

Обработка текстовой информации

По материалам блога “Bash-скрипты: начало”
Автор оригинала: Администратор likegeeks.com
Блог компании RUVDS.com
<https://habr.com/ru/company/ruvds/blog/325522/>
<https://habr.com/ru/post/229501/>

Sed

Утилиту `sed` называют потоковым текстовым редактором. В интерактивных текстовых редакторах, наподобие `nano`, с текстами работают, используя клавиатуру, редактируя файлы, добавляя, удаляя или изменяя тексты. `Sed` позволяет редактировать потоки данных, основываясь на заданных разработчиком наборах правил. Вот как выглядит схема вызова этой команды:

```
$ sed options file
```

По умолчанию `sed` применяет указанные при вызове правила, выраженные в виде набора команд, к `STDIN`. Это позволяет передавать данные непосредственно `sed`. Например, так:

```
$ echo "This is a test" | sed 's/test/another test/'
```

Ниже показан файл, в котором содержится фрагмент текста, и результаты его обработки такой командой:

```
$ sed 's/test/another test' ./myfile
```

Для выполнения нескольких действий с данными, используйте ключ -e при вызове sed. Например, вот как организовать замену двух фрагментов текста:

```
$ sed -e 's/This/That/; s/test/another test/' ./myfile
```

Схема записи команды замены при использовании флагов выглядит так:

`s/pattern/replacement/flags`

Выполнение этой команды можно модифицировать несколькими способами.

При передаче номера учитывается порядковый номер вхождения шаблона в строку, заменено будет именно это вхождение.

Флаг `g` указывает на то, что нужно обработать все вхождения шаблона, имеющиеся в строке.

Флаг `r` указывает на то, что нужно вывести содержимое исходной строки.

Флаг вида `w file` указывает команде на то, что нужно записать результаты обработки текста в файл.

Рассмотрим использование первого варианта команды замены, с указанием позиции заменяемого вхождения искомого фрагмента:

```
$ sed 's/test/another test/2' myfile
```

Теперь опробуем флаг глобальной замены — g:

```
$ sed 's/test/another test/g' myfile
```

Такая команда заменила все вхождения шаблона в тексте.

До сих пор мы вызывали `sed` для обработки всего переданного редактору потока данных. В некоторых случаях с помощью `sed` надо обработать лишь какую-то часть текста — некую конкретную строку или группу строк. Для достижения такой цели можно воспользоваться двумя подходами:

- Задать ограничение на номера обрабатываемых строк.
- Указать фильтр, соответствующие которому строки нужно обработать.

Рассмотрим первый подход. Тут допустимо два варианта. Первый, рассмотренный ниже, предусматривает указание номера одной строки, которую нужно обработать:

```
$ sed '2s/test/another test/' myfile
```

Второй вариант — диапазон строк:

```
$ sed '2,3s/test/another test/' myfile
```

Утилита `sed` годится не только для замены одних последовательностей символов в строках на другие. С её помощью, а именно, используя команду `d`, можно удалять строки из текстового потока.

Вызов команды выглядит так:

```
$ sed '3d' myfile
```

Мы хотим, чтобы из текста была удалена третья строка. Обратите внимание на то, что речь не идёт о файле. Файл останется неизменным, удаление отразится лишь на выводе, который сформирует `sed`.

Вот как применить команду `d` к диапазону строк:

```
$ sed '2,3d' myfile
```

Строки можно удалять и по шаблону:

```
$ sed '/test/d' myfile
```

С помощью sed можно вставлять данные в текстовый поток, используя команды i и a:

- Команда i добавляет новую строку перед заданной.
- Команда a добавляет новую строку после заданной.

Рассмотрим пример использования команды i:

```
$ echo "Another test" | sed 'i\First test '
```

Теперь взглянем на команду a:

```
$ echo "Another test" | sed 'a\First test '
```


Команда `s` позволяет изменить содержимое целой строки текста в потоке данных. При её вызове нужно указать номер строки, вместо которой в поток надо добавить новые данные:

```
$ sed '3c\This is a modified line.' myfile
```

Если воспользоваться при вызове команды шаблоном в виде обычного текста или регулярного выражения, заменены будут все соответствующие шаблону строки:

```
$ sed '/This is/c This is a changed line of text.' myfile
```

Команда `y` работает с отдельными символами, заменяя их в соответствии с переданными ей при вызове данными:

```
$ sed 'y/123/567/' myfile
```

Если вызвать `sed`, используя команду `=`, утилита выведет номера строк в потоке данных:

```
$ sed '=' myfile
```

Если передать этой команде шаблон и воспользоваться ключом `sed -n`, выведены будут только номера строк, соответствующих шаблону:

```
$ sed -n '/test/=' myfile
```

Представим себе такую задачу. Есть файл, в котором имеется некая последовательность символов, сама по себе бессмысленная, которую надо заменить на данные, взятые из другого файла. А именно, пусть это будет файл newfile, в котором роль указателя места заполнения играет последовательность символов DATA. Данные, которые нужно подставить вместо DATA, хранятся в файле data.

Решить эту задачу можно, воспользовавшись командами r и d потокового редактора sed:

```
$ Sed '/DATA>/ {
```

```
r newfile
```

```
d}' myfile
```

AWK

Утилита `awk`, или точнее GNU `awk`, в сравнении с `sed`, выводит обработку потоков данных на более высокий уровень. Благодаря `awk` в нашем распоряжении оказывается язык программирования, а не довольно скромный набор команд, отдаваемых редактору. С помощью языка программирования `awk` можно выполнять следующие действия:

Объявлять переменные для хранения данных.

Использовать арифметические и строковые операторы для работы с данными.

Использовать структурные элементы и управляющие конструкции языка, такие, как оператор `if-then` и циклы, что позволяет реализовать сложные алгоритмы обработки данных.

Создавать форматированные отчёты.

Схема вызова awk выглядит так:

```
$ awk options program file
```

Awk воспринимает поступающие к нему данные в виде набора записей. Записи представляют собой наборы полей. Упрощенно, если не учитывать возможности настройки awk и говорить о некоем вполне обычном тексте, строки которого разделены символами перевода строки, запись — это строка. Поле — это слово в строке.

Рассмотрим наиболее часто используемые ключи командной строки awk:

- -F fs — позволяет указать символ-разделитель для полей в записи.
- -f file — указывает имя файла, из которого нужно прочесть awk-скрипт.
- -v var=value — позволяет объявить переменную и задать её значение по умолчанию, которое будет использовать awk.
- -mf N — задаёт максимальное число полей для обработки в файле данных.
- -mr N — задаёт максимальный размер записи в файле данных.
- -W keyword — позволяет задать режим совместимости или уровень выдачи предупреждений awk.

Одна из основных функций `awk` заключается в возможности манипулировать данными в текстовых файлах. Делается это путём автоматического назначения переменной каждому элементу в строке. По умолчанию `awk` назначает следующие переменные каждому полю данных, обнаруженному им в записи:

`$0` — представляет всю строку текста (запись).

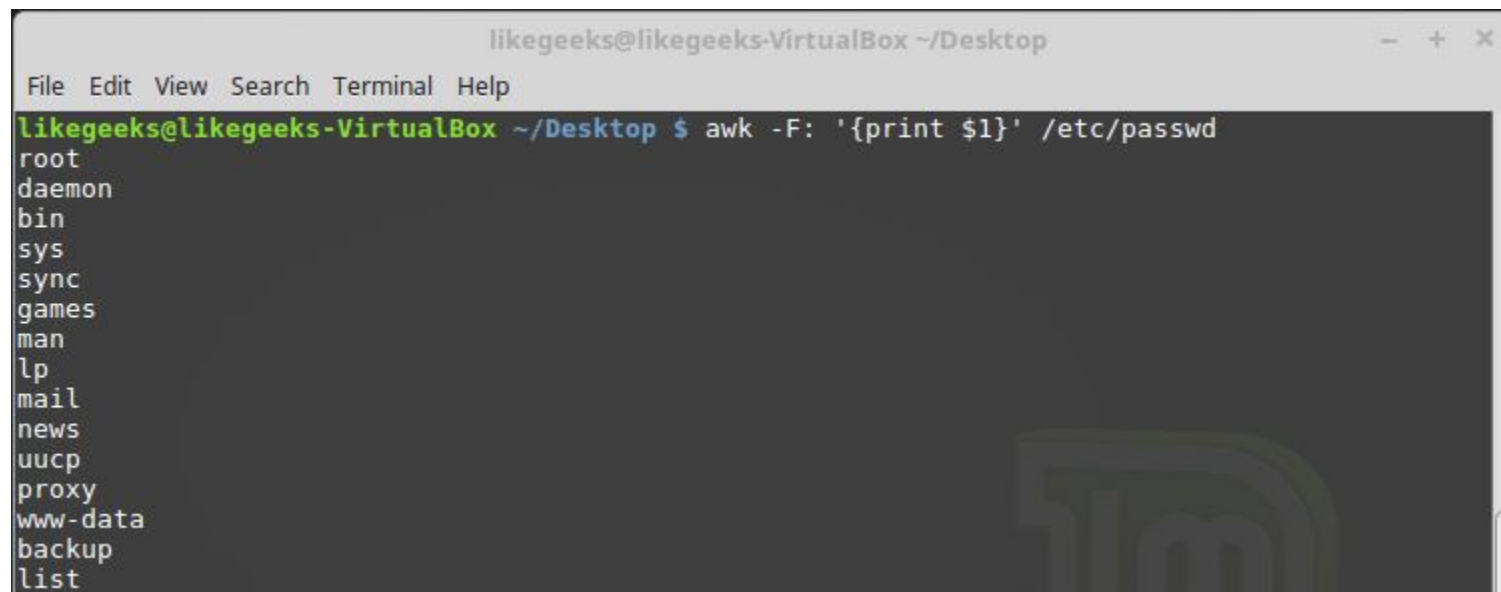
`$1` — первое поле.

`$2` — второе поле.

`$n` — n -ное поле.

Поля выделяются из текста с использованием символа-разделителя. По умолчанию — это пробельные символы вроде пробела или символа табуляции.

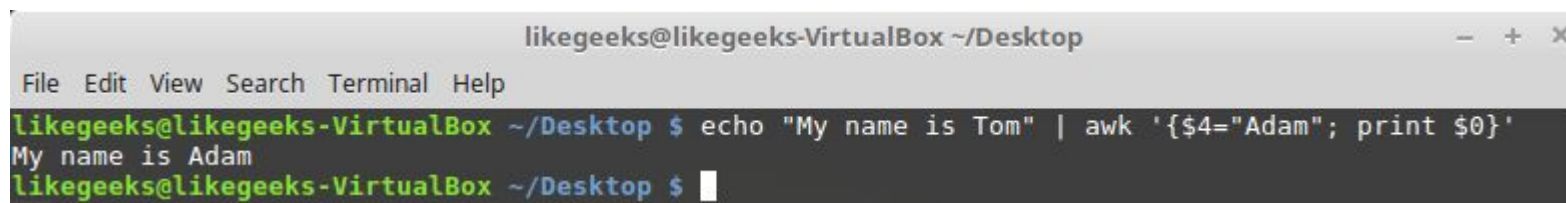
Эта команда выводит первые элементы строк, содержащихся в файле `/etc/passwd`. Так как в этом файле в качестве разделителей используются двоеточия, именно этот символ был передан `awk` после ключа `-F`.



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk -F: '{print $1}' /etc/passwd
root
daemon
bin
sys
sync
games
man
lp
mail
news
uucp
proxy
www-data
backup
list
```

Вызов `awk` с одной командой обработки текста — подход очень ограниченный. `Awk` позволяет обрабатывать данные с использованием многострочных скриптов. Для того, чтобы передать `awk` многострочную команду при вызове его из консоли, нужно разделить её части точкой с запятой:

```
$ echo "My name is Tom" | awk '{$4="Adam"; print $0}'
```

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows a command prompt where the user enters 'echo "My name is Tom" | awk '{\$4="Adam"; print \$0}'' and the output 'My name is Adam' is displayed. The prompt then returns to the user's input line.

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "My name is Tom" | awk '{$4="Adam"; print $0}'
My name is Adam
likegeeks@likegeeks-VirtualBox ~/Desktop $
```


Awk позволяет хранить скрипты в файлах и ссылаться на них, используя ключ -f. Подготовим файл testfile, в который запишем следующее:

```
{print $1 " has a home directory at " $6}
```

Вызовем awk, указав этот файл в качестве источника команд:

```
$ awk -F: -f testfile /etc/passwd
```

Иногда нужно выполнить какие-то действия до того, как скрипт начнёт обработку записей из входного потока. Например — создать шапку отчёта или что-то подобное.

Для этого можно воспользоваться ключевым словом BEGIN. Команды, которые следуют за BEGIN, будут исполнены до начала обработки данных. В простейшем виде это выглядит так:

```
$ awk 'BEGIN {print "Hello World!"}'
```

А вот — немного более сложный пример:

```
$ awk 'BEGIN {print "The File Contents:"}
```

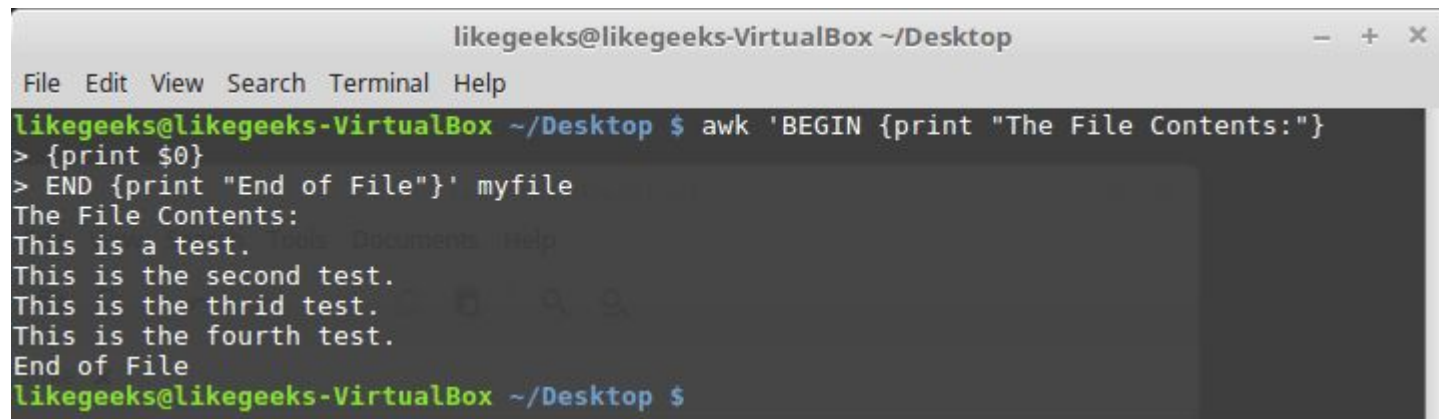
```
{print $0}' myfile
```

Ключевое слово END позволяет задавать команды, которые надо выполнить после окончания обработки данных:

```
$ awk 'BEGIN {print "The File Contents:"}
```

```
{print $0}
```

```
END {print "End of File"}' myfile
```

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the following command and output:

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN {print "The File Contents:"}  
> {print $0}  
> END {print "End of File"}' myfile  
The File Contents:  
This is a test.  
This is the second test.  
This is the thrid test.  
This is the fourth test.  
End of File  
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

```

BEGIN {

print "The latest list of users and shells"

print "  UserName \t HomePath"

print "----- \t -----"

FS=":"

}

{

print $1 " \t " $6

}

END {

print "The end"

}

```

Тут, в блоке `BEGIN`, создаётся заголовок табличного отчёта. В этом же разделе мы указываем символ-разделитель. После окончания обработки файла, благодаря блоку `END`, система сообщит нам о том, что работа окончена.

Awk поддерживает стандартный во многих языках программирования формат условного оператора if-then-else. Однострочный вариант оператора представляет собой ключевое слово if, за которым, в скобках, записывают проверяемое выражение, а затем — команду, которую нужно выполнить, если выражение истинно.

Например, есть такой файл с именем testfile:

10

15

6

33

45

Напишем скрипт, который выводит числа из этого файла, большие 20:

```
$ awk '{if ($1 > 20) print $1}' testfile
```

Если нужно выполнить в блоке if несколько операторов, их нужно заключить в фигурные скобки:

```
$ awk '{
```

```
if ($1 > 20)
```

```
{
```

```
x = $1 * 2
```

```
print x
```

```
}
```

```
}' testfile
```

Условный оператор awk может содержать блок else:

```
$ awk '{
```

```
if ($1 > 20)
```

```
{
```

```
x = $1 * 2
```

```
print x
```

```
} else
```

```
{
```

```
x = $1 / 2
```

```
print x
```

```
}}' testfile
```

Цикл while позволяет перебирать наборы данных, проверяя условие, которое остановит цикл.

Вот файл myfile, обработку которого мы хотим организовать с помощью цикла:

124 127 130

112 142 135

175 158 245

Напишем такой скрипт:

```
$ awk '{
```

```
total = 0
```

```
i = 1
```

```
while (i < 4)
```

```
{
```

```
total += $i
```

```
i++
```

```
}
```

```
avg = total / 3
```

```
print "Average:",avg
```

```
}' testfile
```


Циклы for используются во множестве языков программирования. Поддерживает их и awk. Решим задачу расчёта среднего значения числовых полей с использованием такого цикла:

```
$ awk '{  
  
total = 0  
  
for (i = 1; i < 4; i++)  
  
{  
  
total += $i  
  
}  
  
avg = total / 3  
  
print "Average:",avg  
  
}' testfile
```

Команда `printf` в `awk` позволяет выводить форматированные данные. Она даёт возможность настраивать внешний вид выводимых данных благодаря использованию шаблонов, в которых могут содержаться текстовые данные и спецификаторы форматирования.

Спецификатор форматирования — это специальный символ, который задаёт тип выводимых данных и то, как именно их нужно выводить. `Awk` использует спецификаторы форматирования как указатели мест вставки данных из переменных, передаваемых `printf`.

Первый спецификатор соответствует первой переменной, второй спецификатор — второй, и так далее.

Спецификаторы форматирования записывают в таком виде:

`%[modifier]control-letter`

Вот некоторые из них:

- c — воспринимает переданное ему число как код ASCII-символа и выводит этот символ.
- d — выводит десятичное целое число.
- i — то же самое, что и d.
- e — выводит число в экспоненциальной форме.
- f — выводит число с плавающей запятой.
- g — выводит число либо в экспоненциальной записи, либо в формате с плавающей запятой, в зависимости от того, как получается короче.
- o — выводит восьмеричное представление числа.
- s — выводит текстовую строку.

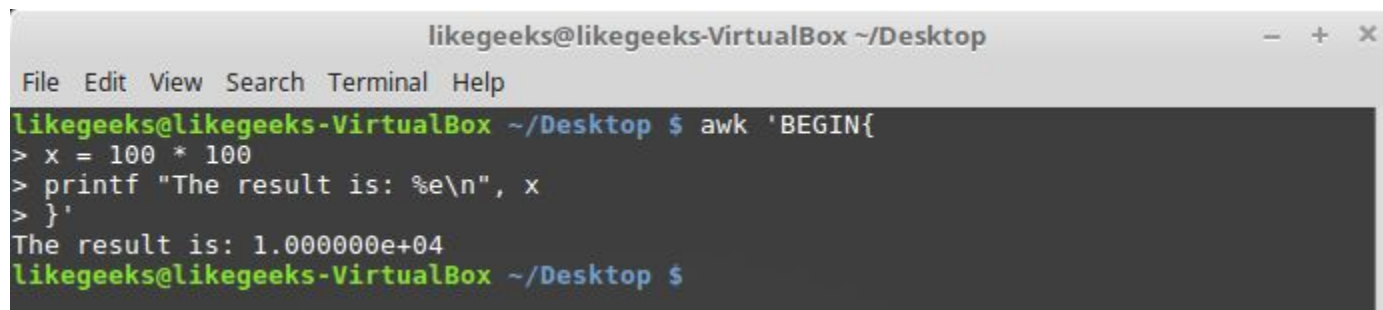
Вот как форматировать выводимые данные с помощью printf:

```
$ awk 'BEGIN{
```

```
x = 100 * 100
```

```
printf "The result is: %e\n", x
```

```
}]'
```

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the following command sequence:

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{  
> x = 100 * 100  
> printf "The result is: %e\n", x  
> }'  
The result is: 1.000000e+04  
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

RegExp

У многих, когда они впервые видят регулярные выражения, сразу же возникает мысль, что перед ними бессмысленное нагромождение символов. Но это, конечно, далеко не так. Взгляните, например, на это регулярное выражение

```
^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$
```

Даже абсолютный новичок сходу поймёт, как оно устроено и зачем нужно. Если же вам не вполне понятно — просто читайте дальше и всё встанет на свои места.

Регулярное выражение — это шаблон, пользуясь которым программы вроде `sed` или `awk` фильтруют тексты. В шаблонах используются обычные ASCII-символы, представляющие сами себя, и так называемые метасимволы, которые играют особую роль, например, позволяя ссылаться на некие группы символов.

При использовании различных символов в регулярных выражениях надо учитывать некоторые особенности. Так, существуют некоторые специальные символы, или метасимволы, использование которых в шаблоне требует особого подхода. Вот они:

`.*[]^${}\+?|()`

Если один из них нужен в шаблоне, его нужно будет экранировать с помощью обратной косой черты (обратного слэша) — `\`.

Например, если в тексте нужно найти знак доллара, его надо включить в шаблон, предварив символом экранирования. Скажем, имеется файл `myfile` с таким текстом:

```
There is 10$ on my pocket
```

Знак доллара можно обнаружить с помощью такого шаблона:

```
$ awk '/\$/ {print $0}' myfile
```

Якорные символы

Существуют два специальных символа для привязки шаблона к началу или к концу текстовой строки. Символ «крышка» — `^` позволяет описывать последовательности символов, которые находятся в начале текстовых строк. Если искомый шаблон окажется в другом месте строки, регулярное выражение на него не отреагирует. Выглядит использование этого символа так:

```
$ echo "welcome to likegeeks website" | awk '/^likegeeks/{print $0}'
```

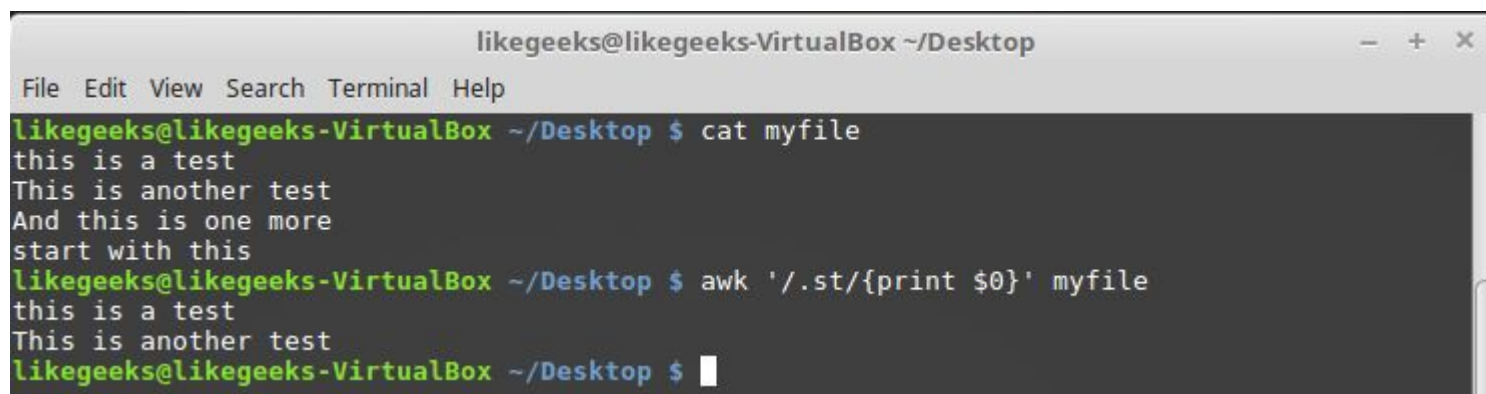
```
$ echo "likegeeks website" | awk '/^likegeeks/{print $0}'
```

В одном и том же шаблоне можно использовать оба якорных символа. Выполним обработку файла `myfile`, содержимое которого показано на рисунке ниже, с помощью такого регулярного выражения:

```
$ awk '/^this is a test$/{print $0}' myfile
```

Точка используется для поиска любого одиночного символа, за исключением символа перевода строки. Передадим такому регулярному выражению файл myfile, содержимое которого приведено ниже:

```
$ awk '/.st/{print $0}' myfile
```



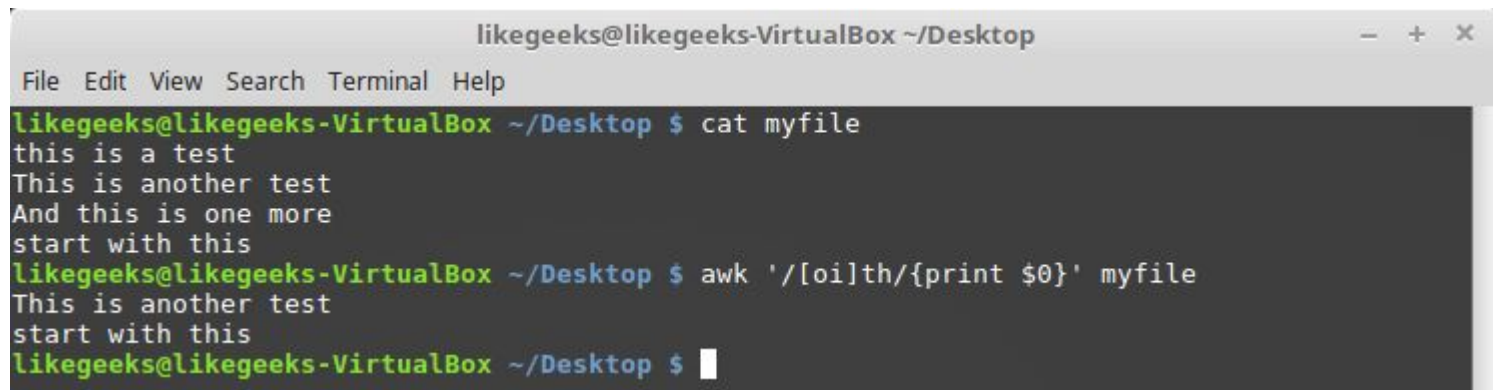
```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
this is a test
This is another test
And this is one more
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/.st/{print $0}' myfile
this is a test
This is another test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```


Классы символов

Точка соответствует любому одиночному символу, но что если нужно более гибко ограничить набор искомых символов? В подобной ситуации можно воспользоваться классами символов.

Благодаря такому подходу можно организовать поиск любого символа из заданного набора. Для описания класса символов используются квадратные скобки — []:

```
$ awk '/[oi]th/{print $0}' myfile
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
this is a test
This is another test
And this is one more
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/[oi]th/{print $0}' myfile
This is another test
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

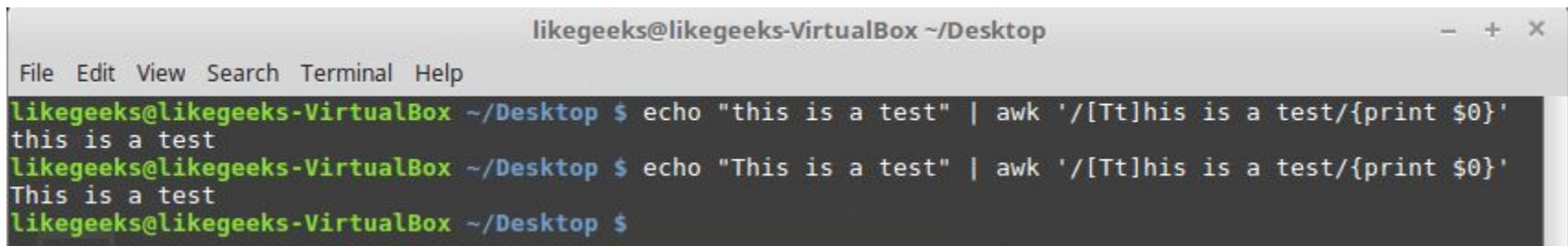
Классы оказываются очень кстати, если выполняется поиск слов, которые могут начинаться как с прописной, так и со строчной буквы:

```
$ echo "this is a test" | awk '/[Tt]his is a test/{print $0}'
```

```
$ echo "This is a test" | awk '/[Tt]his is a test/{print $0}'
```

Поиск слов, которые могут начинаться со строчной или прописной буквы

Классы символов не ограничены буквами. Тут можно использовать и другие символы. Нельзя заранее сказать, в какой ситуации понадобятся классы — всё зависит от решаемой задачи.

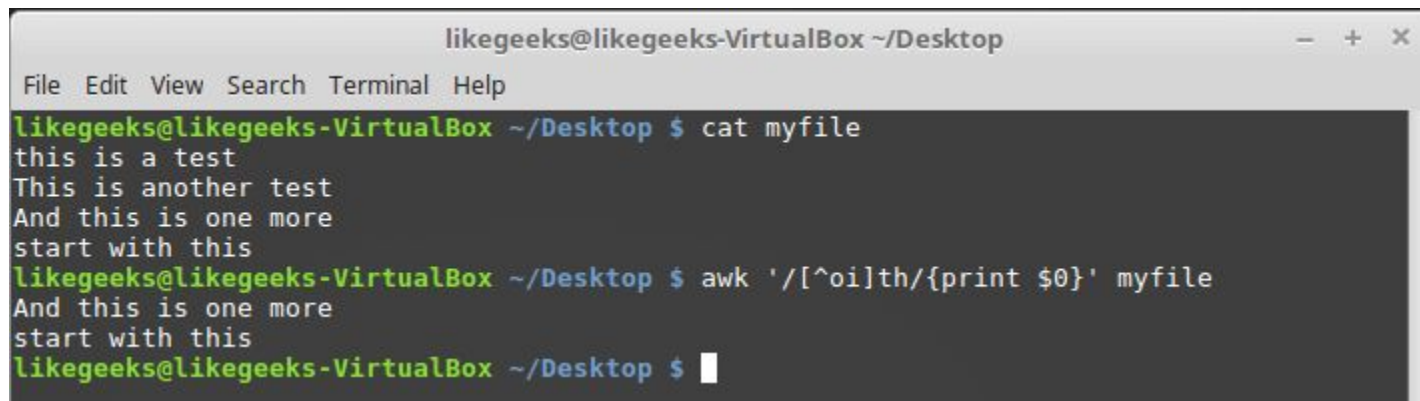


```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "this is a test" | awk '/[Tt]his is a test/{print $0}'
this is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "This is a test" | awk '/[Tt]his is a test/{print $0}'
This is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Классы символов можно использовать и для решения задачи, обратной описанной выше. А именно, вместо поиска символов, входящих в класс, можно организовать поиск всего, что в класс не входит. Для того, чтобы добиться такого поведения регулярного выражения, перед списком символов класса нужно поместить знак `^`. Выглядит это так:

```
$ awk '/[^oi]th/{print $0}' myfile
```

В данном случае будут найдены последовательности символов «th», перед которыми нет ни «o», ни «i».



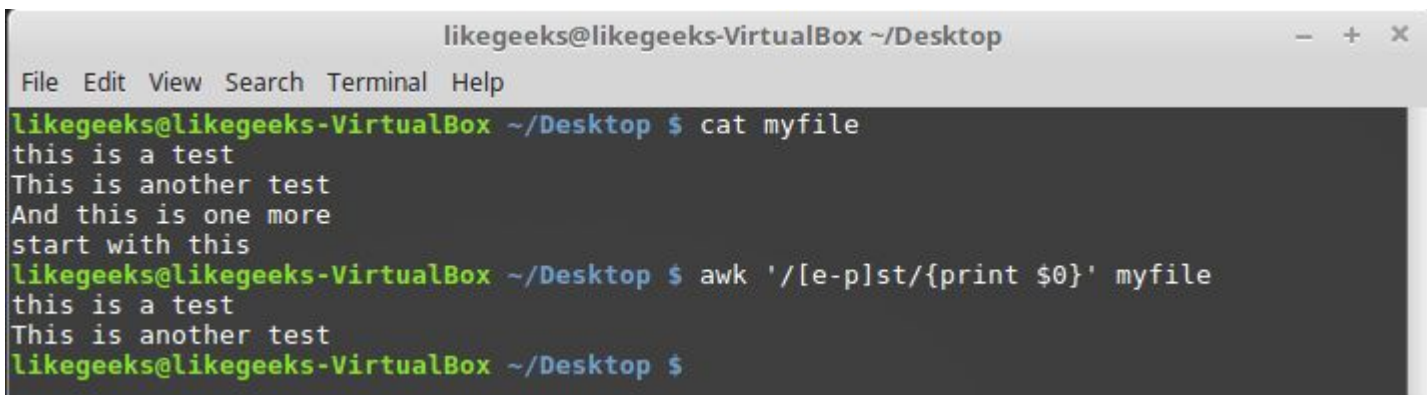
```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
this is a test
This is another test
And this is one more
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/[^oi]th/{print $0}' myfile
And this is one more
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

В символьных классах можно описывать диапазоны символов, используя типе:

```
$ awk '/[e-p]st/{print $0}' myfile
```

В данном примере регулярное выражение реагирует на последовательность символов «st», перед которой находится любой символ, расположенный, в алфавитном порядке, между символами «e» и «p».

Диапазоны можно создавать и из чисел



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
this is a test
This is another test
And this is one more
start with this
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '/[e-p]st/{print $0}' myfile
this is a test
This is another test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Если в шаблоне после символа поместить звёздочку, это будет означать, что регулярное выражение сработает, если символ появляется в строке любое количество раз — включая и ситуацию, когда символ в строке отсутствует.

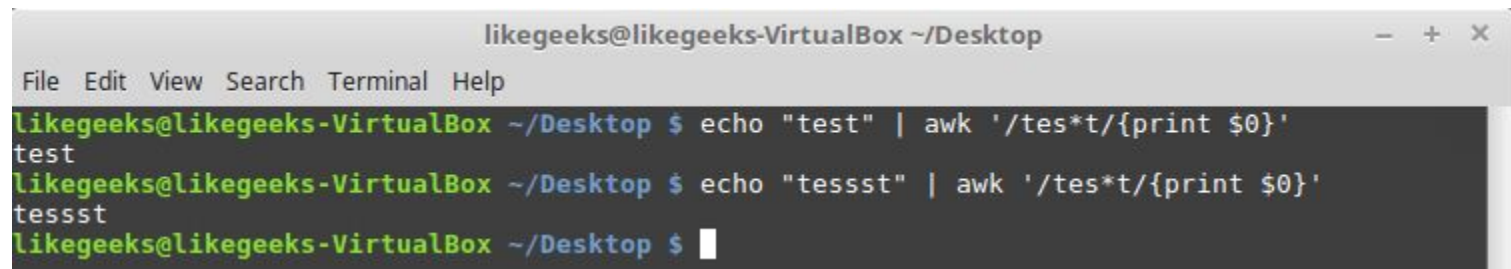
```
$ echo "test" | awk '/tes*t/{print $0}'
```

```
$ echo "tessst" | awk '/tes*t/{print $0}'
```

Этот шаблонный символ обычно используют для работы со словами, в которых постоянно встречаются опечатки, или для слов, допускающих разные варианты корректного написания:

```
$ echo "I like green color" | awk '/colou*r/{print $0}'
```

```
$ echo "I like green colour " | awk '/colou*r/{print $0}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "test" | awk '/tes*t/{print $0}'
test
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "tessst" | awk '/tes*t/{print $0}'
tessst
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Вопросительный знак указывает на то, что предшествующий символ может встретиться в тексте один раз или не встретиться вовсе. Этот символ — один из метасимволов повторений. Вот несколько примеров:

```
$ echo "tet" | awk '/tes?t/{print $0}'
```

```
$ echo "test" | awk '/tes?t/{print $0}'
```

```
$ echo "tesst" | awk '/tes?t/{print $0}'
```

Символ «плюс» в шаблоне указывает на то, что регулярное выражение обнаружит искомое в том случае, если предшествующий символ встретится в тексте один или более раз. При этом на отсутствие символа такая конструкция реагировать не будет:

```
$ echo "test" | awk '/te+st/{print $0}'
```

```
$ echo "teest" | awk '/te+st/{print $0}'
```

```
$ echo "tst" | awk '/te+st/{print $0}'
```

Фигурные скобки, которыми можно пользоваться в ERE-шаблонах, похожи на символы, рассмотренные выше, но они позволяют точнее задавать необходимое число вхождений предшествующего им символа. Указывать ограничение можно в двух форматах:

n — число, задающее точное число искомых вхождений

n, m — два числа, которые трактуются так: «как минимум n раз, но не больше чем m ».

Вот примеры первого варианта:

```
$ echo "tst" | awk '/te{1}st/{print $0}'
```

```
$ echo "test" | awk '/te{1}st/{print $0}'
```

Фигурные скобки в шаблонах, поиск точного числа вхождений

В старых версиях awk нужно было использовать ключ командной строки `--re-interval` для того, чтобы программа распознавала интервалы в регулярных выражениях, но в новых версиях этого делать не нужно.

```
$ echo "tst" | awk '/te{1,2}st/{print $0}'
```

```
$ echo "test" | awk '/te{1,2}st/{print $0}'
```

```
$ echo "teest" | awk '/te{1,2}st/{print $0}'
```

```
$ echo "teeest" | awk '/te{1,2}st/{print $0}'
```

Символ логического «или»

Символ | — вертикальная черта, означает в регулярных выражениях логическое «или». Обработывая регулярное выражение, содержащее несколько фрагментов, разделённых таким знаком, движок сочтёт анализируемый текст подходящим в том случае, если он будет соответствовать любому из фрагментов. Вот пример:

```
$ echo "This is a test" | awk '/test|exam/{print $0}'
```

```
$ echo "This is an exam" | awk '/test|exam/{print $0}'
```

```
$ echo "This is something else" | awk '/test|exam/{print $0}'
```


Группировка фрагментов регулярных выражений

Фрагменты регулярных выражений можно группировать, пользуясь круглыми скобками. Если сгруппировать некую последовательность символов, она будет восприниматься системой как обычный символ. То есть, например, к ней можно будет применить метасимволы повторений. Вот как это выглядит:

```
$ echo "Like" | awk '/Like(Geeks)?/{print $0}'
```

```
$ echo "LikeGeeks" | awk '/Like(Geeks)?/{print $0}'
```

Проверка адресов электронной почты

`^([a-zA-Z0-9_\-\.]+)@`

Это регулярное выражение можно прочесть так: «В начале строки должен быть как минимум один символ из тех, которые имеются в группе, заданной в квадратных скобках, а после этого должен идти знак @».

Теперь — очередь имени хоста — `hostname`. Тут применимы те же правила, что и для имени пользователя, поэтому шаблон для него будет выглядеть так:

`([a-zA-Z0-9_\-\.]+)`

Имя домена верхнего уровня подчиняется особым правилам. Тут могут быть лишь алфавитные символы, которых должно быть не меньше двух (например, такие домены обычно содержат код страны), и не больше пяти.

Всё это значит, что шаблон для проверки последней части адреса будет таким:

```
\.([a-zA-Z]{2,5})$
```

Прочсть его можно так: «Сначала должна быть точка, потом — от 2 до 5 алфавитных символов, а после этого строка заканчивается».

Подготовив шаблоны для отдельных частей регулярного выражения, соберём их вместе:

```
^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$
```

grep

grep — утилита командной строки, которая находит на вводе строки, отвечающие заданному регулярному выражению, и выводит их, если вывод не отменён специальным ключом. Название представляет собой акроним английской фразы «search globally for lines matching the regular expression, and print them» — «искать везде строки, соответствующие регулярному выражению, и выводить их».

Изначально была создана для операционной системы UNIX.

Существуют модификации grep: egrep (с обработкой расширенных регулярных выражений), fgrep (тракующая символы `$*[]^|()\` буквально), rgrep (с включённым рекурсивным поиском). Как сказано в руководстве man «egrep — то же самое, что grep -E. fgrep — то же самое, что grep -F. rgrep — то же самое, что grep -r».

Для начала о том как мы обычно grep'аем файлы.

Используя cat:

```
# cat /var/run/dmesg.boot | grep CPU:
```

```
CPU: Intel(R) Core(TM)2 Quad CPU    Q9550  @ 2.83GHz (2833.07-MHz  
K8-class CPU)
```

Можно и так:

```
# grep CPU: /var/run/dmesg.boot
```

```
CPU: Intel(R) Core(TM)2 Quad CPU    Q9550  @ 2.83GHz (2833.07-MHz  
K8-class CPU)
```

Сделаем тестовый файл

one two three
seven eight one eight three
thirteen fourteen fifteen

sixteen seventeen eighteen seven
sixteen seventeen eighteen
twenty seven

one 504 one
one 503 one
one 504 one
one 504 one

#comment UP
twentyseven
#comment down

twenty1
twenty3
twenty5
twenty7

Опция -w позволяет искать по слову целиком:

```
$ grep -w 'seven' test.txt
```

seven eight one eight three

sixteen seventeen eighteen seven

twenty seven

Стоящие в начале или конце строки?

```
$ grep '^seven' test.txt
```

seven eight one eight three

```
$ grep 'seven$' test.txt
```

sixteen seventeen eighteen seven

twenty seven

twentyseven

Хотите увидеть строки в окрестности искомой?

```
$ grep -C 1 twentyseven test.txt
```

```
#comment UP
```

```
twentyseven
```

```
    #comment down
```

Только снизу или сверху?

```
$ grep -A 1 twentyseven test.txt
```

```
twentyseven
```

```
    #comment down
```

```
$ grep -B 1 twentyseven test.txt
```

```
#comment UP
```

А ещё мы умеем так

```
$ grep "twenty[1-4]" test.txt
```

twenty1

twenty3

И наоборот исключая эти

```
$ grep "twenty[^1-4]" test.txt
```

twenty seven

twentyseven

twenty5

twenty7

А если нужно по началу или концу слова?

```
$ grep '\<seven' test.txt
```

seven eight one eight three

sixteen seventeen eighteen seven

sixteen seventeen eighteen

twenty seven

```
$ grep 'seven\>' test.txt
```

seven eight one eight three

sixteen seventeen eighteen seven

twenty seven

Пару практических примеров

```
$ cat /etc/resolv.conf
```

```
#options edns0
```

```
#nameserver 127.0.0.1
```

```
nameserver 8.8.8.8
```

```
nameserver 77.88.8.8
```

```
nameserver 8.8.4.4
```

Отбираем только строки с ip:

```
$ grep -E "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" /etc/resolv.conf
```

```
#nameserver 127.0.0.1
```

```
nameserver 8.8.8.8
```

```
nameserver 77.88.8.8
```

```
nameserver 8.8.4.4
```

```
$ grep -E '\b[0-9]{1,3}(\.[0-9]{1,3}){3}\b' /etc/resolv.conf
```

```
#nameserver 127.0.0.1
```

```
nameserver 8.8.8.8
```

```
nameserver 77.88.8.8
```

```
nameserver 8.8.4.4
```

Уберём строку с комментарием?

```
$ grep -E '\b[0-9]{1,3}(\.[0-9]{1,3}){3}\b' /etc/resolv.conf | grep -v '#'
```

nameserver 8.8.8.8

nameserver 77.88.8.8

nameserver 8.8.4.4

А теперь выберем только сами ip

```
$ grep -oE '\b[0-9]{1,3}(\.[0-9]{1,3}){3}\b' /etc/resolv.conf | grep -v '#'
```

127.0.0.1

8.8.8.8

77.88.8.8

8.8.4.4

Закомментированная строка вернулась. Это связано с особенностью обработки шаблонов. Как быть? Вот так:

```
$ grep -v '#' /etc/resolv.conf | grep -oE '\b[0-9]{1,3}(\.[0-9]{1,3}){3}\b'
```

8.8.8.8

77.88.8.8

8.8.4.4

Полезные ресурсы

[Команда grep — человеческий man](#)

[О grep для начинающих](#)

[A Beginner's Guide to Grep: Basics and Regular Expressions](#)

[Beginner's Guide to Grep](#)

sort

Команда `sort` сортирует содержимое файла в алфавитном или нумерологическом порядке. Если задать несколько файлов, то команда `sort` соединит их и, рассортировав, выдаст единым выводом. По умолчанию, объектом сортировки будут строки, однако опции позволяют выбирать объект сортировки: колонки, столбцы и прочие элементы форматирования файла. Разделителем между ними служат пробелы, однако соответствующие опции позволяют задать иные разделители.

Команда `sort` весьма древняя, она может служить образцом программирования утилит в ранних 70-х годах прошлого века. У команды множество опций, и их разнообразные сочетания, а также способы задания разделителей, хорошо развивают память и воображение.

файл debts.txt:

Vova: 100\$ -- September 3 2008

Sergey: 10\$ -- December 30 2008

Misha: 25\$ -- May 12 2008

Taras: 500\$ -- June 24 2008

\$ sort debts.txt

Misha: 25\$ -- May 12 2008

Sergey: 10\$ -- December 30 2008

Taras: 500\$ -- June 24 2008

Vova: 100\$ -- September 3 2008

```
$ sort -r debts.txt
```

Vova: 100\$ -- September 3 2008

Taras: 500\$ -- June 24 2008

Sergey: 10\$ -- December 30 2008

Misha: 25\$ -- May 12 2008

```
$ sort -nrk 2 debts.txt
```

Taras: 500\$ -- June 24 2008

Vova: 100\$ -- September 3 2008

Misha: 25\$ -- May 12 2008

Sergey: 10\$ -- December 30 2008

```
$ sort -k 4M debts.txt
```

Misha: 25\$ -- May 12 2008

Taras: 500\$ -- June 24 2008

Vova: 100\$ -- September 3 2008

Sergey: 10\$ -- December 30 2008

```
sort -t '/' -k2 /etc/shells
```

```
/bin/ash
```

```
/bin/bash
```

```
/bin/csh
```

```
/bin/ksh
```

```
/bin/tcsh
```

```
/bin/zsh
```

uniq

uniq — утилита Unix, с помощью которой можно вывести или отфильтровать повторяющиеся строки в файле. Если входной файл задан как («-») или не задан вовсе, чтение производится из стандартного ввода. Если выходной файл не задан, запись производится в стандартный вывод. Вторая и последующие копии повторяющихся соседних строк не записываются. Повторяющиеся входные строки не распознаются, если они не следуют строго друг за другом, поэтому может потребоваться предварительная сортировка файлов.

```
$ echo -e 1234\\n2345\\n3456\\n1111\\n1111\\n1111 | uniq
```

1234

2345

3456

1111

```
echo -e 1111\\n2345\\n1111\\n3456\\n1111 | uniq
```

1111

2345

1111

3456

1111

```
echo -e 1234\\n1111\\n2345\\n1111\\n3456\\n1111 | sort | uniq
```

Опция -с

--count

Сообщит, сколько было одинаковых строк до их урезания:

```
$ echo -e кот\\nконь\\nсобака\\nкрыса\\nкрыса | uniq -c
```

1 кот

1 конь

1 собака

2 крыса

Опция -d

--repeated

Эта опция, наоборот, выведет лишь ту строку, которая повторялась в тексте:

```
$ echo -e кот\\нконь\\нсобака\\нкрыса\\нкрыса | uniq -d
```

крыса

tr

Команда tr служит для перевода (замены) выбранных символов в другие символы или удаления их.

В отличие от большинства других программ командной строки, команда tr не принимает имен файлов в качестве аргумента. Ввод команды tr осуществляется или со стандартного ввода, или с вывода других программ путем перенаправления.

В следующем примере, каждая буква a будет заменена буквой b:

```
$ tr a b
```

Понятно, что возможности команды tr не ограничиваются заменой одной буквы. Команда может заменять любое количество указанных символов на другие символы. В этом случае каждый из наборов символов заключается в квадратные скобки, а скобки, в свою очередь, в кавычки; безразлично, двойные или одинарные.

```
'[набор1]' '[набор2]' или "[набор1]" "[набор2]"
```

В наборе1 один подряд перечисляются символы, подлежащие замене, а в наборе2 - в соответствующем порядке символы, которые их должны заменить:

```
echo cheer | tr 'abcdefghijklmnopqrstuvwxyz' '[hijklmnopqrstuvwxyzabcdefg]'
```

```
jolly
```

WC

Команда `wc` подсчитывает количество строк, слов, байт, или символов в текстовом файле.

Система отвечает строкой в следующем формате:

```
l      w      c      файл
```

где `l` - число строк в файле;

`w` - число слов в файле;

`c` - число символов в файле.

Например, чтобы подсчитать число строк, слов и символов в файле `johnson`, находящегося в текущем справочнике, введите команду:

```
$ wc johnson<CR>
```

```
24 66 406 johnson
```

```
$
```

Система отвечает, что в файле `johnson` 24 строки, 66 слов и 406 символов.

Чтобы получить только число строк, или число слов, или число символов, выберите один из соответствующих форматов командной строки:

`wc -l файл<CR>` (число строк)

`wc -w файл<CR>` (число слов)

`wc -c файл<CR>` (число символов)

Например, если вы используете ключ `-l`, то система напечатает только число строк в файле `sanders`:

```
$ wc -l sanders<CR>
```

```
28 sanders
```

diff

В вычислительной технике diff — утилита сравнения файлов, выводящая разницу между двумя файлами. Эта программа выводит построчно изменения, сделанные в файле (для текстовых файлов). Современные реализации поддерживают также двоичные файлы. Вывод утилиты называется «diff», или, что более распространено, патч, так как он может быть применён с программой patch. Вывод других утилит сравнения файлов также часто называется «diff».

diff: очень гибкая утилита сравнения файлов. Она выполняет построчное сравнение файлов. В отдельных случаях, таких как поиск по словарю, может оказаться полезной фильтрация файлов с помощью sort и uniq перед тем как отдать поток данных через конвейер утилите diff. diff file-1 file-2 -- выведет строки, имеющие отличия, указывая -- какому файлу, какая строка принадлежит.

С ключом --side-by-side, команда diff выведет сравниваемые файлы в две колонки, с указанием несовпадающих строк. Ключи -с и -и так же служат для облегчения интерпретации результатов работы diff.

Файл file1:

test

test2

test3

Файл file2:

test

test23

test3

\$ diff file1 file2

2c2

< test2

> test23

2с2

< test2

> test23

Первое числовое значение соответствует номеру строки (или диапазону номеров строк) из файла с именем file1 (оригинального файла), а последнее значение - номеру строки (или диапазону номеров строк) из файла с именем file2 (нового файла). В качестве буквенного символа может использоваться символ а, указывающий на то, что данные должны быть добавлены, символ d, указывающий на то, что данные должны быть удалены, а также символ с, указывающий на то, что данные должны быть изменены.

Таким образом, строки 2с2 говорит о том, что вторая строка оригинального файла была модифицирована и должна быть заменена на вторую строку из нового файла для того, чтобы файлы стали идентичными. Если вы сравните два файла вручную (file1 и file2) вы сможете самостоятельно убедиться в справедливости данного утверждения.

В случае строк, следующих после строки 2с2 в приведенном выше примере, все гораздо проще: строка, начинающаяся с символа < содержит ничто иное, как вторую строку из файла с именем file1, а строка, начинающаяся с символа > - ничто иное, как интересующую нас строку из файла с именем file2. Строка с тремя дефисами (---) используется исключительно для разделения описанных выше строк.

cut

Команда `cut` используется для выборки колонок из таблицы или полей из каждой строки файла; если применить терминологию баз данных, команда `cut` выполняет операцию проекции отношения. Поля, специфицированные списком, могут быть фиксированной длины, то есть расположенные как на перфокарте (опция `-s`), или переменной длины, изменяющейся от строки к строке; в этом случае границей поля является символ-разделитель, например, символ табуляции (опция `-f`). Команду можно использовать как фильтр: если не указано ни одного файла или задано имя `-`, используется стандартный ввод. Результат всегда поступает на стандартный вывод.

Исходный файл

\$ cat list-of-smartphones-2014.txt

Model:Company:Price:Camera:4G

IPhone4:Apple:1000\$:Yes:Yes

Galaxy:Samsung:900\$:Yes:Yes

Optimus:LG:800\$:Yes:Yes

Sensation:HTC:400\$:Yes:Yes

IPhone4S:Apple:1100\$:Yes:Yes

N9:Nokia:400\$:Yes:Yes

вырезать столбец текста

```
$ cut -d: -f1 list-of-smartphones-2011.txt
```

Model

IPhone4

Galaxy

Optimus

Sensation

IPhone4S

N9

Вырезание определенного количества символов

```
$ cut -c 1-9 list-of-smartphones-2011.txt
```

Model:Com

IPhone4:A

Galaxy:Sa

Optimus:L

Sensation

IPhone4S:

N9:Nokia:

Вырезание по разделителю

```
$ cut -d: -f2 list-of-smartphones-2011.txt
```

Company

Apple

Samsung

LG

HTC

Apple

Nokia

Пример cut + sed

```
$ sed 's/:/t/g' list-of-smartphones-2011.txt | cut -f 1
```

Model

IPhone4

Galaxy

Optimus

Sensation

IPhone4S

N9

первый символ строки

```
$ cut -c 1 list-of-smartphones-2011.txt
```

M

I

G

O

S

I

N

Выведение нескольких столбцов

```
$ cut -d: -f '1 2' list-of-smartphones-2011.txt
```

Model:Company

IPhone4:Apple

Galaxy:Samsung

Optimus:LG

Sensation:HTC

IPhone4S:Apple

N9:Nokia

Дополнительные темы

Текстовый редактор Vi (vim)

Текстовый редактор Emacs

Установка Guest Additions на гостевую Linux-систему