

[КАК СТАТЬ АВТОРОМ](#)

Технотекст

True or false? И при чём здесь кобра

ХОСТИНГ ВЫШЕ ОБЛАКОВ
Аптайм на космическом уровне

CRU VDS

Михаил Корниенко
космонавт, Герой России

Время пребывания в космосе — 44 618 400 с
Средний аптайм сервера RUVDS — 43 518 300 с

Реклама ООО «ИТ ФИНАНС», сайт: [LINUXKARS](#)

**2902.28**

Рейтинг

RUVDS.comVDS/VPs-хостинг. Скидка 15% по коду **HABR15****ru_vds**

2 мая 2017 в 18:04

Bash-скрипты, часть 8: язык обработки данных awk

🕒 14 мин 👁 299K

Блог компании RUVDS.com, Настройка Linux*, Системное администрирование*

[Перевод](#)Автор оригинала: Администратор [likegeeks.com](#)

Bash-скрипты: начало

Bash-скрипты, часть 2: циклы

Bash-скрипты, часть 3: параметры и ключи командной строки

Bash-скрипты, часть 4: ввод и вывод

Bash-скрипты, часть 5: сигналы, фоновые задачи, управление сценариями

Bash-скрипты, часть 6: функции и разработка библиотек

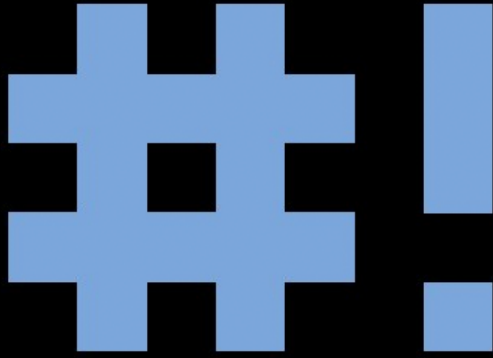
Bash-скрипты, часть 7: sed и обработка текстов

Bash-скрипты, часть 8: язык обработки данных awk

Bash-скрипты, часть 9: регулярные выражения

Bash-скрипты, часть 10: практические примеры

Bash-скрипты, часть 11: expect и автоматизация интерактивных утилит



В прошлый раз мы говорили о потоковом редакторе `sed` и рассмотрели немало примеров обработки текста с его помощью. `Sed` способен решать многие задачи, но есть у него и ограничения. Иногда нужен более совершенный инструмент для обработки данных, нечто вроде языка программирования. Собственно говоря, такой инструмент – `awk`.

Habrahabr10

Промо-код для скидки в 10% на наши виртуальные сервера

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.

Если говорить лишь о возможности создавать форматированные отчёты, которые удобно читать и анализировать, то это оказывается очень кстати при работе с лог-файлами, которые могут содержать миллионы записей. Но `awk` – это намного больше, чем средство подготовки отчётов.

Особенности вызова `awk`

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.

↑ +24 ↓

406



18

предназначенный для чтения данных, их обработки и вывода результатов.

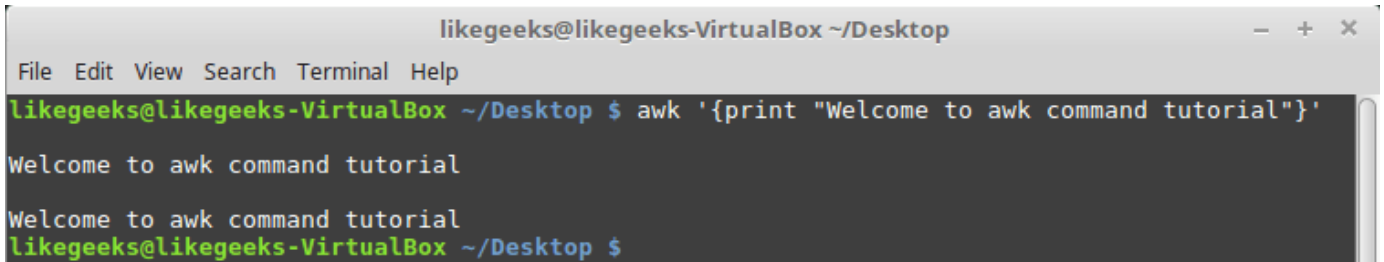
Чтение awk-скриптов из командной строки

Скрипты awk, которые можно писать прямо в командной строке, оформляются в виде текстов команд, заключённых в фигурные скобки. Кроме того, так как awk предполагает, что скрипт представляет собой текстовую строку, его нужно заключить в одинарные кавычки:

```
$ awk '{print "Welcome to awk command tutorial"}'
```

Запустим эту команду... И ничего не произойдёт Дело тут в том, что мы, при вызове awk, не указали файл с данными. В подобной ситуации awk ожидает поступления данных из STDIN. Поэтому выполнение такой команды не приводит к немедленно наблюдаемым эффектам, но это не значит, что awk не работает – он ждёт входных данных из STDIN.

Если теперь ввести что-нибудь в консоль и нажать `Enter`, `awk` обработает введенные данные с помощью скрипта, заданного при его запуске. `Awk` обрабатывает текст из потока ввода построчно, этим он похож на `sed`. В нашем случае `awk` ничего не делает с данными, он лишь, в ответ на каждую новую полученную им строку, выводит на экран текст, заданный в команде `print`.

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 prompt 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' followed by the command 'awk '{print "Welcome to awk command tutorial"}''. The output shows 'Welcome to awk command tutorial' printed twice, once for each line of input. The prompt returns to 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'.

Первый запуск `awk`, вывод на экран заданного текста

Что бы мы ни ввели, результат в данном случае будет одним и тем же – вывод текста. Для того, чтобы завершить работу `awk`, нужно передать ему символ конца файла (EOF, End-of-File). Сделать это можно, воспользовавшись сочетанием клавиш `CTRL + D`.

Неудивительно, если этот первый пример показался вам не особо впечатляющим. Однако, самое интересное – впереди.

Позиционные переменные, хранящие данные полей

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:

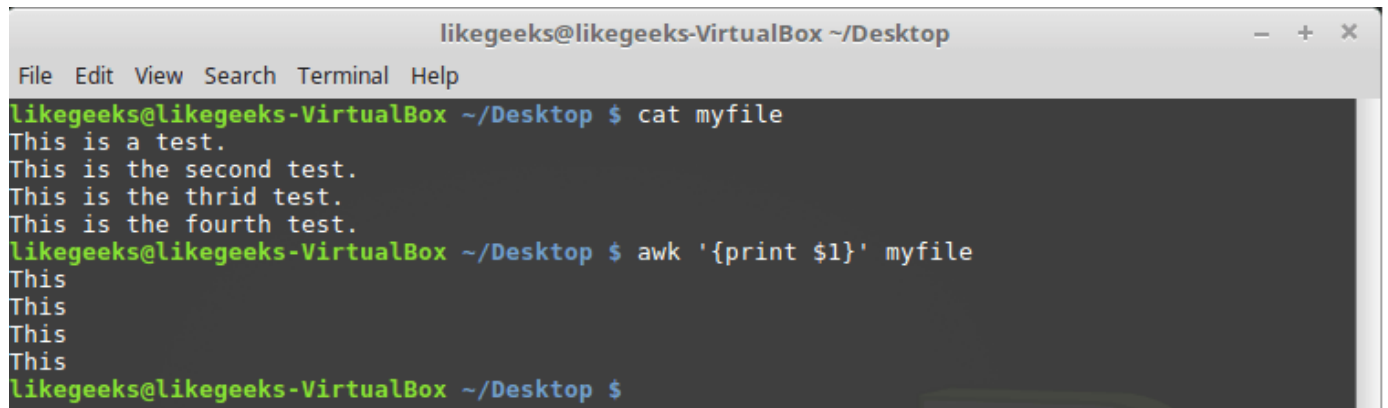
- `$0` — представляет всю строку текста (запись).
- `$1` — первое поле.
- `$2` — второе поле.
- `$n` — `n`-ное поле.

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

Рассмотрим использование этих переменных на простом примере. А именно, обработаем файл, в котором содержится несколько строк (этот файл показан на рисунке ниже) с

помощью такой команды:

```
$ awk '{print $1}' myfile
```



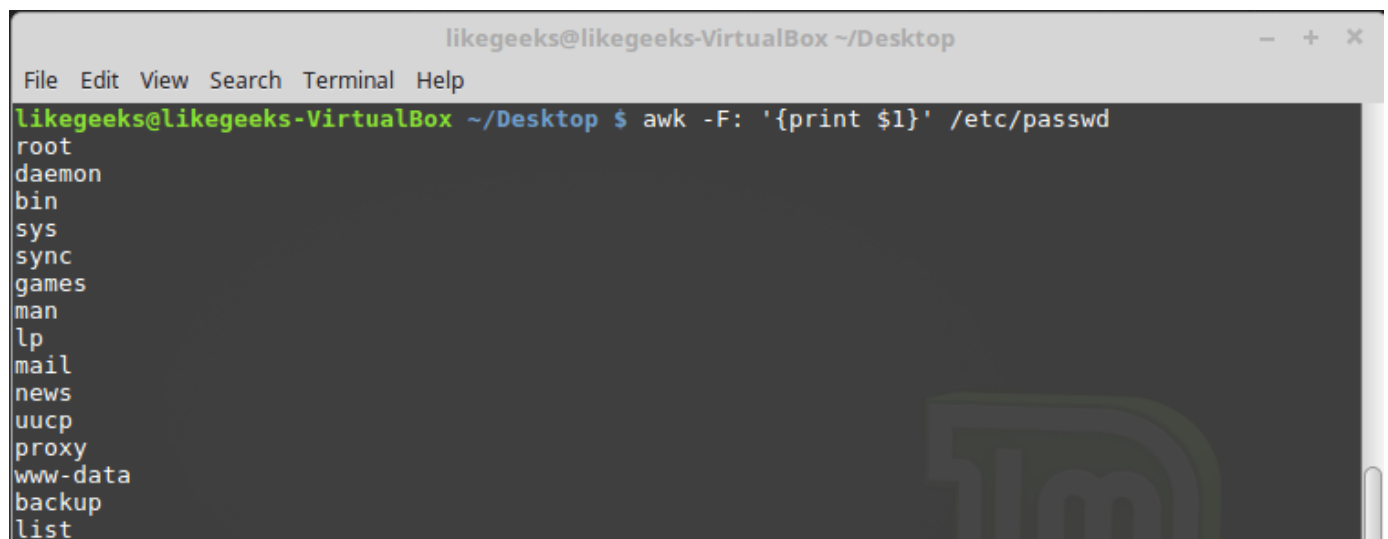
```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
This is a test.
This is the second test.
This is the thrid test.
This is the fourth test.
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{print $1}' myfile
This
This
This
This
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Вывод в консоль первого поля каждой строки

Здесь использована переменная `$1` , которая позволяет получить доступ к первому полю каждой строки и вывести его на экран.

Иногда в некоторых файлах в качестве разделителей полей используется что-то, отличающееся от пробелов или символов табуляции. Выше мы упоминали ключ `awk -F` , который позволяет задать необходимый для обработки конкретного файла разделитель:

```
$ awk -F: '{print $1}' /etc/passwd
```



```
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
```

Указание символа-разделителя при вызове awk

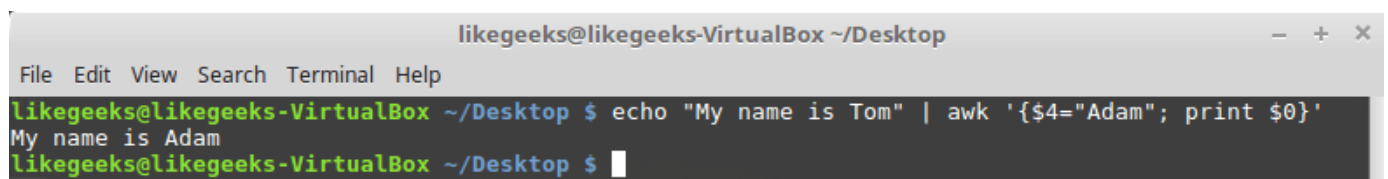
Эта команда выводит первые элементы строк, содержащихся в файле `/etc/passwd` . Так

как в этом файле в качестве разделителей используются двоеточия, именно этот символ был передан awk после ключа `-F`.

Использование нескольких команд

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 terminal shows the command 'echo "My name is Tom" | awk '{\$4="Adam"; print \$0}'' being executed. The output is 'My name is Adam'. The prompt is 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'.

Вызов awk из командной строки с передачей ему многострочного скрипта

В данном примере первая команда записывает новое значение в переменную `$4`, а вторая выводит на экран всю строку.

Чтение скрипта awk из файла

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
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat testfile
{print $1 " has a home directory at " $6}
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk -F: -f testfile /etc/passwd
root has a home directory at /root
daemon has a home directory at /usr/sbin
bin has a home directory at /bin
sys has a home directory at /dev
sync has a home directory at /bin
games has a home directory at /usr/games
man has a home directory at /var/cache/man
lp has a home directory at /var/spool/lpd
mail has a home directory at /var/mail
news has a home directory at /var/spool/news
uucp has a home directory at /var/spool/uucp
proxy has a home directory at /bin
www-data has a home directory at /var/www
backup has a home directory at /var/backups
list has a home directory at /var/list
```

Вызов awk с указанием файла скрипта

Тут мы выводим из файла `/etc/passwd` имена пользователей, которые попадают в переменную `$1`, и их домашние директории, которые попадают в `$6`. Обратите внимание на то, что файл скрипта задают с помощью ключа `-f`, а разделитель полей, двоеточие в нашем случае, с помощью ключа `-F`.

В файле скрипта может содержаться множество команд, при этом каждую из них достаточно записывать с новой строки, ставить после каждой точку с запятой не требуется.

Вот как это может выглядеть:

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

Тут мы храним текст, используемый при выводе данных, полученных из каждой строки обрабатываемого файла, в переменной, и используем эту переменную в команде `print`. Если воспроизвести предыдущий пример, записав этот код в файл `testfile`, выведено будет то же самое.

Выполнение команд до начала обработки данных

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

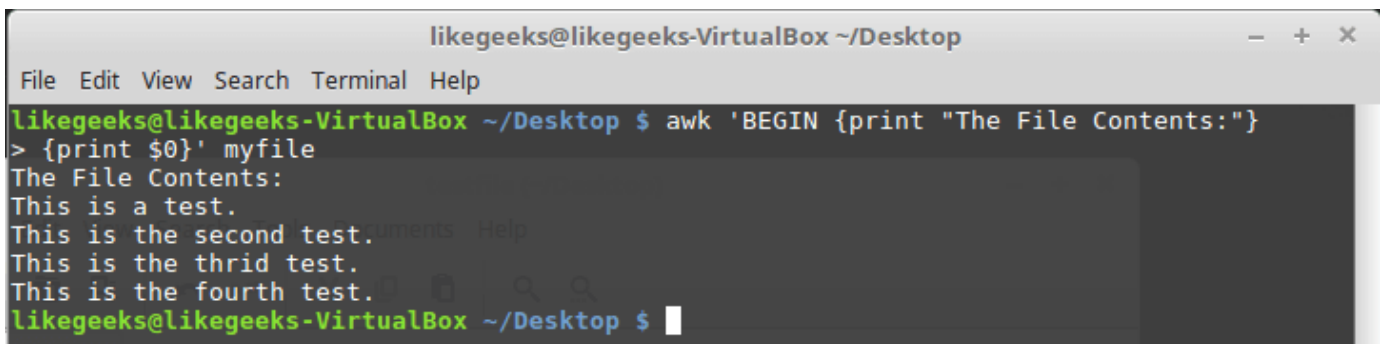
Для этого можно воспользоваться ключевым словом `BEGIN`. Команды, которые следуют за

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
```



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

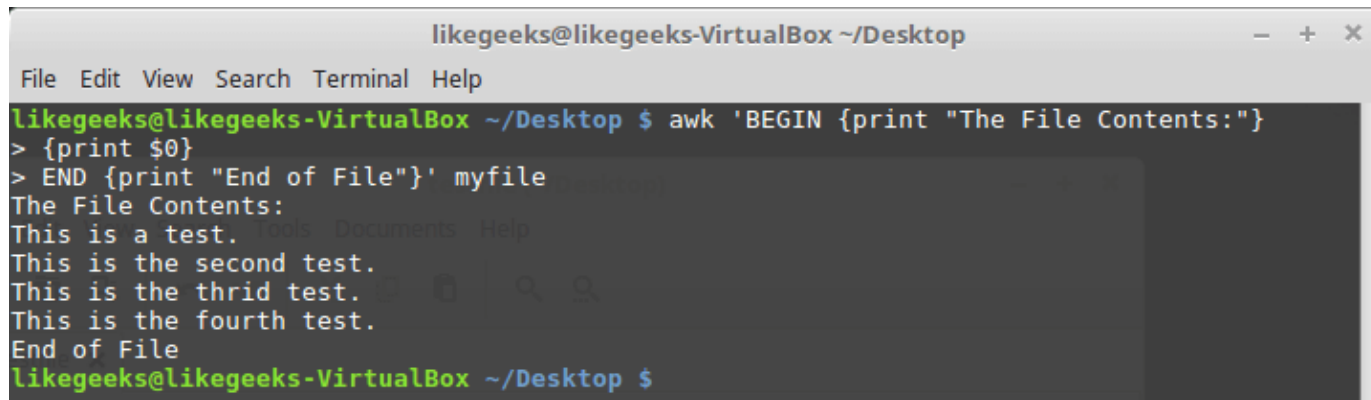
Выполнение команд до начала обработки данных

Сначала awk исполняет блок BEGIN , после чего выполняется обработка данных. Будьте внимательны с одинарными кавычками, используя подобные конструкции в командной строке. Обратите внимание на то, что и блок BEGIN , и команды обработки потока, являются в представлении awk одной строкой. Первая одинарная кавычка, ограничивающая эту строку, стоит перед BEGIN . Вторая – после закрывающей фигурной скобки команды обработки данных.

Выполнение команд после окончания обработки данных

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
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
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 $
```

Результаты работы скрипта, в котором имеются блоки BEGIN и END

После завершения вывода содержимого файла, awk выполняет команды блока END . Это полезная возможность, с её помощью, например, можно сформировать подвал отчёта. Теперь напишем скрипт следующего содержания и сохраним его в файле `myscript` :

```
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.

Запустим скрипт:

```
$ awk -f myscript /etc/passwd
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk -f myscript /etc/passwd
The latest list of users and shells
  UserName      HomePath
  -----
root           /root
daemon         /usr/sbin
bin            /bin
sys            /dev
sync           /bin
games          /usr/games
man            /var/cache/man
lp             /var/spool/lpd
mail           /var/mail
news           /var/spool/news
uucp           /var/spool/uucp
proxy          /bin
www-data       /var/www
backup         /var/backups
list           /var/list
```

Обработка файла /etc/passwd с помощью awk-скрипта

Всё, о чём мы говорили выше – лишь малая часть возможностей awk. Продолжим освоение этого полезного инструмента.

Встроенные переменные: настройка процесса обработки данных

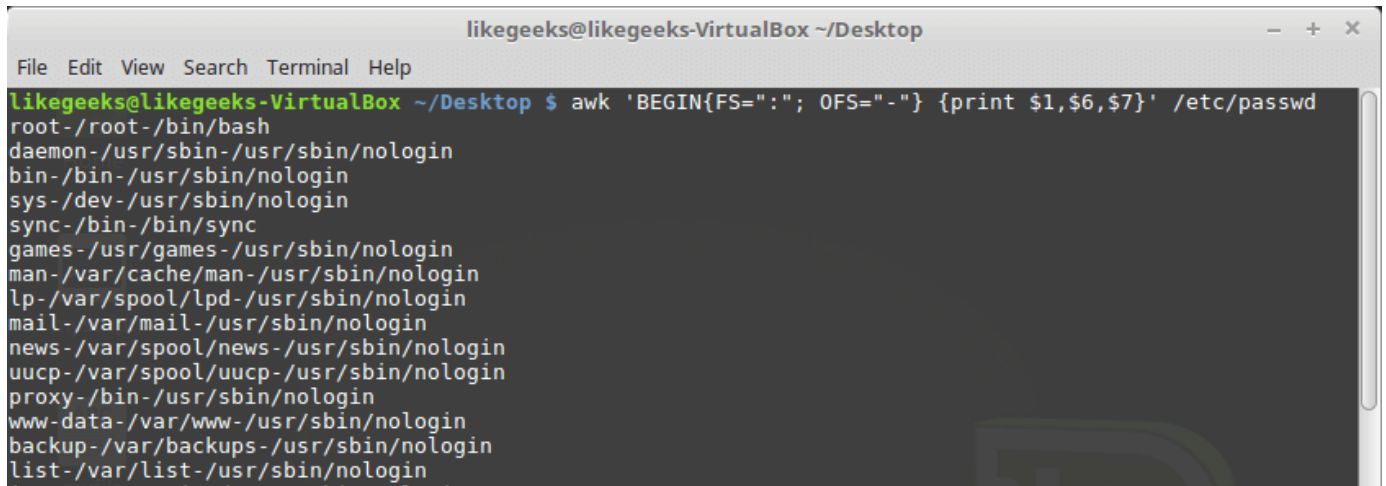
Утилита awk использует встроенные переменные, которые позволяют настраивать процесс обработки данных и дают доступ как к обрабатываемым данным, так и к некоторым сведениям о них.

Мы уже рассматривали позиционные переменные – \$1 , \$2 , \$3 , которые позволяют извлекать значения полей, работали мы и с некоторыми другими переменными. На самом деле, их довольно много. Вот некоторые из наиболее часто используемых:

- FIELDWIDTHS – разделённый пробелами список чисел, определяющий точную ширину каждого поля данных с учётом разделителей полей.
- FS – уже знакомая вам переменная, позволяющая задавать символ-разделитель полей.
- RS – переменная, которая позволяет задавать символ-разделитель записей.
- OFS – разделитель полей на выводе awk-скрипта.
- ORS – разделитель записей на выводе awk-скрипта.

По умолчанию переменная OFS настроена на использование пробела. Её можно установить так, как нужно для целей вывода данных:

```
$ awk 'BEGIN{FS=":"; OFS="-"} {print $1,$6,$7}' /etc/passwd
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{FS=":"; OFS="-"} {print $1,$6,$7}' /etc/passwd
root-/root-/bin/bash
daemon-/usr/sbin-/usr/sbin/nologin
bin-/bin-/usr/sbin/nologin
sys-/dev-/usr/sbin/nologin
sync-/bin-/bin/sync
games-/usr/games-/usr/sbin/nologin
man-/var/cache/man-/usr/sbin/nologin
lp-/var/spool/lpd-/usr/sbin/nologin
mail-/var/mail-/usr/sbin/nologin
news-/var/spool/news-/usr/sbin/nologin
uucp-/var/spool/uucp-/usr/sbin/nologin
proxy-/bin-/usr/sbin/nologin
www-data-/var/www-/usr/sbin/nologin
backup-/var/backups-/usr/sbin/nologin
list-/var/list-/usr/sbin/nologin
```

Установка разделителя полей выходного потока

Переменная `FIELDWIDTHS` позволяет читать записи без использования символа-разделителя полей.

В некоторых случаях, вместо использования разделителя полей, данные в пределах записей расположены в колонках постоянной ширины. В подобных случаях необходимо задать переменную `FIELDWIDTHS` таким образом, чтобы её содержимое соответствовало особенностям представления данных.

При установленной переменной `FIELDWIDTHS` `awk` будет игнорировать переменную `FS` и находить поля данных в соответствии со сведениями об их ширине, заданными в `FIELDWIDTHS`.

Предположим, имеется файл `testfile`, содержащий такие данные:

```
1235.9652147.91
927-8.365217.27
36257.8157492.5
```

Известно, что внутренняя организация этих данных соответствует шаблону 3-5-2-5, то есть, первое поле имеет ширину 3 символа, второе – 5, и так далее. Вот скрипт, который позволит разобрать такие записи:

```
$ awk 'BEGIN{FIELDWIDTHS="3 5 2 5"}{print $1,$2,$3,$4}' testfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{FIELDWIDTHS="3 5 2 5"}{print $1,
$2,$3,$4}' testfile
123 5.965 21 47.91
927 -8.36 52 17.27
362 57.81 57 492.5
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Использование переменной `FIELDWIDTHS`

Посмотрим на то, что выведет скрипт. Данные разобраны с учётом значения переменной `FIELDWIDTHS`, в результате числа и другие символы в строках разбиты в соответствии с заданной шириной полей.

Переменные `RS` и `ORS` задают порядок обработки записей. По умолчанию `RS` и `ORS` установлены на символ перевода строки. Это означает, что `awk` воспринимает каждую новую строку текста как новую запись и выводит каждую запись с новой строки.

Иногда случается так, что поля в потоке данных распределены по нескольким строкам. Например, пусть имеется такой файл с именем `addresses`:

```
Person Name
123 High Street
(222) 466-1234

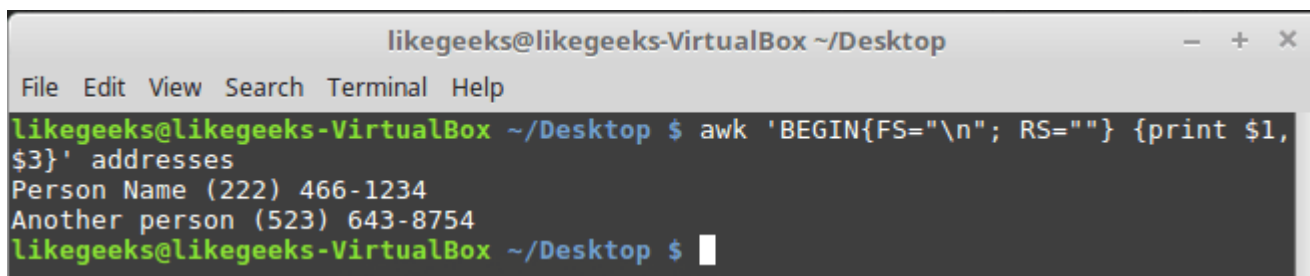
Another person
487 High Street
(523) 643-8754
```

Если попытаться прочесть эти данные при условии, что `FS` и `RS` установлены в значения по умолчанию, `awk` сочтёт каждую новую строку отдельной записью и выделит поля, опираясь на пробелы. Это не то, что нам в данном случае нужно.

Для того, чтобы решить эту проблему, в `FS` надо записать символ перевода строки. Это укажет `awk` на то, что каждая строка в потоке данных является отдельным полем.

Кроме того, в данном примере понадобится записать в переменную `RS` пустую строку. Обратите внимание на то, что в файле блоки данных о разных людях разделены пустой строкой. В результате `awk` будет считать пустые строки разделителями записей. Вот как всё это сделать:

```
$ awk 'BEGIN{FS="\n"; RS="" } {print $1,$3}' addresses
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{FS="\n"; RS="" } {print $1,$3}' addresses
Person Name (222) 466-1234
Another person (523) 643-8754
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Результаты настройки переменных RS и FS

Как видите, awk, благодаря таким настройкам переменных, воспринимает строки из файла как поля, а разделителями записей становятся пустые строки.

Встроенные переменные: сведения о данных и об окружении

Помимо встроенных переменных, о которых мы уже говорили, существуют и другие, которые предоставляют сведения о данных и об окружении, в котором работает awk:

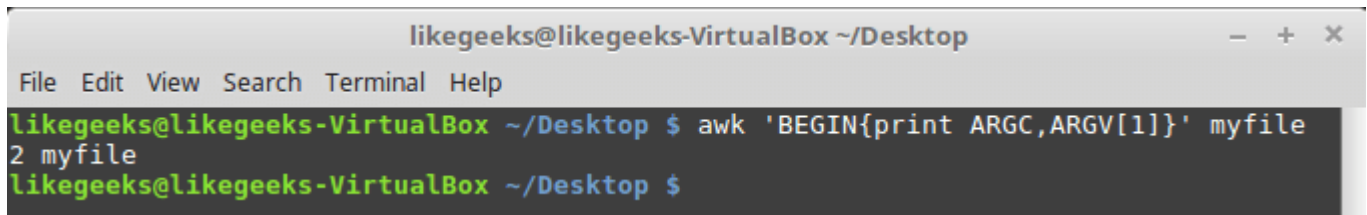
- ARGC – количество аргументов командной строки.
- ARGV – массив с аргументами командной строки.
- ARGIND – индекс текущего обрабатываемого файла в массиве ARGV .
- ENVIRON – ассоциативный массив с переменными окружения и их значениями.
- ERRNO – код системной ошибки, которая может возникнуть при чтении или закрытии входных файлов.
- FILENAME – имя входного файла с данными.
- FNR – номер текущей записи в файле данных.
- IGNORECASE – если эта переменная установлена в ненулевое значение, при обработке игнорируется регистр символов.
- NF – общее число полей данных в текущей записи.
- NR – общее число обработанных записей.

Переменные ARGC и ARGV позволяют работать с аргументами командной строки. При этом скрипт, переданный awk, не попадает в массив аргументов ARGV . Напишем такой скрипт:

```
$ awk 'BEGIN{print ARGC,ARGV[1]}' myfile
```

После его запуска можно узнать, что общее число аргументов командной строки – 2, а

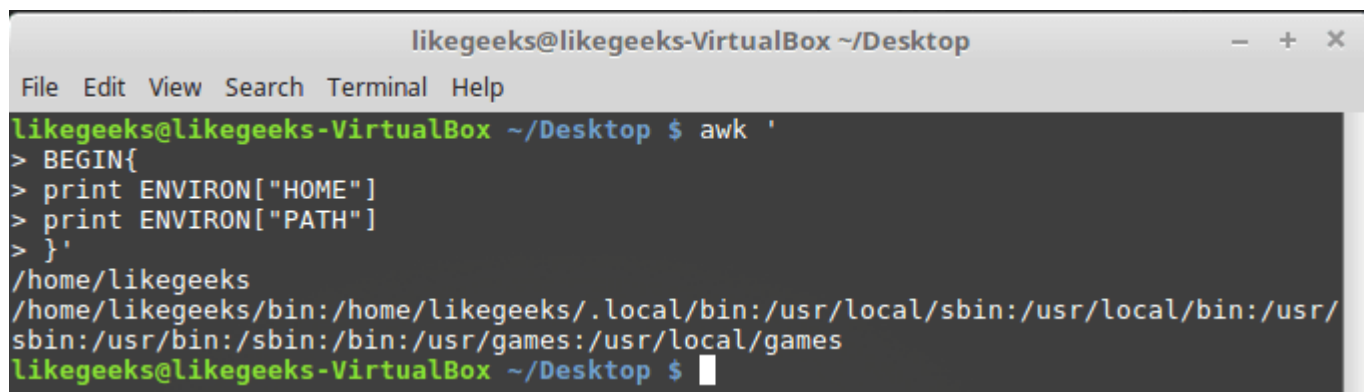
под индексом 1 в массиве ARGV записано имя обрабатываемого файла. В элементе массива с индексом 0 в данном случае будет «awk».

A terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'. The command 'awk 'BEGIN{print ARGV[1]}' myfile' is entered, followed by '2 myfile' on the next line. The prompt returns to 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'.

Работа с параметрами командной строки

Переменная ENVIRON представляет собой ассоциативный массив с переменными среды. Опробуем её:

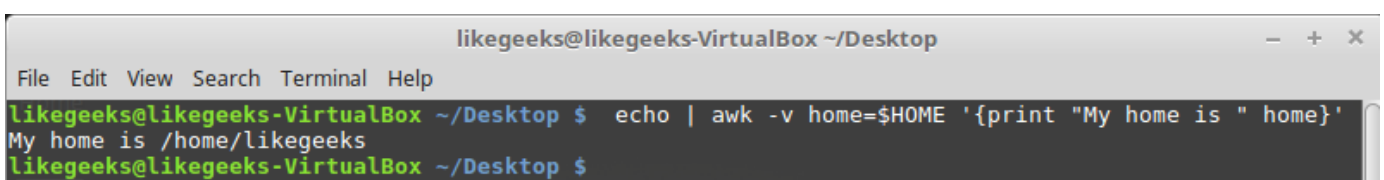
```
$ awk '  
BEGIN{  
  print ENVIRON["HOME"]  
  print ENVIRON["PATH"]  
}'
```

A terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'. The command 'awk 'BEGIN{> print ENVIRON["HOME"]> print ENVIRON["PATH"]> }'' is entered. The output shows the home directory and the PATH variable. The prompt returns to 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'.

Работа с переменными среды

Переменные среды можно использовать и без обращения к ENVIRON. Сделать это, например, можно так:

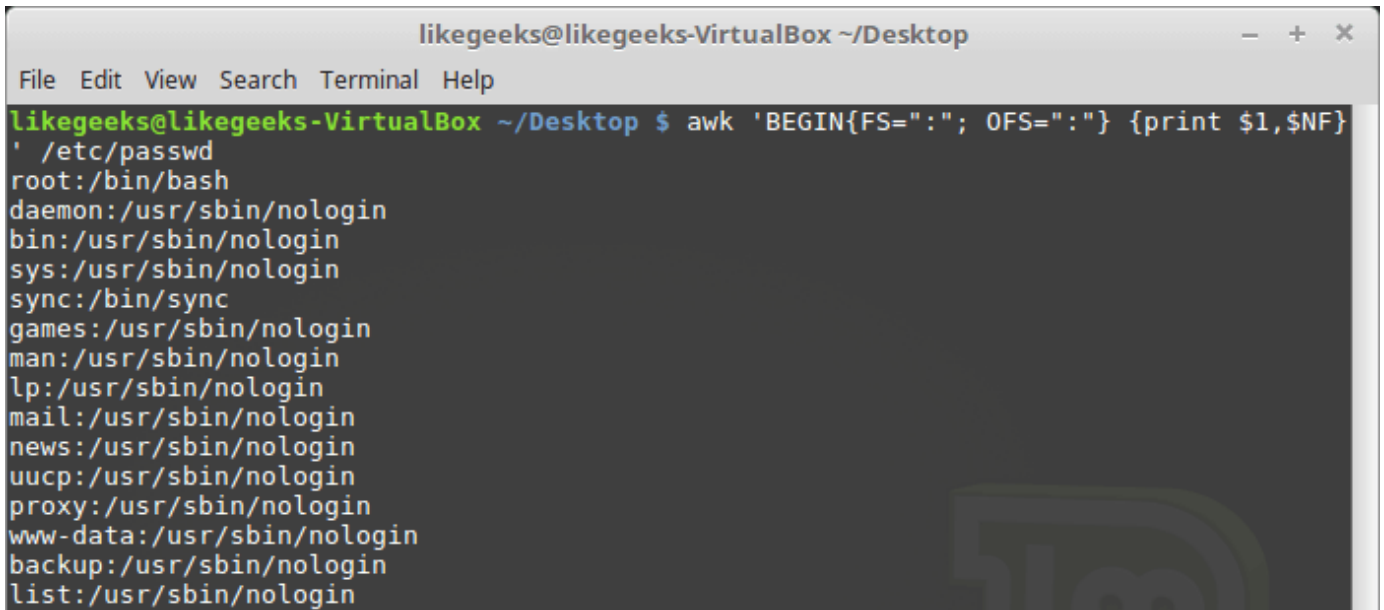
```
$ echo | awk -v home=$HOME '{print "My home is " home}'
```

A terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'. The command 'echo | awk -v home=\$HOME '{print "My home is " home}'' is entered. The output is 'My home is /home/likegeeks'. The prompt returns to 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'.

Работа с переменными среды без использования ENVIRON

Переменная `NF` позволяет обращаться к последнему полю данных в записи, не зная его точной позиции:

```
$ awk 'BEGIN{FS=":"; OFS=":"} {print $1,$NF}' /etc/passwd
```

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The terminal shows the command `awk 'BEGIN{FS=":"; OFS=":"} {print $1,$NF}' /etc/passwd` being executed. The output lists system users and their shells, with the last field of each line printed after a space. The output is: `root:/bin/bash`, `daemon:/usr/sbin/nologin`, `bin:/usr/sbin/nologin`, `sys:/usr/sbin/nologin`, `sync:/bin/sync`, `games:/usr/sbin/nologin`, `man:/usr/sbin/nologin`, `lp:/usr/sbin/nologin`, `mail:/usr/sbin/nologin`, `news:/usr/sbin/nologin`, `uucp:/usr/sbin/nologin`, `proxy:/usr/sbin/nologin`, `www-data:/usr/sbin/nologin`, `backup:/usr/sbin/nologin`, and `list:/usr/sbin/nologin`.

Пример использования переменной `NF`

Эта переменная содержит числовой индекс последнего поля данных в записи. Обратиться к данному полю можно, поместив перед `NF` знак `$`.

Переменные `FNR` и `NR`, хотя и могут показаться похожими, на самом деле различаются. Так, переменная `FNR` хранит число записей, обработанных в текущем файле. Переменная `NR` хранит общее число обработанных записей. Рассмотрим пару примеров, передав `awk` один и тот же файл дважды:

```
$ awk 'BEGIN{FS=","}{print $1,"FNR="FNR}' myfile myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{FS=","}{print $1,"FNR="FNR}' myfile
This is a test. FNR=1
This is the second test. FNR=2
This is the thrid test. FNR=3
This is the fourth test. FNR=4
This is a test. FNR=1
This is the second test. FNR=2
This is the thrid test. FNR=3
This is the fourth test. FNR=4
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Исследование переменной FNR

Передача одного и того же файла дважды равносильна передаче двух разных файлов. Обратите внимание на то, что FNR сбрасывается в начале обработки каждого файла.

Взглянем теперь на то, как ведёт себя в подобной ситуации переменная NR :

```
$ awk '
BEGIN {FS=","}
{print $1,"FNR="FNR,"NR="NR}
END{print "There were",NR,"records processed"}' myfile myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '
> BEGIN {FS=","}
> {print $1,"FNR="FNR,"NR="NR}
> END{print "There were",NR,"records processed"}' myfile myfile
This is a test. FNR=1 NR=1
This is the second test. FNR=2 NR=2
This is the thrid test. FNR=3 NR=3
This is the fourth test. FNR=4 NR=4
This is a test. FNR=1 NR=5
This is the second test. FNR=2 NR=6
This is the thrid test. FNR=3 NR=7
This is the fourth test. FNR=4 NR=8
There were 8 records processed
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

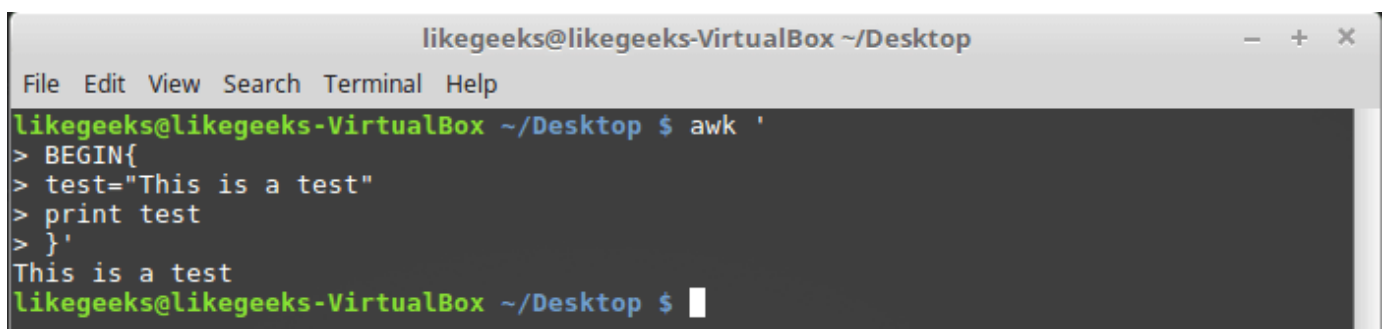
Различие переменных NR и FNR

Как видно, FNR , как и в предыдущем примере, сбрасывается в начале обработки каждого файла, а вот NR , при переходе к следующему файлу, сохраняет значение.

Пользовательские переменные

Как и любые другие языки программирования, awk позволяет программисту объявлять переменные. Имена переменных могут включать в себя буквы, цифры, символы подчёркивания. Однако, они не могут начинаться с цифры. Объявить переменную, присвоить ей значение и воспользоваться ей в коде можно так:

```
$ awk '  
BEGIN{  
test="This is a test"  
print test  
}'
```

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 command 'likegeeks@likegeeks-VirtualBox ~/Desktop \$ awk \'' followed by a multi-line script: '> BEGIN{', '> test="This is a test"', '> print test', and '> }''. The output 'This is a test' is displayed on the next line, followed by the prompt 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'.

Работа с пользовательской переменной

Условный оператор

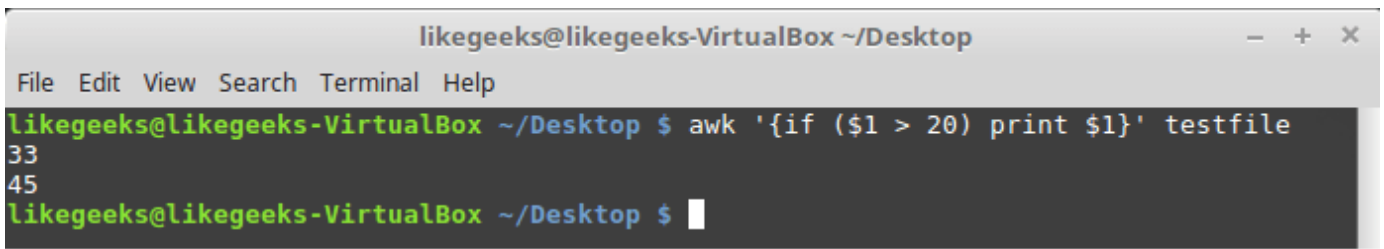
Awk поддерживает стандартный во многих языках программирования формат условного оператора `if-then-else`. Однострочный вариант оператора представляет собой ключевое слово `if`, за которым, в скобках, записывают проверяемое выражение, а затем – команду, которую нужно выполнить, если выражение истинно.

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
```

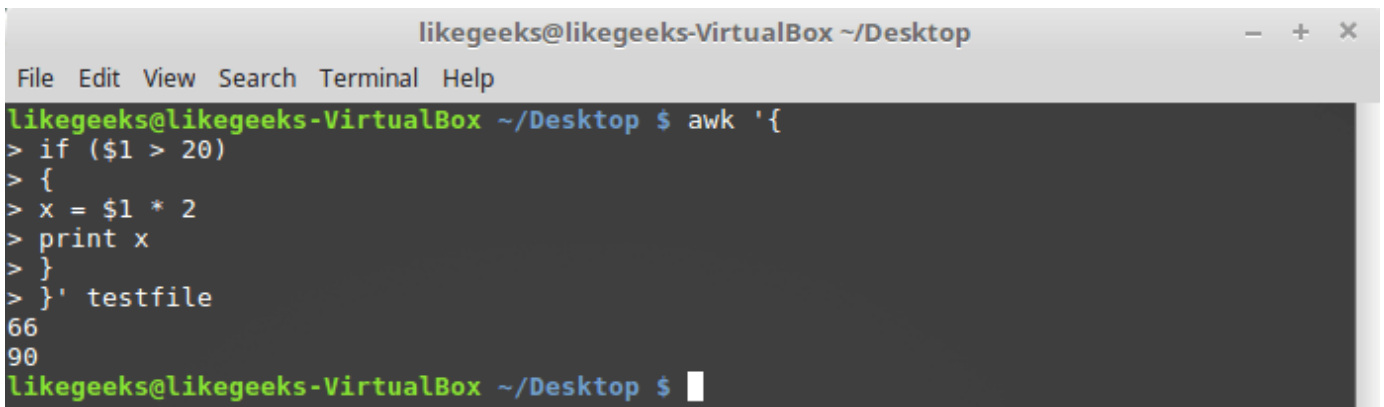


```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{if ($1 > 20) print $1}' testfile
33
45
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Однострочный оператор if

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

```
$ awk '{
if ($1 > 20)
{
x = $1 * 2
print x
}
}' testfile
```



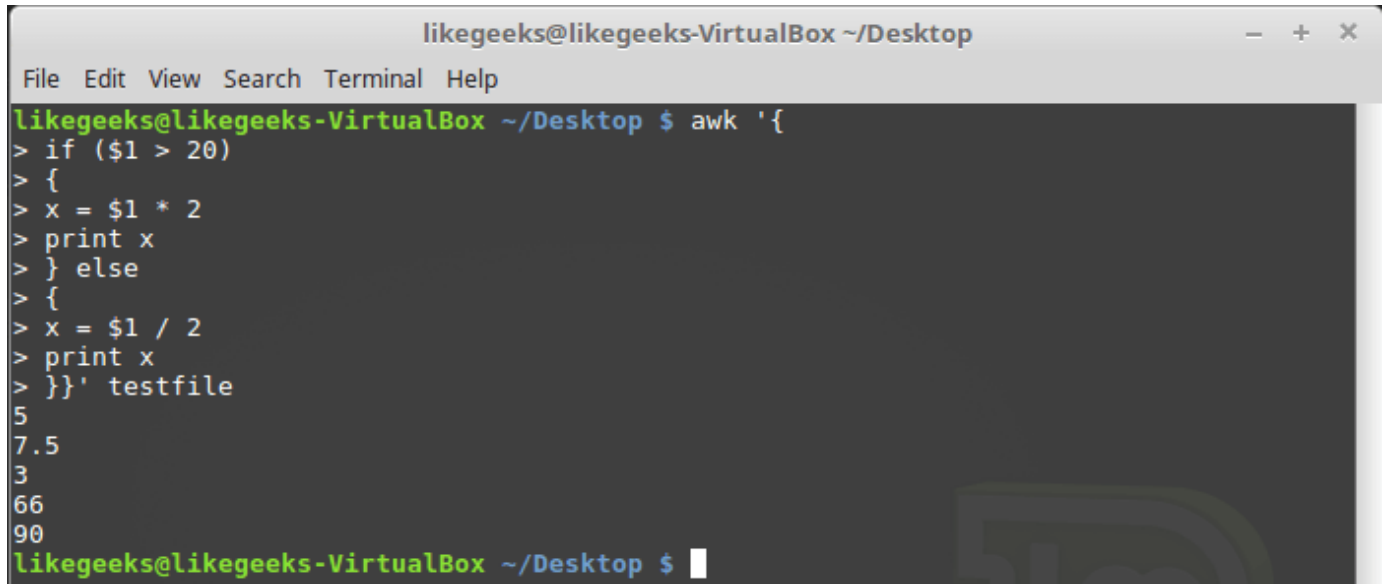
```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{
> if ($1 > 20)
> {
> x = $1 * 2
> print x
> }
> }' testfile
66
90
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Выполнение нескольких команд в блоке if

Как уже было сказано, условный оператор awk может содержать блок `else` :

```
$ awk '{
if ($1 > 20)
{
x = $1 * 2
print x
}
```

```
} else
{
x = $1 / 2
print x
}}' testfile
```

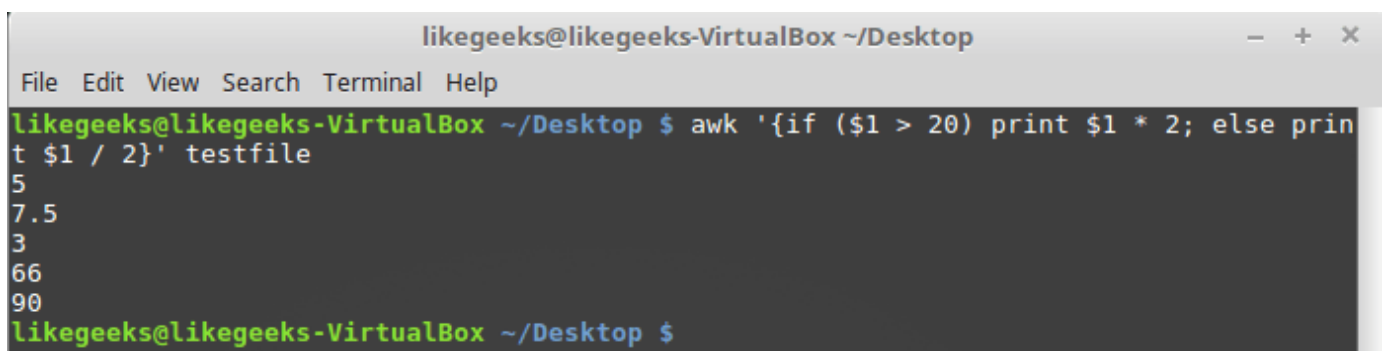


```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{
> if ($1 > 20)
> {
> x = $1 * 2
> print x
> } else
> {
> x = $1 / 2
> print x
> }}' testfile
5
7.5
3
66
90
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Условный оператор с блоком `else`

Ветвь `else` может быть частью однострочной записи условного оператора, включая в себя лишь одну строку с командой. В подобном случае после ветви `if`, сразу перед `else`, надо поставить точку с запятой:

```
$ awk '{if ($1 > 20) print $1 * 2; else print $1 / 2}' testfile
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{if ($1 > 20) print $1 * 2; else print $1 / 2}' testfile
5
7.5
3
66
90
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Условный оператор, содержащий ветви `if` и `else`, записанный в одну строку

Цикл `while`

Цикл `while` позволяет перебирать наборы данных, проверяя условие, которое остановит

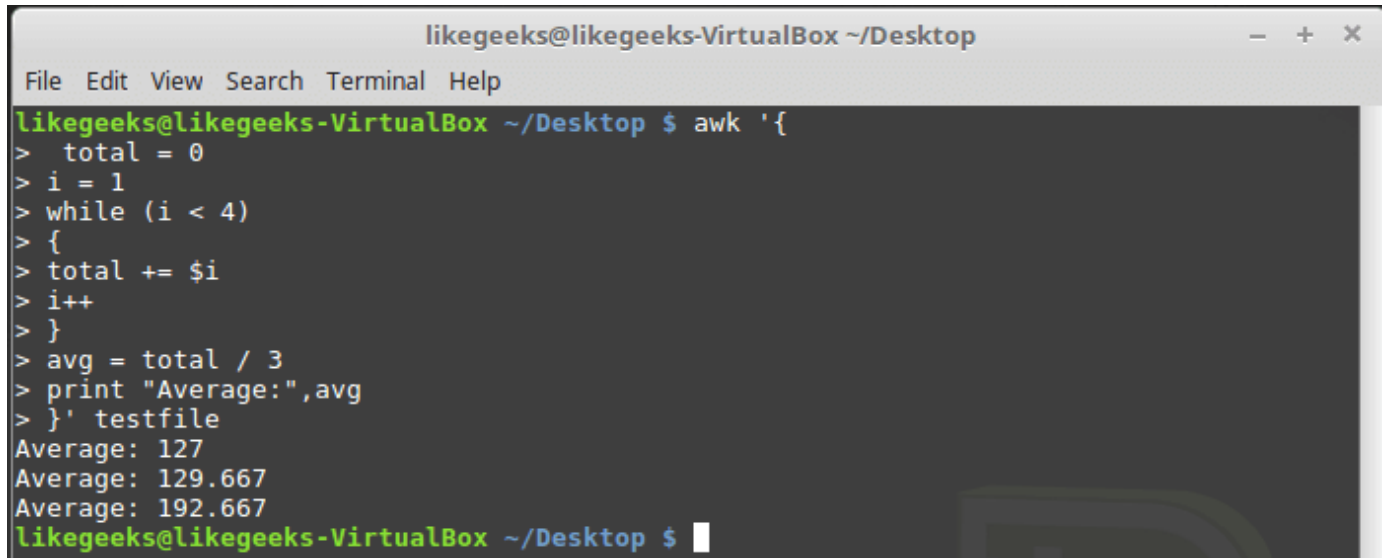
ЦИКЛ.

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
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{
> total = 0
> i = 1
> while (i < 4)
> {
> total += $i
> i++
> }
> avg = total / 3
> print "Average:",avg
> }' testfile
Average: 127
Average: 129.667
Average: 192.667
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Обработка данных в цикле `while`

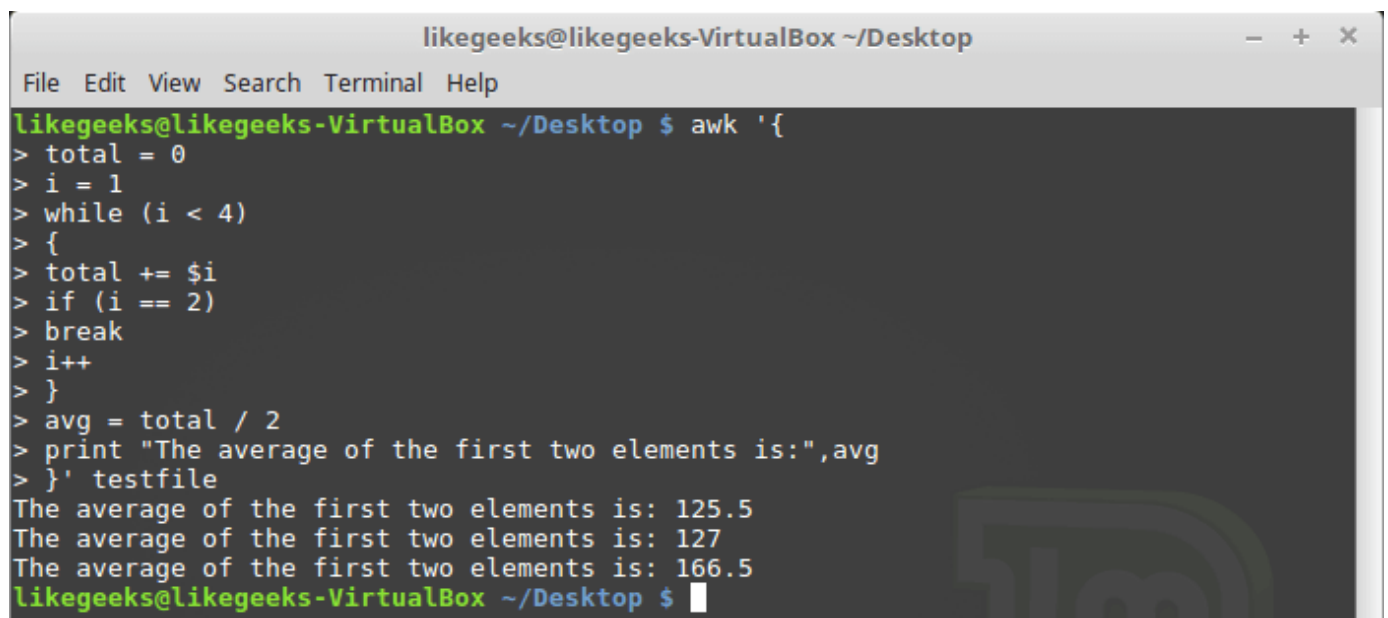
Цикл `while` перебирает поля каждой записи, накапливая их сумму в переменной `total` и увеличивая в каждой итерации на 1 переменную-счётчик `i`. Когда `i` достигнет 4, условие на входе в цикл окажется ложным и цикл завершится, после чего будут выполнены остальные команды – подсчёт среднего значения для числовых полей текущей записи и

вывод найденного значения.

В циклах `while` можно использовать команды `break` и `continue`. Первая позволяет досрочно завершить цикл и приступить к выполнению команд, расположенных после него. Вторая позволяет, не завершая до конца текущую итерацию, перейти к следующей.

Вот как работает команда `break`:

```
$ awk '{
total = 0
i = 1
while (i < 4)
{
total += $i
if (i == 2)
break
i++
}
avg = total / 2
print "The average of the first two elements is:",avg
}' testfile
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{
> total = 0
> i = 1
> while (i < 4)
> {
> total += $i
> if (i == 2)
> break
> i++
> }
> avg = total / 2
> print "The average of the first two elements is:",avg
> }' testfile
The average of the first two elements is: 125.5
The average of the first two elements is: 127
The average of the first two elements is: 166.5
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

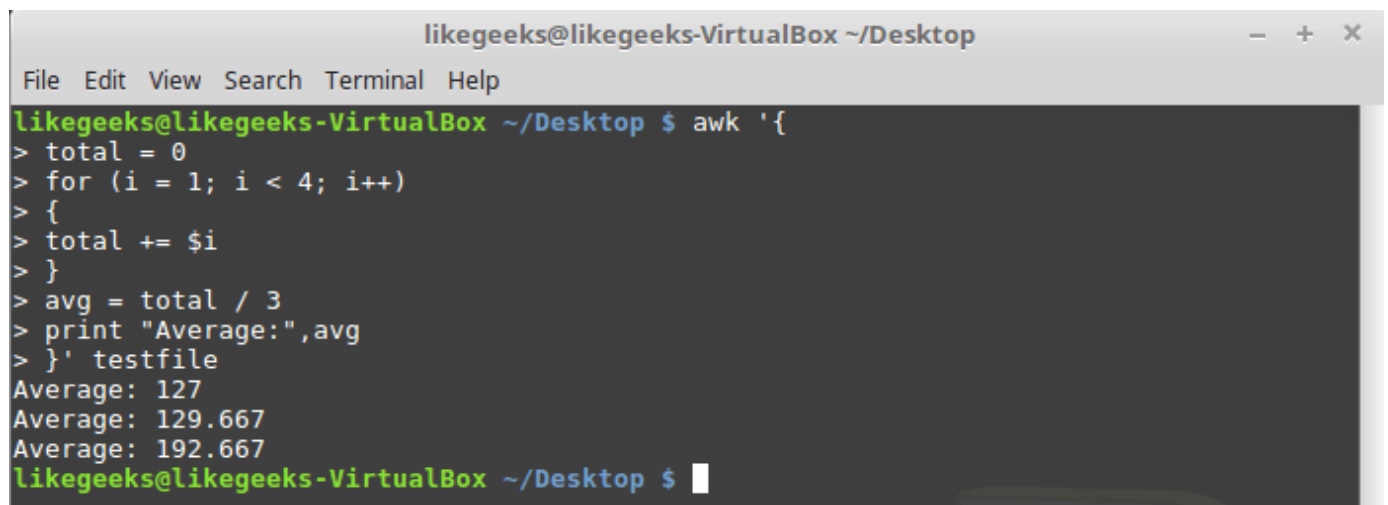
Команда `break` в цикле `while`

Цикл `for`

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
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{
> total = 0
> for (i = 1; i < 4; i++)
> {
> total += $i
> }
> avg = total / 3
> print "Average:",avg
> }' testfile
Average: 127
Average: 129.667
Average: 192.667
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Цикл *for*

Начальное значение переменной-счётчика и правило её изменения в каждой итерации, а также условие прекращения цикла, задаются в начале цикла, в круглых скобках. В итоге нам не нужно, в отличие от случая с циклом `while`, самостоятельно инкрементировать счётчик.

Форматированный вывод данных

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:

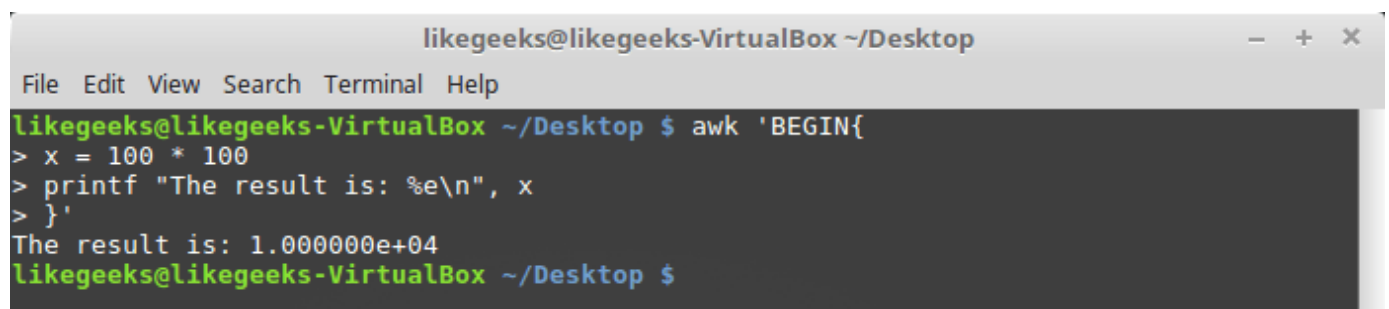
```
%[modifier]control-letter
```

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
}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
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 $
```

Форматирование выходных данных с помощью `printf`

Тут, в качестве примера, мы выводим число в экспоненциальной записи. Полагаем, этого достаточно для того, чтобы вы поняли основную идею, на которой построена работа с

```
printf .
```

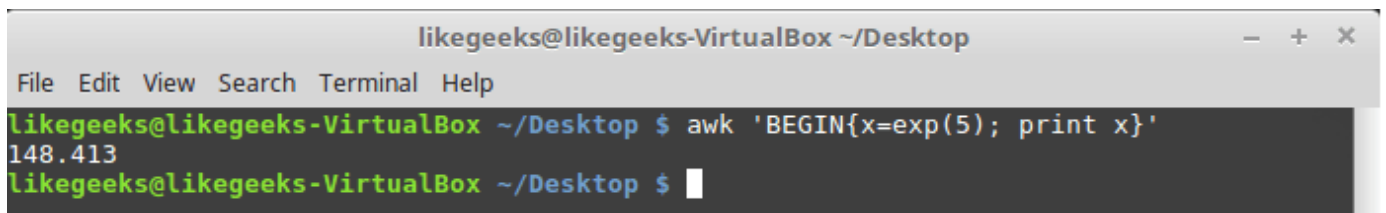
Встроенные математические функции

При работе с awk программисту доступны встроенные функции. В частности, это математические и строковые функции, функции для работы со временем. Вот, например, список математических функций, которыми можно пользоваться при разработке awk-скриптов:

```
cos(x) – косинус x ( x выражено в радианах).  
sin(x) – синус x .  
exp(x) – экспоненциальная функция.  
int(x) – возвращает целую часть аргумента.  
log(x) – натуральный логарифм.  
rand() – возвращает случайное число с плавающей запятой в диапазоне 0 – 1.  
sqrt(x) – квадратный корень из x .
```

Вот как пользоваться этими функциями:

```
$ awk 'BEGIN{x=exp(5); print 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 command 'likegeeks@likegeeks-VirtualBox ~/Desktop \$ awk 'BEGIN{x=exp(5); print x}'' being executed, which outputs '148.413'. The prompt 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' is visible again on the next line.

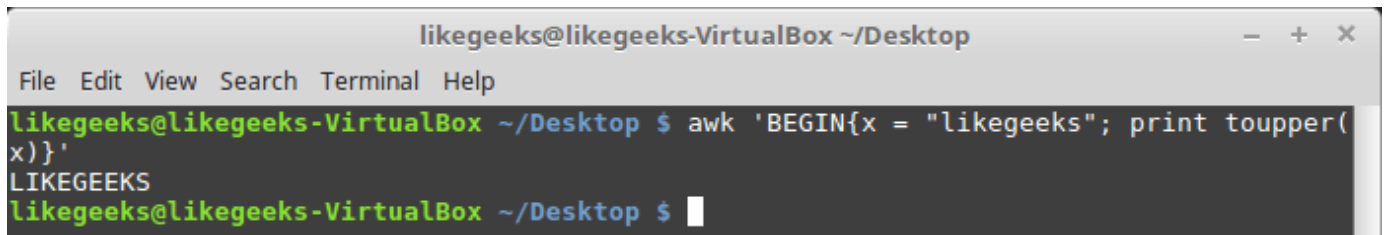
```
likegeeks@likegeeks-VirtualBox ~/Desktop  
File Edit View Search Terminal Help  
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{x=exp(5); print x}''  
148.413  
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Работа с математическими функциями

Строковые функции

Awk поддерживает множество строковых функций. Все они устроены более или менее одинаково. Вот, например, функция `toupper` :

```
$ awk 'BEGIN{x = "likegeeks"; print toupper(x)}'
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{x = "likegeeks"; print toupper(x)}'
LIKEGEEKS
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Использование строковой функции toupper

Эта функция преобразует символы, хранящиеся в переданной ей строковой переменной, к верхнему регистру.

Пользовательские функции

При необходимости вы можете создавать собственные функции awk. Такие функции можно использовать так же, как встроенные:

```
$ awk '
function myprint()
{
printf "The user %s has home path at %s\n", $1,$6
}
BEGIN{FS=":"}
{
myprint()
}' /etc/passwd
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '
> function myprint()
> {
> printf "The user %s has home path at %s\n", $1,$6
> }
> BEGIN{FS=":"}
> {
> myprint()
> }' /etc/passwd
The user root has home path at /root
The user daemon has home path at /usr/sbin
The user bin has home path at /bin
The user sys has home path at /dev
The user sync has home path at /bin
The user games has home path at /usr/games
The user man has home path at /var/cache/man
The user lp has home path at /var/spool/lpd
The user mail has home path at /var/mail
The user news has home path at /var/spool/news
The user uucp has home path at /var/spool/uucp
The user proxy has home path at /bin
The user www-data has home path at /var/www
The user backup has home path at /var/backups
The user list has home path at /var/list
```

Использование собственной функции

В примере используется заданная нами функция `myprint`, которая выводит данные.

Итоги

Сегодня мы разобрали основы awk. Это мощнейший инструмент обработки данных, масштабы которого сопоставимы с отдельным языком программирования.

Вы не могли не заметить, что многое из того, о чём мы говорим, не так уж и сложно для понимания, а зная основы, уже можно что-то автоматизировать, но если копнуть поглубже, вникнуть в документацию... Вот, например, *The GNU Awk User's Guide*. В этом руководстве впечатляет уже одно то, что оно ведёт свою историю с 1989-го (первая версия awk, кстати, появилась в 1977-м). Однако, сейчас вы знаете об awk достаточно для того, чтобы не потеряться в официальной документации и познакомиться с ним настолько близко, насколько вам того хочется. В следующий раз, кстати, мы поговорим о регулярных выражениях. Без них невозможно заниматься серьёзной обработкой текстов в bash-скриптах с применением `sed` и `awk`.

Habrahabr10

Промо-код для скидки в 10% на наши виртуальные сервера

Уважаемые читатели! Уверены, многие из вас периодически пользуются awk. Расскажите, как он помогает вам в работе?

Теги: RuVDS, Linux, Bash, сценарий командной строки, Bash-скрипт, awk, обработка текстов

Хабы: Блог компании RUVDS.com, Настройка Linux, Системное администрирование

Редакторский дайджест



Присылаем лучшие статьи раз в месяц

Электронпочта



RUVDS.com
VDS/VPS-хостинг. Скидка 15% по коду **HABR15**

Telegram ВКонтакте Twitter



450 430.2
Карма Рейтинг

@ru_vds
Пользователь

Комментарии 18

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ



Erwinma1
19 часов назад

Советские игровые автоматы: ностальгический гейминг с запахом резины. Часть 1: как родился «Морской бой»?

Простой 7 мин 4.2K

Ретроспектива

+33

14

20

**dionisdimetor**

19 часов назад

Правда и мифы об энтропии. Как работает второй закон термодинамики?



Средний



22 мин



6.4K



+29



41



43

**GeeksCat**

21 час назад

Sid Meier's Civilization III от Firaxis – история создания



Простой



10 мин



3.8K

Ретроспектива



+27



20



2

**RyabovA**

10 часов назад

У меня зазвонил Репка-фон. Делаем телефон на базе Репка Pi 3 с применением модуля SIM868



Простой



15 мин



1.9K

Тutorial



+19



24



0

**Verz1Lka**

22 часа назад

Анализ системы защиты от ботов на примере letu.ru



Средний



40 мин



2.4K

Из песочницы



+19



33



11

**OlegSivchenko**

12 часов назад

Следуй за человеком. Об ориентировании роботов в толпе



11 мин



817

 +12 10 0

zoldaten

12 часов назад

Как перевести игру 'The Lamplighters League'

 7 мин 703

Тutorial

 +11 6 2

PatientZero

19 часов назад

Почему текст в нижнем регистре сжимается лучше

 Простой 7 мин 4.3K

Обзор

Перевод

 +10 34 13

ibessonov

8 часов назад

CompletableFuture. Глубокое погружение

 Сложный 20 мин 2K +9 22 0

dbalabolin

12 часов назад

Опасен ли разогрев и приготовление еды в микроволновке. Обзор мнений. Мнение автора

 Простой 9 мин 6.8K +8 14 38

Показать еще

Ваш аккаунт	Разделы	Информация	Услуги
Войти	Статьи	Устройство сайта	Корпоративный блог
Регистрация	Новости	Для авторов	Медийная реклама
	Хабы	Для компаний	Нативные проекты
	Компании	Документы	Образовательные
	Авторы	Соглашение	программы
	Песочница	Конфиденциальность	Стартапам



Настройка языка

Техническая поддержка

© 2006–2024, Habr

Партнерская программа RUVDS
ruvds.com

VPS (CPU 1x2ГГц, RAM 512Mb, SSD 10 Gb) – 190 рублей в месяц
ruvds.com

VPS Windows от 523 рублей в месяц. Бесплатный тестовый период 3 дня.
ruvds.com

VDS в Цюрихе. Дата-центр TIER III – швейцарское качество по низкой цене.
ruvds.com

Антивирусная защита виртуального сервера. Легкий агент для VPS.
ruvds.com

VPS в Лондоне. Дата-центр TIER III – английская точность за рубли.
ruvds.com

VPS с видеокартой на мощных серверах 3,4ГГц
ruvds.com

ПРИЛОЖЕНИЯ

**RUVDS Client**

Приложение для мониторинга и управления виртуальными серверами RUVDS с мобильных устройств.

Android iOS

ВИДЖЕТ

The advertisement banner features a blue background with a white cloud logo containing the letters 'RU' and the text 'VDS' to its right. Below this, the text 'Лучшее предложение на рынке' is written in white. The central part of the banner is an isometric illustration of several overlapping, semi-transparent rectangular panels in various colors (blue, purple, pink, red, green, yellow, brown). These panels display different icons and UI elements, such as a person icon, a gear, and a document. At the bottom, the text 'VPS с WINDOWS' is written in large white letters, followed by 'от 523 ₽/месяц' in even larger white letters. Below that, in smaller white text, it says 'Лицензия на ОС включена в стоимость'. At the very bottom, in small white text, it reads 'Реклама. ООО «МТ ФИНАНС». erid: LjN8KDmrk'.

RU VDS

Лучшее предложение
на рынке

VPS с WINDOWS
от 523 ₽/месяц

Лицензия на ОС включена в стоимость

Реклама. ООО «МТ ФИНАНС». erid: LjN8KDmrk

БЛОГ НА ХАБРЕ

19 часов назад

Советские игровые автоматы: ностальгический гейминг с запахом резины. Часть 1: как родился «Морской бой»?

 4.2K  20

19 янв в 20:00

Арифметика первого класса в системе типов Rust

 3.6K  5

19 янв в 16:00

Ручная работа со стеклом. Резка трубчатых заготовок, бутылок

 3.4K  28

18 янв в 20:00

Гроза бэклогов и пристанище необычных игр: как я полюбил Switch

 5.9K  25

18 янв в 16:00

Поделка в стиле 2000-х – блютуз-колонка с пультом, MP3, ЧМ-радио и эффектами

 6.6K  7