sed_examples Documentation

Выпуск 1.1

shmakovpn

нояб. 24, 2019

Contents:

Глава 1

Введение

Даешь беременность! Долой мастурбацию!
Вернулся с работы, возрождай нацию!

Про программу **sed**, как и про любовь, написано очень много всего. И, я думаю, вы, уже с ней знакомы в какой то мере. Поэтому начну я не с ответов, а с вопросов.

1. Что делает делает следующая команда?

```
cat one-ten.txt | sed -n '1!G;h;$!d;p'
```

2. Знаете ответ, отлично! А как она это делает?
3. Не знаете? Птичий язык? Не, это не «мое». Разобраться можно и потом, если вдруг приспичит. А сейчас некогда, надо писать реальные программы на python, java и прочих крутых языках.
4. Как начать думать, чтобы хоть что-то начать понимать?
5. И начав понимать, довести до уровня условных рефлексов, когда все получается само?

Немного странное начало. Но ответ я дам сразу «думать на **sed**» мы будем учиться на **perl**.

Все логично, ведь **sed** это по сути тот же самый интерпретируемый язык программирования. Только сильно «кастрированный» и команды короткие. Поэтому непонятный. Однако, стоит изложить программу **sed**-ову программу на **perl**, как сразу все становиться ясненько-понятненько. Проделав оное 100500 раз в разных вариациях, формируется новый устойчивый навык, происходит личностная трансформация и человек превращается в некоего шамана способного очень быстро оперировать информацией.

Глава 2

Самая простая программа на **sed**

Простота - хуже воровства

Давайте создадим текстовый файл из десяти уникальных строк.

```
one
two
three
four
five
six
seven
eight
nine
ten
```

Назовем его *one-ten.txt*. Он нам понадобиться в дальнейшем для разного рода экспериментов.

А теперь напишем свою первую программу:

```
cat one-ten.txt | sed -re ''
```

Будет выведено десять уникальных строк на консоль. Как если бы простро выполнить команду **cat**, вообще без **sed**.

Давайте посмотрим на ключи:

r использовать регулярные выражения в теле программы

e код программы, в нашем случае „“

Все очень просто, выполнить код после **e** и если в нем есть регулярные выражения, то выполнить и их.

Почему, же мы получили десять уникальных строк? Потому, что **sed** считывает стандартный ввод построчно в буфер «pattern space», далее выполняя код своей программы (ничего) к каждой из строк. По умолчанию, в конце обработки каждой итерации, **sed** выполняет команду **p**. Что означает печать содержимого «pattern space». Используя ключ **n**, можно отключить такой режим.

Таким образом наша первая программа на **sed** не является самой простой. Хотя и является самой короткой. Запишем ее по другому.

```
cat one-ten.txt | sed -nre 'p'
```

Самая простая будет:

```
cat one-ten.txt | sed -nre ''
```

Она ничего не выводит, как и ожидалось. Давайте сделаем ее аналог на **perl**.

sed:

```
cat one-ten.txt | sed -nre ''
```

perl:

```
cat one-ten.txt | perl -e '$hs="";$ps="";while($ps=<>){chomp($ps);}'
```

или (для извращенцев):

```
cat one-ten.txt | perl -e 'while(<>){chomp;}'
```

В **sed** есть две переменные «pattern space» и «hold space». Содержимое которых при запуске прошграмм изначально является пустым. На **perl** это можно явно выразить как присвоение переменным *$ps* и *$hs* пустых значений. Однако, можно этого неделать, что не является ошибкой. Можно вместо *$ps* использовать переменную по умолчанию *$_* , код становится еще короче и непонятнее. Чем больше меньше кода тем незаменимее программист. *while($ps=<>){}* цикл считывающий в *$ps* стандартный ввод. *chomp($ps)* отрезает от *$ps* перевод строки.

Вернемся к самому первому примеру:

sed:

```
cat one-ten.txt | sed -re ''
```

или

```
cat one-ten.txt | sed -nre 'p'
```

perl:

```
cat one-ten.txt | perl -e '$hs="";$ps="";while($ps=<>){chomp($ps);print($ps."\n
↪");}'
```

или (ключ *l* включает автоматическое добавление перевода строки «\n» при вызове *print*):

```
cat one-ten.txt | perl -le '$hs="";$ps="";while($ps=<>){chomp($ps);print($ps);}
↪'
```

можно покороче, но не по талантлевие (на **sed** все равно короче):

```
cat one-ten.txt | perl -le 'while(<>){chomp; print;}'
```

заменим *print* на *say*, ключ *l* в данном случае становится излишним, а *e* надо поменять на *E*:

```
cat one-ten.txt | perl -E 'while(<>){chomp;say;}'
```

Ларри Уол, создатель **perl**, не зря говорил о более чем одном пути:

```
cat one-ten.txt | perl -nE 'chomp;say;'
```

Добавим извращений:

```
cat one-ten.txt | perl -ple ''
```

Чтобы объяснить последний пример, давайте декомпилируем создаваемую интерпритатором **perl** программу. Выполним следующее:

```
perl -MO=Deparse -ple ''
```

Получим:

```
BEGIN { $/ = "\n"; $\ = "\n"; }  # автоматически добавлять перевод строки␣
↪после print
# цикл построчной обработки стандартного ввода
LINE: while (defined($_ = readline ARGV)) {
    chomp $_;  # удалить перевод строки
}
continue {
    die "-p destination: $!\n" unless print $_;  # вывести строку на stdout
}
-e syntax OK
```

Подведем некоторые итоги. **sed** читает вывод построчно и каждый раз сначала отезает от перевод строки, затем помещает строку без перевода строки в буфер «pattern space», далее выполняет код программы после *-re* или *-nre*. После выполнения программы, если нет ключа *-n* будет выполнена команда *p* - вывод «pattern space» плюс перевод строки на стандартный вывод.

Вывод номер два. Одно и тоже на **perl** можно сделать по разному. Очевидно, что в качестве «pattern space» лучше использовать переменную *$_* . *sed -re* соответствует *perl -ple*, а *sed -nre* аналогично *perl -nle*. Или не очевидно? Ответьте себе честно на данный вопрос. Лучше не нарушать божьи заповеди и не лгать особенно самому себе, особенно если вы программист. В дальнейшем я также постараюсь приводить разные **perl** примеры. Будем учиться не просто по честному «думать на **sed**», а «думать красиво».

Глава 3

Комманды sed

БДСМ - база данный системы мониторинга

Введите *man sed* и попробуйте что либо понять.

Ощутили, что лирикой слов точно и просто передать поведение программы, а именно, то как она оперирует стандарным вводом, стандартным выводом, буферами, ветвлениями, переходами и всем остальным, задача отнюдь не тривиальная, и это мягко говоря.

Не зря физика использует язык математики, в музыке ноты, а в геометрии аксиомы и теоремы.

Операции языка **perl** более близки к человеку чем, комманды **sed**.

Эта глава, объясняет поведение команд **sed** куда более человеческим языком.

sed	perl	Описание
Zero-address commands		
#	`#`	Комментарий, распространяется до конца строки или до конца фрагмента скрипта после ключа -e
: label	label:	Устанавливает метку для перехода
}	`}`	Окончание блока комманд
b label	`goto label;`	Безусловный переход на метку label
b	`next;`	Переход на слудующую итерацию
Zero- или One- address commands		

Продолжается на следующей странице

Таблица 1 – продолжение с предыдущей страницы

sed	perl	Описание
=	`print($..."\n");`	Вывод номера текущей строки с добавлением перевода строки

a text	`$hs='';`
	`$ps='';`
	`while($ps=<>) {`
	`  # some code`
	`} continue {`
	`  # some code`
	`  print("$text\n");`
	`}`
		Вывод text плюс перевод строки после выполнения всех действий sed скрипта. Причем все что идет после команды *a*, будет интерпретировано как текст, а не как комманды. Данную команду используют для вставки строк. Но из-за тонкостей своего поведения ее не удобно использовать в однострочниках. Есть другие способы добавлять строки, что будет рассмотренно в дальнейшем

i text	`$hs='';`
	`$ps='';`
	`while($ps=<>) {`
	`  print("$text\n");`
	`  # some code`
	`} continue {`
	`  # some code`
	`}`
		Вывод text плюс перевод строки перед выполнением sed скрипта. Все что идет после команды *i*, будет итерпретировано как текст. Что ее применение в однострочниках, однако, существуют альтернативные способы вставки строк.

q	`print($ps."\n");`
	`exit;`
	`# или просто`
	`exit;`
		Выход из sed. Без дальнейшей обработки ввода. Если включен автоматический вывод (не указан ключ *n*) печать pattern space а затем остановка программы

Q	`exit;`	Немедленная остановка программы sed
Commands which accept address ranges		
{	`{`	Начало блока комманд
c text	`print("text"."\n");` `next;`	Вывод text, вместо строки. Выполните, посмотрите, что будет. `cat one-ten.txt | sed -nre '3cGO'` Все что идет после команды *c* будет интерпретировано как текст. Что опять таки жутко неудобно. В дальнейшем мы покажем, как выполнить замену строки другими способами

Продолжается на следующей странице

Таблица 1 – продолжение с предыдущей страницы

sed	perl	Описание
d	`$ps="";` `next;`	Очищает «pattern space» и переходит к следущей интерации

D	`L: if($ps!~"\n"){`
	`  $ps="";`
	`  next;`
	`} else {`
	`  $ps=~s/^.*\n//;`
	`  goto L;`
	`}`
		Если «pattern space», не содержит перевод строки, выполним команду d. Иначе, вырежем из «pattern space», текст до перевода строки влючительно, и запустим цикл повторно с новым значением «pattern space»

h	`$hs=$ps;`	Копирует «pattern space» в «hold space»
H	`$hs.="\n".$ps;`	Добавляет перевод строки к «hold space», затем добавляет «pattern space»
g	`$ps=$hs;`	Копиреут «hold space» в «pattern space»
G	`$ps.="\n".$hs;`	Добавляет перевод строки к «pattern space», затем добавляет «hold space»
n	`$ps=<>;` `chomp($ps);`	Считывает следующую строку со стандартного ввода в «pattern space»
N	`$ps.="\n".<> or next;` `chomp($ps);`	Добаляет перевод строки к «pattern space». Затем считывает следующую строку со стандартного ввода. Если строк больше нет прекращает выполнение комманд
p	`print($ps."\n");`	Вывод «pattern space» на stdout с добавлением перевода строки
P	`say((split(/\n))[0])`	Вывод «pattern space» до первого перевода строки на стандартный вывод с добавлением перевода строки
s/p/R/	`$ps=~s/p/R/;`	Поиск по паттерну p и замена на R. Практически полный аналог операции замены в perl. Так же работают модификаторы i и g. Специальный символ «&» в R заменяется на текст из «pattern space» подпавший под паттерн p. В perl это «$&». \1-\9 в perl будут $1-$9.

Продолжается на следующей странице

Таблица 1 – продолжение с предыдущей страницы

sed	perl	Описание
t label	`$t=($ps=~s/p/R/);` `goto label if $t;`	Условный переход. Выполняется если операция s/// выполнила замену.
t	`$t=($ps=~s/p/R/);` `next if $t;`	Условный переход на следующий цикл при успешном выполнении s///
T label	`$t=($ps=~s/p/R/);` `goto label if !$t;`	Условный переход. Выполняется если операция s/// не выполнила замену.
T	`$t=($ps=~s/p/R/);` `next if !$t;`	Условный переход на следующий цикл при не успешном выполнении s///
x	`($ps,$hs)=($hs,$ps);`	Обмен содержимым между «hold space» и «pattern space»
y/from/to/	`$ps=~y/from/to/`	Тоже самое что y/// в perl, символы из from будут заменены на соответствующие символы из to

Я сознательно опустил команду *l*. Т.к. не нашел в ней какой либо пользы. Ее задача преодоление “visually unambiguous“,, «визуальной неоднозначности». Связанной с тем, что очень длинные строки не помещаются целиком на терминал и переносятся на следующую строку. Как теперь отличить две отдельные строки от начала и хвоста очень длинной строки перенесенной терминалом на новую строчку? *l* заменяет \ на \. В конце каждого переноса строки добавляется \. При использовании *-l width* с малыми значениями width и наличием \ в строках ввода изменяет длину выводимых строк. Имитация всех нюансов такого поведения на perl становится очень громоздкой. И не несет никакой пратической пользы.

Автором *perl* Ларри Уолом создана утилита s2p (sed to perl), которая конвертирует sed скрипты в perl-код. Возможно кто то найдет ее полезной. https://perldoc.perl.org/5.8.8/s2p.html

Глава 4

s2p

s2p расшифровывается как *set to perl*.

Это программа написанная автром языка **perl** Ларри Уолом. Она позволяет конвертировать **sed** скрипты в программы на языке **perl**.

Установить s2p довольно просто. Введите в консоли.

```
cpan install App::s2p
```

Программа установки будет задавать различные вопросы и предлагать ответы по умолчанию. Приняв все дефолтные ответы и перелогинившись мы получим установленную программу s2p в окружении пользователя.

Запуск s2p

```
s2p -n 'sed скрипт' # sed -nre 'скрипт'
# или
s2p 'sed скрипт'  # sed -re 'скрипт'
```

s2p выведет на стандартный вывод код программы.

Глава 5

Адресация

sed позволяет применять команду или блок команд не ко всем строкам (адресация по умолчанию), а к конкретным.

И так, давайте рассмотрим все на практических примерах, для чего используем заранее заготовленный файл **one-ten.txt**, а так же оключим автоматический вывод **sed** с помощью ключа *-n*.

Адресацию по умолчанию мы наблюдали ранее, когда создавали самую простую программу на **sed**. Следуя принципу: «Повторение - мать учения» и нежелая разрывать логическую связь между тезисами данной главы, будем предельно скучными и пройдем все по порядку.

Давайте выполним.

```
cat one-ten.txt | sed -nre 'p'
```

Получили все строки из *one-ten.txt*. Следовательно, если не указывать никакой адресации, команда или блок комманд применяется ко всем строкам.

Превратим в **perl** в самом подробнейшем виде. Для запуска следует выполнить:

```
cat one-ten.txt | perl -e 'ниже приведенный код'
```

В дальнейшем условимся, что в примерах, где приводится чисто **perl**-овый код, запускать его следуем именно так.

```
$hs="";
$ps="";
while($ps=<>) {
    chomp($ps);
    print($ps."\n");  # команда p
}
```

Применим один адрес к **sed** команде.

```
cat one-ten.txt | sed -nre '3p'
```

Получили только третью строчку. На **perl** следует вопсользовать встроенной переменной содержащий номер текущей строки стандартного ввода *$.*.

```
$hs="";  # инициализируем hold space
$ps="";  # инициализируем pattern space
while($ps=<>) {  # построчно читаем *stdin* в pattern space
    chomp($ps);  # отсекаем перевод строки в конце pattern space
    # начало. собственно сам sed-скрипт
    if($.==3) {  # если идет обработка 3ей строки
        print($ps."\n");
    }
    # конец. собственно сам sed-скрипт
}
```

Вспоминая, что *sed -nre „что-то“* это *perl -nle „что-то“* сократим написанное:

```
cat one-ten.txt | perl -nle 'print if $.==3;'
```

Давайте посмотрим, как **perl** интерпретирует код приведенный выше. Выполним:

```
perl -MO=Deparse -nle 'print if $.==3;'
```

И получим:

```
BEGIN { $/ = "\n"; $\ = "\n"; }
LINE: while (defined($_ = readline ARGV)) {
    chomp $_;
    print $_ if $. == 3;
}
-e syntax OK
```

Следующий пример показывает работу диапазона адресов.

```
cat one-ten.txt | sed -nre '3,5p'
```

Будут выведены строки 3,4 и 5. Таким образом можно указать диапазон строк от одной строки до другой включительно. Существует маленький нуанс.

```
cat one-ten.txt | sed -nre '3,1p'
```

Выведет третью строку. Смысл прост. Представим наш скриптв неком гипотетическом виде:

```
cat one-ten.txt | sed -nre '/условие1/,/условие2/p'
```

Запускать данный пример не надо, он абстрактный и работать не будет.

> Больше кода меньше слов И получиться любов

Давайте добавим в наш простой пример еще одну переменную.

```
$hs='';  # hold space
$ps='';  # pattern space
$af=0;  # address flag
while($ps=<>) {  # читаем стандартный ввод построчно
    chomp($ps);  # отсекаем перевод строки от pattern space
    # выполнить блок кода, если ранее было выполнено условие1
    # если условие1 не было выполнено ранее, проверить условие1,
    # запомнив результат проверки в переменную $af,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    # если результат проверки "истина" выполнить блок кода
    if($af or ($af=(условие1)) ) {
        print($ps."\n");  # выполнение команды p
        $af=0 if условие2; # сбрасываем флаг при выполнению условия 2
    }
}
```

Интерпритатор **sed** сначала проверяет певое условие, второе при этом не проверяется. При выполнении первого условия, выполняется команда или блок команд.

Для всех последующих циклов после успешного выполнения первого условия будет выполняться команда (блок команд) и проверяться второе условие.

Когда будет выполнено второе условие, безусловное исполнение команды (блока команд) для последующих циклов будет отключено и возобновиться проверка первого условия. Т.е. поведение программы вернится к режиму до выполнения условия1.

Давайте сделаем код короче:

```
cat one-ten.txt | sed -nre '2,5p'
# превращается в
cat one-ten.txt | perl -lne 'if($af or ($af=($.==2))){print;$af=0 if $.==5;}'
```

Вместо номеров строк можно использовать **регулярные выражения**, так при следующей выполнении команды будут выведены строки со второй по пятую.

```
cat one-ten.txt | sed -nre '/tw/,5p'
# на perl
cat one-ten.txt | perl -lne 'if($af or ($af=(/tw/))){print;$af=0 if $.==5;}'
```

Еще пример (two, three, four, five)

```
cat one-ten.txt | sed -nre '/tw/,/iv/p'
# perl
cat one-ten.txt | perl -lne 'if($af or ($af=(/tw/))){print;$af=0 if /iv/;}'
```

Адресацию можно инвертировать, указав символ *!* после адреса или диапазона адресов.

```
cat one-ten.txt | sed -nre '5!p'  # выведет все строки кроме пятой
# perl
cat one-ten.txt | perl -nle 'print if $.!=5;'
```

Используем *!* для исключения диапазона адресов

```
cat one-ten.txt | sed -nre '2,9!p'  # выведет первую и последную строку
# хотя правильнее, вырежет строки со второй по девятую включительно
# perl
cat one-ten.txt | perl -lne 'if($af or ($af=($.==2))){$af=0 if $.==9;}else{print;}'
```

Закрепим наши знания на примере с **регулярными выражениями** (1,6,7,8,9,10)

```
cat one-ten.txt | sed -nre '/tw/,/iv/!p'
# perl:
cat one-ten.txt | perl -lne 'if($af or ($af=(/tw/))){$af=0 if /iv/;}else{print;}'
```

В случае регулярного выражения использованного в качестве второго условия, первое условие может быть 0.

cat one-ten.txt | sed -nre „0,/o/p“ cat one-ten.txt | sed -nre „1,/o/p“

Глава 6

Indices and tables

- genindex
- modindex
- search

Алфавитный указатель

Символы