Министерство науки и высшего образования Российской Федерации

Рязанский государственный радиотехнический университет им. В.Ф. Уткина

Утилита awk

Методические указания к практическим занятиям

Рязань 2020

УДК 681.3.06

Утилита awk: методические указания к практическим занятиям / Рязан. гос. радиотехн. ун-т.; сост. А.А. Митрошин, В.Г. Псоянц. – Рязань, 2020. – 16 с.

Содержат описание практического занятия, используемого в курсе «Операционная система Linux». Могут использоваться при изучении других курсов, связанных с операционной системой Linux.

Предназначены для студентов очной, заочной и очно-заочной форм обучения направления подготовки «Информатика и вычислительная техника». Могут использоваться для студентов других направлений подготовки.

Могут использоваться как методические указания к лабораторным работам в курсах, связанных с изучением операционной системы Linux и свободно распространяемого программного обеспечения.

Ил. --. Библиогр.: -- назв.

Операционная система Linux, утилита awk

Печатается по решению редакционно-издательского совета Рязанского государственного радиотехнического университета.

Рецензент: кафедра САПР вычислительных средств Рязанского государственного радиотехнического университета (зав. кафедрой засл. деят. науки и техники РФ В.П.Корячко)

Утилита awk

Составители: Митрошин Александр Александрович

Псоянц Владимир Грикорович

Редактор \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Корректор \_\_\_\_\_\_\_\_\_\_\_\_\_

Подписано в печать \_\_\_\_\_\_\_\_. Формат бумаги 60×84 1/16.

Бумага газетная. Печать трафаретная. Усл. печ. л. 1,0.

Уч-изд. л. 1,0. Тираж 50 экз. Заказ

Рязанский государственный радиотехнический университет.

390005, Рязань, ул. Гагарина, 59/1.

Редакционно-издательский центр РГРТУ.

**Утилита awk**

**Основная задача утилиты awk заключается в просмотре текстового файла или строки с целью нахождения в них информации, соответствующей заданному критерию отбора. После выборки нужных данных к ним применяются функции, выполняющие обработку текста. Сложные сценарии awk обычно применяются для формирования отчетов, основанных на текстовых файлах.**

**Утилита awk** сканирует input (стандартный или указываемый набор файлов), и над строками, удовлетворяющими заданному образцу, выполняет указываемые действия. Строка может содержать максимально до 256 символов. Форматы вызова:

awk [-Fc] [-f file] [files]

awk [-Fc] [prog] [files]

где:

prog

программа, вида: ' образец ${$действие$}$'

file

файл с AWK-программой:

образец {действие}

образец {действие}

...

files

файлы, предназначенные для обработки.

-Fc

устанавливает разделитель полей в «с»

Вызвать утилиту awk можно тремя способами.

Первый способ заключается в передаче ей необходимых команд непосредственно в командной строке:

awk [-F разделитель\_полей] ‘сценарий’ входной\_файл …

В одинарных кавычках указывается список инструкций *языка* awk.

Задавать разделитель полей с помощью опции –F не обязательно; по умолчанию используется пробел.

Второй способ вызова утилиты awk состоит в создании отдельного файла сценария, содержащего список инструкций awk. При этом утилита awk указывается в первой строке сценария в качестве интерпретатора команд:

#! /bin/awk -f

Созданный файл может быть вызван из командной строки.

Выражение #! называется «магической» последовательностью. Подобная инструкция должна быть первой строкой любого сценария. Общий формат инструкции:

#! /путь/программа [командная строка]

Подразумевается, что:

- система, в которой запускается сценарий, распознает эту последовательность;

- указанная программа воспринимает символ как признак комментария.

Когда происходит запуск исполняемого файла, система проверяет, начинается ли он с «магической» последовательности. Если нет, то файл содержит машинные коды и выполняется непосредственно. Если выражение #! обнаружено, то это файл сценария. В этом случае происходит следующее:

1) первая строка сценария заменяет собой командную строку, из которой удаляется «магическая» последовательность;

2) предыдущая команда передается новой командной строке в качестве аргумента.

Опция –f утилиты awk говорит о том, что выполняемые команды находятся в указанном далее файле.

После создания файла сценария необходимо сделать его исполняемым, например, с помощью следующей команды:

chmod u+x имя\_файла\_сценария

При третьем способе вызова все инструкции awk помещаются в отдельный файл, после чего осуществляется вызов этого файла:

awk –f имя\_файла\_сценария входной\_файл …

***Сценарии awk***

Сценарий awk – это набор инструкций, состоящих из *шаблонов* и связанных с ними *процедур*.

Шаблоном может служить любая условная или составная конструкция либо регулярное выражение.

Существуют также два специальных шаблона BEGIN и END. Шаблон BEGIN применяется для инициализации переменных и создания заголовков отчета. Шаблон END используется для вывода итоговых данных и выполнения инструкций по завершении обработки файла.

Если никакой шаблон не указан, то процедура выполняется для каждой записи из входного файла.

Тело процедуры заключается в фигурные скобки. Чаще всего процедура выводит информацию на экран, но она может также содержать операторы присваивания, управляющие конструкции и встроенные функции.

Если процедура не задана, awk выводит на экран все содержимое записи, соответствующей шаблону.

Утилита последовательно просматривает строки файла. Отыскав первый символ-разделитель, она помечает все предшествующие символы как поле номер 1. Символы между первым и вторым разделителем трактуются как поле номер 2 и т.д. Процесс анализа завершается при обнаружении символа новой строки, который по умолчанию считается символом конца записи. После этого содержимое полей сбрасывается и утилита переходит к следующей строке.

На поля текущей записи можно ссылаться следующим образом: $1, $2, … $n. Для обозначения всех полей текущей записи используется идентификатор $0.

Чтобы вывести на экран содержимое записи, можно в теле процедуры использовать команду print со списком нужных полей.

Скопируем в домашний каталог файл /etc/passwd с помощью команды

cp /etc/passwd password.txt

Содержимое файла password.txt показано на рис. 1.

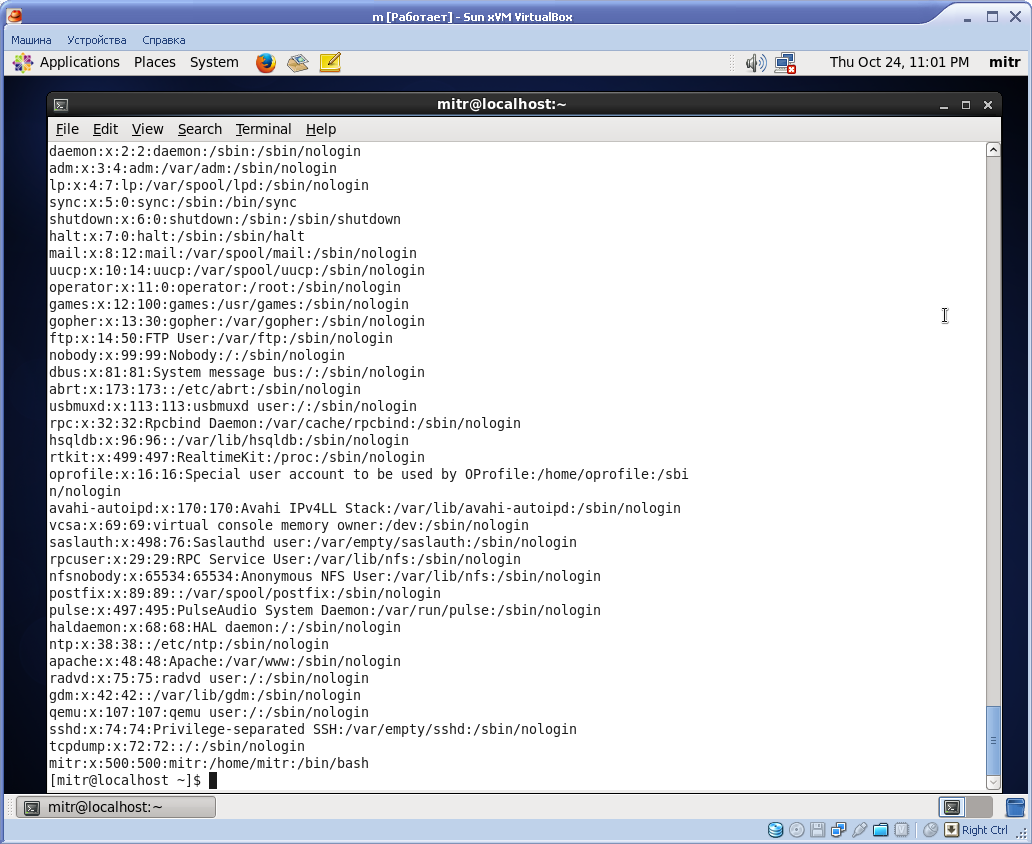


Рисунок 1. Содержимое файла password.txt

Файл представляет собой набор записей, в качестве разделителей полей которого используется символ «:». С помощью утилиты awk выведем список домашних каталогов пользователей, которые содержатся в шестом поле записей файла (рис. 2).

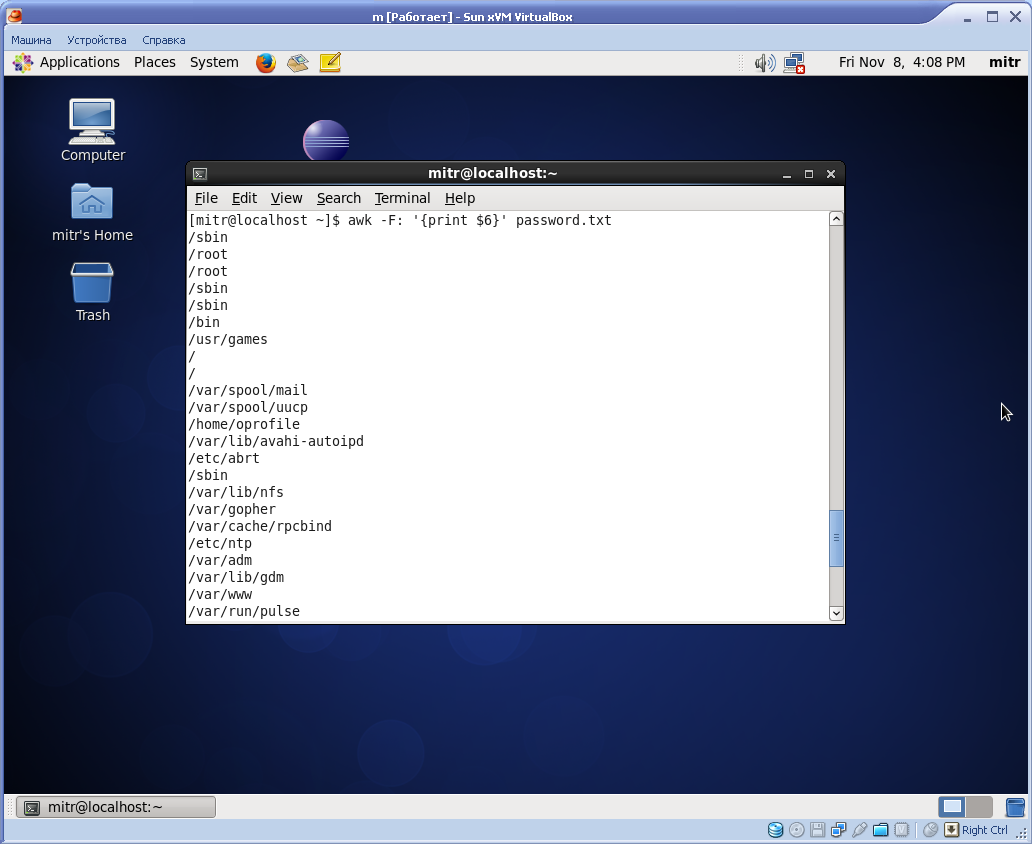


Рисунок 2. Печать содержимого шестого поля с помощью awk

Для того, чтобы сохранить полученные данные в файл, воспользуемся перенаправлением вывода, показанном на рис. 3.

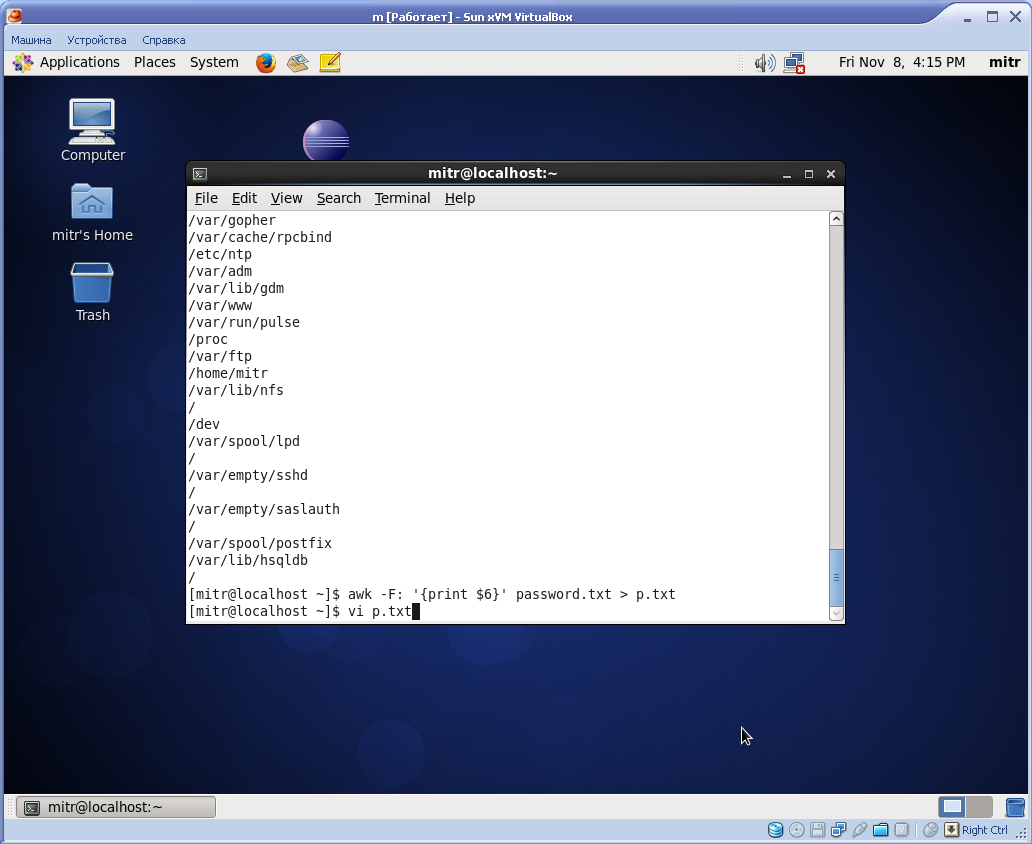


Рисунок 3. Перенаправление вывода awk

Для вывода всех полей записи вместо $6 нужно использовать $0.

Для того чтобы добавить в вывод awk заголовок и резюме отчета, воспользуемся специальными шаблонами BEGIN и END, например:

awk –F: ‘BEGIN {print “Каталог”} {print $2} END {print “Все”}’ password.txt > p.txt

Если не указать файлы, по которым утилита awk производит поиск, то она запрашивает их ввод из командной строки, как это показано на рис. 4. Завершать ввод записей нужно нажатием Ctrl+D.

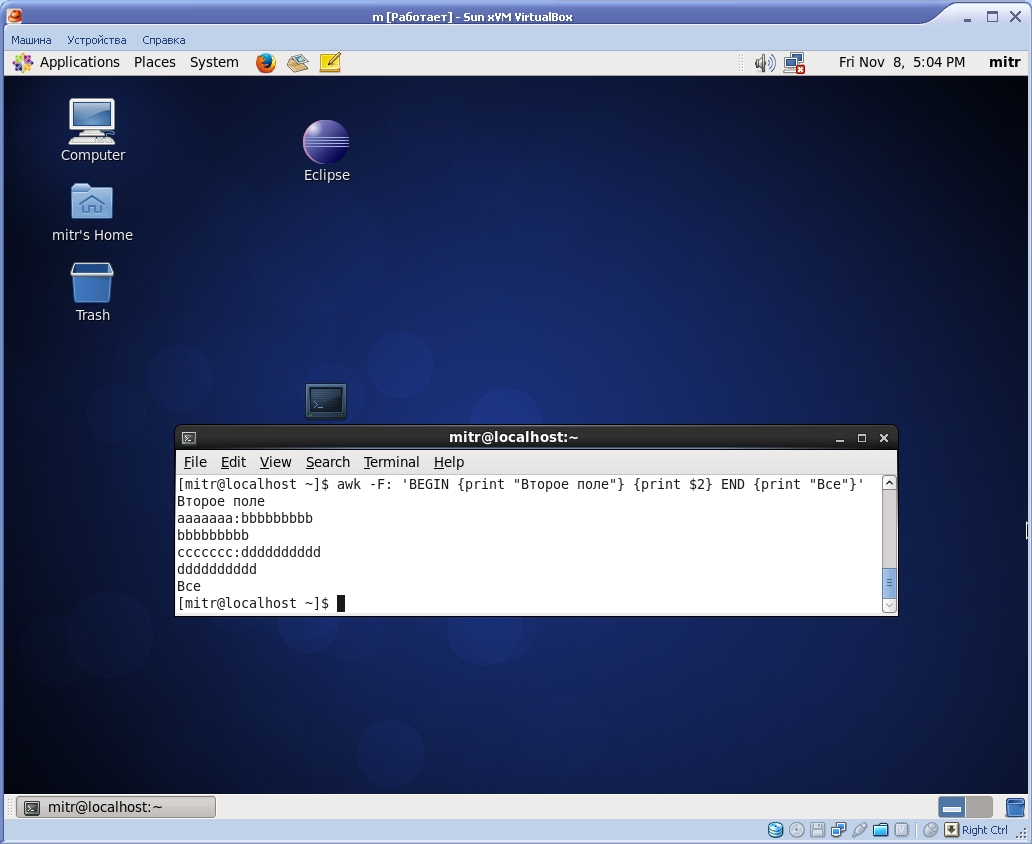


Рисунок 4. Работа awk без указанных входных файлов

***Шаблоны***

Для осуществление поиска в AWK языке допускается использование регулярных выражений, заключенных в «/ /».

Регулярные выражения – мощное средство текстового поиска. Регулярные выражения в той или иной форме используются всеми основными текстовыми редакторами и утилитами. Но, к сожалению, наборы поддерживаемых выражений могут меняться от программы к программе.

Ниже перечислены метасимволы и операторы, применяемые в разных регулярных выражениях.

**\** - Отменяет специальное значение следующего за ним метасимвола. Например, \$ соответствует символу $. Специальными являются следующие символы: $, ., ‘, “, \*, [, ], |, (, ), \, +, ?.

**ˆ** - Соответствует началу строки. Например, ˆ@chapter соответствует @chapter в начале строки.

$ - Соответствует концу строки. Например, p$ соответствует строке, которая закачивается на p.

. – Соответствует любому отдельному символу.

[...] – Соответствует любому символу из числа заключенных в скобки. Чтобы задать диапазон символов, укажите начальный и конечный символы, разделив их символом «-». Например, вместо шаблона [12345] можно использовать шаблон [1-5].

[ˆ ...] – Соответствует любому символу, кроме тех, которые указаны в скобках. Например, [ˆawk] соответствует любой символ, кроме a, w, k.

| - Логическое «или» в регулярном выражении. Например, ˆP|[[:digit:]] соответствует любой строке, которая начинается с символа P или любой цифры.

(...) – Круглые скобки используются для группировки в регулярных выражениях. Используются в регулярных выражениях, содержащих |. Например, @(samp|code) сопоставляется как с @code, так и с @samp.

\* - Указывается на то, что предыдущий шаблон встречается ноль или более раз. awk поддерживает расширенный синтаксис регулярных выражений, где используются также следующие символы:

? - предшествующий шаблон встречается не более одного раза;

+ - предыдущий шаблон встречается один или более раз).

Определены также *классы символов*. Класс символов – специальная нотация для определения списка символов. Классы символов приведены в табл. 1.

Таблица 1. Классы символов

|  |  |
| --- | --- |
| **Класс символов** | **Описание** |
| [:alnum:] | Буквы и цифры |
| [:alpha:] | Буквы |
| [:blank:] | Пробел или табуляция |
| [:cntrl:] | Управляющие символы |
| [:digit:] | Цифры |
| [:graph:] | Отображаемые и печатаемые символы |
| [:lower:] | Буквы в нижнем регистре |
| [:print:] | Печатаемые символы (не управляющие символы) |
| [:punct:] | Знаки пунктуации |
| [:space:] | Пробельные символы, такие как пробел, табуляция, подача новой страницы |
| [:upper:] | Буквы в верхнем регистре |
| [:xdigit:] | Шестнадцатеричные цифры |

Например, чтобы проверить, состоит ли первое поле записи только из букв, можно использовать выражение

$1~/[[:alpha:]]/

Установлены следующие приоритеты выполнения операторов на одном скобочном уровне: [ ] \* + ? конкатенация |.

Например:

|  |  |
| --- | --- |
| /Ivan/ | Указывает на строки, содержащие Ivan. |
| /[Oo]lga|[Mm]ike|[Mm]al/ | Указывает на строки, содержащие Olga или olga или Mike или mike или Mal или mal. |
| /number[0-9]/ | Указывает на строки, содержащие number0 или number1 или … или number9. |

В шаблонах могут указываться выражения отношения, которые бывают двух типов:

*- Выражение* принадлежность *Выражение;*

*- Выражение* логическая операция *Выражение.*

Принадлежность: ~ - содержится; !~ - не содержится.

Логическая операция: <, <=, ==, !=, >=, >.

Например:

|  |  |
| --- | --- |
| $1 ~ /[Ii]van/ | Указывает на строки, первое поле которых содержит Ivan или ivan. |
| $1 >= "s" | Указывает на строки, первое поле которых начинается с символа s или следующих за ним по порядку: t, u, v… |

Допускается логическая комбинация шаблонов с использованием следующих знаков: || - «или»; && - «и»; ! - «не».

Комбинация: «шаблон1, шаблон2» указывает, что действие выполняется над строками, попадающими в указанный диапазон: начиная от строки, удовлетворяющей шаблону «шаблон1» и вплоть до строки, удовлетворяющей шаблону «шаблон2», включая ее саму. Например:

|  |  |
| --- | --- |
| /02\.95/ && ($1 !~ /\.ru/ $2 !~ /\.ru/) | Указывает на строки, содержащие 02.95 и не имеющие .ru одновременно в первом и во втором поле. |
| NR == 100, NR == 200 | Указывает строки с номерами от 100 до 200. |

***Переменные***

В языке awk выделяются две группы *переменных*: предопределенные (встроенные) и декларированные в программе. Исходные значения предопределенных переменных устанавливаются интерпретатором awk в процессе запуска и выполнения awk-программы. К предопределенным относятся переменные, перечисленные в табл. 1.

Таблица 1. Предопределенные переменные

|  |  |
| --- | --- |
| **Предопределенная переменная** | **Значение по умолчанию** |
| NR - номер текущей строки |  |
| NF - число полей в текущей строке |  |
| RS - разделитель строк во входном потоке | "\0" – символ новой строки |
| FS - разделитель полей во входном потоке | Пробел |
| ORS - разделитель строк во входном потоке | Символ новой строки |
| OFS - разделитель полей в выходном потоке | Пробел |
| OFMT - формат вывода чиcел | "%.6g" |
| FILENAME - имя входного файла, который обрабатывается в настоящее время |  |

Другим переменным можно присваивать начальные значения. По умолчанию это "0" или пустая строка.

Над переменными можно выполнять операции, синтаксис которых совпадает с синтаксисом Си: =, +=, -=, \*=, /=, %=, +, /, %, ++, --.  
Awk рассматривает переменную как строковую, пока не возникает необходимость выполнить операции, например:

- если операция - пробел (конкатенация), то переменные - строки;

- если операция – «+», то переменные - числа с плавающей точкой.

Например, при вызове

awk '{a = $3 $4; print a}' a.txt

поля $3 и $4 рассматриваются как строки, над ними выполняется конкатенация (операция - пробел) и значение конкатенации присваивается переменной a. При при вызове

awk '{a = $3+$4; print a}' a.txt

поля $3 и $4 рассматриваются как числа с плавающей точкой, они складываются и их сумма присваивается переменной a.

К переменным могут применяться встроенные функции:

- sin (expr) - синус  expr;

- cos (expr) - косинус expr;

- exp (expr) - экспонента expr;

- log (expr) - натуральный логарифм expr;

- sqrt (expr) - извлечение корня expr;

- int (expr) - целая часть числа expr;

- length (s) - длина строки s;

- printf (fmt, ...)  форматирование (аналогично Си) по спецификации fmt;

- substr (s, m, n) - подстрока в n символов строки s, начинающаяся с m;

- getline () - чтение следующей строки; 0 - конец файла, иначе 1;

- index (s1, s2) - номер позиции, с которой s1 совпадает с s2, иначе 0;

- split (s, M, c) - строка s разбивается на элементы массива M по разделителю c (по умолчанию FS=" ") - возвращает число полей.

Для вызова любых системных команд может применяться встроенная функция system. Например:

awk 'BEGIN { system("pwd") }'

awk 'BEGIN { system("date") }'

***Массивы***

*Массивы* определяются в момент первого использования. Их не нужно объявлять заранее. Не нужно также указывать количество элементов массива.

Индекс массива - любое ненулевое значение или строка. Массивы ассоциативные, т.е. доступ к их элементам осуществляется не по числовому индексу, а по совпадению содержания, например:

day [May][07] = “Вторник”

day [May][09] = “Четверг”

day [Nov][07] = “Четверг”

Для доступа к элементам массива обычно применяется цикл for, имеющий следующий синтаксис:

for (*элемент* in *массив*) действие *массив*[*элемент*],

например:

for (i in myarray) pritn myarray[i]

Пример использования массива:

# Запись 1 для каждого слова

# NF – номер поля текущей записи

{

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

used[$i] = 1

}

# определение числа слов, длина которых больше 10

END {

for (x in used)

if (length(x) > 10) {

++num\_long\_words

print x

}

print num\_long\_words, слов длиннее 10 символов"

}

Удаление элемента массива осуществляется с помощью выражения delete: delete *array*[*index*].

***Управляющие операторы в действиях***

Управляющие операторы, такие как if, while и т.д. управляют потоком выполняемых awk-программ.

***if-else***

Синтаксис:

if (*условие*) *выражение* [else *выражение*]

Пример:

if (x % 2 == 0)

print "x - четное"

else

print "x - нечетное"

***Цикл while***

Синтаксис:

while (условие)

выражение

Пример:

awk ’{ i = 1

while (i <= 3) {

print $i

i++

}

}’ a.txt

***Цикл do-while***

Синтаксис:

do

*выражение*

while (*условие*)

Пример:

{ i = 1

do {

print $0

i++

} while (i <= 10)

}

***Цикл for***

Синтаксис:

for (*инициализация*; *условие*; *приращение*)

*выражение*

Пример:

awk ’{ for (i = 1; i <= 3; i++)

print $i

}’ a.txt

***break***

Оператор break позволяет выйти из цикла.

Пример:

# Поиск наименьшего делителя

{

num = $1

for (div = 2; div\*div <= num; div++)

if (num % div == 0)

break

if (num % div == 0)

printf "Smallest divisor of %d is %d\n", num, div

else

printf "%d is prime\n", num

}

***continue***

Оператор continueпрекращает текущую итерацию цикла и переходит к следующей итерации.

Пример:

BEGIN {

for (x = 0; x <= 20; x++) {

if (x == 5)

continue

printf "%d ", x

}

print ""

}

***next***

Оператор next принуждает awk завершить обработку текущей записи и перейти к следующей записи.

Пример:

# NF – номер поля текущей записи

NF != 4 {

err = sprintf("%s:%d: пропущено: NF != 4\n", FILENAME, FNR)

# NFR – номер текущей записи в файле

print err > "/dev/stderr"

next

}

***Функции пользователя***

В awk-программах возможно создание функций пользователя. Функция описывается следующим образом:

function *имя*([*список параметров*])

{

*Тело функции*

}

Например:

function myprint(num)

{

printf "%6.3g\n", num

}

Тело функции может содержать (а может и не содержать) выражение return, имеющее следующий синтаксис:

return [*expression*]

Выражение return прекращает выполнение функции и определяет значение, возвращаемое функцией. Если необязательный параметр *expression* опущен, то в качестве значения функции возвращается пустая строка.

***Порядок выполнения работы***

1) Изучите теоретический материал.

2) Выполните практическое задание.

3) Ответьте на контрольные вопросы.

***Практическое задание***

1) Объясните, как работает следующая функция.

# mystrtonum --- преобразует строку в число

function mystrtonum(str, ret, n, i, k, c)

{

if (str ~ /^0[0-7]\*$/) {

# восьмеричное

n = length(str)

ret = 0

for (i = 1; i <= n; i++) {

c = substr(str, i, 1)

# index() возвращает 0 если c нет в

# строке,содержит c == "0"

k = index("1234567", c)

ret = ret \* 8 + k

}

} else if (str ~ /^0[xX][[:xdigit:]]+$/) {

# шестнадцатиричное

str = substr(str, 3) # обрезает начальные 0x

n = length(str)

ret = 0

for (i = 1; i <= n; i++) {

c = substr(str, i, 1)

c = tolower(c)

k = index("123456789abcdef", c)

ret = ret \* 16 + k

}

} else if (str ~ \/^[-+]?([0-9]+([.][0-9]\*([Ee][0-9]+)?)?|([.][0-9]+([Ee][-+]?[0-9]+)?))$/) {

# десятичное число, возможно с плавающей точкой

ret = str + 0

} else

ret = "Не число"

return ret

}

***Контрольные вопросы***

1) Для чего предназначена утилита awk?

2) Какими способами можно вызвать утилиту awk?

3) Для чего предназначены шаблоны шаблона BEGIN и END?

4) Расскажите о регулярных выражениях.

5) Что такое «классы символов»? какие классы символов вы знаете?

6) Как определить собственные переменные в скрипте awk? Какие встроенные переменные определены в awk?

7) Расскажите о массивах и их использовании.

8) Расскажите об операторе условного перехода.

9) Расскажите об операторах циклов.

10) Расскажите об операторах break, continue и next.

11) Расскажите о функциях пользователя.

***Библиографический список***

1. Тейнсли Д. Linux и UNIX: программирование в shell. Руководство разработчика. –К.: Издательская группа BHV, 2001.

2. www.gnu.org – Официальный сайт проекта gnu.