

Редактирование файлов с помощью скриптов с помощью ed

Почему ed?

Например `sed`, `ed` это редактор строк. Однако, если вы попытаетесь изменить содержимое файла с `sed` помощью, и файл будет открыт в другом месте и прочитан каким-либо процессом, вы обнаружите, что GNU `sed` и его `-i` опция не позволят вам редактировать файл. Существуют обстоятельства, при которых вам может понадобиться это, например, редактирование активных и открытых файлов, отсутствие GNU или другое `sed`, при наличии опции "на месте".

Почему `ed` ?

- возможно, вы `sed` не поддерживаете редактирование на месте
- возможно, вам нужно быть максимально переносимым
- может быть, вам действительно нужно отредактировать файл в файле (а не создавать новый файл, как GNU `sed`)
- и последнее, но не менее важное: standard `ed` обладает очень хорошими возможностями редактирования и адресации по сравнению со стандартом `sed`

Не поймите меня неправильно, это **не** анти- `sed` статья! Это просто предназначено для того, чтобы показать вам другой способ выполнения работы.

Командующий ed

Поскольку `ed` это интерактивный текстовый редактор, он считывает и выполняет команды, которые поступают из `stdin`. Есть несколько способов передать наши команды в `ed`:

Конвейеры

```
echo '<ED-КОМАНДЫ>' | ed <ФАЙЛ>
```

Для ввода необходимых новых строк и т. Д. Может быть проще использовать встроенную команду `printf` ("help printf"). Здесь показана в качестве примера функция Bash для добавления текста к содержимому файла:

```
# insertHead "$text" "$file"

insertHead() {
printf '%s\n' Н 1i "$1" . w | ed -s "$ 2"
}
```

Здесь-строки

```
ed <ФАЙЛ> <<< '<ED-КОМАНДЫ>'
```

Здесь-документы

```
ed <ФАЙЛ> <
<ED-КОМАНДЫ>
EOF
```

Какой из них вы предпочитаете, это ваш выбор. Я буду использовать here-строки, так как здесь это выглядит лучше всего, ИМХО().

Существуют и другие способы ввода ed . Например, замена процесса. Но этого должно быть достаточно для повседневных нужд.

Поскольку ed требуется, чтобы команды разделялись символами новой строки, я буду использовать специальный метод кавычек Bash, C-подобные строки \$ 'ТЕХТ' , поскольку он может интерпретировать набор различных escape-последовательностей и специальных символов. Я использую эту -s опцию, чтобы сделать ее менее подробной.

Базовый интерфейс

Проверьте ed справочную страницу для получения подробной информации

Подобно vi или vim , ed имеет "командный режим" и "интерактивный режим". Для неинтерактивного использования обычным выбором является командный режим.

Команды должны ed иметь простую и регулярную структуру: ноль, один или два адреса, за которыми следует односимвольная команда, возможно, за которой следуют параметры для этой команды. Эти адреса указывают одну или несколько строк в текстовом буфере. Каждая команда, требующая адресов, имеет адреса по умолчанию, поэтому адреса часто могут быть опущены.

Адресация строки определяется относительно *текущей строки*. Если буфер редактирования не пуст, начальным значением для *текущей строки* должна быть последняя строка в буфере редактирования, в противном случае ноль. Как правило, *текущая строка* - это последняя строка, на которую влияет команда. Все адреса могут адресовать только отдельные строки, а не блоки строк!

Адреса строк или команды, использующие *регулярные выражения*, интерпретируют базовые регулярные выражения POSIX (BRE). Нулевой BRE используется для ссылки на самый последний используемый BRE. Поскольку ed адресация предназначена только для отдельных строк, никакой RE никогда не сможет соответствовать новой строке.

Отладка ваших скриптов ed

По умолчанию `ed` не очень разговорчив и просто печатает "?" при возникновении ошибки. Интерактивно вы можете использовать `h` команду, чтобы получить короткое сообщение, объясняющее последнюю ошибку. Вы также можете включить режим, который `ed` автоматически печатает это сообщение с `h` помощью команды.

Рекомендуется всегда добавлять эту команду в начале ваших сценариев редактирования:

```
bash> файл ed -s <<< '$H \ n,df'
?
скрипт, строка 2: недопустимый суффикс команды
```

Во время работы над своим скриптом вы можете допускать ошибки и уничтожать свой файл, у вас может возникнуть соблазн попробовать, чтобы ваш скрипт делал что-то вроде:

```
# Работает, но есть лучше

# скопируйте мой исходный файл
cp file file.test

# попробуйте мой скрипт для файла
ed -s file.тест <<< '$H\n<ed commands>\nw'

# смотрите результаты
cat file.test
```

Однако есть гораздо лучший способ, вы можете использовать команду `ed p` для печати файла, теперь ваше тестирование будет выглядеть так:

```
файл ed -s <<< '$H \n<команды редактирования>\n, p'
```

`,` (запятая) перед `p` командой - это ярлык, для `1`, `$` которого определяется диапазон адресов от первой до последней строки, `,` `p` что означает печать всего файла после его изменения. Когда ваш скрипт работает успешно, вам нужно только заменить `,` `p` на `a w`.

Конечно, даже если файл не был изменен `p` командой, **всегда полезно иметь резервную копию!**

Редактирование ваших файлов

Большинство из этих вещей можно сделать с `sed` помощью. Но есть также вещи, которые невозможно выполнить `sed` или которые могут быть выполнены только с помощью очень сложного кода.

Простые замены слов

Например `sed`, `ed` также знает общую `s/FROM/TO/` команду, и она также может принимать адреса строк. **Если в адресных строках не производится замена, это считается ошибкой.**

Замены по всему файлу

```
ed -s test.txt <<< '$, s/ Windows(R)-совместимый/ POSIX-соответствующий/g\nw'
```

Примечание: Запятая как оператор единого адреса является псевдонимом для `1, $` ("все строки").

Замены в определенных строках

В строке, содержащей `fruits`, выполните замену:

```
ed -s test.txt <<< '$/фрукты / фрукты/яблоко / банан/g \nw'
```

В 5-й строке после строки, содержащей `fruits`, выполните замену:

```
ed -s test.txt <<< '$/фрукты/+5s/яблоко/банан/g\nw'
```

Блокировать операции

Удаление блока текста

Самый простой - это хорошо известный (по позиции) блок текста:

```
# удалить строки с номерами 2-4 (2, 3, 4)
ed -s test.txt <<< '$2,5d\ nw'
```

При этом удаляются все строки, соответствующие определенному регулярному выражению:

```
# удалить все строки, соответствующие foobar
ed -s test.txt <<< '$g/foobar/d\nw'
```

`g/regex/` применяет следующую за ней команду ко всем строкам, соответствующим регулярному выражению

Перемещение блока текста

... с помощью `m` команды: `<ADDRESS> m <TARGET-ADDRESS>`

Это определено то, что нелегко сделать с помощью `sed`.

```
# перемещение строк 5-9 в конец файла
ed -s test.txt <<< '$5,9млн$\nw'
```

```
# перемещение строк 5-9 в строку 3
ed -s test.txt <<< '$5,9м3\nw'
```

Скопируйте блок текста

... с помощью `t` команды: `<ADDRESS> t <TARGET-ADDRESS>`

Вы используете `t` команду точно так же, как вы используете команду `m` (переместить).

```
# сделайте копию строк 5-9 и поместите ее в конец файла
ed -s test.txt <<< '$5,9t$\nw'

# сделайте копию строк 5-9 и поместите ее в строку 3
ed -s test.txt <<< '$5,9t3\nw'
```

Объединить все строки

... но оставьте последнюю новую строку нетронутой. Это делается дополнительной командой: `j` (join).

```
файл ed -s <<< '$1,$ j \ nw'
```

По сравнению с двумя другими методами (с использованием `tr` или `sed`), вам не нужно удалять все новые строки и вручную добавлять одну в конце.

Операции с файлами

Вставьте другой файл

Как вставить другой файл? Как и в случае с `sed`, вы используете команду `r` (читать). Это вставляет другой файл в строку перед последней строкой (и выводит результат в стандартный вывод - , `p`):

```
ed -s FILE1 <<< '$$-1 r FILE2\n,p'
```

Для сравнения, вот возможное `sed` решение, которое должно использовать арифметику Bash и внешнюю программу `wc`:

```
sed "$((${wc -l < FILE1}-1))r FILE2" FILE1

# ОБНОВЛЕНИЕ вот один, который использует параметр GNU sed "e" для s-
команды
# он выполняет команды, найденные в пространстве шаблонов. Я буду счи-
тать, что
# риск для безопасности, но, знаете, иногда GNU> безопасность...
sed '${h;s/./cat FILE2/e;G}' FILE1
```

Другой подход, в двух вызовах `sed`, который полностью исключает использование внешних команд:

```
sed '${s/$/\n-||-/;r FILE2\n}' FILE1 | sed '0,-||-/#!/ h;N;///
D};$G'
```

Подводные камни

ed - это не sed

ed и sed могут выглядеть одинаково, но одни и те же команды могут действовать по-разному:

/foo/d

В sed /foo/d удалит все строки, соответствующие foo, в ed команды не повторяются в каждой строке, поэтому эта команда будет искать следующую строку, соответствующую foo, и удалять ее. Если вы хотите удалить все строки, соответствующие foo, или выполнить замену во всех строках, соответствующих foo, вы должны сообщить ed об этом с помощью команды g (global):

```
echo $'1 \n1 \ n3'> файл

#замените все строки, соответствующие 1, на "replacement"
файл ed -s <<< '$g/1/s/1/replacement/\n,p'

#замените первую строку, соответствующую 1, на "замена"
#(потому что поиск начинается с последней строки)
файл ed -s <<< '$s/1/replacement/\n,p'
```

ошибка останавливает скрипт

Вы можете подумать, что это не проблема и что то же самое происходит с sed, и вы правы, за исключением того, что если ed не находит шаблон, это ошибка, в то время как sed просто переходит к следующей строке. Например, предположим, что вы хотите изменить foo на bar в первой строке файла и добавить что-то после следующей строки, ed остановится, если не сможет найти foo в первой строке, sed продолжится.

```
#Gnu sed версия
sed -e '1s/foo/bar/' -e '$ a \something' файл

#Первая версия ed, ничего не делает, если foo не найден в первой строке:
ed -s file <<< '$N\n1s/foo/bar/\na\nsomething\n.\nw'
```

Если вы хотите такого же поведения, вы можете использовать g/foo/, чтобы обмануть ed. g/foo/ применит команду ко всем строкам, соответствующим foo, таким образом, замена будет успешной, и ed не выдаст ошибку, если foo не найден:

```
#Вторая версия добавит строку с "чем-то", даже если foo не найден
ed -s file <<< '$N\n1g/foo/s/foo/bar/\na \ nsomething\n.\nw'
```

На самом деле, даже подстановка, которая завершается с ошибкой после команды g/ /, похоже, не вызывает ошибки, т. Е. Вы можете использовать такой трюк, как g/./s/foo/bar/, чтобы попытаться выполнить подстановку во всех непустых строках

здесь документы

параметры оболочки расширены

Если вы не заключаете разделитель в кавычки, \$ имеет особое значение. Это звучит очевидно, но об этом легко забыть, когда вы используете адреса типа \$-1 или команды типа \$ a . Либо заключите в кавычки \$, либо разделитель:

```
#сбой
файла ed -s << EOF
$
последняя строка
.
w
EOF

#ok
файл ed -s << EOF
\$
последняя строка
.
w
EOF

#снова ok
файл ed -s << 'EOF'
$
последняя строка
.
w
EOF
```

"." не является командой

The . используемая для завершения команда "а" должна быть единственной в строке. будьте осторожны, если вы делаете отступы в командах:

```
#ed не заботится о пробелах перед командами, но . должно быть единств
енное, что есть в строке:
ed -s file << EOF

моего содержимого
.
w
EOF
```

Имитация других команд

Имейте в виду, что во всех приведенных ниже примерах весь файл будет считан в память.

Простой grep

```
файл ed -s <<< 'g/foo/p'
```

```
# эквивалентный  
файл ed -s <<< 'g/foo/'
```

Название `grep` происходит от обозначения `g/RE/p` (глобальное \Rightarrow регулярное выражение \Rightarrow печать). ссылка <http://www.catb.org/~esr/jargon/html/G/grep.html> (<http://www.catb.org/~esr/jargon/html/G/grep.html>)

wc -l

Поскольку по умолчанию для команды `ed` "напечатать номер строки" используется последняя строка, простой `=` (знак равенства) напечатает этот номер строки и, следовательно, количество строк в файле:

```
файл ed -s <<< '='
```


cat

Да, это шутка...

```
файл ed -s <<< $',p'
```

... но аналогично `cat` отображению окончаний строк и экранирования можно выполнить с `list` помощью команды (`l`):

```
файл ed -s <<< $',l'
```

 продолжение следует

Ссылки

Ссылка:

- Gnu ed (http://www.gnu.org/software/ed/manual/ed_manual.html) - если нам нужно было угадать, вы, вероятно, используете этот.
- POSIX ed (http://pubs.opengroup.org/onlinepubs/9699919799/utilities/ed.html#tag_20_38), `ex` (http://pubs.opengroup.org/onlinepubs/9699919799/utilities/ex.html#tag_20_40) и `vi` (http://pubs.opengroup.org/onlinepubs/9699919799/utilities/vi.html#tag_20_152)
- <http://sdf.lonestar.org/index.cgi?tutorials/ed> (<http://sdf.lonestar.org/index.cgi?tutorials/ed>) - эд чит - лист на sdf.org

Разное информация / учебные пособия:

- Как я могу заменить строку другой строкой в переменной, потоке, файле или во всех файлах в каталоге? (<http://mywiki.woledge.org/BashFAQ/021>)- BashFAQ

- <http://wolfram.schneider.org/bsd/7thEdManVol2/edtut/edtut.pdf>
(<http://wolfram.schneider.org/bsd/7thEdManVol2/edtut/edtut.pdf>) - Старый, но все еще актуальный учебник по редактированию.

Обсуждение

noname, [2010/04/12 05:45 \(\)](#)

ис -l Поскольку текущая строка (строка, адресуемая, когда не указан явный адрес) является последней строкой после запуска, а = (знак равенства) там напечатает номер строки последней строки, то есть количество строк файла: знак равенства = не зависиттам, где мы сейчас находимся, по умолчанию всегда выводится номер строки \$ try **ed -s /etc/passwd «< \$ ' 1 \ n=**

Ян Шампера, [2010/04/12 14:19 \(\)](#)

Аааа, ты абсолютно прав. На моей ed странице руководства упоминаются "адреса по умолчанию". Спасибо!

Стив Терпе (<https://github.com/sterpe>), [2014/12/02 16:42 \(\)](#)

Действительно сумасшедший, используйте `sed` для `редактирования` нескольких файлов на месте:

```
ls *.js | sed "s/./ed & << EOF\\  
0a\\  
\\/** @jsx реагирует.DOM *\\//\\  
.\\  
wq\\  
EOF/" | sh
```

foo, [2015/05/25 07:18 \(\)](#)

удалить строки с 2 по 4 (2, 3, 4)

Комментарий или код должны быть изменены.

~ \$ cat » f

111

222

333

444

```
555
```

```
666
```

```
~ $ ed -s f «< $ '2,5d \ nw'
```

```
~ $ cat f
```

```
111
```

```
666
```

 [howto/edit-ed.txt](#)  Последнее редактирование: 08.05.08 20:00 автор: bill_thomson

Этот сайт поддерживается Performing Databases - вашими
экспертами по администрированию баз данных

Bash Hackers Wiki



Except where otherwise noted, content on this wiki is licensed under the following license:
GNU Free Documentation License 1.3