Приложение B. Маленький учебник по Sed и Awk

В этом приложении содержится очень краткое описание приемов работы с утилитами обработки текста **sed** и **awk**. Здесь будут рассмотрены лишь несколько базовых команд, которых, в принципе, будет достаточно, чтобы научиться понимать простейшие конструкции sed и awk внутри сценариев на языке командной оболочки.

sed: неинтерактивный редактор текстовых файлов

awk: язык обработки шаблонов с C-подобным синтаксисом

При всех своих различиях, эти две утилиты обладают похожим синтаксисом, они обе умеют работать с регулярными выражениями, обе, по-умолчанию, читают данные с устройства `stdin` и обе выводят результат обработки на устройство `stdout`. Обе являются утилитами UNIX-систем, и прекрасно могут взаимодействовать между собой. Вывод от одной может быть перенаправлен, по конвейеру, на вход другой. Их комбинирование придает сценариям, на языке командной оболочки, мощь и гибкость языка Perl.


Одно важное отличие состоит в том, что в случае с sed, сценарий легко может передавать дополнительные аргументы этой утилите, в то время, как в случае с awk (см. Пример 33-3 и Пример 9-22), это более сложная задача .

B.1. Sed

Sed -- это неинтерактивный строчный редактор. Он принимает текст либо с устройства `stdin`, либо из текстового файла, выполняет некоторые операции над строками и затем выводит результат на устройство `stdout` или в файл. Как правило, в сценариях, sed используется в конвейерной обработке данных, совместно с другими командами и утилитами.

Sed определяет, по заданному *адресному пространству*, над какими строками следует выполнить операции. [1] Адресное пространство строк задается либо их порядковыми номерами, либо шаблоном. Например, команда `3d` заставит sed удалить третью строку, а команда `/windows/d` означает, что все строки, содержащие "windows", должны быть удалены.

Из всего разнообразия операций, мы остановимся на трех, используемых наиболее часто. Это **p** -- печать (на `stdout`), **d** -- удаление и **s** -- замена.

Таблица B-1. Основные операции sed

Операция	Название	Описание
`[диапазон строк]/p`	print	Печать [указанного диапазона строк]

Операция	Название	Описание
`[диапазон строк]/d`	delete	Удалить [указанный диапазон строк]
`s/pattern1/pattern2/`	substitute	Заменить первое встреченное соответствие шаблону pattern1, в строке, на pattern2
`[диапазон строк]/s/pattern1/pattern2/`	substitute	Заменить первое встреченное соответствие шаблону pattern1, на pattern2, в указанном *диапазоне строк*
`[диапазон строк]/y/pattern1/pattern2/`	transform	заменить любые символы из шаблона pattern1 на соответствующие символы из pattern2, в указанном *диапазоне строк*(эквивалент команды **tr**)
`g`	global	Операция выполняется над *всеми* найденными соответствиями внутри каждой из заданных строк

Без оператора `g` (*global*), операция замены будет производиться только для первого найденного совпадения, с заданным шаблоном, в каждой строке.

В отдельных случаях, операции sed необходимо заключать в кавычки.

```
sed -e '/^$/d' $filename
#  Ключ -e говорит о том, что далее следует строка, которая должна
интерпретироваться
#+ как набор инструкций редактирования.
# (При передаче одной инструкции, ключ "-e" является необязательным.)
#  "Строгие" кавычки ('') предотвращают интерпретацию символов
регулярного выражения,
#+ как специальных символов, командным интерпретатором.
#
# Действия производятся над строками, содержащимися в файле $filename.
```

В отдельных случаях, команды редактирования не работают в одиночных кавычках.

```
filename=file1.txt
pattern=BEGIN

  sed "/^$pattern/d" "$filename"  # Результат вполне предсказуем.
# sed '/^$pattern/d' "$filename"    дает иной результат.
#         В данном случае, в "строгих" кавычках (' ... '),
#+      не происходит подстановки значения переменной "$pattern".
```

 Sed использует ключ `-e` для того, чтобы определить, что следующая строка является

инструкцией, или набором инструкций, редактирования. Если инструкция является единственной, то использование этого ключа не является обязательным.

```
sed -n '/xzy/p' $filename
# Ключ -n заставляет sed вывести только те строки, которые совпадают с
указанным шаблоном.
# В противном случае (без ключа -n), будут выведены все строки.
# Здесь, ключ -e не является обязательным, поскольку здесь стоит
единственная команда.
```

Таблица B-2. Примеры операций в sed

Операция	Описание
`8d`	Удалить 8-ю строку.
`/^$/d`	Удалить все пустые строки.
`1,/^$/d`	Удалить все строки до первой пустой строки, включительно.
`/Jones/p`	Вывести строки, содержащие "Jones" (с ключом -n).
`s/Windows/Linux/`	В каждой строке, заменить первое встретившееся слово "Windows" на слово "Linux".
`s/BSOD/stability/g`	В каждой строке, заменить все встретившиеся слова "BSOD" на "stability".
`s/ *$//`	Удалить все пробелы в конце каждой строки.
`s/00*/0/g`	Заменить все последовательности ведущих нулей одним символом "0".
`/GUI/d`	Удалить все строки, содержащие "GUI".
`s/GUI//g`	Удалить все найденные "GUI", оставляя остальную часть строки без изменений.

Замена строки пустой строкой, эквивалентна удалению части строки, совпадающей с шаблоном. Остальная часть строки остается без изменений. Например, **`s/GUI//`**, изменит следующую строку

`The most important parts of any application are its GUI and sound effects`
на
`The most important parts of any application are its  and sound effects`

Символ обратного слэша представляет символ перевода строки, как символ замены. В этом случае, замещающее выражение продолжается на следующей строке.

```
s/^  */\
/g
```

Эта инструкция заменит начальные пробелы в строке на символ перевода строки. Ожидаемый результат -- замена отступов в начале параграфа пустыми строками.

Указание диапазона строк, предшествующее одной, или более, инструкции может потребовать заключения инструкций в фигурные скобки, с соответствующими символами перевода строки.

```
/[0-9A-Za-z]/,/^$/{
/^$/d
}
```

В этом случае будут удалены только первые из нескольких, идущих подряд, пустых строк. Это может использоваться для установки однострочных интервалов в файле, оставляя, при этом, пустые строки между параграфами.


Быстрый способ установки двойных межстрочных интервалов в текстовых файлах -- **sed G filename.**

Примеры использования sed в сценариях командной оболочки, вы найдете в:

1. Пример 33-1
2. Пример 33-2
3. Пример 12-2
4. Пример A-3
5. Пример 12-12
6. Пример 12-20
7. Пример A-13
8. Пример A-19
9. Пример 12-24
10. Пример 10-9
11. Пример 12-33
12. Пример A-2
13. Пример 12-10
14. Пример 12-8
15. Пример A-11
16. Пример 17-11

Ссылки на дополнительные сведения о sed, вы найдете в разделе *Литература*.

Примечания

[1] Если адресное пространство не указано, то, по-умолчанию, к обработке принимаются *все* строки.

B.2. Awk

Awk -- это полноценный язык обработки текстовой информации с синтаксисом, напоминающим синтаксис языка **C**. Он обладает довольно широким набором возможностей, однако, мы рассмотрим лишь некоторые из них -- наиболее употребимые в сценариях командной оболочки.

Awk "разбивает" каждую строку на отдельные *поля*. По-умолчанию, поля -- это последовательности символов, отделенные друг от друга пробелами, однако имеется возможность назначения других символов, в качестве разделителя полей. Awk анализирует и обрабатывает каждое поле в отдельности. Это делает его идеальным инструментом для работы со структурированными текстовыми файлами, осбенно с таблицами.

Внутри сценариев командной оболочки, код awk, заключается в "строгие" (одиночные) кавычки и фигурные скобки.

```
awk '{print $3}' $filename
# Выводит содержимое 3-го поля из файла $filename на устройство
stdout.

awk '{print $1 $5 $6}' $filename
# Выводит содержимое 1-го, 5-го и 6-го полей из файла $filename.
```

Только что, мы рассмотрели действие команды **print**. Еще, на чем мы остановимся -- это переменные. Awk работает с переменными подобно сценариям командной оболочки, но более гибко.

```
{ total += ${column_number} }
```

Эта команда добавит содержимое переменной *column_number* к переменной "total". Чтобы, в завершение вывести "total", можно использовать команду **END**, которая открывает блок кода, отрабатывающий после того, как будут обработаны все входные данные.

```
END { print total }
```

Команде **END**, соответствует команда **BEGIN**, которая открывает блок кода, отрабатывающий перед началом обработки входных данных.

Примеры использования awk в сценариях командной оболочки, вы найдете в:

1. Пример 11-10
2. Пример 16-7
3. Пример 12-24
4. Пример 33-3
5. Пример 9-22
6. Пример 11-16
7. Пример 27-1
8. Пример 27-2
9. Пример 10-3
10. Пример 12-42
11. Пример 9-26
12. Пример 12-3
13. Пример 9-12
14. Пример 33-11
15. Пример 10-8

Это все, что я хотел рассказать об awk. Дополнительные ссылки на информацию об awk, вы найдете в разделе *Литература*.
