Bash-скрипты, часть 8: язык обработки данных awk

<https://likegeeks.com/awk-command/>

* [Блог компании RUVDS.com](https://habr.com/company/ruvds/),
* [Настройка Linux](https://habr.com/hub/linux/),
* [Системное администрирование](https://habr.com/hub/sys_admin/)
* Перевод

[Bash-скрипты: начало](https://habrahabr.ru/company/ruvds/blog/325522/)  
[Bash-скрипты, часть 2: циклы](https://habrahabr.ru/company/ruvds/blog/325928/)  
[Bash-скрипты, часть 3: параметры и ключи командной строки](https://habrahabr.ru/company/ruvds/blog/326328/)  
[Bash-скрипты, часть 4: ввод и вывод](https://habrahabr.ru/company/ruvds/blog/326594/)  
[Bash-скрипты, часть 5: сигналы, фоновые задачи, управление сценариями](https://habrahabr.ru/company/ruvds/blog/326826/)  
[Bash-скрипты, часть 6: функции и разработка библиотек](https://habrahabr.ru/company/ruvds/blog/327248/)  
[Bash-скрипты, часть 7: sed и обработка текстов](https://habrahabr.ru/company/ruvds/blog/327530/)  
[Bash-скрипты, часть 8: язык обработки данных awk](https://habrahabr.ru/company/ruvds/blog/327754/)  
[Bash-скрипты, часть 9: регулярные выражения](https://habrahabr.ru/company/ruvds/blog/327896/)  
[Bash-скрипты, часть 10: практические примеры](https://habrahabr.ru/company/ruvds/blog/328346/)  
[Bash-скрипты, часть 11: expect и автоматизация интерактивных утилит](https://habrahabr.ru/company/ruvds/blog/328436/)

[](https://habrahabr.ru/company/ruvds/blog/327754/)  
  
В прошлый раз мы говорили о потоковом редакторе [sed](https://habrahabr.ru/company/ruvds/blog/327530/) 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.  
  
[https://habrastorage.org/files/1ba/550/d25/1ba550d25e8846ce8805de564da6aa63.png](https://ruvds.com/ru-rub/#order)  
  
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.

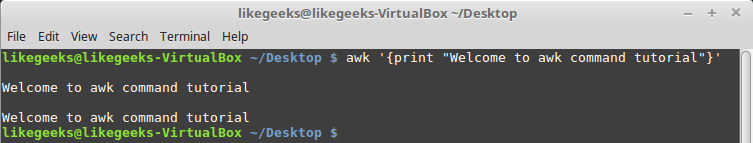
Настоящая мощь awk скрывается в той части команды его вызова, которая помечена выше как program. Она указывает на файл awk-скрипта, написанный программистом и предназначенный для чтения данных, их обработки и вывода результатов.

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](https://habrahabr.ru/company/ruvds/blog/326594/). 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 введённые 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.



*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.

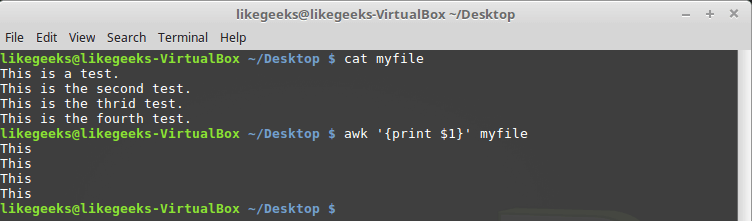
Позиционные переменные, хранящие данные полей

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.

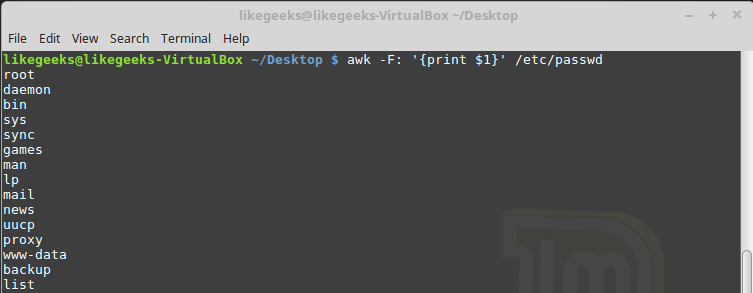
Поля выделяются из текста с использованием символа-разделителя. По умолчанию — это пробельные символы вроде пробела или символа табуляции.  
  
Рассмотрим использование этих переменных на простом примере. А именно, обработаем файл, в котором содержится несколько строк (этот файл показан на рисунке ниже) с помощью такой команды:

$ awk '{print $1}' myfile



*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

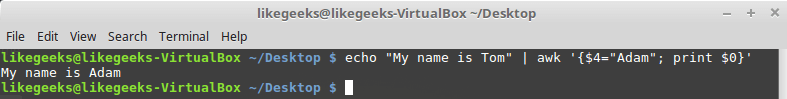


*Указание символа-разделителя при вызове awk*  
  
Эта команда выводит первые элементы строк, содержащихся в файле /etc/passwd. Так как в этом файле в качестве разделителей используются двоеточия, именно этот символ был передан awk после ключа -F.

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}'



*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.

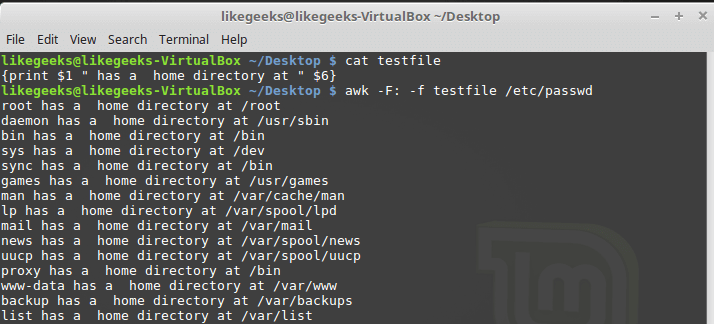
Чтение скрипта awk из файла

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



*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

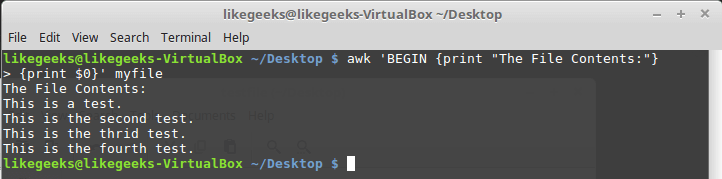
Иногда нужно выполнить какие-то действия до того, как скрипт начнёт обработку записей из входного потока. Например — создать шапку отчёта или что-то подобное.  
  
Для этого можно воспользоваться ключевым словом BEGIN. Команды, которые следуют за BEGIN, будут исполнены до начала обработки данных. В простейшем виде это выглядит так:

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

А вот — немного более сложный пример:

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

{print $0}' myfile



*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.

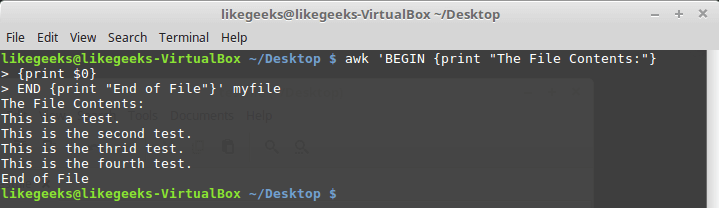
Выполнение команд после окончания обработки данных

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



*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

}

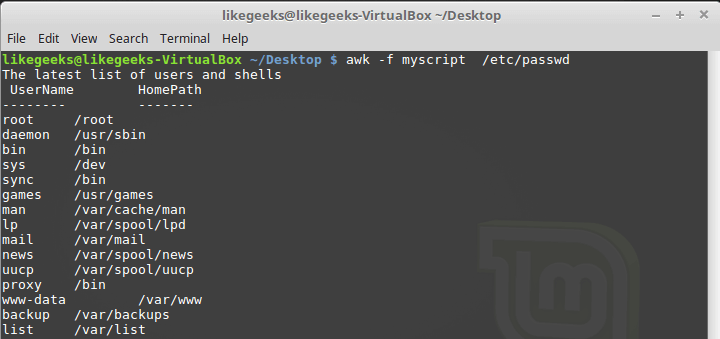
END {

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



*Обработка файла /etc/passwd с помощью awk-скрипта*  
  
Всё, о чём мы говорили выше — лишь малая часть возможностей awk. Продолжим освоение этого полезного инструмента.

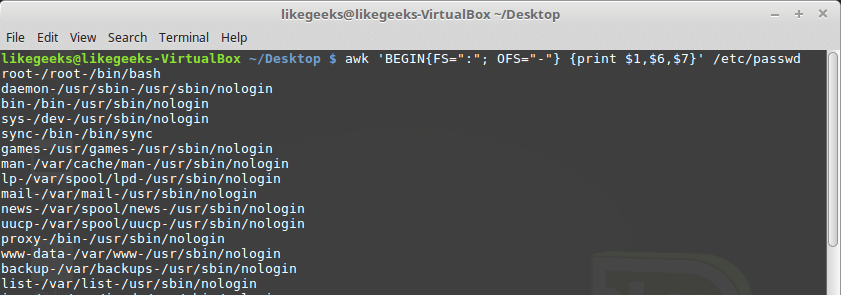
Встроенные переменные: настройка процесса обработки данных

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



*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:

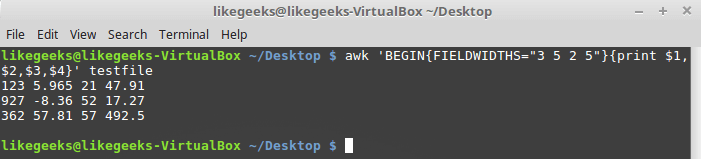
1235.9652147.91

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



*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

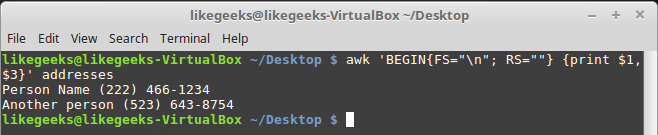
Another person

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



*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

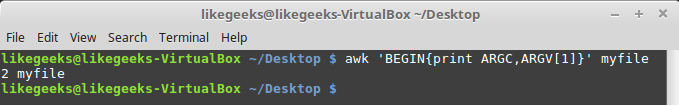
Помимо встроенных переменных, о которых мы уже говорили, существуют и другие, которые предоставляют сведения о данных и об окружении, в котором работает awk:

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».



*Работа с параметрами командной строки*  
  
Переменная ENVIRON представляет собой ассоциативный массив с переменными среды. Опробуем её:

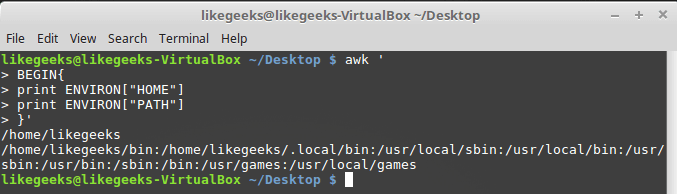
$ awk '

BEGIN{

print ENVIRON["HOME"]

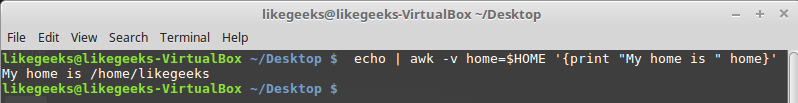
print ENVIRON["PATH"]

}'



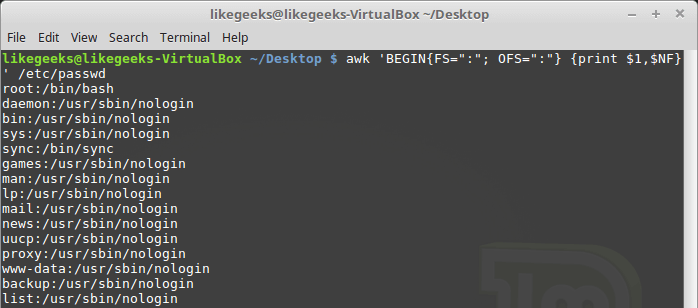
*Работа с переменными среды*  
  
Переменные среды можно использовать и без обращения к ENVIRON. Сделать это, например, можно так:

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



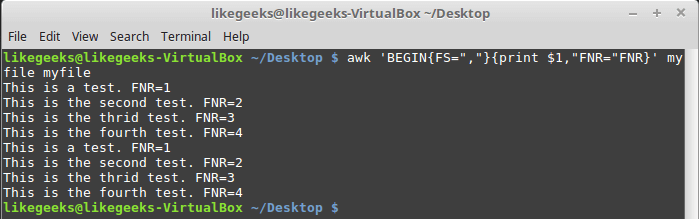
*Работа с переменными среды без использования ENVIRON*  
  
Переменная NF позволяет обращаться к последнему полю данных в записи, не зная его точной позиции:

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



*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



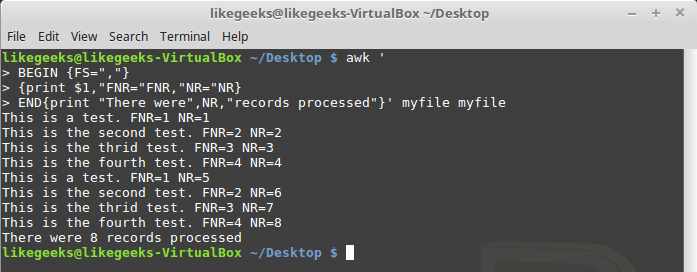
*Исследование переменной FNR*  
  
Передача одного и того же файла дважды равносильна передаче двух разных файлов. Обратите внимание на то, что FNR сбрасывается в начале обработки каждого файла.  
  
Взглянем теперь на то, как ведёт себя в подобной ситуации переменная NR:

$ awk '

BEGIN {FS=","}

{print $1,"FNR="FNR,"NR="NR}

END{print "There were",NR,"records processed"}' myfile myfile



*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:

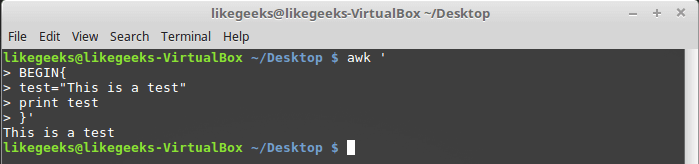
$ awk '

BEGIN{

test="This is a test"

print test

}'



*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

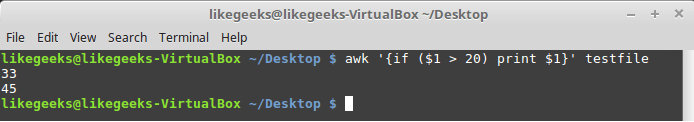
6

33

45

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

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



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

$ awk '{

if ($1 > 20)

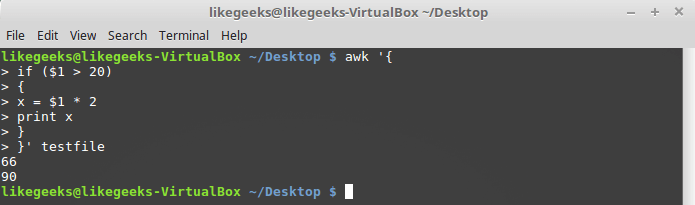
{

x = $1 \* 2

print x

}

}' testfile



*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

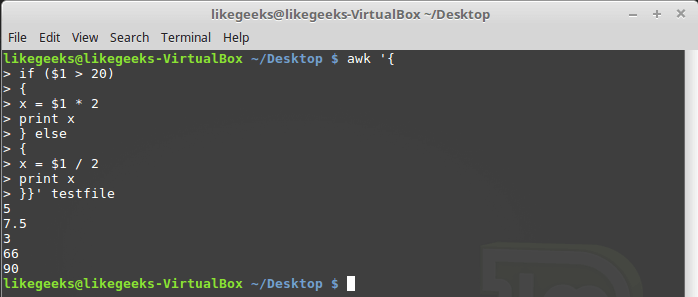
} else

{

x = $1 / 2

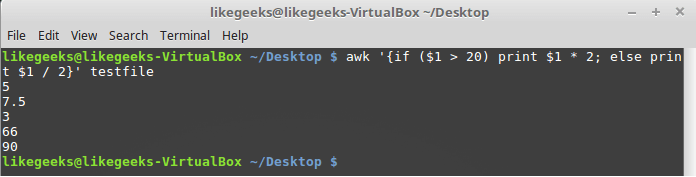
print x

}}' testfile



*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



*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

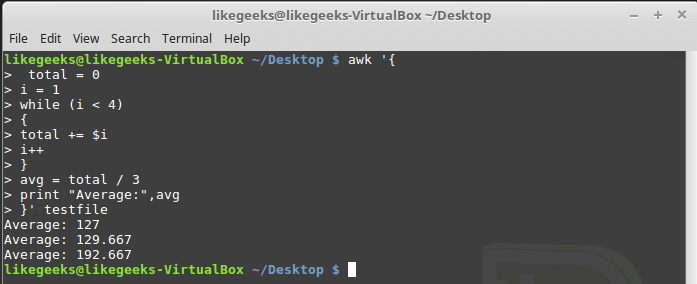
i++

}

avg = total / 3

print "Average:",avg

}' testfile



*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

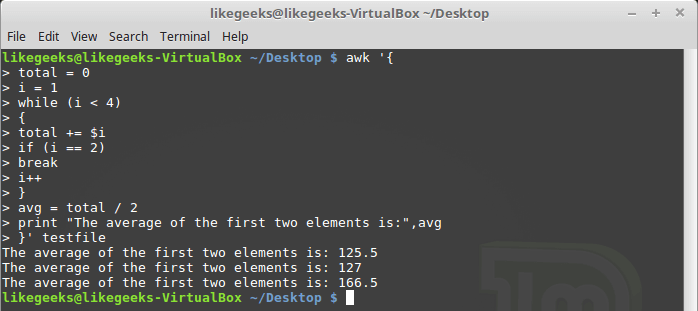
i++

}

avg = total / 2

print "The average of the first two elements is:",avg

}' testfile



*Команда break в цикле while*

Цикл for

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

$ awk '{

total = 0

for (i = 1; i < 4; i++)

{

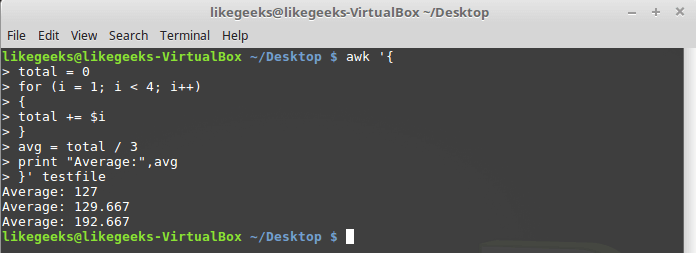
total += $i

}

avg = total / 3

print "Average:",avg

}' testfile



*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.

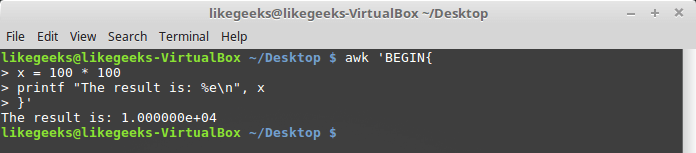
Вот как форматировать выводимые данные с помощью printf:

$ awk 'BEGIN{

x = 100 \* 100

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

}'



*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.

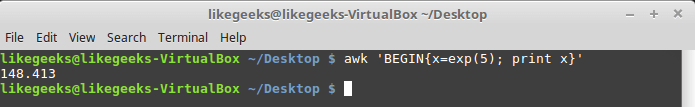
Встроенные математические функции

При работе с awk программисту доступны [встроенные функции](https://www.gnu.org/software/gawk/manual/html_node/Built_002din.html#Built_002din). 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}'

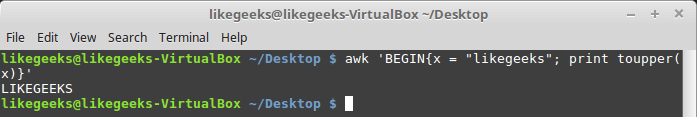


*Работа с математическими функциями*

Строковые функции

Awk поддерживает множество [строковых функций](https://www.gnu.org/software/gawk/manual/html_node/String-Functions.html#String-Functions). Все они устроены более или менее одинаково. Вот, например, функция toupper:

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



*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

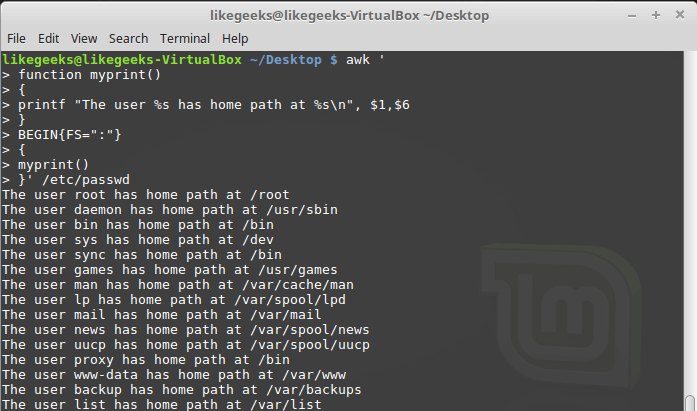
}

BEGIN{FS=":"}

{

myprint()

}' /etc/passwd



*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](https://www.gnu.org/software/gawk/manual/html_node/index.html). 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.  
  
[https://habrastorage.org/files/1ba/550/d25/1ba550d25e8846ce8805de564da6aa63.png](https://ruvds.com/ru-rub/#order)