

8. Выполняются стартовые файлы POSIX ($ENV) вместо обычных файлов Bash.
9. Преобразование тильды выполняется лишь в назначениях, предшествующих имени команды, а не во всех.
10. По умолчанию история записывается в файл ~/.sh_history (принятое по умолчанию значение $HISTFILE).
11. Операторы перенаправления не преобразуют имена файлов в word, если оболочка не является интерактивной.
12. Операторы перенаправления не расщепляют слова в word.
13. Имена функций должны быть действительными именами оболочки, т. е. они не могут содержать символов, не являющихся буквами, цифрами или знаком подчёркивания, а также не могут начинаться с цифры. Объявление функции с непригодным именем приводит к критической ошибке синтаксиса в неинтерактивных оболочках.
14. Имена функций могут не совпадать с именами специальных внутренних команд POSIX.
15. Специальные внутренние команды POSIX отыскиваются до функций оболочки.
16. При выводе определений функций оболочки (например, с помощью type) Bash не печатает ключевое слово function.
17. Символ ~ в качестве первого символа в элементах PATH не преобразуется, как описано в параграфе 3.5.2. Преобразование тильды.
18. Зарезервированное слово time может применяться в качестве команды. В этом случае выводится временная статистика оболочки и завершённых потомков. Формат вывода задаёт переменная TIMEFORMAT.
19. При анализе и преобразовании ${ . . . } для выражений в двойных кавычках одинарные кавычки уже не имеют особого значения и не могут служить для «экранирования» закрывающей скобки или других специальных символов, пока оператор не является одним из тех, которые указаны для удаления шаблона. В этом случае они не обязаны быть парными.
20. Анализатор не распознает time как зарезервированное слово, если следующий маркер начинается с -.
21. Символ ! не вводит преобразования истории для строки в двойных кавычках даже при включённой опции histexpand.
22. Если внутренняя команда POSIX special возвращает статус ошибки, неинтерактивная оболочка завершает работу. Критические ошибки указаны в стандарте POSIX и включают передачу некорректных опций, ошибки перенаправления, ошибки назначения переменных для присвоений до имени команды и т. п.
23. Неинтерактивная оболочка завершается со статусом ошибки, если при назначении переменной имя команды не указано после оператора присваивания. Такие ошибки могут возникать, например, при попытке задать значение переменной, доступной лишь для чтения.
24. Неинтерактивная оболочка завершается со статусом ошибки, если при назначении переменной оператор присваивания предшествует специальной встроенной команде, а не другой простой команде.
25. Неинтерактивная оболочка завершается со статусом ошибки, если переменная цикла for или переменная выбора в операторе select доступна лишь для чтения.
26. Неинтерактивная оболочка завершается, если не найден файл из . filename.
27. Неинтерактивная оболочка завершается, если синтаксическая ошибка в арифметическом выражении приводит к некорректному выражению.
28. Неинтерактивная оболочка завершается, если возникает ошибка преобразования параметров.
29. Неинтерактивная оболочка завершается, если имеется синтаксическая ошибка в сценарии, прочитанном встроенной командой . или source, или при обработке строки внутренней командой eval.
30. Подстановка процессов не доступна.
31. Косвенность переменных доступна, но не может применяться к специальным параметрам # и ?.
32. При преобразовании специального параметра * в контексте шаблона, где преобразуется выражение в двойных кавычках, не считающее последовательность $* заключённой в двойные кавычки.
33. Операторы присваивания, предшествующие специальным внутренним командам POSIX, сохраняются в среде оболочки после выполнения внутренней команды.
34. Операторы присваивания, предшествующие вызовам функций оболочки, сохраняются в среде оболочки после возврата из функции как при исполнении специальных внутренних команд POSIX.
35. Внутренняя команда command не препятствует внутренним командам, которые принимают операторы присваивания в качестве аргументов, преобразовывать их как операторы присваивания. В иных режимах внутренние операторы присваивания, заданные как аргументы, теряют свои свойства при обработке command.
36. Внутренняя команда bg использует для описания каждого задания, помещаемого в фоновый режим, обязательный формат, не указывающий, является задание текущим или предыдущим.
37. Вывод kill -l включает имена всех сигналов в одной строке через пробелы без суффикса SIG.
38. Внутренняя команда kill не воспринимает имена сигналов с префиксом SIG.
39. Внутренние команды export и readonly используют для вывода формат, требуемый POSIX.
40. Внутренняя команда trap выводит имена сигналов без префикса SIG.

41. Внутренняя функция trap проверяет первый аргумент на предмет спецификации сигнала и сохраняет исходную обработку сигналов (при наличии), если этот аргумент не включает только цифры действительного номера сигнала. Если пользователь хочет сбросить обработчик для данного сигнала в исходное состояние, следует использовать - в качестве первого аргумента.
42. Внутренние команды . и source не просматривают текущий каталог для аргумента filename, если файл не найден в PATH.
43. Включение режима POSIX ведёт к установке опции inherit_errexit, поэтому субоболочки, вызываемые для выполнения подстановки команд, наследуют значение опции -e от родительской оболочки. При выключенной опции inherit_errexit интерпретатор Bash сбрасывает -e в таких субоболочках.
44. Включение режима POSIX ведёт к установке опции shift_verbose option, поэтому числовые аргументы shift, выходящие за пределы числа позиционных параметров, приводят к сообщениям об ошибке.
45. При выводе определений псевдонимов внутренняя команда alias не использует ‘alias ’, если нет опции -p.
46. При вызове внутренней команды set без опций не выводится имён и определений функций оболочки.
47. При вызове внутренней команды set без опций она выводит значения переменных без кавычек, если они не включают метасимволов , даже если в результате возникают непечатаемые символы.
48. Когда внутренняя команда cd вызывается в логическом режиме, а путь, созданный из $PWD и указанного в аргументе имени каталога, не указывает существующий каталог, cd будет давать отказ вместо перехода в физический режим.
49. Когда внутренняя команда cd не может сменить каталог в результате того, что имя, созданное из $PWD и указанного аргументом каталога, превышает по размеру PATH MAX при преобразовании символьных ссылок, cd будет давать отказ вместо попытки использовать лишь предоставленное имя каталога.
50. Внутренняя команда pwd проверяет совпадение выводимого значения с текущим каталогом, даже если такая проверка не запрошена с помощью опции -P.
51. При выводе истории внутренняя команда fc не указывает наличие изменений в записях истории.
52. По умолчанию fc использует редактор ed.
53. Внутренние команды type и command не сообщают о нахождении неисполняемого файла, хотя оболочка будет пытаться выполнить такой файл, если он окажется единственным в $PATH.
54. Режим редактирования vi будет вызывать редактор vi напрямую по команде v вместо проверки переменных $VISUAL и $EDITOR.
55. При включённой опции xpg_echo оболочка Bash не будет пытаться интерпретировать какие-либо аргументы для эхо-вывода как опции. Каждый аргумент отображается после преобразования символов экранирования.
56. Внутренняя команда ulimit использует размер блока 512 байтов для опций -c и -f.
57. Получение сигнала SIGCHLD при установленной для него ловушке не прерывает внутреннюю команду и не ведёт к немедленному возврату. Команда trap запускается однократно для каждого завершения работы потомка.
58. Внутренняя команда read может быть прервана сигналом, для которого установлена ловушка. Если Bash получает захваченный сигнал при выполнении команды read, выполняется обработчик прерывания и read возвращает статус выхода больше 128.
59. Bash удаляет статус завершённого фонового процесса из списка таких состояний после использования внутренней функции wait для получения статуса.

Имеется ряд аспектов поведения POSIX, которые Bash не реализует по умолчанию даже в режиме POSIX.

1. Внутренняя команда fc проверяет $EDITOR как программу для редактирования записей истории, если не установлена переменная FCEDIT, вместо использования напрямую редактора ed. fc применяет ed, если переменная EDITOR не задана.
2. Как отмечено выше, Bash требует включения опции xpg_echo для полного соответствия стандарту внутренней команды echo.

Bash можно настроить на соответствие стандарту POSIX по умолчанию путём указания --enable-strict-POSIX-default при сборке (10.8. Дополнительные возможности).

7. Управление заданиями

В этой главе описано управление заданиями в Bash и способы доступа к такому управлению.

7.1. Основы управления заданиями

Управление заданиями позволяет селективно останавливать (приостанавливать) выполнение процесса, а затем продолжать (восстанавливать) процесс. Пользователь обычно применяет эти средства через интерактивный интерфейс, предоставляемый вместе с драйвером терминала в ядре и Bash.

Оболочка связывает задание с каждый конвейером и хранит таблицу выполняемых в данный момент заданий, которую можно просматривать по команде jobs. При асинхронном запуске задания Bash выводит строку вида

```
[1] 25647
```

показывающую, что задание имеет номер 1, а идентификатор последнего процесса в конвейере, связанном с этим заданием, имеет значение 25647. Все процессы конвейера относятся к одному заданию. Bash использует абстракцию задания в качестве базы для управления заданиями.

Для упрощения реализации пользовательского интерфейса управления заданиями операционная система поддерживает понятие текущего идентификатора группы терминальных процессов. Члены этой группы (процессы, у который идентификатор группы совпадает с идентификатором группы терминальных процессов) получают генерируемые клавиатурой сигналы, такие как SIGINT. Эти процессы называют приоритетными (foreground). Фоновыми (background) процессами называют процессы, у которых идентификатор группы отличается о идентификатора текущей группы терминальных процессов. Такие процессы не подвержены воздействию клавиатурных сигналов. Только приоритетным процессам разрешено чтение с терминала (и запись при установке пользователем stty tostop). Фоновые процессы, которые пытаются читать с терминала (или писать при включённом stty tostop) передают сигнал SIGTTIN (SIGTTOU) через драйвер терминала в ядре, который приостанавливает процесс, если не будет перехвачен.

Если операционная система, где работает Bash, поддерживает управление заданиями, Bash позволяет упростить это. Ввод символов приостановки (обычно ^Z, Control-Z) при работе процесса, останавливает процесс и возвращает управление Bash. Ввод символов отложенной приостановки (обычно ^Y, Control-Y) останавливает процесс после попытки чтения с терминала и управление возвращается в Bash. Пользователь может менять состояние заданий, используя команду bg для продолжения задания в фоновом режиме, fg для продолжения в приоритетном режиме или kill для полного прекращения . Клавиши ^Z действуют сразу, но вызывают побочный эффект в форме отбрасывания ожидающих выходных данных и автозаполнения (typeahead).

Существует много способов ссылки на задания в оболочке. Символ % указывает спецификацию задания (jobspec). Задание с номером n можно указать как %n. Символы %% и %+ обозначают оболочку текущего задания, которое является последним, остановленным в приоритетном режиме или запущенным в фоновом. Отдельный символ % без указания номера задания также обозначает текущее задание. Предыдущее задания можно указать символами %-. Если имеется лишь одно задание, его можно указать как %+, так и %-. В относящемся к заданиям выводе (например, в выводе команд задания) текущее задание всегда помечается +, а предыдущее - '-'.

Задание можно указать с помощью префикса имени, использованного при запуске, или подстроки команды. Например, %ce указывает остановленное задание ce, а %?ce - задание, содержащее строку ce в команде. Если префикс или подстрока соответствуют нескольким заданиям, Bash сообщает об ошибке.

Задание можно сделать приоритетным (foreground) простым указанием его имени. Например %1 является синонимом fg %1 и переводит фоновое задание 1 в приоритетное. Аналогично, %1 & переводит задание 1 в фоновый режим как команда bg %1

Оболочка сразу узнает о смене состояния заданий. Обычно Bash ждёт вывода приглашения перед выводом данных о смене состояния заданий, чтобы не прерывать другой вывод. Если включена опция -b внутренней команды set, Bash сообщает об изменения незамедлительно (4.3.1. Внутренняя команда set). Любая ловушка SIGCHLD выполняется для каждого завершаемого дочернего процесса.

При попытке завершения Bash с остановленным заданием (или работающим при включённой опции checkjobs - 4.3.2. Внутренняя команда shopt) выводится предупреждение, а при включённой опции checkjobs - список заданий с их статусом. После этого можно использовать команду для проверки статуса заданий. При повторной попытке выхода без промежуточных команд Bash не выдаёт предупреждений и просто прерывает остановленные задания.

Когда оболочка ждёт выполнения задания или процесса по внутренней команде wait и управление заданиями включено, wait будет возвращать смену статуса задания. Опция -f будет заставлять wait ожидать завершения процесс или задания перед возвратом управления.

7.2. Внутренние средства управления заданиями

bg

```
bg [jobspec ...]
```

Возобновляет каждое остановленное задание jobspec в фоновом режиме как при запуске с оператором &. Если аргумент jobspec не задан, используется текущее задание. Статус возврата будет 0, если команда не была введена при отключённом управлении заданиями, указанное аргументом jobspec задание не было найдено или указанное задание было запущено без управления.

fg

```
fg [jobspec]
```

Возобновляет приоритетное (foreground) задание jobspec и делает его текущим. Если аргумент jobspec не задан, используется текущее задание. Статусом возврата будет статус команды, помещённой в режим foreground, или отличное от 0 значение, если управление заданиями отключено, при включённом управлении jobspec не указывает пригодного задания или указывает задание, запущенное без управления.

jobs

```
jobs [-lnprs] [jobspec]
jobs -x command [arguments]
```

Первая форма выводит список активных заданий. Опции приведены в таблице.

`-l` Добавляет список идентификаторов процессов.
`-n` Вывод информации лишь о заданиях, статус которых изменился с последнего уведомления пользователя.
`-p` Вывод только идентификатора процесса лидера группы процессов.
`-r` Вывод только работающих заданий.
`-s` Вывод только остановленных заданий.

При указании jobspec вывод ограничивается информацией об указанном задании, иначе выводятся все.

При наличии опции -x команда jobs заменяет все jobspec в команде или аргументах идентификаторами групп процессов и выполняет команду command, передавая ей arguments и возвращая статус выхода.

kill

```
kill [-s sigspec] [-n signum] [-sigspec] jobspec или pid
kill -l|-L [exit_status]
```

Передаёт сигнал, указанный в sigspec или signum, процессу, заданному jobspec или pid. Аргумент sigspec указывает имя сигнала без учёта регистра символов (префикс SIG не обязателен) или его номер. При отсутствии sigspec и signum используется SIGTERM. Имена сигналов выводит опция -l или -L. Если с этой опцией указаны аргументы, выводятся соответствующие им имена сигналов и возвращается статус 0. Статусом выхода служит

номер сигнала или статус прерванного сигналом процесса. Статус возврата равен -, если успешно передан хотя бы один сигнал, и отличен от 0 при возникновении ошибки или наличии непригодной опции.

wait

```
wait [-fn] [jobspec или pid ...]
```

Задаёт ожидание завершения каждого процесса, указанного pid или jobspec, и возвращает статус последней выполненной команды. При указании jobspec ожидается выполнение всех процессов задания. Если задание не указано, ожидание применяется ко всем заданиям. Если аргументы не заданы, ожидание применяется ко всем активным в данный момент дочерним процессам и в итоге возвращается статус 0. При наличии опции -n команда wait ждёт завершения любого из заданий и возвращает статус его выхода. При наличии опции -f и включённом управлении заданиями wait заставляет завершиться каждый процесс pid или задание jobspec перед возвратом его статуса вместо возврата управления при смене статуса. Если jobspec и pid не указывают активных дочерних процессов оболочки, возвращается статус 127.

disown

```
disown [-ar] [-h] [jobspec ... | pid ... ]
```

Без опций удаляет каждое задания jobspec из таблицы активных заданий. При наличии опции -h задания не удаляются из таблицы, а помечаются так, что сигнал SIGHUP не передаётся заданию при получении SIGHUP оболочкой. Если jobspec отсутствует и нет опции -a или -r, используется текущее задание. Без jobspec опция -a задаёт удаление или маркировку всех заданий, а -r без jobspec ограничивает операцию работающими заданиями.

suspend

```
suspend [-f]
```

Приостанавливает выполнение данной оболочки до получения сигнала SIGCONT. Оболочку входа (login) обычно приостановить нельзя, но опция -f позволяет форсировать приостановку.

При отключённом управлении заданиями внутренние команды kill и wait не принимают jobspec и нужно указывать pid.

7.3. Переменные управления заданиями

auto_resume

Эта переменная определяет взаимодействие оболочки с пользователем и управлением заданиями. При наличии переменной простые команды из одного слова считаются кандидатами на возобновление имеющегося задания. Двусмысленность не допускается и при наличии нескольких заданий, начинающихся с набранной строки, будет выбрано то, к которому наименее давно был доступ. Именем остановленного задания в этом контексте является командная строка, использованная для запуска задания. Если переменная имеет значение exact, строка должна в точности совпадать с именем остановленного задания, а при значении substring достаточно совпадения введённой строки с частью имени остановленного задания. Значение substring обеспечивает функциональность, аналогичную идентификатору задания %? (7.1. Основы управления заданиями). При установке в переменной любого другого значения введённая строка должна быть префиксом имени остановленного задания, аналогично идентификатору задания %.

8. Редактирование командной строки

В этой главе описаны базовые возможности командного интерфейса gnu для редактирования строк. Редактирование выполняется с помощью библиотеки Readline, используемой разными программами, включая Bash. Редактирование командной строки включено по умолчанию для интерактивной оболочки, если при вызове не была задана опция --noediting. Редактирование строк применяется также с опцией -e для внутренней команды read (4.2. Внутренние команды bash). По умолчанию команды редактирования строк похожи на команды Emacs, но доступен также интерфейс со стилем редактора vi. Редактирование строк можно включить в любой момент опцией -o emacs или -o vi внутренней команды set (4.3.1. Внутренняя команда set) или отключить опцией +o emacs или +o vi команды set.

8.1. Введение в редактирование строк

В приведённых ниже описаниях C-k означает Control-K, т. е. нажатие клавиши k при удерживаемой клавише Control. M-k указывает Meta-K, т. е. нажатие клавиши k при удерживаемой клавише Meta (если она имеется). Клавиша Meta на клавиатуре часто обозначена ALT. На клавиатурах с двумя клавишами ALT (обычно слева и справа от клавиши пробела) левая клавиша ALT обычно служит в качестве клавиши Meta. Правую клавишу ALT также можно настроить в качестве Meta или иного модификатора (например, Compose для ввода символов со знаком ударения).

Если нет клавиш Meta, ALT или иной клавиши, работающей как Meta, идентичные последовательности можно создавать, нажимая клавишу ESC, а затем k.

M-C-k означает Meta-Control-k, т. е. нажатие клавиши k при удерживаемых клавишах Meta и Control.

Для некоторых клавиш применяются специальные имена. В частности, клавиши DEL, ESC, LFD, SPC, RET, TAB обозначаются по именам в тексте документа и файле инициализации (8.3. Файл инициализации Readline). Если на клавиатуре нет клавиши LFD, её заменит нажатие C-j. Клавиша RET может называться Return или Enter.

8.2. Взаимодействие с Readline

Зачастую при работе в интерактивной сессии можно заметить ошибку в начале набранной строки, уже введя достаточно много символов. Библиотека Readline обеспечивает множество команд для манипуляций с набранным текстом, позволяя исправлять ошибки, добавлять или удалять текст. Для использования команд редактирования нужно переместить курсор в нужное место строки и удалить ненужные символы, внести изменения или добавить пропущенные символы. Добившись нужного результата, просто нажмите клавишу RET. Перемещать курсор в конец строки перед нажатием RET не требуется.

8.2.1. Основы Readline

Для ввода символов в строку нужно просто набрать их и символ появится в позиции указанной курсором, а курсор будет перемещён вправо. При ошибочном ввода можно использовать символ стирания для возврата и удалить ошибочный символ. Иногда ошибка может остаться незамеченной сразу и в таких случаях можно использовать клавиши C-b для перемещения курсора влево и исправления ошибки. После этого курсор можно переместить вправо клавишами C-f.

При добавлении символов в середину строки символы справа от добавляемых смещаются вправо, освобождая место для новых. При удалении текста символы справа от удаляемых смещаются влево, занимая освободившееся место. Ниже приведены наиболее важные команды редактирования строк.

C-b
Сдвиг курсора на одну позицию назад (влево).
C-f
Сдвиг курсора на одну позицию вперёд (вправо).
DEL или Backspace
Удаление символа слева от курсора.
C-d
Удаление символа в позиции курсора[1].
Символьные клавиши
Вставка символа в позицию курсора.
C-_ или C-x C-u
Отмена последней команды редактирования. Можно отметить все, вплоть до пустой строки.

8.2.2. Команды перемещения в Readline

В дополнение к описанным выше командам имеются комбинации клавиш быстрого перемещения.

C-a
Перенос курсора в начало строки.
C-e
Перенос курсора в конец строки.
M-f
Перенос курсора на одно слово вперёд (вправо). Словом считается последовательность букв и цифр.
M-b
Перенос курсора на одно слово назад (влево).
C-l
Очистка экрана с выводом текущей строки в верхней позиции.

Отметим, что C-f перемещает курсов на один символ, а M-f - на одно слово. Это свободное соглашение, связывающее клавишу Control с символами, а Meta - со словами.

8.2.3. Команды уничтожения в Readline

Уничтожение текста означает его удаление из строки с сохранением для последующего применения (обычно путём восстановления - yanking). В современной терминологии используются cut и paste вместо kill и yank. Если описание команды говорит, что она «убивает» текст, можно быть уверенным в возможности вернуть его в то же или иное место.

При использовании команды kill текст сохраняется в буфере kill-ring. Произвольное число уничтожений текста сохраняется вместе и текст можно восстановить. Буфер не связан с конкретными строками и уничтоженный текст можно потом поместить в ту же или другую строку. Команды уничтожения текста приведены ниже.

`C-k` Уничтожает текст от курсора до конца строки.
`M-d` Уничтожает текст от курсора до конца текущего слова или до конца следующего слова, если курсор находится между словами. Границы слов совпадают с используемыми M-f.
`M-DEL` Уничтожает текст от курсора до начала текущего слова или до начала предыдущего слова, если курсор находится между словами. Границы слов совпадают с используемыми M-b.
`C-w` Уничтожает текст от курсора до предыдущего пробела. Это отличается от M-DEL различием в границах слов.

Ниже приведены команды восстановления текста (yank), когда в строку копируется уничтоженный последним текст.

`C-y` Поместить уничтоженный последним текст, начиная с позиции курсора.
`M-y` «Провернуть» kill-ring и восстановить новый «верхний» текст. Команда работает только после C-y или M-y.

8.2.4. Аргументы Readline

Можно передать команде Readline числовые аргументы, которые могут служить счётчиками повторов, а иногда содержат знак, который меняет действие команды. При передаче отрицательного аргумента команде, которая обычно действует в прямом направлении, эта команда будет применена в обратном направлении. Например, для уничтожения текста до начала строки можно использовать M-- C-k.

Общим способом передачи числовых аргументов является нажатие клавиши Meta с цифровой клавишей перед командой. Если вместо первой цифры будет нажата клавиша -, аргумент получит отрицательное значение. После нажатия Meta-цифра можно нажать остальные цифры, а затем ввести команду. Например, для передачи команде C-d аргумента 10 можно нажать клавиши M-1 0 C-d, что приведёт к удалению следующих после курсора 10 символов.

8.2.5. Поиск команд в истории

Readline поддерживает команды для поиска в истории команд (9.1. Средства Bash History)строк, содержащих указанную подстроку. Поддерживается два режима поиска - инкрементный и неинкрементный.

Инкрементный поиск начинается до завершения пользователем ввода искомой строки. По мере ввода символов Readline отображает следующую запись истории, соответствующую набранному. Для поиска в прямом направлении по списку истории служит команда C-s, в обратном - C-r. Для завершения инкрементного поиска служат символы из переменной isearch-terminators. Если значение переменной не задано, поиск можно завершить клавишами ESC и C-J, а C-g будет прерывать инкрементный поиск и восстанавливать исходную строку. По завершении поиска запись истории, содержащая введённые символы, становится текущей строкой.

Для перехода к другим записям истории, соответствующим набранному, служат команды C-r и C-s. Любая другая комбинация клавиш, с которой связана команда Readline, будет завершать поиск и выполнять команду. Например, клавиша RET ведёт к завершению поиска и восприятию команды с её выполнением. Команды перемещения будут

[1]В зависимости от настройки Backspace может удалять символ слева от курсора, а DEL - в позиции курсора как C-d.

прерывать поиск и переводить в режим редактирования найденной строки. Readline запоминает последнюю строку инкрементного поиска и при нажатии C-r подряд без промежуточных символов, определяющих поиск, используется такая строка.

При неинкрементном поиске искомая строка считывается полностью до начала поиска в истории. Строка поиска может быть введена пользователем или быть частью содержимого текущей строки.

8.3. Файл инициализации Readline

Хотя библиотека Readline распространяется с набором привязок клавиш в стиле Emacs, можно использовать другой набор привязок. Любой пользователь может настроить программы, использующие Readline, помещая команды в файл inputrc (обычно в домашнем каталоге). Имя этого файла берётся из значения переменной окружения INPUTRC, если переменная не задана, используется ~/.inputrc. Если указанный файл отсутствует или недоступен для чтения, используется файл /etc/inputrc.

При запуске программы, использующей библиотеку Readline, считывается файл инициализации и задаются привязки клавиш. Кроме того, команда C-x C-r перечитывает файл инициализации, активизируя все внесённые в него изменения.

8.3.1. Синтаксис файла инициализации

Имеется небольшое число базовых конструкций, разрешённых в файле инициализации Readline. Пустые строки в файле игнорируются, строки, начинающиеся с символа #, являются комментариями. Строки, начинающиеся с символа $, задают условные конструкции (8.3.2. Условные конструкции в файле инициализации). Остальные строки задают переменные и привязки клавиш.

Установка переменных

Можно изменить поведение Readline, меняя значения переменных в файле инициализации командой set.

```
set variable value
```

Например, для смены стиля команд Emacs на стиль vi можно указать

```
set editing-mode vi
```

Имена и значения переменных, когда это возможно, не учитывают регистр символов. Нераспознанные переменные игнорируются. Логические переменные (те, что могут иметь значения on или off) устанавливаются в on, если имеют значение null (пусто), on (без учёта регистра) или 1. Все прочие значения ведут к установке off.

Команда bind -V выводит имена и значения текущих переменных Readline (4.2. Внутренние команды bash). Поведение библиотеки при работе определяется прежде всего перечисленными ниже переменными.

bell-style	Управляет поведением Readline при «звонках» в терминал (bell). При значении none звонок не используется, visible задаёт использование видимого звонка, если он доступен. Принятое по умолчанию значение audible задаёт использование звукового сигнала.
bind-tty-special-chars	Принятое по умолчанию значение on ведёт к попытке Readline привязать управляющие символы с особой обработкой драйвером терминала в ядре к их эквивалентам в Readline.
blink-matching-paren	При значении on Readline пытается кратковременно переместить курсов к открывающей скобке при вводе закрывающей (по умолчанию off).
colored-completion-prefix	Если установлено значение on, Readline при перечислении дополнений указывает общий префикс набора возможных дополнений другим цветом в соответствии с переменной среды LS_COLORS (по умолчанию off).
colored-stats	Если установлено значение on, Readline выводит возможные дополнения с выделением типов цветом в соответствии с переменной среды LS_COLORS (по умолчанию off).
comment-begin	Текст для вставки в начало строки по команде insert-comment (по умолчанию #).
completion-display-width	Число символов в строке экрана, используемых для вывода возможных совпадений при дополнении. Отрицательные и превышающие ширину экрана значения игнорируются. Значение 0 задаёт вывод совпадений по одному в строке. По умолчанию -1.
completion-ignore-case	Если установлено значение on, Readline учитывает регистр символов при сопоставлении имён файлов и дополнении (по умолчанию off).
Completion-map-case	Если установлено значение on и включено completion-ignore-case, Readline считает символы дефиса (-) и подчёркивания (_) эквивалентными при сопоставлении имён файлов и дополнении без учёта регистра (по умолчанию off).
completion-prefix-display-length	Число символов общего префикса списка возможных дополнений, которые выводятся без изменения. При положительном значении более длинные общие префиксы заменяются многоточием.
completion-query-items	Число возможных дополнений, при котором у пользователя спрашивают о выводе списка, если набор дополнений больше. При меньшем числе список выводится без вопроса. Значение должно быть неотрицательным целым числом, при отрицательных значениях Readline не будет задавать вопрос (по умолчанию 100).
convert-meta	При значении on Readline будет преобразовывать символы с установленным старшим (8) битом в последовательность ASCII без старшего бита с ESC-префиксом, а затем - в последовательность с префиксом meta. По умолчанию установлено значение on, но оно будет заменено на off, если установленный язык включает 8-битовые символы.
disable-completion	При значении on Readline будет запрещать дополнение слов. Символы дополнения будут помещаться в строку как при самовставке (по умолчанию off).

echo-control-characters	При заданном по умолчанию значении on и поддержке операционной системой Readline будет выводить эхо-символы по сигналам клавиатуры.
editing-mode	Управляет используемыми привязками клавиш и может принимать значение emacs или vi. По умолчанию Readline использует режим Emacs.
emacs-mode-string	При включённом режиме show-mode-in-prompt эта строка (по умолчанию @) выводится непосредственно перед последней строкой основного приглашения в режиме emacs. Значение преобразуется подобно привязкам клавиш и доступны стандартные префиксы Meta- и Control, а также экранирование \. Комбинации \1 и \2 указывают начало и конец последовательности непечатаемых символов, которую можно включить в последовательность управления терминалом в строке режима.
enable-bracketed-paste	Если установлено значение on, Readline будет настраивать терминал на размещение каждой вставки в буфере редактирования как отдельной строки символов вместо того, что считать каждый символ вводом с клавиатуры. Это предотвращает трактовку введённых символов как команд редактирования (по умолчанию off).
enable-keypad	Если установлено значение on, Readline пытается включить клавиатуру приложения при её вызове. Некоторым системам это нужно для включения клавиш со стрелками. По умолчанию off.
enable-meta-key	Если установлено значение on, Readline пытается включить любую клавишу модификатора Meta, поддержку которой заявляет терминал. На многих терминалах клавиша Meta служит для ввода 8-битовых символов. По умолчанию on.
expand-tilde	Значение on задаёт преобразование тильды при попытке Readline дополнять слова (по умолчанию off).
history-preserve-point	При значении on код истории пытается установить курсор в одну позицию для каждой строки истории,извлекаемой командами previous-history и next-history (по умолчанию off).
history-size	Максимальное число записей в списке истории. При значении 0 имеющиеся записи удаляются, при отрицательных значениях размер списка не ограничивается (принято по умолчанию). При задании в переменной нечислового значения устанавливается предел 500 записей.
horizontal-scroll-mode	Значение on задаёт горизонтальную прокрутку текста при размере строки, превышающем размер экрана, вместо перехода на новую строку (по умолчанию off).
input-meta	Если установлено значение on, Readline включает 8-битовый ввод (не сбрасывает старший бит), независимо от заявленной терминалом его поддержки. По умолчанию установлено off, но Readline будет устанавливать on, если выбранный язык использует 8-битовые символы. Переменную называют также meta-flag.
isearch-terminators	Строка символов прерывания инкрементного поиска без последующего выполнения команды из дальнейших символов (8.2.5. Поиск команд в истории). Если переменная не задана, используется ESC или C-J.
keymap	Задаёт представление Readline о текущей раскладке клавиатуры для привязки клавиш. Встроенные таблицы называются emacs, emacs-standard, emacs-meta, emacs-ctlx, vi, vi-move, vi-command, vi-insert. Раскладки vi и vi-command (vi-move является синонимом), а также emacs и emacs-standard эквивалентны. Приложения могут добавлять свои имена. По умолчанию применяется emacs. Переменная editing-mode также влияет на принятую по умолчанию раскладку.
keyseq-timeout	Время ожидания ввода (мсек, по умолчанию 500) символа при считывании неоднозначной последовательности нажатий клавиш (непонятно, завершён ли ввод последовательности). По истечении времени Readline считает последовательность завершённой. Readline использует значение для определения доступности ввода из текущего источника (по умолчанию rl_instream). Если переменная не содержит положительное число, Readline будет ждать нажатия другой клавиши для принятия решения.
mark-directories	При значении on (принято по умолчанию) имена каталогов завершаются символом \.
mark-modified-lines	Значение on заставляет Readline выводить символ * в начале строк истории, которые были изменены (по умолчанию off).
mark-symlinked-directories	При значении on к полным именам, которые являются символьными ссылками на каталог, добавляется \ в зависимости от mark-directories (по умолчанию off).
match-hidden-files	Значение on (принято по умолчанию) заставляет Readline сопоставлять имена файлов, начинающиеся с точки (скрытые) при дополнении имён файлов. При значении off пользователь должен сам указать точку в начале.
menu-complete-display-prefix	Если установлено значение on, при дополнении меню выводится общий префикс списка возможных дополнений (может быть пустым) перед прокручиванием списка (по умолчанию off).
output-meta	Если установлено значение on, Readline будет выводить символы с восьмым битом напрямую, без Meta-префиксов. По умолчанию установлено значение off, но Readline устанавливает on для языков с 8-битовыми символами.

page-completions	Если установлено значение on (принято по умолчанию), Readline использует внутреннее разбиение на страницы в стиле more при отображении многостраничных списков возможных дополнений.
print-completions-horizontally	При значении on Readline выводит дополнения по горизонтали, сортируя в алфавитном порядке, вместо построчного вывода (по умолчанию off).
revert-all-at-newline	Если установлено значение on, Readline отменяет все изменения в строках истории перед возвратом при выполнении accept-line. По умолчанию строки могут быть изменены и сохраняют отдельные списки отмены изменений между вызовами readline.
show-all-if-ambiguous	При установке on для слов с несколькими вариантами дополнения совпадения выводятся сразу вместо подачи звукового сигнала (по умолчанию off).
show-all-if-unmodified	При установке on для слов с несколькими вариантами дополнения без возможности частичного завершения (варианты не имеют общего префикса) совпадения выводятся сразу вместо подачи звукового сигнала (по умолчанию off).
show-mode-in-prompt	Установка значения on добавляет в начало приглашения строку, указывающую режим редактирования (emacs, vi command, vi insertion), которую пользователь может изменить (например, emacs-mode-string). По умолчанию off.
skip-completed-text	Установка значения on меняет принятое по умолчанию поведение при вставке в строку одного совпадения в середине слова. При включённом режиме readline не вставляет символы из завершения, соответствующие символам после указателя в дополняемом слове, поэтому части слов после курсора не дублируются. Например, попытка дополнить после символа e слово Makefile даст в результате Makefile, а не Makefilefile, при наличии одного завершения (по умолчанию off).
vi-cmd-mode-string	Если переменная show-mode-in-prompt включена, эта строка (по умолчанию (cmd)) выводится непосредственно перед последней строкой основного приглашения в командном режиме vi. Значение преобразуется подобно привязке клавиш, поэтому доступен стандартный набор префиксов Meta и Control, а также \-экранирование. Комбинации \1 и \2 указывают начало и конец последовательности непечатаемых символов, которую можно включить в последовательность управления терминалом в строке режима.
vi-ins-mode-string	Если переменная show-mode-in-prompt включена, эта строка (по умолчанию (ins)) выводится непосредственно перед последней строкой основного приглашения в режиме вставки vi. Значение преобразуется подобно привязке клавиш, поэтому доступен стандартный набор префиксов Meta и Control, а также \-экранирование. Комбинации \1 и \2 указывают начало и конец последовательности непечатаемых символов, которую можно включить в последовательность управления терминалом в строке режима The default is ‘(ins)’.
visible-stats	При установке значения on символ, указывающий тип файла, добавляется к имени файла при выводе возможных дополнений (по умолчанию off).

Привязка клавиш

Синтаксис привязки клавиш управления в файле инициализации достаточно прост. Ниже приведены таблицы имён команд, принятые по умолчанию привязки клавиш и краткие описания действия команд.

Когда имя команды понятно, нужно просто найти строку в файле инициализации или добавить её и указать комбинацию клавиш для привязки, отделив её двоеточием от имени команды. Пробел между именем и двоеточием будет считаться частью имени. Клавиши можно указывать разными способами. Кроме команд readline позволяет связать комбинацию клавиш со строкой, которая будет вставляться при нажатии клавиш (макрос).

Команда bind -p выводит имена функций Readline с их привязками в формате, который подходит для файла инициализации (4.2. Внутренние команды bash).

keyname: function-name или macro

Поле keyname задаёт имя клавиши на английском языке, например,

```
Control-u: universal-argument
Meta-Rubout: backward-kill-word
Control-o: "> output"
```

В этом примере C-u привязывается к функции universal-argument, M-DEL - к функции backward-kill-word, а C-o - к макросу, указанному справа (вставка ‘> output’ в строку).

В привязках распознаются символьные имена клавиш DEL, ESC, ESCAPE, LFD, NEWLINE, RET, RETURN, RUBOUT, SPACE, SPC, TAB.

"keyseq": function-name or macro

Поле keyseq отличается от keyname тем, что позволяет указывать всю последовательность клавиш в двойных кавычках. Это позволяет использовать некоторые комбинации в стиле Emacs, но имена специальных символов не распознаются.

```
"\C-u": universal-argument
"\C-x\C-r": re-read-init-file
"\e[11~": "Function Key 1"
```

В примере комбинация C-u снова привязана к функции universal-argument, C-x C-r - к функции re-read-init-file, а ESC [1 1 ~ служит для вставки текста Function Key 1.

При указании комбинаций доступны приведённые ниже escape-последовательности в стиле Emacs.

\C- Префикс Control
\M- Префикс Meta
\e Символ экранирования (escape)
**** Обратная дробная черта

\" Символ двойной кавычки
\' Символ одинарной кавычки или апостроф

В дополнение к последовательностям в стиле Emacs поддерживается \-экранирование, приведённое ниже.

\a Звонок (bell)
\b Забой (backspace)
\d Удаление (delete)
\f Перевод страницы (form feed)
\n Перевод строки (newline)
\r Возврат каретки (carriage return)
\t Горизонтальная табуляция
\v Вертикальная табуляция
\nnn 8-битовый символ с восьмеричным кодом nnn (1 - 3 цифры)
\xHH 8-битовый символ с шестнадцатеричным кодом HH (1 или 2 цифры)

При вводе текста в определении макроса должны применяться одинарные или двойные кавычки. Текст без кавычек считается именем функции. В теле макроса описанное выше \-экранирование преобразуется. Символ \ экранирует в определении макроса любые символы, включая одинарные и двойные кавычки. Например, для привязки C-x \ к вставке в строку одного символа \ служит "\C-x\\": "\\".

8.3.2. Условные конструкции в файле инициализации

Readline реализует функции, похожие на условную компиляцию в препроцессорах C, которые позволяют задавать переменные и привязки клавиш с учётом проверки условий. Анализатор понимает 4 директивы, описанные ниже.

$if

Конструкция $if позволяет создавать привязки на основе режима редактирования, применяемого терминала или приложения, использующего Readline. Проверяемый текст после оператора сравнения продолжается до конца строки, если явно не указано иное. Для выделения текста не требуется специальных символов.

mode Форма mode= директивы $if служит для проверки работы Readline в режиме emacs или vi и может применяться с командой set keymap, например, для привязки раскладок emacs-standard и emacs-ctlx только при работе Readline в режиме emacs.

term Форма term= может служить для включения раскладок под определённые терминалы, возможно с учётом функциональных клавиш терминала. Слово справа от знака = сравнивается с полным именем терминала и частью имени до первого символа -. Это позволяет, например, указать sun для соответствия терминалам sun и sun-cmd.

version Проверка version может служить для работы с учётом версии Readline. Поле version преобразуется в текущую версию Readline. Набор операторов сравнения включает = (и ==), !=, <=, >=, <, >. Номер версии, указываемый справа от оператора, состоит из старшей части, необязательной точки и необязательной младшей части (например, 7.1). Если младшая часть не указана, предполагается 0. Оператор может отделяться пробелами от строки version и от номера версии. Приведённый ниже пример устанавливает переменную для Readline версии 7.0 и выше.

```
$if version >= 7.0
set show-mode-in-prompt on
$endif
```

application Конструкция application служит для задания установок в зависимости от приложения. Каждая программа, использующая библиотеку Readline, задаёт своё имя, что позволяет выполнить сравнение. Например, приведённая ниже команда добавляет комбинацию, заключающую текущее и предыдущее слово в кавычки при использовании Bash.

```
$if Bash
"\C-xq": "\eb\"\ef\""
$endif
```

variable Конструкция variable обеспечивает простую проверку равенства для переменных Readline, поддерживая операторы =, == и !=. Имя переменной должно, а значение может отделяться от оператора пробелом. Проверки возможны для строковых и логических (on или off) переменных. Ниже приведён пример проверки mode=emacs.

```
$if editing-mode == emacs
set show-mode-in-prompt on
$endif
```

$endif

Эта директива завершает конструкцию $if.

$else

Команды этой ветви конструкции $if выполняются при отрицательном результате проверки.

$include

Эта директива принимает аргументом имя файла и читает из него команды и привязки. Например,

```
$include /etc/inputrc
```

8.3.3. Пример файла инициализации

```
# This file controls the behaviour of line input editing for
# programs that use the GNU Readline library. Existing
# programs include FTP, Bash, and GDB.
#
# You can re-read the inputrc file with C-x C-r.
# Lines beginning with '#' are comments.
#
# First, include any system-wide bindings and variable
# assignments from /etc/Inputrc
$include /etc/Inputrc
#
# Set various bindings for emacs mode.
set editing-mode emacs
```

```
$if mode=emacs
Meta-Control-h:       backward-kill-word Text after the function name is ignored
#
# Arrow keys in keypad mode
#
#"\M-OD":      backward-char
#"\M-OC":      forward-char
#"\M-OA":      previous-history
#"\M-OB":      next-history
#
# Arrow keys in ANSI mode
"\M-[D":       backward-char
"\M-[C":       forward-char
"\M-[A":       previous-history
"\M-[B":       next-history
#
# Arrow keys in 8 bit keypad mode
#
#"\M-\C-OD":   backward-char
#"\M-\C-OC":   forward-char
#"\M-\C-OA":   previous-history
#"\M-\C-OB":   next-history
#
# Arrow keys   in 8 bit ANSI mode
#
#"\M-\C-[D":   backward-char
#"\M-\C-[C":   forward-char
#"\M-\C-[A":   previous-history
#"\M-\C-[B":   next-history
C-q: quoted-insert

$endif
# An old-style binding. This happens to be the default.
TAB: complete
# Macros that are convenient for shell interaction
$if Bash
# edit the path
"\C-xp": "PATH=${PATH}\e\C-e\C-a\ef\C-f"
# prepare to type a quoted word --
# insert open and close double quotes
# and move to just after the open quote
"\C-x\"": "\"\"\C-b"
# insert a backslash (testing backslash escapes
# in sequences and macros)
"\C-x\\": "\\"
# Quote the current or previous word
"\C-xq": "\eb\"\ef\""
# Add a binding to refresh the line, which is unbound
"\C-xr": redraw-current-line
# Edit variable on current line.
"\M-\C-v": "\C-a\C-k$\C-y\M-\C-e\C-a\C-y="
$endif
# use a visible bell if one is available
set bell-style visible
# don't strip characters to 7 bits when reading
set input-meta on
# allow iso-latin1 characters to be inserted rather
# than converted to prefix-meta sequences
set convert-meta off
# display characters with the eighth bit set directly
# rather than as meta-prefixed characters
set output-meta on
# if there are more than 150 possible completions for
# a word, ask the user if he wants to see all of them
set completion-query-items 150
# For FTP
$if Ftp
"\C-xg": "get \M-?"
"\C-xt": "put \M-?"
"\M-.": yank-last-arg
$endif
```

8.4. Клавиатурные команды Readline

В этом разделе описаны команды Readline, которые могут быть связаны с комбинациями клавиш. Можно посмотреть список привязок с помощью команды bind -P или bind -p (в более сжатом формате, пригодном для файла inputrc (4.2. Внутренние команды bash). Имена команд без комбинации клавиш по умолчанию не привязаны. В последующих описаниях точка указывает текущую позицию курсора, а метка (mark) - позицию курсора, сохранённую командой set-mark. Текст между точкой и меткой называют также областью (region).

8.4.1. Команды перемещения

beginning-of-line (C-a)

Перемещение в начало текущей строки.

end-of-line (C-e)

Перемещение в конец строки.

forward-char (C-f)
Перемещение на 1 символ вперёд.
backward-char (C-b)
Перемещение на 1 символ назад.
forward-word (M-f)
Перемещение в конец следующего слова (последовательность букв и цифр).
backward-word (M-b)
Перемещение в начало текущего или предыдущего слова (последовательность букв и цифр).
shell-forward-word ()
Перемещение в конец следующего слова (слова ограничиваются метасимволами без кавычек).
shell-backward-word ()
Перемещение в начало текущего или предыдущего слова (слова ограничиваются метасимволами без кавычек).
previous-screen-line ()
Пытается поместить указатель в ту же позицию предыдущей экранной строки. Это не будет работать, если текущая строка Readline не занимает нескольких экранных строк или позиция указателя не превышает сумму размера приглашения и ширины экрана.
next-screen-line ()
Пытается поместить указатель в ту же позицию следующей экранной строки. Это не будет работать, если текущая строка Readline не занимает нескольких экранных строк или позиция указателя не превышает сумму размера приглашения и ширины экрана.
clear-screen (C-l)
Очищает экран и заново выводит текущую строку наверху экрана.
redraw-current-line ()
Обновляет текущую строку. По умолчанию не имеет привязки.

8.4.2. Команды работы с историей

accept-line (Newline или Return)
Принимает строку независимо от позиции курсора. Непустая строка добавляется в историю в соответствии с переменными HISTCONTROL и HISTIGNORE. Изменённая строка истории восстанавливается.
previous-history (C-p)
Возврат в списке истории с выборкой предыдущей команды.
next-history (C-n)
Перемещение вперёд по списку истории с выборкой следующей команды.
beginning-of-history (M-<)
Перемещение к первой строке истории.
end-of-history (M->)
Перемещение в конец истории ввода (т. е. к вводимой в данный момент строке).
reverse-search-history (C-r)
Инкрементный поиск по списку истории от текущей строки к началу (вверх).
forward-search-history (C-s)
Инкрементный поиск по списку истории от текущей строки к концу (вниз).
non-incremental-reverse-search-history (M-p)
Неинкрементный поиск заданной пользователем строки по списку истории от текущей строки к началу (вверх).
non-incremental-forward-search-history (M-n)
Неинкрементный поиск заданной пользователем строки по списку истории от текущей строки к концу (вниз).
history-search-forward ()
Неинкрементный поиск вперёд по истории строки символов от начала текущей строки до курсора. Искомая строка должна быть началом строки истории. По умолчанию команда не привязана.
history-search-backward ()
Неинкрементный поиск назад по истории строки символов от начала текущей строки до курсора. Искомая строка должна быть началом строки истории. По умолчанию команда не привязана.
history-substring-search-forward ()
Неинкрементный поиск вперёд по истории строки символов от начала текущей строки до курсора. Искомая строка может находиться в любом месте строки истории. По умолчанию команда не привязана.
history-substring-search-backward ()
Неинкрементный поиск назад по истории строки символов от начала текущей строки до курсора. Искомая строка может находиться в любом месте строки истории. По умолчанию команда не привязана.
yank-nth-arg (M-C-y)
Вставка первого аргумента в предыдущую команду (обычно второе слово предыдущей строки) в позиции курсора. При наличии аргумента n вставляется n-е слово из предыдущей команды (отсчёт с 0), при отрицательном значении вставляется n-е слово от конца предыдущей команды. После вычисления n аргумент извлекается как при преобразовании истории !n.
yank-last-arg (M-. или M-_)
Вставка последнего аргумента в предыдущую команду (последнее слово предыдущей записи истории). С числовым аргументом похожа на yank-nth-arg. Последовательные вызовы yank-last-arg перемещают назад по списку истории, вставляя последнее (или заданное аргументом первого вызова) слово в каждую строку по очереди. Числовой аргумент этих последовательных вызовов определяет направление в списке истории. Используются средства преобразования истории для извлечения последнего аргумента как при задании !$.

8.4.3. Команды изменения текста

end-of-file (обычно C-d)
Символ, указывающий конец файла, как установлено, например, stty. Если символ прочитан при отсутствии других символов в строке и курсор находится в начале строки, Readline считает это завершением ввода и возвращает eof.
delete-char (C-d)
Удаляет символ в позиции курсора. Если функция привязана к той же комбинации, как символ tty eof (обычно C-d), действие идентично предыдущей команде.

backward-delete-char (Rubout[1])
Удаляет символ перед курсором (слева). Числовой аргумент задаёт уничтожение (kill) символов вместо удаления.

forward-backward-delete-char ()
Удаляет символ в позиции курсора, если это не конец строки, когда удаляется символ перед курсором (слева). По умолчанию команда не привязана.

quoted-insert (C-q или C-v)
Добавляет следующий введённый символ в строку дословно, например, как комбинация вставки символов C-q.

self-insert (a, b, A, 1, !, ...)
Вставляет введённый символ (самое себя).

bracketed-paste-begin ()
Эта функция предназначена для привязки к последовательности «вставка в скобках» (bracketed paste), передаваемой некоторыми терминалами, и такая привязка задана по умолчанию. Это позволяет Readline вставлять текст одним блоком без трактовки каждого символа как ввода с клавиатуры. Символы вставляются как с привязкой self-insert вместо выполнения команд редактирования.

transpose-chars (C-t)
Меняет местами символы перед курсором (слева) и в позиции курсора, а затем перемещает курсор на одну позицию вперёд (вправо). Если курсор находится в конце строки, просто меняются местами 2 последних символа. Отрицательные аргументы не дают эффекта.

transpose-words (M-t)
Меняет местами слово в позиции курсора и предшествующее ему слово, устанавливая курсор в конец второго (правого) слова. Если курсор находится в конце строки, просто меняются местами 2 последних слова строки.

upcase-word (M-u)
Переводит в верхний регистр текущее или следующее слово. С отрицательным аргументом переводит в верхний регистр предыдущее слово без перемещения курсора.

downcase-word (M-l)
Переводит в нижний регистр текущее или следующее слово. С отрицательным аргументом переводит в верхний регистр предыдущее слово без перемещения курсора.

capitalize-word (M-c)
Меняет на заглавную первую букву текущего или следующего слова. С отрицательным аргументом переводит в верхний регистр первую букву предыдущего слова без перемещения курсора.

overwrite-mode ()
Переключает режим вставки-замены. С явным положительным числовым аргументом переходит в режим замены, с явным неположительным - в режим вставки. Команда работает только в режиме emacs, а в режиме vi переключение происходит иначе. Каждый вызов readline() начинается в режиме вставки. В режиме замены введённый символ помещается в текущую позицию вместо имеющегося. Символы, привязанные к backward-delete-char заменяют символ перед курсором (слава) пробелом. По умолчанию команда не привязана.

8.4.4. Уничтожение и восстановление

kill-line (C-k)
Уничтожает текст от курсора до конца строки.

backward-kill-line (C-x Rubout)
Уничтожает текст от курсора до начала текущей строки.

unix-line-discard (C-u)
Уничтожает текст от курсора до начала текущей строки.

kill-whole-line ()
Уничтожает все символы текущей строки независимо от позиции курсора. По умолчанию не привязана.

kill-word (M-d)
Уничтожает текст от курсора до конда текущего или следующего (если курсор находится между словами) слова. Границы слов такие же как для forward-word.

backward-kill-word (M-DEL)
Уничтожает слово перед курсором. Границы слов такие же как для forward-word.

shell-kill-word ()
Уничтожает текст от курсора до конда текущего или следующего (если курсор находится между словами) слова. Границы слов такие же как для shell-forward-word.

shell-backward-kill-word ()
Уничтожает слово перед курсором. Границы слов такие же как для shell-backward-word.

unix-word-rubout (C-w)
Уничтожает слово перед курсором, используя пробел в качестве границы. Текст сохраняется в kill-ring.

unix-filename-rubout ()
Уничтожает слово перед курсором, используя пробел и / в качестве границы. Текст сохраняется в kill-ring.

delete-horizontal-space ()
Удаляет пробелы и символы табуляции по обе стороны курсора. По умолчанию не привязана.

kill-region ()
Уничтожает текст в текущей области (region). По умолчанию не привязана.

copy-region-as-kill ()
Копирует текст области в kill-буфер с возможностью вставить его позже. По умолчанию не привязана.

copy-backward-word ()
Копирует слово перед курсором в kill-буфер. Границы слов как для backward-word. По умолчанию не привязана.

copy-forward-word ()
Копирует слово после курсора в kill-буфер. Границы слов как для forward-word. По умолчанию не привязана.

yank (C-y)
Помещает верхнюю запись kill-ring в позицию курсора.

yank-pop (M-y)
Прокручивает kill-ring и извлекает новую верхнюю запись. Может использоваться лишь вслед за yank или yank-pop.

[1]Клавиша удаления последнего введённого символа, например, Backspace.

8.4.5. Задание числовых аргументов

digit-argument (M-0, M-1, ... M--)

Добавляет введённую цифру к имеющемуся аргументу или создаёт новый аргумент. M-- создаёт отрицательный аргумент.

universal-argument ()

Обеспечивает другой способ задания аргумента. Если за этой командой следует одна или несколько цифр (возможно со знаком - в начале), эти цифры определяют аргумент. Если за командой следуют цифры, повторное выполнение universal-argument завершает числовой аргумент (в остальных случаях игнорируется). Особым случаем является ввод после команды символа, не являющегося - или цифрой, - число аргументов команды умножается на 4. Изначально имеется один аргумент, поэтому первое выполнение команды увеличивает число аргументов до 4, второе - до 16 и т. д. По умолчанию команда не привязана.

8.4.6. Завершение строк из Readline

complete (TAB)

Пытается завершить текст до точки. Реальное дополнение зависит от приложения. Bash пытается завершить текст как переменную (если он начинается с $), имя пользователя (начинается с ~), имя хоста (начинается с @) или команду (включая псевдонимы и функции). Если ни один из вариантов не обеспечивает соответствия, выполняется завершение имени файла.

possible-completions (M-?)

Список возможных завершений текста до точки. При выводе завершений Readline устанавливает число отображаемых символов в соответствии с completion-display-width, значением переменной COLUMNS или шириной экрана в указанном порядке.

insert-completions (M-*)

Вставляет все завершения текста перед точкой, которые были бы сгенерированы possible-completions.

menu-complete ()

Похожа на complete, но заменяет дополняемое слово одним совпадением из списка возможных дополнений. Повтор дополнения меню из списка возможных дополнений вставляет каждое совпадение по очереди. В конце списка подаётся звуковой сигнал (в зависимости от настройки) и восстанавливается исходный текст. Аргумент n задаёт шаг перемещения вперёд по списку совпадений, при отрицательном значении перемещение происходит назад. Команда предназначена для привязки к клавише TAB, но по умолчанию не привязана.

menu-complete-backward ()

Идентична menu-complete, но с обратным перемещением как при отрицательном аргументе menu-complete.

delete-char-or-list ()

Удаляет символ в позиции курсора, если это не начало и не конец строки (как delete-char). В конце строки идентична possible-completions. Команда по умолчанию не привязана.

complete-filename (M-/)

Пытается завершить текст перед курсором как полное имя файла.

possible-filename-completions (C-x /)

Выводит возможные завершения текста до курсора, считая его именем файла.

complete-username (M-~)

Пытается завершить текст перед курсором как имя пользователя.

possible-username-completions (C-x ~)

Выводит возможные завершения текста до курсора, считая его именем пользователя.

complete-variable (M-$)

Пытается завершить текст перед курсором как имя переменной среды.

possible-variable-completions (C-x $)

Выводит возможные завершения текста до курсора, считая его именем переменной среды.

complete-hostname (M-@)

Пытается завершить текст перед курсором как имя хоста.

possible-hostname-completions (C-x @)

Выводит возможные завершения текста до курсора, считая его именем хоста.

complete-command (M-!)

Пытается завершить текст перед курсором как имя команды, сопоставляя текст с псевдонимами, зарезервированными словами, функциями и внутренними командами оболочки, а также именами исполняемых файлов в указанном порядке.

possible-command-completions (C-x !)

Выводит возможные завершения текста до курсора, считая его именем команды.

dynamic-complete-history (M-TAB)

Пытается завершить текст перед курсором, сравнивая его со строками из списка истории для дополнения.

dabbrev-expand ()

Пытается завершить меню по тексту до курсора, сравнивая его со строками из списка истории для дополнения.

complete-into-braces (M-{)

Дополняет имя файла и вставляет список возможных дополнений в фигурных скобках, чтобы он был доступен для оболочки (3.5.1. Раскрытие скобок).

8.4.7. Клавиатурные макросы

start-kbd-macro (C-x ()

Начинает сохранение введённых символов в текущем клавиатурном макросе.

end-kbd-macro (C-x))

Останавливает сохранение введённых символов в текущем клавиатурном макросе и сохраняет макрос.

call-last-kbd-macro (C-x e)

Заново выполняет определённый последним клавиатурный макрос, как будто символы вводятся с клавиатуры.

print-last-kbd-macro ()

Выводит определённый последним клавиатурный макрос в форме, пригодной для файла inputrc.

8.4.8. Прочие команды

re-read-init-file (C-x C-r)
Читает файл inputrc и включает найденные привязки и назначения переменных.

abort (C-g)
Прерывает текущую команду редактирования с подачей звукового сигнала (в зависимости от bell-style).

do-lowercase-version (M-A, M-B, M-x, ...)
Если символ x имеет верхний регистр, запускается команда, привязанная к соответствующему символу в нижнем регистре. При x в нижнем регистре поведение становится неопределённым.

prefix-meta (ESC)
Добавляет префикс Meta к следующему введённому символу. Это рассчитано на клавиатуры без клавиши Meta. Нажатие ESC f эквивалентно M-f.

undo (C-_ или C-x C-u)
Инкрементный откат, раздельно запоминаемый для каждой строки.

revert-line (M-r)
Отменяет все изменения данной строки, как многократное выполнение команды undo для возврата к началу.

tilde-expand (M-&)
Выполняет преобразование тильды для текущего слова.

set-mark (C-@)
Устанавливает маркер в позицию курсора. При наличии числового аргумента маркер устанавливается в заданную им позицию.

exchange-point-and-mark (C-x C-x)
Меняет местами позицию курсора и маркер. Курсор помещается в сохранённую позицию, а прежняя позиция сохраняется как маркер.

character-search (C-])
Символ считывается и курсор перемещается к следующему вхождению этого символа. Отрицательное значение задаёт поиск предыдущего вхождения.

character-search-backward (M-C-])
Символ считывается и курсор перемещается к предыдущему вхождению этого символа. Отрицательное значение задаёт поиск следующего вхождения.

skip-csi-sequence ()
Считывается достаточное число символов для потребления многосимвольной последовательности, такой как для клавиш Home и End. Последовательности этого типа начинаются с индикатора управляющей последовательности (CSI[1]), которым обычно служит ESC-[. Если последовательность привязана к \e[, создающие её символы не будут давать эффекта, если они явно не связаны с командой readline вместо вставки в буфер редактирования. Команда по умолчанию не привязана, но обычно её связывают с клавишами ESC-[.

insert-comment (M-#)
Без числового аргумента вставляется значение переменной comment-begin в начало текущей строки. При наличии числового аргумента команда действует как переключатель - если символы в начале строки не совпадают с comment-begin, вставляется значение этой переменной, в противном случае значение comment-begin удаляется из начала строки. В любом случае строка воспринимается как при вводе символа новой строки. Принятое по умолчанию значение comment-begin заставляет делать текущую строку комментарием для оболочки. Если числовой аргумент ведёт к удалению комментария, строка будет выполнена оболочкой как команда.

dump-functions ()
Выводит все функции и их привязки в выходной поток Readline. При наличии числового аргумента вывод форматируется так, что его можно включить в файл inputrc. По умолчанию команда не привязана.

dump-variables ()
Выводит все устанавливаемые переменные и их значения в выходной поток Readline. При наличии числового аргумента вывод форматируется так, что его можно включить в файл inputrc. По умолчанию команда не привязана.

dump-macros ()
Выводит все комбинации клавиш Readline, привязанные к макросам, и создаваемые ими строки. При наличии числового аргумента вывод форматируется для включения в файл inputrc. По умолчанию команда не привязана.

glob-complete-word (M-g)
Слово перед курсором трактуется как шаблон преобразования пути с неявным добавлением *. Этот шаблон служит для создания списка совпадающих имён файлов для возможного дополнения.

glob-expand-word (C-x *)
Слово перед курсором считается шаблоном преобразования пути и вместо него вставляется список совпадающих имён файлов. При наличии числового аргумента перед преобразованием имён в конец добавляется *.

glob-list-expansions (C-x g)
Выводится список преобразования, которые будут созданы glob-expand-word и строка печатается заново. При наличии числового аргумента перед преобразованием имён в конец добавляется *.

display-shell-version (C-x C-v)
Выводится информация о версии текущего экземпляра Bash.

shell-expand-line (M-C-e)
Строка преобразуется, как это делает оболочка. Преобразуются псевдонимы и история, а также выполняется преобразование слов оболочкой (3.5. Преобразования).

history-expand-line (M-^)
Выполняет преобразование истории для текущей строки.

magic-space ()
Выполняет преобразование истории для текущей строки и вставляет пробел (9.3. Преобразование истории).

alias-expand-line ()
Выполняет преобразование псевдонимов для текущей строки (6.6. Псевдонимы).

history-and-alias-expand-line ()
Выполняет преобразование истории и псевдонимов для текущей строки.

insert-last-argument (M-. or M-_)
Синоним yank-last-arg.

[1]Control Sequence Indicator.

operate-and-get-next (C-o)

Принимает текущую строку для выполнения и извлекает следующую строку истории, помещая её в текущую строку для редактирования. При наличии числового аргумента он задаёт строку истории, которую следует использовать взамен текущей.

edit-and-execute-command (C-x C-e)

Вызывает редактор текущей строки команды и выполняет результат как команды оболочки. Bash пытается вызывать в качестве редактора $VISUAL, $EDITOR, emacs в указанном порядке.

8.5. Readline в режиме vi

Хотя библиотека Readline не включает полного набора функций редактирования vi, она поддерживает простые операции редактирования строк. Readline в режиме vi ведёт себя в соответствии со стандартом POSIX.

Для интерактивного переключения между режимами редактирования emacs и vi служат команды set -o emacs и set -o vi (4.3.1. Внутренняя команда set). Readline по умолчанию работает в режиме emacs.

При вводе строки в режиме vi используется режим вставки, как при нажатии клавиши i. Клавиша ESC переключает в режим команд, где можно редактировать текст строки с использованием стандартных команд перемещения vi, переходить к предыдущей (k) или следующей (j) строке истории и т. п.

8.6. Программируемое дополнение

При попытке завершения слова в аргументе команды, для которой определена спецификация дополнения (compspec) с использованием внутренней команды complete (8.7. Внутренние функции программируемого дополнения), вызываются функции программируемого дополнения.

Сначала идентифицируется имя команды. Если для команды была определена спецификация compspec, она служит для создания списка возможных дополнений слова. Если слово команды является пустой строкой (попытка дополнения в начале строки), применяется любая спецификация compspec, определённая с опцией -E. Если слово команды является полным путём, сначала отыскивается compspec для полного пути, а при отсутствии предпринимается попытка найти compspec для части, следующей за последним символом /. Если этот поиск не даёт compspec, по умолчанию используется любая спецификация compspec, определённая с опцией -D. Если заданной по умолчанию спецификации compspec нет, Bash пытается преобразовать псевдоним для слова команды, а затем найти compspec для преобразованного слова.

После нахождения спецификации compspec она применяется для создания списка соответствующих слов. Если compspec найти не удалось, выполняется принятое по умолчанию дополнение Bash (8.4.6. Завершение строк из Readline). Сначала применяются действия, заданные compspec, при этом возвращаются только совпадения, которые начинаются с дополняемого слова. При использовании опции -f или -d для дополнения имени файла или каталога, применяется переменная оболочки FIGNORE для фильтрации совпадений (5.2. Переменные Bash).

Затем генерируются совпадения, заданные шаблоном преобразования имён файлов для опции -G. Слова, создаваемые шаблоном не обязаны совпадать с дополняемым словом, переменная оболочки GLOBIGNORE не используется для фильтрации совпадений, но переменная FIGNORE применяется.

Далее рассматривается строка, заданная аргументом опции -W. Сначала строка расщепляется ч использованием в качестве ограничителей символов из переменной IFS. Кавычки в строке принимаются во внимание для обеспечения механизма преобразования слов с метасимволами оболочки или символами из IFS. Затем каждое слово преобразуется с учётом преобразований скобок, тильды параметров и переменных, как описано в параграфе 3.5. Преобразования. Результат расщепляется по правилам параграфа 3.5.7. Расщепление слов.

Результатом преобразования является совпадение начала с дополняемым словом и совпадающие слова становятся возможными дополнениями.

После генерации совпадений вызывается функция или команда оболочки, заданная опциями -F и -C при вызове, при этом переменным COMP_LINE, COMP_POINT, COMP_KEY и COMP_TYPE присваиваются значения, как описано в параграфе 5.2. Переменные Bash. При вызове функции оболочки устанавливаются также переменные COMP_WORDS и COMP_CWORD. При вызове команды или функции первым аргументом ($1) является имя команды, чьи аргументы дополняются, вторым ($2) - дополняемое слово, а третьим ($3) слово, предшествующее дополняемому в строке текущей команды. Дополнения не фильтруются, функция или команда имеют полную свободу генерации совпадений.

Функция, заданная с -F, вызывается первой и может использовать для генерации совпадений любые средства оболочки, включая встроенные функции compgen и compopt, описанные в параграфе 8.7. Внутренние функции программируемого дополнения. Она должна поместить возможные дополнения в переменную (массив) COMPREPLY по одному дополнению в элемент. Затем вызывается команда указанная с -C, в среде эквивалентной подстановке команд. Ей следует вывести список дополнений (по 1 в строке) на стандартный вывод. Для экранирования перевода строки (newline) можно использовать \.

После создания списка всех возможных дополнений к нему применяются фильтры, заданные опцией -X. Фильтр является шаблоном, какие применяются для преобразования имён файлов - символ & в шаблоне заменяется текстом дополняемого слова. Сам символ & может экранироваться символом \, который удаляется до попытки сопоставления. Любое дополнение, соответствующее шаблону, удаляется из списка. Символ ! перед шаблоном инвертирует его и удаляются дополнения, не соответствующие шаблону. Если включена опция nocasematch (4.3.2. Внутренняя команда shopt), регистр символов при сопоставлении не учитывается. В заключение к каждому элементу списка дополнений добавляются префикс и суффикс, заданные опциями -P и -S, и результат возвращается коду завершения Readline как список возможных дополнений.

Если описанные выше действия не привели к созданию списка совпадений и была задана опция -o dirnames для дополнения при заданной спецификации compspec, выполняется попытка дополнить имена каталогов.

Если была указано опция -o plusdirs для дополнения при заданной спецификации compspec, выполняется попытка дополнить имена каталогов и совпадения добавляются к результатам предшествующих действий.

По умолчанию, если найдена спецификация compspec, все созданное этой спецификацией возвращается коду дополнения как полный набор возможных вариантов. Принятые по умолчанию дополнения Bash не применяются и принятое по умолчанию дополнение имён файлов Readline отключено. Если была указана опция -o bashdefault для дополнения с compspec и эта спецификация не дала совпадений, предпринимается попытка принятого по умолчанию дополнения Bash. Если была задана опция -o default для дополнения с compspec и эта спецификация, а также дополнение Bash не дали совпадений, применяется заданное по умолчанию дополнение Readline.

Когда compspec указывает желательность дополнения имён каталогов, программируемые функции дополнения заставляют Readline добавлять символ / к дополненным именам, которые являются символьными ссылками на каталоги, в соответствии со значением переменной mark-directories в Readline, независимо от установки переменной mark-symlinked-directories в Readline.

Имеется некоторая поддержка динамически меняющихся дополнений, которая может быть полезна в сочетании с принятым по умолчанию дополнением, заданным с опцией -D. Для функций оболочки, выполняемых как обработчики дополнений, можно указать потребность в повторе дополнения путём возврата статуса 124. Если функция оболочки возвращает 124 и меняет спецификацию compspec, связанную с командой, для которой выполняется дополнение (первый аргумент при выполнении функции), программируемое дополнение запускается сначала в попытке найти новую спецификацию compspec для этой команды. Это позволяет создавать динамический набор дополнений вместо загрузки всех спецификаций сразу.

Предположим, например, что имеется библиотека спецификаций compspec, каждая из которых хранится в отдельном файле, соответствующем имени команды. Приведённая ниже функция используемого по умолчанию дополнения будет загружать дополнения динамически.

```
_completion_loader()
{
        . "/etc/bash_completion.d/$1.sh" >/dev/null 2>&1 && return 124
}
complete -D -F _completion_loader -o bashdefault -o default
```

8.7. Внутренние функции программируемого дополнения

Для работы с программируемыми функциями дополнения имеется 3 команды, одна из которых задаёт способ дополнения аргументов конкретной команды, а две другие позволяют менять режим дополнения.

compgen

```
compgen [option] [word]
```

Генерирует список возможных дополнений для слова в соответствии с опциями (любые опции внутренней команды complete, кроме -p и -r) и выдаёт совпадения на стандартный вывод. При указании опции -F или -C переменные оболочки, задаваемые различными средствами программируемого дополнения, будут доступны, но бесполезны.
Совпадения будут генерироваться так же, как это делает программируемый код дополнения по спецификации с такими же флагами. Если задан параметр word, будут выводиться лишь соответствующие ему дополнения.
Функция возвращает значение true, если нет непригодных опций и найдены совпадения.

complete

```
complete [-abcdefgjksuv] [-o comp-option] [-DEI] [-A action] [-G globpat]
[-W wordlist] [-F function] [-C command] [-X filterpat]
[-P prefix] [-S suffix] name [name ...]
complete -pr [-DEI] [name ...]
```

Задаёт способ дополнения аргументов для каждого параметра name. При наличии опции -p или полном отсутствии опций выводится имеющаяся спецификация дополнения в форме пригодной для ввода. Опция -r удаляет спецификацию дополнения для каждого значения name или все спецификации, если параметр name не задан. Опция -D указывает, что следует применять другие заданные опции и действия к «принятому по умолчанию» дополнению command, т. е. выполняется дополнение команды, которое ранее не было определено. Опция -E указывает, что следует применять другие заданные опции и действия к дополнению «пустой» команды, т. е. пустой строки command. Опция -I указывает, что следует применять другие заданные опции и действия к дополнению изначально не заданного слова в строке или после разделителя (такого как ; или |), который обычно является завершение имени команды. При наличии множества опций -D имеет приоритет над -E и обе эти опции более приоритетны чем -I. Если задана любая из опций -D, -E, -I, все остальные аргументы name игнорируются и дополнения относятся лишь к заданному опцией случаю.
Процесс применения этих спецификаций дополнения описан в параграфе 8.6. Программируемое дополнение.
Назначение остальных опций описано ниже. Аргументы опций -G, -W и -X (при необходимости и опций -P и -S) следует заключать в кавычки, чтобы предотвратить их преобразование до вызова внутренней команды complete.

-o comp-option

Опция comp-option управляет несколькими аспектами поведения compspec сверх простой генерации дополнений и может принимать одно из приведённых ниже значений.

bashdefault	Выполнять принятые по умолчанию дополнения Bash, если compspec не даёт совпадений.
default	Выполнять принятое по умолчанию дополнение имён файлов Readline, если compspec не даёт совпадений.
dirnames	Выполнять дополнение имён каталогов, если compspec не даёт совпадений.
filenames	Говорит Readline, что генерирует имена файлов, поэтому можно выполнять соответствующую обработку (например, добавление / к именам каталогов, экранирование специальных символов, исключение пробелов в конце). Опция предназначена для использования с функциями оболочки, заданными -F.
noquote	Говорит Readline, что не нужно применять экранирование дополненных имён файлов (принято по умолчанию).
nosort	Говорит Readline, что не нужно сортировать список дополнений по алфавиту.
nospace	Говорит Readline, что не нужно добавлять пробел к слову, дополненному в конце строки (принято по умолчанию).
plusdirs	После создания всех дополнений, заданных compspec, выполняется дополнение имён каталогов и совпадения добавляются к результатам других действий.

-A action

Параметр action может принимать одно из перечисленных ниже значений при создании списка дополнений.

alias	Имена псевдонимов. Может также применяться сокращение -a.
arrayvar	Имена переменных массивов.
binding	Имена привязок клавиш Readline (8.4. Клавиатурные команды Readline).
builtin	Имена внутренних команд оболочки. Может также применяться сокращение -b.
command	Имена команд. Может также применяться сокращение -c.
directory	Имена каталогов. Может также применяться сокращение -d.
disabled	Имена отключённых внутренних функций оболочки.
enabled	Имена включённых внутренних функций оболочки.
export	Имена экспортируемых переменных оболочки. Может также применяться сокращение -e.
file	Имена файлов. Может также применяться сокращение -f.
function	Имена функций оболочки.
group	Имена групп. Может также применяться сокращение -g.
helptopic	Темы справки, принимаемые внутренней командой help (4.2. Внутренние команды bash).
hostname	Имена хостов из файла, заданного переменной оболочки HOSTFILE (5.2. Переменные Bash).
job	Имена заданий, если активно управление заданиями. Может также применяться сокращение -j.
keyword	Зарезервированные слова оболочки. Может также применяться сокращение -k.
running	Имена работающих заданий, если активно управление заданиями.
service	Имена служб. Может также применяться сокращение -s.
setopt	Действительные аргументы опции -o внутренней команды set (4.3.1. Внутренняя команда set).
shopt	Имена опций оболочки, воспринимаемые функцией shopt (4.2. Внутренние команды bash).
signal	Имена сигналов.
stopped	Имена остановленных заданий, если активно управление заданиями.
user	Имена пользователей. Может также применяться сокращение -u.
variable	Имена всех переменных оболочки. Может также применяться сокращение -v.

-C command

Команда command выполняется в среде субоболочки и вывод используется как возможные дополнения.

-F function

Функция оболочки, выполняемая в среде текущей оболочки. При выполнении $1 указывает имя команды, аргументы которой дополняются, $2 - дополняемое слово, $3 - слово, предшествующее дополняемому, как описано в параграфе 8.6. Программируемое дополнение. По завершении список возможных дополнений извлекается как переменная массива COMPREPLY.

-G globpat

Шаблон преобразования имён файлов globpat преобразуется для генерации возможных дополнений.

-P prefix

Префикс prefix добавляется в начале каждого возможного дополнения после применения всех других опций.

-S suffix

Суффикс suffix добавляется в конце каждого возможного дополнения после применения всех других опций.

-W wordlist

Список wordlist расщепляется с использованием символов специальной переменной IFS в качестве разделителей и каждое полученное слово преобразуется. Возможными дополнениями являются элементы списка, соответствующие дополняемому слову.

-X filterpat

Шаблон filterpat служит для преобразования имён файлов и применяется к списку возможных дополнений, созданному предшествующими опциями и аргументами, с удалением всех элементов, соответствующих filterpat. Символ ! А начале filterpat инвертирует операцию и удаляются несоответствующие элементы.

Возвращается значение true, если нет непригодных опций, не представлены без аргументов опции, отличные от -p и -r, не было попыток удалить отсутствующую спецификацию дополнения для имени и не было ошибок при добавлении спецификации дополнения.

compopt

```
compopt [-o option] [-DEI] [+o option] [name]
```

Меняет опции дополнения в соответствии с option для каждого параметра name или для текущего дополнения при отсутствии name. При отсутствии опций выводятся имеющиеся опции для каждого name или текущего дополнения. Возможными значениями параметра option являются пригодные для внутренней функции complete, как описано выше. Опция -D указывает, что следует применять другие заданные опции к «принятому по умолчанию дополнению команды, т. е. предпринимается попытка дополнения команды, которе не было задано раньше. Опция -E указывает, что следует применять другие заданные опции и действия к дополнению «пустой» команды, т. е. пустой строки command. Опция -I указывает, что следует применять другие заданные опции и действия к дополнению изначально не заданного слова в строке или после разделителя (такого как ; или |), который обычно является завершение имени команды. При наличии множества опций -D имеет приоритет над -E и обе эти опции более приоритетны чем -I.

Возвращается значение true, если нет непригодных опций, не было попыток изменить опции для name без спецификации дополнения и не возникло ошибок.

8.8. Пример программируемого дополнения

Наиболее распространенным способом расширения функциональности дополнения за пределы операций complete и compgen является использование функций оболочки с их привязкой к конкретной команде с помощью complete -F.

Приведённая ниже функция обеспечивает дополнения для внутренней команды cd. Это достаточно хороший пример использования shell-функции для дополнения. Функция использует слово, переданное как аргумент $2, для определения имени дополняемого каталога. Можно применять переменную массива COMP_WORDS для индексирования текущего слова.

Функция использует внутренние команды complete и compgen для основной работы, добавляя от внутренней команды Bash cd преобразование тильды (3.5.2. Преобразование тильды), поиск каталогов в $CDPATH (4.1. Внутренние элементы Bourne Shell) и базовую поддержку опции оболочки cdable_vars (4.3.2. Внутренняя команда shopt). Функция

_comp_cd меняет значение IFS, чтобы оно включало лишь символ newline для имён файлов с пробелами и символами табуляции, compgen выводит возможные дополнения по одному в строке.

Возможные дополнения помещаются в переменную массива COMPREPLY (по одному в элемент). Система программируемого дополнения отыскивает дополнения в возвращённом результате.

```
# Функция дополнения для внутренней команды cd на основе функции
# дополнения cd из пакета bash_completion
_comp_cd()
{
        local IFS=$' \t\n'    # нормализация IFS
        local cur _skipdot _cdpath
        local i j k
        # Преобразование тильды в полный путь
        case "$2" in
        \~*)    eval cur="$2" ;;
        *)      cur=$2 ;;
        esac
        # Нет cdpath или абсолютного пути, прямое дополнение каталога
        if [[ -z "${CDPATH:-}" ]] || [[ "$cur" == @(./*|../*|/*) ]]; then
                # compgen выводит пути по одному в строке и может применять цикл while
                IFS=$'\n'
                COMPREPLY=( $(compgen -d -- "$cur") )
                IFS=$' \t\n'
        # CDPATH + каталоги в текущем каталоге, если их нет в CDPATH
        else
                IFS=$'\n'
                _skipdot=false
                # Обработка CDPATH для преобразования пустых имён каталогов в .
                _cdpath=${CDPATH/#:/.:}
                _cdpath=${_cdpath//::/:.:}
                _cdpath=${_cdpath/%:/:.}
                for i in ${_cdpath//:/$'\n'}; do
                        if [[ $i -ef . ]]; then _skipdot=true; fi
                        k="${#COMPREPLY[@]}"
                        for j in $( compgen -d -- "$i/$cur" ); do
                                COMPREPLY[k++]=${j#$i/}       # Вырезание каталога
                        done
                done
                $_skipdot || COMPREPLY+=( $(compgen -d -- "$cur") )
                IFS=$' \t\n'
        fi
        # Имена переменных, если задана подходящая опция и нет дополнений
        if shopt -q cdable_vars && [[ ${#COMPREPLY[@]} -eq 0 ]]; then
                COMPREPLY=( $(compgen -v -- "$cur") )
        fi
        return 0
}
```

Функция дополнения устанавливается с помощью опции -F команды complete

```
# Указывает readline экранировать нужное, добавлять \ к каталогам и
# использовать принятое по умолчанию дополнение bash для других аргументов
complete -o filenames -o nospace -o bashdefault -F _comp_cd cd
```

Чтобы Bash и Readline принимали во внимание другие детали, для этого используются опции Bash и Readline. Опция -o filenames говорит Readline что возможные дополнения следует считать именами файлов и применять экранирование. Эта опция также ведёт к добавлению символа / в имена, которые Readline считает каталогами (поэтому можно расширить _comp_cd для добавления / при использовании каталогов, найденных через CDPATH - Readline не может считать эти дополнения каталогами). Опция -o указывает Readline не добавлять пробел к имени каталога. Опция -o bashdefault включает остальные «принятые по умолчанию» дополнения Bash к набору «принятых по умолчанию» в Readline. Это включает дополнение имён команд, дополнение переменных для слов, начинающихся с {, дополнения с шаблонами преобразования путей (3.5.8. Преобразование имён файлов) и т. д.

После установки в команде complete функция _comp_cd будет вызываться при попытке дополнения слов для cd.

В проекте дополнения bash имеется ещё много примеров для команд GNU, Unix и Linux, которые по умолчанию устанавливаются во многих дистрибутивах GNU/Linux. Созданный Ian Macdonald проект сейчас размещён на сайте http://bash-completion.alioth.debian.org/. Имеются варианты для таких систем как Solaris и Mac OS X. Старые версии пакета дополнения bash распространяются с bash в каталоге examples/complete.

9. Интерактивное использование истории

В этой главе описана работа с библиотекой gnu History в интерактивном режиме.

9.1. Средства Bash History

При включённой опции -o history внутренней команды set (4.3.1. Внутренняя команда set) оболочка предоставляет доступ к истории команд, т. е. списку ранее введённых команд. Значение переменной оболочки HISTSIZE задаёт число записей, хранящихся в списке истории)по умолчанию 500). Оболочка сохраняет команды в списке до преобразования параметров и переменных с учётом значений HISTIGNORE и HISTCONTROL.

При запуске оболочки история инициализируется из файла, указанного в переменной HISTFILE (по умолчанию ~/.bash_history). При необходимости заданный в HISTFILE файл усекается, чтобы число строк в нем не превышало значение переменной HISTFILESIZE. При завершении оболочки со включённой историей последние $HISTSIZE строк копируются из списка истории в файл, заданный в $HISTFILE. Если установлена опция оболочки histappend (4.2. Внутренние команды bash), строки добавляются в конец файла истории, а при выключенной опции файл

перезаписывается. После сохранения истории файл усекается до числа строк, не превышающего значение $HISTFILESIZE. Если переменная HISTFILESIZE не установлена, пуста или содержит нечисловое значение или число меньше 0, файл истории не усекается.

При установленной переменной HISTTIMEFORMAT временная метка, связанная с каждой записью истории сохраняется в файле истории, помеченная символом комментарии. При чтении истории строки, в которых сразу за символом комментария следует цифра, считаются временными метками для следующей записи.

Можно применять внутреннюю команду fc для просмотра, редактирования или повторного выполнения части списка истории. Внутренняя команда history позволяет просматривать и редактировать список, а также манипулировать файлом истории. При редактировании командной строки доступны команды поиска в каждом режиме редактирования, обеспечивающем доступ к списку истории (8.4.2. Команды работы с историей).

Оболочка позволяет управлять сохранением команд в истории с помощью переменных HISTCONTROL и HISTIGNORE. Включение опции оболочки cmdhist ведёт к попытке сохранения всех строк многострочной команды в одной записи истории с добавлением символов ; для обеспечения синтаксической корректности. Опция lithist заставляет оболочку сохранять команды с символами новой строки вместо ;. Для установки этих опций служит внутренняя команда shopt (4.3.2. Внутренняя команда shopt).

9.2. Внутренние команды истории Bash

Bash поддерживает две внутренних команды для манипулирования списком и файлом истории.

fc

```
fc [-e ename] [-lnr] [first] [last]
fc -s [pat=rep] [command]
```

Первая форма выбирает из списка истории команды от first до last и выводит или редактирует и заново выполняет их все. Оба параметра first и last могут быть заданы строками (выбирается наиболее свежая команда, начинающаяся с заданной строки) или номером (индекс списка истории, где отрицательные значения задают смещение от номера текущей команды). Если параметр last не задан, он принимает значение first. Если не задан параметр first, устанавливается предыдущая команда при редактировании и -16 для списка. Флаг -l задаёт печать списка на стандартном устройстве вывода, флаг -n отключает вывод номеров команд в списке, а -r обращает порядок списка. В остальных случаях вызывается редактор, заданные параметром ename, а при отсутствии параметра - редактор, заданный преобразованием переменной ${FCEDIT:-${EDITOR:-vi}}. Это выражение задаёт использования значения переменной FCEDIT (при наличии) или EDITOR (при наличии), а при отсутствии обеих вызывается редактор vi. По завершении редактирования исправленная команда выводится и выполняется.

Во втором варианте команда command выполняется заново после замены каждого pat в выбранной команде на rep. Параметр command интерпретируется так же, как описанный выше параметр first.

Полезным псевдонимом для работы с командой fc является r='fc -s', ввод команды r cc запустит последнюю команду, начинающуюся с cc, а r повторит выполнение последней команды (6.6. Псевдонимы).

history

```
history [n]
history -c
history -d offset
history -d start-end
history [-anrw] [filename]
history -ps arg
```

При запуске без опций выводит список записей истории с номерами строк, указывая префиксом * изменённые строки. Аргумент n ограничивает вывод последними n строками. Если переменная оболочки HISTTIMEFORMAT задана и не пуста, она служит строкой формата для strftime при выводе временных меток записей истории. Метки не отделяются пробелом от строк истории.

Опции команды описаны ниже.

-c	Очищает список истории и может сочетаться с другими опциями для полной замены списка.
-d offset	Удаляет запись, указанную смещением offset. Положительные значения считаются номерами в списке, отрицательные отсчитываются от последней записи + 1, т. е. команда history -d -1 удалит последнюю команду.
-d start-end	Удаляет записи истории между позициями start и end, включительно. Положительные и отрицательные значения трактуются как для offset.
-a	Добавляет новые записи в файл истории. Это будут строки, введённые с начала текущей сессии Bash, но ещё не добавленные в файл.
-n	Добавляет строки, ещё не прочитанные из файла истории в текущий список. Это будут строки, добавленные в конец файла истории с начала текущей сессии Bash.
-r	Читает файл истории и добавляет его содержимое в конец списка.
-w	Записывает текущий список в файл истории.
-p	Выполняет подстановку в истории по arg и печатает результат на стандартном устройстве вывода без сохранения в списке.
-s	Параметр arg добавляется в конец списка как одна запись.

Если задано имя файла при наличии любой из опций -w, -r, -a, -n, оно считается именем файла истории. В остальных случаях используется значение переменной HISTFILE.

9.3. Преобразование истории

Библиотека History обеспечивает функции преобразования истории, похожие на функции оболочки csh. В этом разделе описан синтаксис для работы с данными истории.

Преобразование истории вводит слова из истории во входной поток, упрощая повтор команд, вставку аргументов из предшествующих команд в текущую и быстрое исправления ошибок в предыдущей команде. Преобразование выполняется сразу после прочтения полной строки до её разбиения оболочкой на отдельные слова и происходит отдельно в каждой строке. Bash пытается информировать функции преобразования истории о кавычках из прошлых строк, которые сохраняют действие.

Преобразование состоит из двух частей. Сначала определяется строка из списка истории, которую следует использовать в процессе подстановки. Затем выбирается часть этой строки для включения в текущую строку. Выбор строки из истории называют событием, а используемые части этой строки - словами. Для манипуляция с выбранными словами можно применять различные модификаторы. Строка разбивается на слова обычными методами Bash и несколько слов в кавычках рассматриваются как одно слово. Преобразование истории инициируется специальным символом (по умолчанию !).

При преобразовании используются похожие на применяемые в оболочке соглашения о кавычках. Для отмены специальной трактовки символов перед ним может помещаться символ \, последовательность символов в одинарных кавычках трактуется дословно. Для символов внутри двойных кавычек может применяться преобразование истории, поскольку символу преобразования может предшествовать \, но не символ одинарных кавычек, который не имеет специального значения внутри двойных кавычек. При использовании оболочки лишь символы \ и ’ могут служить для экранирования символа преобразования истории, но этот символ считается заключённым в кавычки также в том случае, когда он непосредственно предшествует закрывающей кавычке в строк с двойными кавычками.

Несколько опций, устанавливаемых внутренней командой shopt (4.3.2. Внутренняя команда shopt), могут применяться для настройки преобразования истории. Если включена опция histverify и применяется Readline, подстановки в истории не передаются сразу же анализатору оболочки и преобразованная строка заново загружается в буфер редактирования Readline для дополнительного изменения. При использовании Readline со включённой опцией histreedit неудачное преобразование истории будет заново загружаться в буфер Readline для исправления. Можно использовать опцию -p внутренней команды history для просмотра преобразования истории перед его использованием. Опция -s команды history позволяет добавлять команды в конец списка истории без их реального выполнения, чтобы они были доступны для последующих вызовов. Это наиболее полезно в сочетании с Readline.

Оболочка позволяет управлять символами, используемыми в механизмах преобразования истории, с помощью переменной histchars (5.2. Переменные Bash). Оболочка использует символ комментария в истории для включения в файл истории временных меток.

9.3.1. Обозначения событий

Указатель события - это ссылка на запись команды в списке истории. Если ссылка не является абсолютной, событие относится к текущей позиции в списке истории.

!
: Начало подстановки истории за исключением тех случаев, когда за восклицательным знаком следует пробел, символ табуляции, завершение строки = или ((если включена опция extglob с помощью внутренней команды shopt).

!n
: Указывает строку команды n.

!-n
: Указывает команду n назад.

!!
: Указывает предыдущую команду (синоним !-1).

!string
: Указывает последнюю команду, предшествующую текущей позиции в списке истории и начинающуюся со string.

!?string[?]
: Указывает последнюю команду, предшествующую текущей позиции в списке истории и содержащую string. Символ ? в конце можно опустить, если непосредственно за ним следует newline.

^string1^string2^
: Быстрая подстановка. Повторяется последняя команда с заменой string1 на string2 (эквивалент !!:s/string1/string2/).

!#
: Набрана все команда.

9.3.2. Обозначения слов

Указатель слов применяется для выбора нужных слов из события. Указатель слова отделяется от спецификации события двоеточием, которое можно опустить, если указатель слова начинается с ^, $, *, - или %. Слова нумеруются от начала строки, начиная с 0 для первого слова. Слова вставляются в текущую строку через пробел. Например,

`!!` указывает предшествующую команду.
`!!:$` указывает последний аргумент предшествующей команды и может сокращаться до !$.
`!fi:2` указывает второй аргумент последней команды, начинающейся с fi.

Указатель слов приведены ниже.

`0` Нулевое слово, которое для многих приложений является словом команды.
`N` N-е слово.
`^` Первый аргумент (слово 1).
`$` Последний аргумент.
`%` Слово, соответствующее последнему поиску ‘?string?’.
`x-y` Диапазон слов (-y является сокращение 0-y).
`*` Все слова кроме нулевого (сокращение 1-$). Использование * не является ошибкой при наличии в событии единственного слова, просто будет возвращена пустая строка.
`x*` Сокращение для x-$
`x-` Сокращение для x-$, похожее на x*, но без возврата последнего слова.

Если указатель слова приведён без спецификации события, в качестве события служит предыдущая команда.

9.3.3. Модификаторы

После необязательного указателя слова можно добавить один или несколько приведённых ниже модификаторов, помещая перед каждым двоеточие (:).

`h` Удаляет конечную часть пути, оставляя только «голову».

t	Удаляет начальные компоненты пути, оставляя только «хвост».
r	Удаляет суффикс .suffix (расширение), оставляя лишь базовое имя.
e	Удаляет все, кроме расширения (суффикса).
p	Выводит новую команду без её выполнения.
q	Помещает подставленные слова в кавычки, экранируя дальнейшую подстановку.
x	Помещает подставленные слова в кавычки (как q), но разбивает их по символам пробела, табуляции и новой строки.
s/old/new/	Подставляет new вместо первого вхождения old в строке события. Вместо / можно использовать любые ограничители. В old и new ограничитель можно экранировать символом \. При наличии символа & в new, он заменяется old. Одиночный символ / экранирует &. Конечный ограничитель не обязателен, если это последний символ в строке ввода.
&	Повторяет предыдущую подстановку.
g	Применяет изменения ко всей строке события и используется вместе с s как в gs/old/new/ или с &.
G	Применяет следующий модификатор s один раз к каждому слову в событии.

10. Установка Bash

В этой главе приведены базовые инструкции по установке Bash на разных поддерживаемых платформах. Оболочка поддерживает операционные системы gnu, практически все версии Unix, а также ряд других систем, таких как BeOS и Interix. Имеются также варианты для ms-dos, os/2 и Windows.

10.1. Базовая установка

Ниже описан простейший вариант установки Bash с компиляцией исходных кодов.

1. Из каталога с исходным кодом оболочки следует ввести команду ./configure для настройки Bash под конкретную систему. При работе с csh или старыми версиями System V может потребоваться команда sh ./configure, чтобы интерпретатор csh не пытался настраивать себя.

   Работа сценария настройки займёт некоторое время и на консоль будут выводиться сообщения о проверки разных возможностей и функций.

2. Команда make запускает компиляцию Bash и сборку сценария bashbug для сообщений об ошибках.

3. Для запуска набора тестов Bash можно ввести команду make tests.

4. Команда make install служит для установки bash и bashbug, а также страниц man и файла Info.

Сценарий configure пытается предсказать корректные значения зависимых от системы переменных в процессе настройки и использует эти значения для создания файлов Makefile в каждом каталоге пакета (основной каталог, builtins, doc и support, каждый каталог в lib и некоторые другие). Создаётся также файл config.h с зависимыми от системы определениями. В заключение создаётся сценарий оболочки config.status, который можно потом использовать для восстановления текущей конфигурации, файл config.cache с результатами проверок для ускорения последующей настройки и config.log с комментированным выводом процесса настройки (полезен в основном для отладки configure). Если config.cache содержит неприемлемые для вас результаты, файл можно удалить или отредактировать.

Для просмотра опций и аргументов настройки служит вводимая из каталога исходного кода Bash команда

```
bash-4.2$ ./configure --help
```

Если нужно собрать Bash в отдельном каталоге за пределами дерева кодов (например при сборке для нескольких вариантов архитектуры) можно указать полный путь к сценарию configure. В приведённом ниже примере bash будет собираться в каталоге /usr/local/build/bash-4.4 с использованием исходного кода из /usr/local/src/bash-4.4.

```
mkdir /usr/local/build/bash-4.4
cd /usr/local/build/bash-4.4
bash /usr/local/src/bash-4.4/configure
make
```

Сборка в отдельном каталоге более подробно описана в параграфе 10.3. Компиляция для разной архитектуры.

Если нужны какие-то особые настройки при компиляции Bash, следует описать свои потребности и отправить по адресу bash-maintainers@gnu.org, чтобы их можно было рассмотреть в следующем выпуске.

Файл configure.ac служит для создания сценария configure программой autoconf. Этот файл потребуется лишь для изменения или повторного создания сценария с использованием новой версии autoconf (не ниже 2.50).

Результаты сборки можно удалить из каталога исходных кодов командой make clean. Для удаления также файлов, созданных сценарием configure (например, перед сборкой Bash для другой системы) служит команда make distclean.

10.2. Компиляторы и опции

В некоторых системах нужны необычные опции компиляции или компоновки, о которых сценарий configure не знает. Можно задать для configure начальные значения переменных путём их установки в среде. Например в оболочках, совместимых с Bourne, можно использовать команды вида

```
CC=c89 CFLAGS=-O2 LIBS=-lPOSIX ./configure
```

В системах с программой env можно воспользоваться командами вида

```
env CPPFLAGS=-I/usr/local/include LDFLAGS=-s ./configure
```

Процесс настройки использует компилятор GCC для сборки Bash, если этот компилятор доступен.

10.3. Компиляция для разной архитектуры

Можно собрать Bash сразу для нескольких типов компьютеров, помещая объектные файлы для каждой архитектуры в свой каталог. Для этого нужна версия make, поддерживающая переменную VPATH, например. GNU make. Для настройки нужно перейти в каталог, куда следует помещать объектные и исполняемые файлы, и выполнить сценарий

configure из каталога с исходным кодом (10.1. Базовая установка). Можно использовать аргумент —srcdir=PATH, который скажет сценарию configure, где расположен исходный код. Сценарий автоматически проверить код в каталоге, находится он сам и в каталоге '..'.

При использовании make без поддержки переменной VPATH, можно собрать Bash для одной архитектуры в каталоге с исходным кодом. После установки Bash для одной архитектуры следует использовать команду make distclean перед сборкой для другой архитектуры.

Если система поддерживает символьные ссылки, можно использовать сценарий support/mkclone для создания дерева сборки с символьными ссылками на каждый файл в каталоге исходных кодов. Ниже приведён пример создания каталога сборки в текущем каталоге из каталога исходных кодов /usr/gnu/src/bash-2.0

```
bash /usr/gnu/src/bash-2.0/support/mkclone -s /usr/gnu/src/bash-2.0 .
```

Для сценария mkclone требуется оболочка Bash, поэтому перед его использованием нужно собрать Bash хотя бы для одной архитектуры перед созданием каталогов сборки для других вариантов архитектуры.

10.4. Каталог для установки

По умолчанию команда make install устанавливает файлы оболочки в каталоги /usr/local/bin, /usr/local/man и т. п. Можно задать другой префикс установки с помощью опции configure —prefix=PATH или задания значения переменной DESTDIR для make при вызове ' make install.

Можно указать отдельные префиксы установки для зависимых и независимых от архитектуры файлов. Если использовать опцию configure --exec-prefix=PATH, команда make install будет использовать значение PATH в качестве префикса для установки программ и библиотек, а документация и другие файлы данных будут размещены с использованием обычного префикса.

10.5. Указание типа системы

Некоторые параметры configure могут не настраиваться автоматически, но в любом случае нужно определить тип хоста, на котором будет работать Bash. Обычно configure может определить тип хоста автоматически, но если выводится сообщение о невозможности определения, следует использовать опцию --host=TYPE. Параметр TYPE может быть коротким именем системы (например, sun4) или каноническим именем с тремя полями CPU-COMPANY-SYSTEM (например, i386-unknown-freebsd4.2). Возможные значения полей указаны в файле support/config.sub.

10.6. Общее использование принятых по умолчанию значений

При желании можно сделать параметры сценария configure общими, создав сценарий config.site с принятыми по умолчанию значениями таких переменных как CC, cache_file, prefix. Сценарий configure ищет такой файл сначала в каталоге PREFIX/share/config.site, затем в PREFIX/etc/config.site. Можно также задать переменную среды CONFIG_SITE, указывающую этот файл. Следует отметить, что сценарий Bash configure ищет этот файл, но это делают не все сценарии configure.

10.7. Управление настройкой конфигурации

Сценария configure распознает перечисленные ниже опции управления его работой.

--cache-file=file Использовать и сохранить результаты тестов в файле file вместо ./config.cache. Указание в качестве файла /dev/null отключает кэширование для отладки configure.

--help Выводит краткую информацию о параметрах configure и завершает работу.

--quiet
--silent
-q Отключает вывод сообщений о выполняемых проверках.

--srcdir=dir Указывает размещение исходного кода Bash в каталоге dir. Обычно configure может определить каталог автоматически.

--version Выводит информацию о версии Autoconf, использованной для создания сценария configure и завершает работу.

Другие опции configure можно увидеть по команде configure --help.

10.8. Дополнительные возможности

Сценарий Bash configure имеет множество опций вида --enable-feature для включения необязательных возможностей Bash. Имеется также несколько опций вида --with-package, где package указывает тот или иной пакет, например, bash-malloc или purify. Для отключения принятого по умолчанию пакета служит опция --without-package, для отключения той или иной включённой по умолчанию функции Bash - --disable-feature.

Ниже приведён полный список опций --enable- и --with-, поддерживаемых сценарием Bash configure.

--with-afs Определяет использование файловой системы AFS[1] для Transarc.

--with-bash-malloc Задаёт использование Bash-версии malloc из каталога lib/malloc. Это отличается от malloc из gnu libc, но старая версия основана на 4.2 bsd malloc. Используемая версия malloc работает очень быстро, но требует большого пространства для каждого размещения. Опция включена по умолчанию. В файле NOTES приведён список систем, для которых опцию следует отключить и configure автоматически отключает опцию для многих систем.

--with-curses Задаёт использование библиотеки curses вместо termcap. Это следует включать для систем, которые не имеют адекватной и полной базы termcap.

--with-gnu-malloc Синоним для --with-bash-malloc.

[1]Andrew File System - распределенная файловая система на основе защищённых серверов.

--with-installed-readline[=PREFIX]	Связывает Bash с локально установленной версией Readline вместо версии из lib/readline и работает только для Readline версии 5.0 и выше. Если PREFIX имеет значение yes или отсутствует, configure использует значения переменных make includedir и libdir, которые по умолчанию являются подкаталогами prefix, для поиска установленной версии Readline, если её нет в стандартных каталогах include и library. Если для PREFIX задано значение no, Bash привязывается к версии из lib/readline. Если PREFIX имеет любое другое значение, считает его именем каталога и ищет установленную версию Readline в его подкаталогах (включаемые файлы в PREFIX/include и библиотеки в PREFIX/lib).
--with-purify	Задаёт использование блока проверки выделения памяти Purify от Rational Software.
--enable-minimal-config	Задаёт сборку оболочки с минимальным набором возможностей, похожей на классическую оболочку Bourne. Имеется несколько опций --enable-, меняющих сборку и компоновку Bash, а не работу оболочки.
--enable-largefile	Включает поддержку больших файлов, если операционная система требует специальных опций компилятора для сборки программ, способных работать с большими файлами. Опция включена по умолчанию, если операционная система поддерживает большие файлы.
--enable-profiling	Задаёт сборку двоичного файла Bash, создающего данные профилировки для обработки gprof при каждом запуске.
--enable-static-link	Задаёт статическую компоновку Bash, если применяется gcc. Опцию можно использовать для сборки root-оболочки.

Опция minimal-config может служить для отключения всех перечисленных ниже опций, но она обрабатывается раньше, поэтому некоторые опции можно включить далее в команде с помощью --enable-feature. Все указанные ниже опции, за исключением --disabled-builtins, --direxpand-default и --xpg-echo-default, по умолчанию включены, если операционная система обеспечивает их поддержку.

--enable-alias	Разрешает преобразование псевдонимов и включает внутренние команды alias и unalias (6.6. Псевдонимы).
--enable-arith-for-command	Включает поддержку дополнительной формы оператора for, аналогичную оператору for в языке C (3.2.4.1. Конструкции с циклами).
--enable-array-variables	Включает поддержку переменных одномерных массивов (6.7. Массивы).
--enable-bang-history	Включает поддержку подстановки истории в стиле csh (9.3. Преобразование истории).
--enable-brace-expansion	Включает преобразование скобок в стиле csh (b{a,b}c → bac bbc). 3.5.1. Раскрытие скобок
--enable-casemod-attributes	Включает поддержку атрибутов, изменяющих регистр внутренней команды declare и операторов присваивания. Например, при назначении переменных символы верхнего регистра могут переводиться в нижний регистр.
--enable-casemod-expansion	Включает поддержку преобразования слов с изменением регистра.
--enable-command-timing	Включает поддержку зарезервированного слова time и отображение временной статистики для конвейеров (3.2.2. Конвейеры). Это позволяет контролировать время для конвейеров, а также внутренних команд и функций.
--enable-cond-command	Включает поддержку условной команды [[. (3.2.4.2. Конструкции с условием).
--enable-cond-regexp	Включает поддержку сопоставлений с регулярными выражениями POSIX, использующих бинарный оператор =~ в условной конструкции [[. (3.2.4.2. Конструкции с условием).
--enable-coprocesses	Включает поддержку копроцессов и зарезервированного слова coproc (3.2.2. Конвейеры).
--enable-debugger	Включает поддержку отладчика bash (распространяется отдельно).
--enable-dev-fd-stat-broken	Включает обход ситуаций, когда вызов stat для /dev/fd/N возвращает результат, отличный от вызова fstat для дескриптора N. Это имеет значение для условных конструкций, проверяющих атрибуты.
--enable-direxpand-default	Включает опцию оболочки direxpand (4.3.2. Внутренняя команда shopt) по умолчанию при запуске оболочки (обычно по умолчанию отключена).
--enable-directory-stack	Включает поддержку стека каталогов в стиле csh и внутренних команд pushd, popd и dirs (6.8. Стек каталогов).
--enable-disabled-builtins	Позволяет вызывать внутренние команды с помощью builtin xxx даже при отключении xxx командой enable -n xxx (см. описание команд builtin и enable в параграфе 4.2. Внутренние команды bash).
--enable-dparen-arithmetic	Включает поддержку команд ((...)) (3.2.4.2. Конструкции с условием).
--enable-extended-glob	Включает поддержку расширенного сопоставления с шаблонами, описанного в параграфе 3.5.8.1. Сопоставление с шаблоном.
--enable-extended-glob-default	Включает по умолчанию опцию оболочки extglob, описанную в параграфе 4.3.2. Внутренняя команда shopt.
--enable-function-import	Включает поддержку импорта определений функций, экспортируемых другим экземпляром оболочки из среды. Опция включена по умолчанию.
--enable-glob-asciirange-default	Включает по умолчанию опцию оболочки globasciiranges, описанную в параграфе 4.3.2. Внутренняя команда shopt. Это управляет поведением диапазонов символов при сравнении с шаблонами при использовании выражений в скобках.
--enable-help-builtin	Включает внутреннюю команду help которая выводит справку о внутренних функциях и переменных оболочки (4.2. Внутренние команды bash).
--enable-history	Включает историю команд, а также внутренние команды fc и history (9.1. Средства Bash History).
--enable-job-control	Включает средства управления заданиями (7. Управление заданиями), если операционная система поддерживает их.

`--enable-multibyte`	Включает поддержку многобайтовых символов, если операционная система поддерживает их.
`--enable-net-redirections`	Включает особую обработку имён файлов вида /dev/tcp/host/port и /dev/udp/host/port в перенаправлении (3.6. Перенаправление).
`--enable-process-substitution`	Включает подстановку процессов (3.5.6. Подстановка процессов), если операционная система поддерживает её.
`--enable-progcomp`	Включает средства программируемого дополнения (8.6. Программируемое дополнение). Опция работает лишь при включённой поддержке Readline.
`--enable-prompt-string-decoding`	Включает интерпретацию ряда символов с \-экранированием в строках приглашений $PS0, $PS1, $PS2 и $PS4 (6.9. Управление формой приглашения).
`--enable-readline`	Включает поддержку редактирования и истории команд с версией Bash библиотеки Readline (8. Редактирование командной строки).
`--enable-restricted`	Включает поддержку ограниченной оболочки, когда Bash при вызове по команде rbash переходит в ограниченный режим (6.10. Ограниченная оболочка).
`--enable-select`	Включает составную команду select, позволяющую создавать простые меню (3.2.4.2. Конструкции с условием).
`--enable-separate-helpfiles`	Задаёт использование внешних файлов документации внутренней командой help вместо встраивания текста справки в программу.
`--enable-single-help-strings`	Задаёт сохранение каждого раздела справочной системы, выводимой внутренней командой help, в форме отдельной строки. Это может помочь при переводе текста. Однако опцию придётся выключить, если компьютер не может корректно выводить длинные строки.
`--enable-strict-POSIX-default`	Включает совместимость Bash с POSIX по умолчанию (6.11. Режим POSIX).
`--enable-usg-echo-default`	Синоним для --enable-xpg-echo-default.
`--enable-xpg-echo-default`	Заставляет встроенную команду echo преобразовывать по умолчанию символы с \-экранированием без опции -e. Это устанавливает для переменной оболочки xpg_echo значение on по умолчанию, что делает команду Bash echo более похожей на версию, заданную спецификацией Single Unix Specification версии 3 (4.2. Внутренние команды bash).

Файл config-top.h содержит операторы препроцессора C #define для опций, которые не устанавливаются из configure. Некоторые из этих опций не предназначены для изменения. При редактировании файла следует внимательно читать комментарии к каждому определению.

Приложение A. Уведомление об ошибках

При обнаружении ошибок в Bash о них следует сообщать. Но сначала нужно убедиться в том, что это действительно ошибка и она присутствует в последней версии Bash, доступной на ftp://ftp.gnu.org/pub/gnu/bash/. Когда ясно, что ошибка действительно имеется, следует воспользоваться командой bashbug для представления сведений. Если ошибку удалось исправить самостоятельно, следует включить исправления в отчёт. Предложения и «философские» сообщения об ошибках следует направлять по адресу bug-bash@gnu.org или в конференцию Usenet gnu.bash.bug.

В отчёт. об ошибке следует включать:

- номер версии Bash;
- указание оборудования и операционной системы;
- компилятор, использованный для сборки Bash;
- описание ошибочного поведения;
- краткий сценарий или «задание» для воспроизведения ошибки.

Программа bashbug три первых элемента определяет автоматически по шаблону сообщения. Информацию следует направлять по адресу bug-bash@gnu.org.

Приложение B. Основные отличия от Bourne Shell

Bash использует в точности такую же грамматику, преобразование слов и переменных, перенаправление и экранирование как Bourne Shell. Bash использует стандарт POSIX в качестве спецификации для реализации этих свойств. Имеются некоторые различия между традиционной оболочкой Bourne и Bash, которые кратко перечислены ниже. Многие из этих различий подробно рассмотрены в предыдущих главах. Для сравнения использована версия sh, включённая в SVR4.2 (последняя версия Bourne).

- Bash следует стандарту POSIX, даже при отличии POSIX от традиционного поведения sh (6.11. Режим POSIX).
- Bash поддерживает многосимвольные опции вызова (6.1. Вызов Bash).
- Bash включает редактирование команд (8. Редактирование командной строки) и внутреннюю команду bind.
- Bash обеспечивает механизм программируемого дополнения слов (8.6. Программируемое дополнение) и внутренние команды complete, compgen, compopt для управления им.
- Bash поддерживает историю команд (9.1. Средства Bash History) и внутренние команды history и fc для работы с ней. Список истории Bash поддерживает временные метки и использует переменную HISTTIMEFORMAT для их вывода.
- Bash реализует преобразование истории в стиле csh (9.3. Преобразование истории).

- Bash поддерживает переменные одномерных массивов (6.7. Массивы), а также соответствующие преобразования назначение переменных для них. Некоторые внутренние команды Bash принимают опции для работы с массивами. Bash имеет множество внутренних переменных-массивов.
- Поддерживается синтаксис $'...', расширяющий \-экранирование ANSI-C внутри одинарных кавычек (3.1.2.4. Кавычки ANSI-C).
- Bash Поддерживает синтаксис $"..." для зависимых от языка трансляций символов в двойных кавычках. Опции вызова -D, --dump-strings и --dump-po-strings выводят транслируемые строки, найденные в сценарии (3.1.2.5. Зависимая от языка трансляция).
- Bash поддерживает ключевое слово ! Для инвертирования значений, возвращаемых конвейерами (3.2.2. Конвейеры). Это очень полезно, когда оператор if должен действовать лишь при отрицательном результате проверки. Опция Bash -o pipefail ведёт к возврату конвейером отказа при отказе любой из его команд.
- Bash поддерживает зарезервированное слово time и синхронизацию команд (3.2.2. Конвейеры). Выводом статистики синхронизации можно управлять с помощью переменной TIMEFORMAT.
- Bash реализует конструкцию for ((expr1 ; expr2 ; expr3)) для команд подобно языку C (3.2.4.1. Конструкции с циклами).
- Bash включает составную команду select для создания простых меню (3.2.4.2. Конструкции с условием).
- Bash включает составную команду [[, которая делает проверку условия частью грамматики (3.2.4.2. Конструкции с условием), включая необязательное сопоставление с регулярным выражением.
- Bash поддерживает необязательное сопоставление без учёта регистра символов в конструкциях case и [[.
- Bash включает преобразование скобок (3.5.1. Раскрытие скобок) и тильды (3.5.2. Преобразование тильды).
- Bash реализует псевдонимы команд и внутренние команды alias и unalias (6.6. Псевдонимы).
- Bash поддерживает арифметику оболочки, составную команду (((3.2.4.2. Конструкции с условием) и арифметические преобразования (6.5. Арифметика командного процессора).
- Переменные из начального окружения оболочки автоматически экспортируются в дочерние процессы. Оболочка Bourne обычно не делает этого без явной маркировки переменных для экспорта.
- Bash поддерживает оператор присваивания += который добавляет значение в конец переменной, указанной слева от оператора.
- Bash включает шаблоны POSIX %, #, %% и ## для удаления подстрок в начале или в конце значений переменных (3.5.3. Преобразование параметров оболочки).
- Поддерживается преобразование ${#xx}, возвращающее размер ${xx} (3.5.3. Преобразование параметров оболочки).
- Имеется расширение ${var:offset[:length]}, которое преобразует подстроку значения var размером length, начиная со смещения offset (3.5.3. Преобразование параметров оболочки).
- Доступно преобразование ${var/[/]pattern[/replacement]}, которое сопоставляет с шаблоном pattern и выполняет замену на replacement в значении переменной var (3.5.3. Преобразование параметров оболочки).
- Доступно преобразование ${!prefix*}, которое извлекает имена всех переменных оболочки, начинающиеся с prefix (3.5.3. Преобразование параметров оболочки).
- Bash поддерживает непрямое преобразование переменных ${!word} (3.5.3. Преобразование параметров оболочки).
- Bash может извлекать позиционные параметры сверх $9, используя ${num}.
- Реализована форма подстановки команд POSIX $() (3.5.4. Подстановка команд), которая предпочтительней формы Bourne ‘‘ (поддерживается для совместимости).
- Bash включает подстановку процессов (3.5.6. Подстановка процессов).
- Bash автоматически назначает переменные, предоставляющие информацию о текущем пользователе (UID, EUID, GROUPS), хосте (HOSTTYPE, OSTYPE, MACHTYPE, HOSTNAME) и работающем экземпляре Bash (BASH, BASH_VERSION, BASH_VERSINFO), как описано в разделе 5.2. Переменные Bash.
- Используется переменная IFS для расщепления лишь результатов преобразования, а не всех слов (3.5.7. Расщепление слов). Это закрывает давно известный дефект защиты.
- Код преобразования имён файлов в скобках использует символы ! и ^ для инвертирования набора символов внутри скобок. Оболочка Bourne использует лишь !.
- Bash реализует полный набор операторов преобразования имён файлов POSIX, включая классы символов, классы эквивалентности и сортировку символов (3.5.8. Преобразование имён файлов).
- Bash реализует расширенное сопоставление с шаблонами при включённой опции оболочки extglob (3.5.8.1. Сопоставление с шаблоном).
- Могут существовать одноимённые переменная и функция. Оболочка sh не разделяет эти пространства имён.
- Функции Bash могут использовать локальные переменные с помощью внутренней команды local, что позволяет создавать рекурсивные функции (4.2. Внутренние команды bash).

- Назначение переменных перед командой влияет лишь на эту команду, внутренние команды и функции (3.7.4. Окружение). В sh все назначения переменных перед командами являются глобальными, если команда не выполняется из файловой системы.
- Bash выполняет преобразование для имён файлов, заданных как операнды для операторов перенаправления ввода и вывода (3.6. Перенаправление).
- Bash включает оператор перенаправления <>, позволяющий открыть файл для чтения и записи, а также оператор &> для перенаправления стандартного вывода и вывод ошибок в один файл (3.6. Перенаправление).
- Bash включает оператор перенаправления <<<, позволяющий использовать строку в качестве стандартного ввода для команды.
- Bash включает операторы перенаправления [n]<&word и [n]>&word, переносящие один дескриптор файла в другой.
- Bash использует особую трактовку некоторых имён файлов в операторах перенаправления (3.6. Перенаправление).
- Bash позволяет создавать сетевые соединения с любыми машинами и службами в операторах перенаправления (3.6. Перенаправление).
- Поддерживается опция noclobber, позволяющая переписывать существующие файлы при перенаправлении вывода (4.3.1. Внутренняя команда set). Оператор >| позволяет переопределить опцию noclobber.
- Внутренние команды Bash cd и pwd (4.1. Внутренние элементы Bourne Shell) принимают опции -L и -P для переключения между физическим и логическим режимом.
- Bash позволяет функциям переопределять одноимённые внутренние команды и обеспечивает доступ к функциональности этих внутренних команд с помощью внутренних команд builtin и command (4.2. Внутренние команды bash).
- Внутренняя команда command позволяет селективно отключать функции при поиске команд (4.2. Внутренние команды bash).
- Внутренние команды можно селективно включать и отключать внутренней командой enable (4.2. Внутренние команды bash).
- Внутренняя команда Bash exec принимает опции, позволяющие пользователям управлять содержимым окружения, передаваемого выполняемой команде, а также передаваемым команде нулевым аргументом (4.1. Внутренние элементы Bourne Shell).
- Функции оболочки могут экспортироваться в дочерние процессы через окружение с помощью export -f (3.3. Функции оболочки).
- Внутренние функции Bash export, readonly и declare могут принимать опцию -f для воздействия на функции оболочки, опцию -p для вывода переменных с различными атрибутами в формате, пригодном для ввода, опцию -n для удаления атрибутов переменных и аргументы name=value для задания одновременно переменных и значений.
- Внутренняя команда Bash hash позволяет связать имя с произвольным именем файла, даже если файл нельзя найти по переменной $PATH, с помощью hash -p (4.1. Внутренние элементы Bourne Shell).
- Bash включает внутреннюю команду help для получения справок о возможностях оболочки (4.2. Внутренние команды bash).
- Внутренняя команда printf обеспечивает форматированный вывод (4.2. Внутренние команды bash).
- Внутренняя команда Bash read (4.2. Внутренние команды bash) будет читать строки, завершающиеся символом \, с опцией -r и использовать переменную REPLY по умолчанию, если нет аргументов, не являющихся опциями. Эта команда также принимает строку приглашения с опцией -p и будет использовать Readline для получения строк, если задана опция -e. Команда read также поддерживает опции управления вводом - -s выключает эхо вводимых символов при их чтении, -t позволяет задать тайм-аут для ввода, -n позволяет считывать лишь заданное число символов, а -d будет читать до ввода определённого символа, а не новой строки.
- Внутренняя команда return может служить для прерывания работы сценариев, вызванных по внутренней команде . или source (4.1. Внутренние элементы Bourne Shell).
- Bash включает внутреннюю функцию shopt для тонкой настройки дополнительных возможностей (4.3.2. Внутренняя команда shopt) и позволяет устанавливать и сбрасывать опции при вызове оболочки (6.1. Вызов Bash).
- Bash обеспечивает множество опций контроля поведения с помощью внутренней команды set (4.3.1. Внутренняя команда set).
- Опция -x (xtrace) выводит команды, отличающиеся от простых, при трассировке выполнения (4.3.1. Внутренняя команда set).
- Внутренняя команда test (4.1. Внутренние элементы Bourne Shell) несколько отличается, поскольку она реализует алгоритм POSIX, определяющий поведение на основе множества аргументов.
- Bash включает внутреннюю команду caller, выводящую контекст любого активного вызова подпрограммы (shell-функции или сценария, выполняемого по команде . или source). Это позволяет отлаживать bash.
- Внутренняя команда trap (4.1. Внутренние элементы Bourne Shell) позволяет задать псевдосигнал DEBUG, похожий на EXIT. Команды с ловушкой DEBUG выполняются перед каждой простой командой, командами for,

case и select, каждой арифметической командой и первой командой, выполняемой в shell-функции. Ловушка DEBUG не наследуется функциями оболочки, если им не назначен атрибут trace или не включена опция functrace с помощью внутренней команды shopt. Опция оболочки extdebug оказывает дополнительное влияние на DEBUG.

Внутренняя функция trap (4.1. Внутренние элементы Bourne Shell) позволяет задать псевдосигнал ERR, похожий на EXIT и DEBUG. Команды, заданные с ловушкой ERR, выполняются после отказов простых команд с некоторыми исключениями. Ловушка ERR не наследуется функциями оболочки, если не включена опция -o внутренней функцией set.

Внутренняя команда trap (4.1. Внутренние элементы Bourne Shell) позволяет задать псевдосигнал RETURN, похожий на EXIT и DEBUG. Команды, заданные с ловушкой RETURN, выполняются перед возобновлением исполнения после возврата из shell-функции или сценария, запущенного командой . или source. Ловушка RETURN не наследуется функциями оболочки, если им не назначен атрибут trace или не включена опция functrace с помощью внутренней команды shopt.

- Внутренняя команда Bash type расширена и даёт больше информации о найденных именах (4.2. Внутренние команды bash).
- Внутренняя команда Bash umask поддерживает опцию -p для вывода в формате пригодном для ввода (4.1. Внутренние элементы Bourne Shell).
- Bash реализует стек каталогов в стиле csh и внутренние команды pushd, popd, dirs для работы с ним (6.8. Стек каталогов). Bash также позволяет просматривать стек каталогов как значение переменной оболочки DIRSTACK.
- Bash интерпретирует специальные символы с \-экранированием в строке приглашения при интерактивном режиме (6.9. Управление формой приглашения).
- Ограниченный режим в Bash (6.10. Ограниченная оболочка) более полезен, нежели в SVR4.2.
- Внутренняя команда disown может удалять задания из внутренней таблицы оболочки (7.1. Основы управления заданиями) или предотвращать отправку заданиям сигналов SIGHUP при завершении оболочки по сигналу SIGHUP.
- Bash включает множество возможностей для отладки сценариев оболочки.
- Оболочка SVR4.2 имеет две связанных с привилегиями внутренних команды (mldmode и priv), отсутствующих в Bash.
- Bash не имеет внутренних команд stop и newgrp.
- Bash не использует переменную SHACCT и не выполняет учёта.
- SVR4.2 sh использует переменную TIMEOUT, в Bash - TMOUT.

Уникальные для Bash возможности описаны в главе 6. Свойства Bash.

B.1 Отличия от оболочки SVR4.2

Поскольку Bash является совершенно новой реализацией, здесь нет многих ограничений, присущих оболочке SVR4.2.

- Bash не создаёт субоболочку при переходе в структуру управления (или из неё), такую как оператор if или while.
- Bash не разрешает несбалансированные кавычки. Оболочка SVR4.2 в некоторых случаях просто вставляет недостающие кавычки перед EOF, что может приводить к трудно обнаруживаемым ошибкам.
- Оболочка SVR4.2 использует схему управления памятью в стиле «барокко» (baroque) на основе ловушек SIGSEGV. Если оболочка запускается из процесса с блокировкой SIGSEGV (например, путём вызова функции system() из библиотеки C), она ведёт себя некорректно.
- В сомнительной попытке обеспечения безопасности оболочка SVR4.2 при вызове без опции -p меняет действительные и эффективные значения uid и gid, если они меньше некого порогового значения (обычно 100). Это может приводить к непредсказуемым результатам.
- Оболочка SVR4.2 не разрешает пользователям ловушки SIGSEGV, SIGALRM, SIGCHLD.
- Оболочка SVR4.2 не разрешает пользователям отменять переменные IFS, MAILCHECK, PATH, PS1, PS2.
- Оболочка SVR4.2 трактует ^ как недокументированный эквивалент |.
- Bash допускает использование при вызове нескольких аргументов (-x -v), а SVR4.2 - только один (-xv). Фактически некоторые версии оболочки завершают работу аварийно (dump core), если второй аргумент начинается с -.
- Оболочка SVR4.2 выходит из сценария при отказе встроенной команды, а Bash - только при отказе некоторых особых внутренних команд POSIX и лишь при определённых обстоятельствах, указанных в стандарте POSIX.
- Оболочка SVR4.2 иначе ведёт себя при вызове как jsh (включает управление заданиями).

Приложение C. GNU Free Documentation License

Version 1.3, 3 November 2008

http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document free in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ascii without markup, Texinfo input format, LaTEX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be

used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C) year your name.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled ``GNU Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with the Front-Cover Texts being list, and with the Back-Cover Texts being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru