Internet Engineering Task Force (IETF)
Request for Comments: 8792
Category: Informational
ISSN: 2070-1721

K. Watsen
Watsen Networks
E. Auerswald
Individual Contributor
A. Farrel
Old Dog Consulting
Q. Wu
Huawei Technologies

June 2020

Handling Long Lines in Content of Internet-Drafts and RFCs

Обработка длинных строк в Internet-Draft и RFC

Аннотация

Этот документ определяет две стратегии обработки длинных строк в документах с ограниченной шириной текста. Одна стратегия (single backslash) основана на сложившейся практике применения символа \ для указания раздела строки с продолжением с первого непробельного ( ) символа следующей строки. Другая стратегия (double backslash) расширяет первую, добавляя ещё один символ \ для указания продолжения, что позволяет обрабатывать более сложные ситуации. В обеих стратегиях применяется самоописывающий заголовок, позволяющий автоматически восстанавливать исходное содержимое.

Статус документа

Документ не содержит стандарта Internet и публикуется с информациоными целями.

Документ является результатом работы IETF[1] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[2]. Не все документы, одобренные IESG претендуют на статус Internet Standard. Дополнительную информацию о стандартах Internet можно найти в разделе 2 в RFC 7841.

Информация о текущем статусе документа, найденных ошибках и способах обратной связи доступна по ссылке https://www.rfc-editor.org/info/rfc8792.

Оглавление

[1]Internet Engineering Task Force - комиссия по решению инженерных задач Internet.
[2]Internet Engineering Steering Group - комиссия по инженерным разработкам Internet.

1. Введение

[RFC7994] задаёт требования к текстовым документам RFC и указывает, что каждая строка RFC (и Internet-Draft) должна быть не длиннее 72 символов, после чего должен следовать characters followed by the character sequence that denotes an end-of-line (EOL).

В Internet-Draft и RFC часто включаются примеры текста и фрагменты кода, в которых строки могут включать более 72 символов. Утилита xml2rfc [xml2rfc] на момент публикации этого документа не пыталась фальцевать (wrap)содержимое таких включений, а просто выдавала предупреждения для строк длиннее 69 символов. Исторически сложилось так, что RFC Editor не давал каких-либо рекомендаций по обработке длинных строк в таких включениях, кроме совета авторам указывать произведенные манипуляции.

Этот документ определяет две стратегии обработки длинных строк в документах с ограниченной шириной текста. Одна стратегия (single backslash) основана на сложившейся практике применения символа \ для указания раздела строки с продолжением с первого непробельного ( ) символа следующей строки. Другая стратегия (double backslash) расширяет первую, добавляя ещё один символ \ для указания продолжения, что позволяет обрабатывать более сложные ситуации. В обеих стратегиях применяется самоописывающий заголовок, позволяющий автоматически восстанавливать исходное содержимое.

Описанные в документе стратегии работают с любым текстовым содержимым, но предназначены в основном для структурированных последовательностей строк, указанных элементом <sourcecode>, определённым в параграфе 2.48 [RFC7991], а не «двухмерных» изображений, таких как отмеченные элементом <artwork>, определенным в параграфе 2.5 [RFC7991].

Отметим, что текстовые файлы представляются строками, начинающимися с позиции 1 и имеющими размер N символов, где последний символ имеет номер N и сразу за ним следуют символы завершения строки.

2. Заявление о применимости

Описанные здесь форматы и алгоритмы могут применяться в любом контексте, где требуется структурированная фальцовка документов IETF и иных документов.

В рамках IETF эта работа нацелена, прежде всего, на элемент xml2rfc v3 <sourcecode> (параграф 2.48 в [RFC7991]) и элемент xml2rfc v2 <artwork> (параграф 2.5 в [RFC7749]), которые, за неимением лучшего варианта, используются в xml2rfc v2 для исходного кода и иллюстраций. Эта работа может применяться и с элементом xml2rfc v3 <artwork> (параграф 2.5 в [RFC7991]), но, как указано в параграфе 5.1, в общем случае это не рекомендуется.

3. Уровни требований

Ключевые слова **должно** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не следует** (SHALL NOT), **следует** (SHOULD), **не нужно** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **не рекомендуется** (NOT RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе интерпретируются в соответствии с BCP 14 [RFC2119] [RFC8174] тогда и только тогда, когда они выделены шрифтом, как показано здесь.

4. Цели

4.1. Автоматическая фальцовка длинных строк тестового содержимого

Автоматическая фальцовка длинных строк нужна для поддержки документов, которые компилируются динамически с включением строк, которые могут быть не ограничены по размеру. Например, процесс сборки может включать текст из других локальных файлов или динамически создаваемое содержимое от внешних процессов. Оба эти случая рассмотрены ниже.

Во многие документы нужно включать содержимое из локальных файлов (например, XML, JSON, ABNF, ASN.1). Перед включением содержимого процессу сборки **следует** сначала проверить эти файлы с помощью соответствующих формату инструментов. Чтобы такие инструменты могли обработать файлы, те должны быть в своём исходном состоянии, что может предполагать наличие длинных строк. Таким образом, эти файлы потребуется сфальцевать перед включением в документ XML, чтобы выполнить ограничения по размеру строк в xml2rfc.

Точно так же документы могут включать динамически создаваемый вывод обычно от внешних процессов, работающих с такими же исходными файлами, которые упомянуты выше. Например, процесс может преобразовывать входной формат в иной, или отображать отчёт, представление или другой файл. В некоторых случаях создаваемый динамически вывод может включать строки, выходящие за пределы ограничения xml2rfc.

В обоих случаях нужна фальцовка и её **следует** автомтизировать для сокращения объёма работы и числа ошибок, возникающих при ручной обработке.

4.2. Автоматическое восстановление исходного содержимого

Автоматическое восстановление исходного содержимого требуется для проверки извлеченного из документа текста.

Например, модули YANG [RFC7950] извлекаются из Internet-Draft и проверяются в процессе представления. Кроме того, обсуждалось желание проверять экземпляры примеров (например, документов XML и JSON), содержащихся в Internet-Drafts [yang-doctors-thread].

5. Ограничения

5.1. Не рекомендуется для графических элементов

Хотя представленное здесь решение работает с текстовым содержимым любого типа, оно наиболее полезно для представления исходного кода (XML, JSON и т. п.) или, в более общем случае, содержимого, которое не простирается в двух измерениях (например, диаграммы).

По сути, проблема состоит в читаемости содержимого после фальцовки. Непредсказуемый текст особенно плохо выгляти в сфальцованном виде и в эту категорию попадает большинство диаграмм унифицированного языка моделирования (Unified Modeling Language или UML), деревьев YANG, «рисунков» ASCII в целом.

Не рекомендуется применять представленное в документе решение для графических элементов.

5.2. Работает не так хорошо, как связанные с форматом опции

Представленное здесь решение в целом подходит для любого текстового содержимого, поскольку то отображается в форме обычного текста. Однако различные форматы иногда имеют встроенные механизмы для более эффективного предотвращения длинных строк.

Например, утилиты pyang [pyang] и yanglint [yanglint] имеют опцию команды tree-line-length, которая позволяет указать желаемый максимальный размер строки при генерации диаграмм деревьев YANG [RFC8340].

В другом случае некоторые исходные форматы (например, YANG [RFC7950]) позволяют любую строку в кавычках разбить на несколько подстрок, разделённых символом конкатенации (например, +), каждая из которых может размещаться в отдельной строке текста.

Авторам **рекомендуется** использовать возможности выбранного формата для предотвращения длинных строк.

6. Две стратегии фальцовки

Этот документ задаёт две похожих стратегии фальцовки текстового содержимого.

Single Backslash ('\')
: Используется символ обратной дробной черты (\) в конце строки, где происходит разбиение и предполагается, что строка продолжается с первого непробельного ( ) символа следующей строки.

Double Backslash ('\\')
: Используется символ \ в конце строки, где происходит разбиение и предполагается, что строка продолжается с первого символа после \ в следующей строке.

6.1. Сравнение

Первая стратегия создаёт более удобочитаемый результат. Однако (1) весьма вероятно встретить нефальцуемый ввод (например, длинная строка пробелов) и (2) длинные строки, которые можно автоматически сфальцевать, могут приводить к необходимости специальной обработки для предотвращения ошибок.

Вторая стратегия создаёт менее удобочитаемый вывод, но она вряд ли столкнётся с нефальцуемыми данными, нет длинных строк, которые нельзя разделить и не требует специальной обоаботки.

6.2. Рекомендация

Реализациям **рекомендуется** сначала использовать стратегию с одним символом \, и лишь в маловероятном случае её непригодности, когда фальцовка не удаётся или логика не может справиться с непредвиденными обстоятельствами, возникающими в желаемой точке разделения строки, переходить ко второй стратегии (\\).

7. Стратегия Single Backslash (\)

7.1. Сфальцованная структура

Содержимое, фальцуемое с применением этой стратегии, **должно** соответствовать описанной ниже структуре.

7.1.1. Заголовок

Заголовок включает 2 строки. Первая строка содержит приведённую ниже строку из 36 символов, которая **может** быть окружена любыми печатаемыми символами. Сама эта строка не может быть разделена.

```
NOTE: '\' line wrapping per RFC 8792[1]
```

[1]Разбиение (перенос строки) в соответствии с RFC 8792.

Вторая строка является пустой, т. е. содержит лишь символы новой строки. Это предназначено для визуального выделения и удобочитаемости.

7.1.2. Тело

Кодирование символов совпадает с описанным в разделе 2 [RFC7994], за исключением того, что [RFC7991] запрещает применение символов табуляции.

Строки с символом \ в конце считаются фальцуемыми (разделяемыми).

Очень длинные строки **можно** делить на множество частей.

7.2. Алгоритм

В этом параграфе описан процесс разделения (folding) и сборки (unfolding) длинных строк, когда они встречаются в текстовом содержимом. Процесс описан полностью, но реализации **могут** применять свою последовательность действий.

Когда в большом документе неоднократно встречается содержимое, требующее разделения и сборки, другой процесс должен извлечь (или вставить) отдельные блоки с длинными строками до использования описанных здесь алгоритмов. Например, это может делать утилита xiax [xiax].

7.2.1. Разделение

Определяется желаемый размер строки от ввода до процесса разделения (line-wrapping), например из параметра командной строки. Если значение не задано явно, **следует** использовать размер 69 символов. Желаемый максимум должен быть не меньше размера заголовка, составляющего 36 символов. Если желаемый размер меньше этого значения, процесс завершается (содержимое не фальцуется).

Текст сканируется в поиске символов горизонтальной табуляции, которые при обнаружении заменяются пробелами или работа завершается с рекомендацией поставщику входных данных сначала преобразовать табуляцию в пробелы.

Текст сканируется в поиске хотя бы одной строки, превышающей по размеру желаемый максимум. Если таких строк не найдено, работа завершается (фальцовка не нужна).

Сканируется текст в поиске строк, завершающихся символом \, которые могут приводить к неоднозначному результату. Если такая строка найдена и её размер меньше желаемого максимума, строку **следует** пометить для принудительной (forced) фальцовки (хотя она не требуется). Если реализация не поддерживает принудительную фальцовку, работа **должна** завершаться.

Если содержимое требует фальцовки и она возможна, в него помещается заголовок (7.1.1. Заголовок) так чтобы вместе с окружающими заголовок символами размер строки не превышал желаемый максимум.

Для каждой строки текста (сверху вниз), если строка превышает по размеру желаемый максимум или требует принудительной фальцовки, выполняются указанные ниже действия.

1. Определяется точка разделения строки, которая **должна** размещаться не дальше желаемого максимума от начала строки и **недопустимо** выбирать для разделения точку строки, за которой сразу следует символ пробела ( ). При форсированной фальцовке выбирается точка между символом \ и символами новой строки. Если нужной точки не найдено, работа завершается (фальцовка невозможна).
2. В найденное место строки помещается символ \ и символы завершения строки.
3. В следующую помещается любое число символов пробела ( ), но результат должен быть не больше желаемого максимума.

В результате предыдущей операции получается следующая строка с произвольным числом символов пробела в начале, за которыми следуют символы, размещённые в исходной строке после точки разделения.

Описанные выше действия продолжаются до завершения текстового содержимого. Отметим, что алгоритм подходит для случаев, когда после первого разделения оставшаяся часть строки длиннее желаемого ваксимума и должна быть разделена снова. Этот процесс может продолжаться неограниченно.

Описанный здесь процесс иллюстрирует функция fold_it_1() в Приложении A.

7.2.2. Сборка

Сканируется начало текстового содержимого в поиске заголовка (7.1.1. Заголовок). При отсутствии заголовка работа завершается (сбора не нужна).

Удаляются две строки заголовка из текстового содержимого.

Для каждой строки тестового содержимого (сверху вниз) выполняется поиск символа \ в конце строки и при обнаружении выполяется объединение с последующей строкой и удаление символа \ вместе с символами завершения строки и символами пробела в начале следующей строки. Это продолжается для следующих строк, образующих разделенную на части строку исходного соедержимого (если они имеются).

Операции продолжаются до конца текстового содержимого.

Пример процесса показан в функции unfold_it_1() Приложения A.

8. Стратегия Double Backslash (\\)

8.1. Сфальцованная структура

Содержимое, фальцуемое с применением этой стратегии, **должно** соответствовать описанной ниже структуре.

8.1.1. Заголовок

Заголовок включает 2 строки. Первая строка содержит приведённую ниже строку из 37 символов, которая может быть окружена любыми печатаемыми символами. Сама эта строка не может быть разделена.

```
NOTE: '\\' line wrapping per RFC 8792[1]
```

Вторая строка является пустой, т. е. содержит лишь символы новой строки. Это предназначено для визуального выделения и удобочитаемости.

8.1.2. Тело

Кодирование символов совпадает с описанным в разделе 2 [RFC7994], за исключением того, что [RFC7991] запрещает применение символов табуляции.

Строки с символом \ в конце, за которым сразу же следуют символы перевода строки, а следующая строка начинается с символа \, считаются фальцуемыми (разделяемыми).

Очень длинные строки **можно** делить на множество частей.

8.2. Алгоритм

В этом параграфе описан процесс разделения (folding) и сборки (unfolding) длинных строк, когда они встречаются в текстовом содержимом. Процесс описан полностью, но реализации **могут** применять свою последовательность действий.

Когда в большом документе неоднократно встречается содержимое, требующее разделения и сборки, другой процесс должен извлечь (или вставить) отдельные блоки с длинными строками до использования описанных здесь алгоритмов. Например, это может делать утилита xiax [xiax].

8.2.1. Разделение

Определяется желаемый размер строки от ввода до процесса разделения (line-wrapping), например из параметра командной строки. Если значение не задано явно, **следует** использовать размер 69 символов. Желаемый максимум должен быть не меньше размера заголовка, составляющего 37 символов. Если желаемый размер меньше этого значения, процесс завершается (содержимое не фальцуется).

Текст сканируется в поиске символов горизонтальной табуляции, которые при обнаружении заменяются пробелами или работа завершается с рекомендацией поставщику входных данных сначала преобразовать табуляцию в пробелы.

Текст сканируется в поиске хотя бы одной строки, превышающей по размеру желаемый максимум. Если таких строк не найдено, работа завершается (фальцовка не нужна).

Сканируется текст в поиске строк, завершающихся символом \, за которой следует строка, начинающаяся с символа \, что могут приводить к неоднозначному результату. Если такая строка найдена и её размер меньше желаемого максимума, строку **следует** пометить для принудительной (forced) фальцовки (хотя она не требуется). Если реализация не поддерживает принудительную фальцовку, работа **должна** завершаться.

Если содержимое требует фальцовки и она возможна, в него помещается заголовок (8.1.1. Заголовок) так чтобы вместе с окружающими заголовок символами размер строки не превышал желаемый максимум.

Для каждой строки текста (сверху вниз), если строка превышает по размеру желаемый максимум или требует принудительной фальцовки, выполняются указанные ниже действия.

1. Определяется точка разделения строки, которая **должна** размещаться не дальше желаемого максимума от начала строки. При форсированной фальцовке выбирается точка между символом \ и символами новой строки. Если нужной точки не найдено, работа завершается (фальцовка невозможна).
2. В найденное место строки помещается символ \ и символы завершения строки.
3. В следующую помещается любое число символов пробела ( ), но результат должен быть не больше желаемого максимума, затем помещается символ \.

В результате предыдущей операции получается следующая строка с произвольным числом символов пробела в начале, за которыми следует символ \ и символы, размещённые в исходной строке после точки разделения.

Описанные выше действия продолжаются до завершения текстового содержимого. Отметим, что алгоритм подходит для случаев, когда после первого разделения оставшаяся часть строки длиннее желаемого ваксимума и должна быть разделена снова. Этот процесс может продолжаться неограниченно.

Описанный здесь процесс иллюстрирует функция fold_it_2() в Приложении A.

8.2.2. Сборка

Сканируется начало текстового содержимого в поиске заголовка (8.1.1. Заголовок). При отсутствии заголовка работа завершается (сбора не нужна).

Удаляются две строки заголовка из текстового содержимого.

Для каждой строки тестового содержимого (сверху вниз) выполняется поиск символа \ в конце строки и символа \ в начале следующей строки, а при обнаружении выполяется объединение с последующей строкой и удаление символа \ вместе с символами завершения строки, \ и символами пробела в начале следующей строки. Это продолжается для следующих строк, образующих разделенную на части строку исходного соедержимого (если они имеются).

Операции продолжаются до конца текстового содержимого.

Пример процесса показан в функции unfold_it_2() Приложения A.

[1]Разбиение (перенос строки) в соответствии с RFC 8792.

9. Примеры

Ниже приведены самодокументирующие примеры фальцовки текстового содержимого. Исходный текст не представлен здесь, поскольку он был бы сфальцован. Поэтому приведены лишь результаты.

9.1. Пример с граничными условиями

Входные данные содержат 7 строк, каждая из которых длиннее предыдущей. Числа служат для удобства учёта. Использовано принятое по умолчанию значение желаемого максимума - 69.

9.1.1. Использование \

```
========== NOTE: '\' line wrapping per RFC 8792 ===========

1234567890123456789012345678901234567890123456789012345678901234567890123456
1234567890123456789012345678901234567890123456789012345678901234567890123456\
7
1234567890123456789012345678901234567890123456789012345678901234567890123456\
78
1234567890123456789012345678901234567890123456789012345678901234567890123456\
789
1234567890123456789012345678901234567890123456789012345678901234567890123456\
7890
1234567890123456789012345678901234567890123456789012345678901234567890123456\
78901
1234567890123456789012345678901234567890123456789012345678901234567890123456\
789012
```

9.1.2. Использование \\

```
========== NOTE: '\\' line wrapping per RFC 8792 ==========

1234567890123456789012345678901234567890123456789012345678901234567890123456
1234567890123456789012345678901234567890123456789012345678901234567890123456\
7
1234567890123456789012345678901234567890123456789012345678901234567890123456\
78
1234567890123456789012345678901234567890123456789012345678901234567890123456\
789
1234567890123456789012345678901234567890123456789012345678901234567890123456\
7890
1234567890123456789012345678901234567890123456789012345678901234567890123456\
78901
1234567890123456789012345678901234567890123456789012345678901234567890123456\
789012
```

9.2. Пример с несколькими частями одной строки

Этот пример показывает разделение очень длинной строки на несколько частей. Входная строка содержит 280 символов. Числа служат для удобства учёта. Использовано принятое по умолчанию значение желаемого максимума - 69.

9.2.1. Использование \

```
========== NOTE: '\' line wrapping per RFC 8792 ===========

1234567890123456789012345678901234567890123456789012345678901234567890123456\
9012345678901234567890123456789012345678901234567890123456789012345678901234\
7890123456789012345678901234567890123456789012345678901234567890123456789012\
5678901234567890123456789012345678901234567890123456789012345678901234567890\
34567890
```

9.2.2. Использование \\

```
========== NOTE: '\\' line wrapping per RFC 8792 ==========

1234567890123456789012345678901234567890123456789012345678901234567890123456\
\9012345678901234567890123456789012345678901234567890123456789012345678901234\
\6789012345678901234567890123456789012345678901234567890123456789012345678901\
\3456789012345678901234567890123456789012345678901234567890123456789012345678\
\01234567890
```

9.3. Примеры «эффективной» фальцовки

Этот пример показывает улучшение читаемости за счёт «эффективной» фальцовки, которая выполняется в удобных для формата точках. Фальцовка была выполнена вручную, поскольку сценарий из Приложения A не реализует «эффективной» фальцовки. Отметим, что заголовки окружены разными печатаемыми символы, которые не использовались в созданных сценарием примерах.

9.3.1. Использование \

```
[NOTE: '\' line wrapping per RFC 8792]
<yang-library
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"
    xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">

  <module-set>
    <name>config-modules</name>
    <module>
      <name>ietf-interfaces</name>
      <revision>2018-02-20</revision>
      <namespace>\
        urn:ietf:params:xml:ns:yang:ietf-interfaces\
      </namespace>
    </module>
```

```
      ...
    </module-set>
    ...
  </yang-library>
```

Ниже приведён эквивалентный текст, созданный с использованием сценария из Приложения А.

```
  ========== NOTE: '\' line wrapping per RFC 8792 ===========
  <yang-library
      xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"
      xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">

    <module-set>
      <name>config-modules</name>
      <module>
        <name>ietf-interfaces</name>
        <revision>2018-02-20</revision>
        <namespace>urn:ietf:params:xml:ns:yang:ietf-interfaces</namesp\
  ace>
      </module>
      ...
    </module-set>
    ...
  </yang-library>
```

9.3.2. Использование \\

```
[NOTE: '\\' line wrapping per RFC 8792]
  <yang-library
      xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"
      xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">

    <module-set>
      <name>config-modules</name>
      <module>
        <name>ietf-interfaces</name>
        <revision>2018-02-20</revision>
        <namespace>\
          \urn:ietf:params:xml:ns:yang:ietf-interfaces\
        \</namespace>
      </module>
      ...
    </module-set>
    ...
  </yang-library>
```

Ниже приведён эквивалентный текст, созданный с использованием сценария из Приложения А.

```
  ========== NOTE: '\\' line wrapping per RFC 8792 ==========

  <yang-library
      xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"
      xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">

    <module-set>
      <name>config-modules</name>
      <module>
        <name>ietf-interfaces</name>
        <revision>2018-02-20</revision>
        <namespace>urn:ietf:params:xml:ns:yang:ietf-interfaces</namesp\
  \ace>
      </module>
      ...
    </module-set>
    ...
  </yang-library>
```

9.4. Пример принудительной фальцовки

Здесь показано, как можно обработать «неподходящие» последовательности с использованием принудительной фальцовки, когда разделение строки выполняется, даже если оно не нужно.

```
  Следующая строка длиннее 68 символов и её нужно фальцевать.
  1234567890123456789012345678901234567890123456789012345678901234567890123456789

  Эта строка завершается символом обратной дробной черты \

  Эта строка завершается символом обратной дробной черты \
  \ Эта строка начинается символом обратной дробной черты

  Эти строки содержат блок символов обратной дробной черты 3x3
     \\\
     \\\
     \\\
```

Фальцовка была выполнена вручную, поскольку сценарий из Приложения А не реализует принудительной фальцовки.

Отметим, что заголовки содержат символ # вместо знака равенства (=), применённого в примерах выше.

*9.4.1. Использование *

```
# NOTE: '\' line wrapping per RFC 8792
   Следующая строка длиннее 68 символов и её нужно фальцевать.
   12345678901234567890123456789012345678901234567890123456789012345678\
   89

   Эта строка завершается символом обратной дробной черты \\

   Эта строка завершается символом обратной дробной черты \\
   \ Эта строка начинается символом обратной дробной черты

   Эти строки содержат блок символов обратной дробной черты 3x3
      \\\\

      \\\\

      \\\
```

*9.4.2. Использование *

```
# NOTE: '\\' line wrapping per RFC 8792
   Следующая строка длиннее 68 символов и её нужно фальцевать.
   12345678901234567890123456789012345678901234567890123456789012345678\
   \89

   Эта строка завершается символом обратной дробной черты \

   Эта строка завершается символом обратной дробной черты \\
   \
   \ Эта строка начинается символом обратной дробной черты

   Эти строки содержат блок символов обратной дробной черты 3x3
      \\\\
      \
      \\\\
      \
      \\\
```

10. Вопросы безопасности

Этот документ не связан с безопасностью.

11. Взаимодействие с IANA

Этот документ не требует действий IANA.

12. Литература

12.1. Нормативные документы

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7991] Hoffman, P., "The "xml2rfc" Version 3 Vocabulary", RFC 7991, DOI 10.17487/RFC7991, December 2016, <https://www.rfc-editor.org/info/rfc7991>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

12.2. Дополнительная литература

[bash] "GNU Bash Manual", <https://www.gnu.org/software/bash/manual>.

[pyang] "pyang", <https://pypi.org/project/pyang/>.

[RFC7749] Reschke, J., "The "xml2rfc" Version 2 Vocabulary", RFC 7749, DOI 10.17487/RFC7749, February 2016, <https://www.rfc-editor.org/info/rfc7749>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7994] Flanagan, H., "Requirements for Plain-Text RFCs", RFC 7994, DOI 10.17487/RFC7994, December 2016, <https://www.rfc-editor.org/info/rfc7994>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[xiax] "The 'xiax' Python Package", <https://pypi.org/project/xiax/>.

[xml2rfc] "xml2rfc", <https://pypi.org/project/xml2rfc/>.

[yang-doctors-thread] Watsen, K., "[yang-doctors] automating yang doctor reviews", message to the yang-doctors mailing list, 18 April 2018, <https://mailarchive.ietf.org/arch/msg/yang-doctors/DCfBqgfZPAD7afzeDFlQ1Xm2X3g>.

[yanglint] "yanglint", commit 1b7d73d, February 2020, <https://github.com/CESNET/libyang#yanglint>.

Приложение A. Сценарий bash rfcfold

Это ненормативное прилоежение включает сценарий bash [bash], который может фальцевать и собирать текст с применением обеих стратегий, описанных в разделах 7 и 8. Этот сценарий rfcfold доступен по ссылке <https://github.com/ietf-tools/rfcfold>.

Сценарий предназначен для применения к одному экземпляру текстового содержимого. Если нужно обработать несколько экземпляров текстового содержимого в большом документе (например, Internet-Draft или RFC), нужно использовать другой инструмент для извлечения содержимого из документа перед использованием сценария.

Для удобочитаемости в сценарии принят минимальный размер строки на 8 символов больше размера заголовков, заданных в параграфах 7.1.1 и 8.1.1, чтобы можно было добавить символ пробела ( ) и 3 знака равенства (=) с каждой стороны необработанного заголовка.

При обнаружении во входном файле символа табуляции выдаётся сообщение

```
Error: infile contains a tab character, which is not allowed.
```

Сценарий проверяет доступность GNU awk (gawk) для проверки наличия символов управления ASCII и символов других кодировок (не ASCII) во входном файле (см. ниже). Отметим, что тестирование выявило недостатки применяемой по умолчанию версии awk на некоторых платформах, поэтому сценарий использует лишь gawk и выдаёт предупреждение при недоступности этой утилиты

```
Debug: no GNU awk; skipping checks for special characters[1].
```

При доступности gawk и обнаружении во входном файле символов управления ASCII выдаётся предупреждение

```
Warning: infile contains ASCII control characters (unsupported).
```

При доступности gawk и обнаружении во входном файле символов не-ASCII выдаётся предупреждение

```
Warning: infile contains non-ASCII characters (unsupported).
```

Сценарий не поддерживает логику исключения пробелов, описанную в параграфе 7.2.1. При обнаружении такой ситуации сценарий завершает работу с сообщением об ошибке

```
Error: infile has a space character occurring on the folding
column.  This file cannot be folded using the '\' strategy[2].
```

Хотя сценрий может собрать строки с принудительной фальцовкой, он не способен принудительно фальцевать текст, как описано в параграфах 7.2.1 и 8.2.1. При обнаружении необходимости принудительной фальцовки входного файла сценарий будет выдавать одно из указанных ниже сообщений об ошибке.

Для ****

```
Error: infile has a line ending with a '\' character.  This file
cannot be folded using the '\' strategy without there being false
positives produced in the unfolding (i.e., this script does not
force-fold such lines, as described in RFC 8792)[3].
```

Для ****

```
Error: infile has a line ending with a '\' character followed by a
'\' character as the first non-space character on the next line.
This script cannot fold this file using the '\\' strategy without
there being false positives produced in the unfolding (i.e., this
script does not force-fold such lines, as described in RFC 8792)[4].
```

Символы \ на уровне оболочки были специально добавлены в сценарий, чтобы он гарантированно не был сфальцован в этом документе и его можно было бы извлечь для применения с помощью операций copy-paste. Поскольку сценарий не содержит обязательного заголовка, описанного в параграфе 7.1.1, эти символы \ не предназначены для фальцовки (например, как описано в разделе 7).

```
<CODE BEGINS> file "rfcfold"
#!/bin/bash --posix

# Этот сценарий может потребовать действий для работы в конкретной
# системе. Например, может потребоваться установка утилиты gsed.
# Также следует обеспечить применение оболочки bash, а не sh.

# Copyright (c) 2020 IETF Trust, Kent Watsen, and Erik Auerswald.
# Все права защищены.
#
# Распространение и использование в исходной или двоичной форме с
# изменениями или без таковых разрешено при ряда выполнении условий:
#
#   * При распространении в исходном коде должны сохраняться указание
#     авторских прав, этот список условий и отказ от ответственности.
#
#   * При распространении в двоичном коде должны приводиться указание
#     авторских прав, этот список условий и отказ от ответственности
#     в документации или ином сопроводительном документе.
#
#   * Internet Society, IETF, IETF Trust или имена конкретных
#     участников работы не могут применяться для поддержки или
#     продвижения этой программы без соответствующего письменного
#     разрешения.
#
```

[1] GNU awk не обнаружено и проверка наличия специальных символов не выполнялась.
[2] Входной файл содержит пробел в точке разделения и его нельзя обработать с использованием стратегии с одним символом \.
[3] Входной файл содержит строку, завершающуюся символом \. Такой файл нельзя обработать в стратегии с одним символом \ без ложных срабатываний при последующей сборке (т. е. сценарий не фальцует такие строки принудительно, как указано в RFC 8792).
[4] Входной файл содержит строку, завершающуюся символом \, за которой следует строка, начинающаяся символом \. Такой файл нельзя обработать в стратегии с двумя символами \ без ложных срабатываний при последующей сборке (т. е. сценарий не фальцует такие строки принудительно, как указано в RFC 8792).

```
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE
# COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
# (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
# SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
# STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
# ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

print_usage() {
  printf "\n"
  printf "Folds or unfolds the input text file according to"
  printf " RFC 8792.\n"
  printf "\n"
  printf "Usage: rfcfold [-h] [-d] [-q] [-s <strategy>] [-c <col>]"
  printf " [-r] -i <infile> -o <outfile>\n"
  printf "\n"
  printf "  -s: strategy to use, '1' or '2' (default: try 1,"
  printf " else 2)\n"
  printf "  -c: column to fold on (default: 69)\n"
  printf "  -r: reverses the operation\n"
  printf "  -i: the input filename\n"
  printf "  -o: the output filename\n"
  printf "  -d: show debug messages (unless -q is given)\n"
  printf "  -q: quiet (suppress error and debug messages)\n"
  printf "  -h: show this message\n"
  printf "\n"
  printf "Exit status code: 1 on error, 0 on success, 255 on no-op."
  printf "\n\n"
}

# Глобальные переменные, которые не следует менять
strategy=0 # auto
debug=0
quiet=0
reversed=0
infile=""
outfile=""
maxcol=69  # Принято по умолчанию и может быть задано параметром
col_gvn=0  # maxcol переопределено?
hdr_txt_1="NOTE: '\\' line wrapping per RFC 8792"
hdr_txt_2="NOTE: '\\\\' line wrapping per RFC 8792"
equal_chars="============================================================"
space_chars="                                                            "
temp_dir=""
prog_name='rfcfold'

# Функции для диагностических сообщений
prog_msg() {
  if [[ "$quiet" -eq 0 ]]; then
    format_string="${prog_name}: $1: %s\n"
    shift
    printf -- "$format_string" "$*" >&2
  fi
}

err() {
  prog_msg 'Error' "$@"
}

warn() {
  prog_msg 'Warning' "$@"
}

dbg() {
  if [[ "$debug" -eq 1 ]]; then
    prog_msg 'Debug' "$@"
  fi
}

# Определяет имя утилиты [g]sed
type gsed > /dev/null 2>&1 && SED=gsed || SED=sed

# Предупреждение при использовании не-GNU sed
"$SED" --version < /dev/null 2> /dev/null | grep -q GNU || \
warn 'not using GNU `sed` (likely cause if an error occurs).'

cleanup() {
  rm -rf "$temp_dir"
}
trap 'cleanup' EXIT
```

```
  fold_it_1() {
    # Проверка того, что входной файл уже не был сфальцован
    if [[ -n "$("$SED" -n '/\\$/p' "$infile")" ]]; then
      err "infile '$infile' has a line ending with a '\\' character."\
          "This script cannot fold this file using the '\\' strategy"\
          "without there being false positives produced in the"\
          "unfolding."
      return 1
    fi

    # Точка фальцовки (разделения)
    foldcol=$(expr "$maxcol" - 1) # Для вставленного символа \

    # Проверка наличия символа пробела в точке деления
    grep -q "^\(.\{$foldcol\}\)\{1,\} " "$infile"
    if [[ $? -eq 0 ]]; then
      err "infile '$infile' has a space character occurring on the"\
          "folding column.  This file cannot be folded using the"\
          "'\\' strategy."
      return 1
    fi

    # Центровка текста заголовка
    length=$(expr ${#hdr_txt_1} + 2)
    left_sp=$(expr \( "$maxcol" - "$length" \) / 2)
    right_sp=$(expr "$maxcol" - "$length" - "$left_sp")
    header=$(printf "%.*s %s %.*s" "$left_sp" "$equal_chars"\
                    "$hdr_txt_1" "$right_sp" "$equal_chars")

    # Создание выходного файла
    echo "$header" > "$outfile"
    echo "" >> "$outfile"
    "$SED" 's/\(.\{'"$foldcol"'\}\)\(..\)/\1\\\n\2/;t M;b;:M;P;D;'\
      < "$infile" >> "$outfile" 2> /dev/null
    if [[ $? -ne 0 ]]; then
      return 1
    fi
    return 0
  }

  fold_it_2() {
    # Точка фальцовки (разделения)
    foldcol=$(expr "$maxcol" - 1) # Для вставленного символа \

    # Проверка наличия в файле символов фальцовки (fold-sequence)
    if [[ -n "$("$SED" -n '/\\$/{N;s/\\\n[ ]*\\/&/p;D}' "$infile")" ]]
    then
      err "infile '$infile' has a line ending with a '\\' character"\
          "followed by a '\\' character as the first non-space"\
          "character on the next line.  This script cannot fold"\
          "this file using the '\\\\' strategy without there being"\
          "false positives produced in the unfolding."
      return 1
    fi

    # Центровка текста заголовка
    length=$(expr ${#hdr_txt_2} + 2)
    left_sp=$(expr \( "$maxcol" - "$length" \) / 2)
    right_sp=$(expr "$maxcol" - "$length" - "$left_sp")
    header=$(printf "%.*s %s %.*s" "$left_sp" "$equal_chars"\
                    "$hdr_txt_2" "$right_sp" "$equal_chars")

    # Создание выходного файла
    echo "$header" > "$outfile"
    echo "" >> "$outfile"
    "$SED" 's/\(.\{'"$foldcol"'\}\)\(..\)/\1\\\n\\\2/;t M;b;:M;P;D;'\
      < "$infile" >> "$outfile" 2> /dev/null
    if [[ $? -ne 0 ]]; then
      return 1
    fi
    return 0
  }

  fold_it() {
    # Проверка наличия во входном файле символов табуляции
    grep -q $'\t' "$infile"
    if [[ $? -eq 0 ]]; then
      err "infile '$infile' contains a tab character, which is not"\
          "allowed."
      return 1
    fi

    # Фальцовка строк с символами управления ASCII или кодами не-ASCII
    # может приводить к ошибкам разделения строк и не поддерживается
    if type gawk > /dev/null 2>&1; then
      env LC_ALL=C gawk '/[\000-\014\016-\037\177]/{exit 1}' "$infile"\
```

```
      || warn "infile '$infile' contains ASCII control characters"\
              "(unsupported)."
      env LC_ALL=C gawk '/[^\000-\177]/{exit 1}' "$infile"\
      || warn "infile '$infile' contains non-ASCII characters"\
              "(unsupported)."
    else
      dbg "no GNU awk; skipping checks for special characters."
    fi

    # Нужна ли фальцовка файла?
    testcol=$(expr "$maxcol" + 1)
    grep -q ".\{$testcol\}" "$infile"
    if [[ $? -ne 0 ]]; then
      dbg "nothing to do; copying infile to outfile."
      cp "$infile" "$outfile"
      return 255
    fi

    if [[ "$strategy" -eq 1 ]]; then
      fold_it_1
      return $?
    fi
    if [[ "$strategy" -eq 2 ]]; then
      fold_it_2
      return $?
    fi
    quiet_sav="$quiet"
    quiet=1
    fold_it_1
    result=$?
    quiet="$quiet_sav"
    if [[ "$result" -ne 0 ]]; then
      dbg "Folding strategy '1' didn't succeed; trying strategy '2'..."
      fold_it_2
      return $?
    fi
    return 0
  }

  unfold_it_1() {
    temp_dir=$(mktemp -d)

    # Вывод всего, кроме двух первых строк в файл wip
    awk "NR>2" "$infile" > "$temp_dir/wip"

    # Сборка (unfold) файла wip
    "$SED" '{H;$!d};x;s/^\n//;s/\\\n *//g' "$temp_dir/wip" > "$outfile"

    return 0
  }

  unfold_it_2() {
    temp_dir=$(mktemp -d)

    # Вывод всего, кроме двух первых строк в файл wip
    awk "NR>2" "$infile" > "$temp_dir/wip"

    # Сборка (unfold) файла wip
    "$SED" '{H;$!d};x;s/^\n//;s/\\\n *\\//g' "$temp_dir/wip"\
      > "$outfile"

    return 0
  }

  unfold_it() {
    # Нужна ли сборка (unfold) файла?
    line=$(head -n 1 "$infile")
    line2=$("$SED" -n '2p' "$infile")
    result=$(echo "$line" | fgrep "$hdr_txt_1")
    if [[ $? -eq 0 ]]; then
      if [[ -n "$line2" ]]; then
        err "the second line in '$infile' is not empty."
        return 1
      fi
      unfold_it_1
      return $?
    fi
    result=$(echo "$line" | fgrep "$hdr_txt_2")
    if [[ $? -eq 0 ]]; then
      if [[ -n "$line2" ]]; then
        err "the second line in '$infile' is not empty."
        return 1
      fi
      unfold_it_2
      return $?
    fi
```

```
    dbg "nothing to do; copying infile to outfile."
    cp "$infile" "$outfile"
    return 255
  }

  process_input() {
    while [[ "$1" != "" ]]; do
      if [[ "$1" == "-h" ]] || [[ "$1" == "--help" ]]; then
        print_usage
        exit 0
      elif [[ "$1" == "-d" ]]; then
        debug=1
      elif [[ "$1" == "-q" ]]; then
        quiet=1
      elif [[ "$1" == "-s" ]]; then
        if [[ "$#" -eq "1" ]]; then
          err "option '-s' needs an argument (use -h for help)."
          exit 1
        fi
        strategy="$2"
        shift
      elif [[ "$1" == "-c" ]]; then
        if [[ "$#" -eq "1" ]]; then
          err "option '-c' needs an argument (use -h for help)."
          exit 1
        fi
        col_gvn=1
        maxcol="$2"
        shift
      elif [[ "$1" == "-r" ]]; then
        reversed=1
      elif [[ "$1" == "-i" ]]; then
        if [[ "$#" -eq "1" ]]; then
          err "option '-i' needs an argument (use -h for help)."
          exit 1
        fi
        infile="$2"
        shift
      elif [[ "$1" == "-o" ]]; then
        if [[ "$#" -eq "1" ]]; then
          err "option '-o' needs an argument (use -h for help)."
          exit 1
        fi
        outfile="$2"
        shift
      else
        warn "ignoring unknown option '$1'."
      fi
      shift
    done

    if [[ -z "$infile" ]]; then
      err "infile parameter missing (use -h for help)."
      exit 1
    fi

    if [[ -z "$outfile" ]]; then
      err "outfile parameter missing (use -h for help)."
      exit 1
    fi

    if [[ ! -f "$infile" ]]; then
      err "specified file '$infile' does not exist."
      exit 1
    fi

    if [[ "$col_gvn" -eq 1 ]] && [[ "$reversed" -eq 1 ]]; then
      warn "'-c' option ignored when unfolding (option '-r')."
    fi

    if [[ "$strategy" -eq 0 ]] || [[ "$strategy" -eq 2 ]]; then
      min_supported=$(expr ${#hdr_txt_2} + 8)
    else
      min_supported=$(expr ${#hdr_txt_1} + 8)
    fi
    if [[ "$maxcol" -lt "$min_supported" ]]; then
      err "the folding column cannot be less than $min_supported."
      exit 1
    fi

    # this is only because the code otherwise runs out of equal_chars
    max_supported=$(expr ${#equal_chars} + 1 + ${#hdr_txt_1} + 1\
         + ${#equal_chars})
    if [[ "$maxcol" -gt "$max_supported" ]]; then
      err "the folding column cannot be more than $max_supported."
      exit 1
```

```
    fi
}

main() {
  if [[ "$#" -eq "0" ]]; then
     print_usage
     exit 1
  fi

  process_input "$@"

  if [[ "$reversed" -eq 0 ]]; then
    fold_it
    code=$?
  else
    unfold_it
    code=$?
  fi
  exit "$code"
}

main "$@"
<CODE ENDS>
```

Благодарности

Авторы благодарят RFC Editor за подтверждение того, что на момент публикации не было установленного соглашения по обработке длинных строк, при включении исходного кода, что послужило мотивом для этой работы.

Спасибо Ben Kaduk, Benoit Claise, Gianmarco Bruno, Italo Busi, Joel Jaeggli, Jonathan Hansford, Lou Berger, Martin Bjorklund, Rob Wilton (по алфавиту) за вклад в подготовку этого документа.

Адреса авторов

Kent Watsen
Watsen Networks
Email: kent+ietf@watsen.net

Erik Auerswald
Individual Contributor
Email: auerswal@unix-ag.uni-kl.de

Adrian Farrel
Old Dog Consulting
Email: adrian@olddog.co.uk

Qin Wu
Huawei Technologies
Email: bill.wu@huawei.com

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru