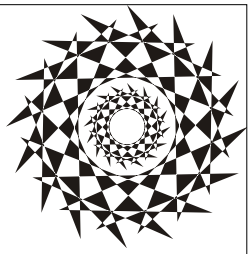

Автоматизация обработки текста средствами gawk

1. Введение в gawk

gawk (GNU awk) — небольшой язык программирования, ориентированный на обработку текстовых файлов. Язык awk (далее будем называть этот язык просто awk) удобно использовать для обработки дампов баз данных, системных файлов. Он подобен языку Perl, но проще в использовании. По сути, awk является прародителем Perl. Как и Perl, awk построен на регулярных выражениях и средствах для обработки шаблонов.

Язык awk назван по первым буквам фамилий его создателей: Alfred V. Aho, Peter J. Weinberger и Brian W. Kernighan.

Основное назначение языка, как уже было сказано, — обработка текстовых файлов, поэтому awk пригодится администраторам для анализа системных журналов и создания различных систем статистики.

2. Основы языка

2.1. Образцы и действия

Сценарий awk пишется, как и сценарий для оболочек `bash` и `tcsh`: это обычный текстовый файл, в начале которого задается интерпретатор:

```
#!/usr/bin/awk -f
```

Сценарий на языке awk состоит из одной или нескольких строк вида:

```
образец { действие }
```

Образец — это строка (или регулярное выражение), которая должна быть в обрабатываемом сценарием текстовом файле. Действие будет применено к строке, соответствующей образцу. Если образец — это регулярное выражение, то он заключается в слэши:

```
/выражение/
```

Если образец не указан, то действие будет применено ко всем строкам обрабатываемого файла.

Особого внимания заслуживают образцы `BEGIN` и `END`. Первый образец запускает команды до обработки файла, а второй — после обработки файла. Например:

```
BEGIN {
    print "Мой сценарий\n"
    counter=0
}
{
    print "Основная программа\n"
}
END {
    print "Сценарий завершен"
    printf ("\nСчетчик:\t%d\n", counter)
}
```

2.2. Операторы

Основные операторы языка awk представлены в табл. 1.

Таблица 1. *Основные операторы языка awk*

Оператор	Описание
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно
==	Равно
!=	Не равно
!~	Отрицание. Используется при задании образцов. Означает, что поле или переменная не соответствует регулярному выражению
~	Поле или переменная соответствует регулярному выражению
||	Логическое ИЛИ (OR)
&&	Логическое И (AND)
*	Умножение
/	Деление
%	Получение остатка от деления
+	Сложение
-	Вычитание

=	Присваивает переменной значение или результат выражения
++	Увеличивает значение переменной
--	Уменьшает значение переменной

2.3. Переменные

Строковые переменные инициализируются пустой строкой, а числовые переменные — нулем:

```
counter=0
name=""
```

Служебные переменные языка awk приведены в табл. 2.

Таблица 2. *Служебные переменные языка awk*

Переменная	**Описание**
$0	Текущая запись (в одной переменной)
$n, n — число	Поле с номером *n* в текущей записи
FILENAME	Имя текущего файла, null для стандартного ввода
FS	Разделитель полей
NF	Количество полей в записи
NR	Номер записи
OFS	Разделитель полей при выводе (по умолчанию пробел)
ORS	Разделитель записей при выводе (по умолчанию символ новой строки)
RS	Разделитель записей при вводе (по умолчанию символ новой строки)

2.4. Ассоциативные массивы

В отличие от других языков программирования, awk поддерживает ассоциативные массивы (правда, такие массивы поддерживает еще PHP и некоторые другие языки, но во многих языках таких массивов нет). В ассоциативном массиве в качестве индекса выступает не число, а строка.

Пример объявления такого массива:

```
массив[строка] = значение
```

Затем для перебора массива используется вот такая конструкция:

```
for (элемент in массив) действие
```

Например:

```
for (item in count) printf "%-20s%-20s\n"
```

2.5. Функции

В табл. 3 приведены функции, которые язык awk предоставляет для обработки чисел и строк.

Таблица 3. *Функции языка awk*

Функция	Описание
`length(str)`	Возвращает длину строки (количество символов в `str`). Если строка не указана, возвращает количество символов в текущей записи
`int(num)`	Возвращает целую часть числа `num`
`index(str1,str2)`	Возвращает индекс строки `str2` в строке `str1` или 0, если строки `str2` нет в строке `str1`
`split(str,arr,del)`	Разбивает строку `str` на элементы массива `arr`, параметр `del` — это разделитель, который используется для разделения строки
`sprintf(fmt,args)`	Форматирует аргументы (параметр `args`) в соответствии со строкой формата `fmt`
`substr(str,pos,len)`	Возвращает подстроку строки `str` длины `len`, которая начинается на позиции `pos`
`tolower(str)`	Возвращает копию строки `str`, где все символы будут в нижнем регистре
`toupper(str)`	Возвращает копию строки `str`, где все символы будут в верхнем регистре

2.6. Вывод с помощью *printf*

Команда `printf` используется для вывода аргументов в соответствии с заданным форматом:

```
printf "строка формата", аргумент1, …, аргументN
```

Строка формата определяет, как будут отображены аргументы. В строке формата вы можете использовать `\t` для вставки символа табуляции и `\n` для новой строки на экране.

Для каждого аргумента в строке формата должен быть свой модификатор вывода. Модификатор имеет формат:

```
%[-][x[.y]]conv
```

Символ `%` обязателен, и он означает, что перед нами — модификатор. Символ "-" означает, что вывод должен быть отформатирован по левому краю. `x` — минимальная ширина поля, а `y` — количество цифр после запятой (при выводе числа с дробной частью), а `conv` задает формат аргумента (табл. 4).

Таблица 4. *Формат аргумента*

conv	Описание
d	Десятичное число
e	Экспоненциальная запись числа

Таблица 4 *(окончание)*

conv	Описание
f	Число с плавающей запятой
o	Беззначное восьмеричное число
s	Строка
x	Беззначное шестнадцатеричное число

2.7. Управляющие структуры

Условный оператор *if..else*

Условный оператор работает так же, как и в любом другом языке программирования:

```
if (условие)
          {команды1}
      [else
          {команды2}]
```

Если условие истинно (значение выражения — true), то будут выполнены `команды1`, в противном случае будут выполнены `команды2`. Вот пример условного оператора:

```
if (num == "one")
              print "Число: 1"
           else
              print "Другое число:", num
        }
```

Цикл *while*

Цикл `while` выполняется до тех пор, пока условие истинно. Синтаксис цикла следующий:

```
while (условие)
    {команды}
```

Рассмотрим небольшой пример:

```
count = 1

    while (count <= 10)
```

```
{
print n
n++
}
```

Цикл *for*

Синтаксис цикла `for` следующий:

```
for (инициализация; условие; инкремент)
    {команды}
```

Цикл `for` — это цикл со счетчиком. Он выполняется, пока истинно условие. Перед запуском цикла выполняются команды инициализации, как правило, сбрасывается счетчик (или инициализируется). Далее проверяется условие: если оно истинно, выполняются команды. После выполнения команд происходит инкремент счетчика.

Пример цикла `for`:

```
for (n=1; n <= 10; n++)
        print n
```

Для ассоциативных массивов синтаксис цикла `for` немного другой:

```
for (элемент in массив)
    {команды}
```

В теле цикла вы можете использовать команды `break` (прерывает цикл) и `continue` (прерывает текущую итерацию).

3. Примеры

Теперь рассмотрим несколько примеров. Предположим, у нас есть файл friends с данными о наших друзьях в таком формате:

```
ник год_рождения e-mail        icq
```

Количество полей, сами понимаете, может быть произвольным. Для большей определенности файл friends представлен в листинге 1.

Листинг 1. Файл friends

```
den    1983   den@localhost 111111111
evg    1982   evg@localhost 111111112
max    1987   max@localhost 111111114
fox    1980   fox@localhost 111111113
dm     1979   dm@locahost   111111119
vvv    1980   vvv@localhost 111111117
ppt    1982   ppt@localhost 111111115
```

Самая простая программа на языке awk выглядит так:

```
{print}
```

Данная программа просто выведет весь этот файл. Для ее запуска введите команду:

```
gawk '{print}' friends
```

Попробуем усложнить программу и вывести всех, кто родился в 1980 году. Для этого нужно использовать регулярные выражения. Регулярные выражения заключаются в слэши:

```
gawk '/1980/' friends
```

Вывод будет таким:

```
fox 1980   fox@localhost 111111113
vvv 1980   vvv@localhost 111111117
```

Спрашивается, почему мы не оформляем наши программы в сценарии, а запускаем их из командной строки? Да потому что так проще — ради программы, состоящей из одной строки, не хочется создавать файл, устанавливать его права, передавать ему параметры. А так все видно: каким интерпретатором обрабатывается программа (gawk), текст программы и обрабатываемый файл (friends).

Теперь попробуем вывести только определенные поля. Порядок вывода полей значения не имеет:

```
gawk '{print $2, $1}' friends
```

Мы вывели второе и первое поле:

```
1983 den
1982 evg
1987 max
...
```

Можно усложнить программу и вывести только несколько полей, но которые соответствуют регулярному выражению, например:

```
gawk '/1980/ {print $2, $1}' friends
```

Вывод программы:

```
1980 fox
1980 vvv
```

Выведем все записи, где в первом поле есть буква d:

```
gawk '$1 ~ /d/' friends
```

Вывод программы:

```
den 1983   den@localhost 111111111
dm  1979   dm@locahost   111111119
```

До этого мы выводили записи, которые соответствуют определенному регулярному выражению. Давайте выведем записи, где поля соответствуют тому или ино-

му регулярному выражению. Например, вот программа, выводящая записи, где второе поле равно 1982:

```
gawk '$2 == 1982' friends
```

Вывод программы:

```
evg 1982   evg@localhost 111111112
ppt 1982   ppt@localhost 111111115
```

Теперь напишем awk-сценарий — более сложную программу, которая не помещается в одну строку. Программа подсчитывает, сколько человек родилось в 1980 году. Данную программу можно было бы записать в одну строку, но мы используем образцы `BEGIN` и `END` для улучшения вывода программы.

Листинг 2. Сценарий на awk

```
#!/usr/bin/awk -f
BEGIN {
                print "Список друзей"
                year=0
}
{
                if ($2 == "1980") year=year+1
}
 END {
                printf ("\tВсего записей:\t%d\n", NR)
                printf ("\tВ 1980 году родилось друзей:\t%d\n", year)
}
```

Чтобы запустить сценарий, нужно сделать его исполнимым и передать ему файл друзей:

```
$ chmod +x friends.awk
$ ./friends.awk friends
```

Теперь усложним задачу: вычислим сумму столбца в текстовом файле. Команда достаточно проста, поэтому ее даже не буду оформлять в виде листинга. Здесь мы подсчитываем и выводим сумму 4-го столбца файла somefile. Думая, зная bash и awk, вы без проблем оформите приведенную далее команду в виде bash-сценария, чтобы не вводить ее каждый раз:

```
cat somefile | awk '{s += $4} END {print s}'
```

Awk - очень гибкий язык, с его помощью можно решить практически все задачи обработки текста. Представим, что нам нужно найти все битые ссылки на нашем сайте. Можно, конечно, перелопатить все страницы и проанализировать каждую ссылку, но гораздо проще проанализировать журнал Apache, чтобы вывести все 404-ые ошибки: мы найдем отсутствующие на нашем сервере ресурсы и сможем

исправить ссылки (возможно, ресурсы не отсутствуют, а мы просто допустили ошибку в имени файла). Вывод всех 404-ых ошибок достигается так:

```
awk '$9 == 404 {print $7}' /var/log/apache2/access_log | uniq -c | sort -rn | more
```

Да, мы не найдем битые ссылки, указывающие на другие ресурсы, но зато поймем, каких файлов не хватает на нашем сервере, что тоже ничего, особенно, если учесть, что с этой задачей мы справились с помощью одной команды.