ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Р. Е. Алексеева
Институт радиоэлектроники и информационных технологий

Кафедра «Информационные радиосистемы»

Разработка сценариев на языке оболочки bash

Методические указания к лабораторной работе №6
по дисциплине «Системное программирование»
для студентов специальности 210302 «Радиотехника»
дневной формы обучения

Нижний Новгород
2009

Составитель С.Б. Сидоров

УДК 519.6

Разработка сценариев на языке оболочки bash: Метод. указания к лаб. работе №6 по дисциплине «Системное программирование» для студентов специальности «Радиотехника» дневной формы обучения. / НГТУ; Сост.: С.Б. Сидоров. Н. Новгород, 2009.– 18 с.

Изложены краткие сведения о правилах разработки сценариев на языке программирования оболочки bash. Приведены базовые синтаксические конструкции языка программирования. Рассмотрены основные команды и приведены примеры их использования. Изложена методика отладки сценариев. Определён порядок выполнения лабораторной работы, сформулированы вопросы для самопроверки.

Редактор _____

Подп. к печ._____. Формат 60x84$^{1}/_{16}$. Бумага_____. Печать офсетная.
Печ. л.___1,25____. Уч.-изд.л._____. Тираж_____экз. Заказ_____.

Нижегородский государственный технический университет им. Р.Е. Алексеева.
Типография НГТУ.
Адрес университета и полиграфического предприятия:
603950, ГСП-41, г. Нижний Новгород, ул. Минина, 24.

1. Цель работы

Изучение принципов разработки сценариев на языке программирования оболочки bash. Разработка собственного сценария с использованием основных синтаксических элементов и команд языка. Знакомство с методикой отладки сценариев для поиска и устранения ошибок.

2. Краткие сведения

2.1. Основы разработки сценариев оболочки

Взаимодействие пользователя с операционной системой осуществляется через оболочку, которая представляет собой внешнюю программу.

Сразу после запуска оболочки производится её инициализация для установки ряда параметров. При этом оболочкой производится чтение двух файлов: */etc/profile* и *~/.profile*. В первом из них содержатся настройки параметров, общие для всех пользователей. Во втором файле каждый пользователь может разместить свои собственные настройки для работы с оболочкой.

Пользовательская оболочка может быть запущена на выполнение в двух режимах – интерактивном и не интерактивном. Когда оболочка выдаёт пользователю приглашение, она работает в **интерактивном** режиме. Это означает, что оболочка принимает ввод от пользователя и выполняет команды, которые пользователь укажет. Оболочка называется интерактивной, поскольку она взаимодействует с пользователем. Завершение работы с оболочкой в этом случае происходит по команде пользователя.

В **не интерактивном** режиме оболочка не взаимодействует с пользователем. Вместо этого она читает команды из некоторого файла и выполняет их. Когда будет достигнут конец файла, оболочка завершится. Запуск оболочки в не интерактивном режиме можно осуществить следующим способом:

$ /bin/sh имя_файла

Здесь *имя_файла* – имя файла, содержащего команды для выполнения. Такой файл называется *сценарием оболочки*. Он является текстовым файлом и может быть создан любым доступным текстовым редактором.

Пользователю может представлять неудобство каждый раз указывать имя программы-оболочки */bin/sh* для выполнения сценария. Для того чтобы иметь возможность выполнять сценарий, набирая только его имя, прежде всего

необходимо сделать его исполняемым. Для этого необходимо установить соответствующие права доступа к файлу с помощью команды *chmod* следующим способом:

chmod u+x имя_файла

Кроме этого, необходимо явно указать, какая оболочка должна использоваться для выполнения данного сценария. Это можно сделать, разместив в первой строке сценария последовательность символов *#!/bin/sh* Здесь указывается, что для выполнения сценария необходимо использовать оболочку */bin/sh*

Сценарий может содержать *комментарии*. Комментарий – это оператор, который может размещаться в сценарии оболочки, но оболочкой не исполняется. Комментарий начинается с символа # и продолжается до конца строки.

Ниже приведён пример короткого сценария:

```
#!/bin/sh
#пример короткого сценария
date ; who ;
```

2.2. Переменные

Переменная – это «слово», которому приписано значение. Оболочка разрешает создавать и удалять переменные, а также присваивать им значения. В большинстве случаев разработчик ответственен за управление переменными в сценариях. Использование переменных позволяет создавать гибкие, легко адаптируемые сценарии. Определяются переменные следующим образом (отсутствие пробелов до и после символа «=» существенно):

имя_переменной=значение

Например, *MY_NAME=Sergey* определяет переменную с именем *MY_NAME* и присваивает ей значение *Sergey*. Имена переменных определяются по тем же правилам, что и в языке программирования Си. В переменной можно сохранять любое нужное значение. Особый случай – когда это значение содержит пробелы. Для правильного выполнения такого действия указанное значение достаточно заключить в кавычки.

Для того чтобы получить значение переменной, перед её именем необходимо поставить знак *$*. В том случае, когда некоторая переменная становится ненужной, её можно удалить операцией *unset имя_переменной*. Ниже приведён пример, иллюстрирующий работу с переменными в сценариях.

```
#!/bin/sh
#пример операций над переменными
MY_NAME=Sergey
MY_FULL_NAME="Sergey B. Sidorov"
echo name = $MY_NAME  and full name = $MY_FULL_NAME
unset MY_NAME
```

В результате выполнения команды *echo* на терминал будет выдано сообщение: *name = Sergey and full name = Sergey B. Sidorov*

Все рассмотренные выше примеры переменных – это примеры *скалярных* переменных, то есть таких, которые могут хранить только одно значение. Наряду с ними можно использовать *переменные-массивы*. Доступ к элементам массива осуществляется операцией индексирования *[]*. Язык программирования оболочки не требует предварительного объявления переменной массивом с указанием его размерности. При этом отдельные элементы массива создаются по мере доступа к ним. Ниже приведён пример работы с массивом *NAME*.

```
#!/bin/sh
#пример использования массива
NAME[0]=Сергей
NAME[10]=Катя
NAME[2]=Лиза
echo все имена: ${NAME[*]}
echo ${NAME[10]} и ${NAME[2]} сестры
```

Если вместо индекса элемента использовать *, результатом выражения будет список значений всех элементов массива (в данном случае таковых три).

2.3. Область видимости переменных

Каждая переменная имеет свою *область видимости*. Допустим, в сценарии определена некоторая переменная. Что будет с ней после завершения этого сценария, доступна ли она из других сценариев, вызываемых исходным? В языке оболочки все переменные делятся на три категории: локальные переменные, переменные окружения и переменные оболочки.

Локальная переменная – это такая переменная, которая существует только внутри конкретного экземпляра оболочки. Она не доступна программам, запускаемым на выполнение из этой оболочки. Все рассматривавшиеся ранее переменные были локальными.

Переменная окружения – это переменная, которая доступна любой

программе, запущенной из данной оболочки. Некоторые программы нуждаются в определённых переменных окружения. В таком случае в сценарии их можно определить.

Переменная оболочки – это специальная переменная, которая устанавливается оболочкой и необходима ей для корректной работы. Некоторые из них являются переменными окружения, а некоторые – локальными. В переменных с именами $1, 2, 3, \ldots$ сохраняются параметры командной строки. То есть если некий сценарий *script* запустить в виде *script something*, то в его оболочке *$1=something*.

Переменную можно разместить в окружении, выполнив команду экспорта: *export имя_переменной*. В результате выполнения следующего сценария на терминал будут выданы значения всех переменных окружения оболочки и, в том числе, переменной *MY_NAME*.

```
#!/bin/sh
#пример размещения переменной в окружении
MY_NAME=Sergey ; export MY_NAME;
set
```

Ниже в таблице приводится список некоторых стандартных переменных окружения и переменных оболочки с указанием их назначения и примера их инициализации.

Имя переменной	***Краткое описание***
HOME	Содержит путь к домашнему каталогу пользователя
TERM	Тип используемого терминала *TERM=vt100*
PATH	Определяет список каталогов, в которых оболочка ищет запускаемые на выполнение программы *PATH=/bin:/usr/bin*
MANPATH	Определяет список каталогов, в которых программа *man* ищет справочную информацию по запрошенной команде *MANPATH=/usr/man:/usr/share/man*
PWD	Содержит путь к текущему рабочему каталогу
RANDOM	Формирует случайное целое число из диапазона 0..32767 каждый раз при доступе к этой переменной
?	Результат выполнения предыдущей команды
HOSTNAME	Имя узла (компьютера), на котором выполняется оболочка
SECONDS	Число секунд, прошедших с момента запуска оболочки.

2.4. Средства ввода-вывода

Задачи ввода–вывода могут быть решены посредством использования специальных команд *echo*, *printf*, *read*. Команда *echo* выдаёт на стандартное устройство вывода значения всех своих параметров. Команда *printf* подобна своему аналогу — функции *printf* в программах на языке Си. Первым параметром указывается форматная строка, а далее перечисляются выдаваемые значения.

Команда *read имя1 имя2 ... имяN* считывает со стандартного устройства ввода строку, выделяет из неё отдельные слова (группы символов, отделяемые пробелами) и каждое слово заносит в указанные в качестве параметров соответствующие переменные. При этом если переменных меньше, чем выделенных слов, то в последнюю из них будет занесена оставшаяся часть строки.

В качестве примера приведён сценарий, в котором от пользователя запрашивается строка, после чего она выводится на терминал. Параметр *–n* запрещает перевод на следующую строку по окончании вывода командой *echo*, разрешённый по умолчанию.

```
#!/bin/sh
#пример использования средств ввода-вывода
echo –n Введите строку:
read INPUT
printf "%s\n" "$INPUT"
```

2.5. Подстановки

Выделяют следующие виды подстановок:

- **подстановка имён файлов** – позволяет выполнять преобразование строки, содержащей шаблоны, в список имён файлов;
- **подстановка переменной** – позволяет управлять значением переменной, основываясь на её состоянии;
- **подстановка команды** – позволяет захватить вывод команды и подставить его в другую команду;
- **арифметическая подстановка** – позволяет выполнять простые математические вычисления, используя оболочку.

Подстановки имён файлов основывается на использовании различных шаблонов для задания группы файлов в кратком виде вместо полного их перечисления. Шаблон * отождествляется с любым количеством любых символов в имени файла. Шаблон *?* соответствует одному произвольному

символу, а шаблон *[группа_символов]* явно определяет набор символов, допустимых в месте его расположения. Ниже приведены примеры использования шаблонов:

```
*.cpp          #список файлов, оканчивающихся на .cpp
module?.h      #список файлов с одним произвольным символом вместо ?
module[aA].cpp     #modulea.cpp и/или moduleA.cpp
```

Механизм подстановки команды состоит в выполнении оболочкой указанного набора команд и последующей подстановки их вывода вместо самих команд. Подстановка команды выполняется при записи команды в обратных апострофах. Например, в результате выполнения команды *DATE=`date`* переменная *DATE* примет значение – вывод команды *date*.

Для вычисления значения арифметического выражения *expression* необходимо использовать команду следующего вида: *$((expression))* При этом в выражении могут использоваться операции сложения, вычитания, умножения и целочисленного деления, а также круглые скобки для задания порядка вычислений. Например, в результате выполнения следующей команды переменная *foo* примет значение 3.

```
foo=$(( ((5 + 3*2) – 4) / 2 ))
```

2.6. Команды ветвления

Язык оболочки содержит два оператора ветвления:

- оператор *if*;
- оператор *case*.

Оператор *if* выполняет действия в зависимости от истинности заданного условия и имеет следующий синтаксис:

```
if list1
then
        commands1
elif list2
then
        commands2
else
        commands3
fi
```

В приведённой записи как *elif*, так и *else* могут отсутствовать. Если тело оператора *if* небольшое, то обычно используют запись в одну строку такую,

как: *if list1 ; then commands1 ; else commands2 ; fi;* При этом важно правильно ставить *;*. Алгоритм выполнения оператора *if* следующий:

1. выполняется *list1*;
2. если код завершения *list1* 0 (истина), то выполняется *commands1* и оператор *if* завершается;
3. иначе выполняется *list2* и проверяется его код завершения;
4. если *list2* возвратил 0, то выполняется *commands2* и оператор завершается;
5. если *list2* возвратил не нулевое значение, то выполняется *commands3*.

В качестве условий *list1* и *list2* могут использоваться обычные команды, однако наиболее часто – это вызов одной или нескольких *test* команд в виде *test выражение* или (более кратко) *[выражение]*. Подробно команда *test* рассмотрена в следующем разделе. Ниже приведён пример использования оператора *if*:

```
#!/bin/sh
#выполнение программы с контролем её существования
if [ –x $HOME/script ] ; then $HOME/script ; fi ;
```

В данном примере проверяется, существует ли в домашнем каталоге пользователя файл с именем *script* и является ли он исполняемым (атрибут '*x*'). Если это так, то сценарий запускает *script* на выполнение.

Если необходимо сделать выбор из многих альтернатив, то более удобно использовать оператор *case*, имеющий следующий синтаксис:

```
case значение in
        шаблон1) список1 ;;
        шаблон2) список2 ;;
        ...
esac
```

Оператор *case* является аналогом оператора *switch* в языке Си и работает похожим образом. Строка *значение* сравнивается с каждым из указанных *шаблонов* до тех пор, пока не будет обнаружено соответствие. После обнаружения соответствия выполняется *список*, указанный после шаблона. Два символа *';;'* после списка имеют тот же смысл, что и оператор *break* в программах на языке Си. Если никаких соответствий не обнаружено, то оператор *case* завершается без выполнения каких-либо действий.

Ниже приведён пример использования оператора *case*. Сценарий ожидает в качестве параметра название фрукта и выдаёт его описание. Если при вызове параметр не указан, то выдаётся диагностическое сообщение.

```
#!/bin/sh
#печать информации о фруктах
if [ -z «$1» ]
then
        echo «Ошибка: пропущен параметр»
        exit 1
fi
case "$1" in
        apple) echo "Яблоки очень полезны" ;;
        banana) echo "Бананы весьма вкусны" ;;
        *) echo "Ничего не могу сказать про" $1 ;;
esac
```

2.7. Синтаксис команды test

Команда *test выражение* проверяет указанное *выражение* и заканчивает свою работу с кодом завершения 0 (истина) или 1 (ложь). В качестве *выражение* могут использоваться выражения 3 типов:

- проверки характеристик файлов;
- сравнения строк;
- числовые сравнения.

Основной синтаксис для проверки характеристик файлов следующий: *test опция имя_файла* В приведённой ниже таблице сведены возможные опции команды *test* для проверки характеристик файлов.

Опция	***Описание***
–b	Истина, если файл существует и является блочным специальным файлом.
–c	Истина, если файл существует и является символьным специальным файлом.
–d	Истина, если файл существует и является каталогом.
–e	Истина, если файл существует.
–f	Истина, если файл существует и является обычным файлом.
–h	Истина, если файл существует и является символической ссылкой.
–r	Истина, если файл существует и доступен на чтение.
–s	Истина, если файл существует и имеет ненулевой размер.
-S	Истина, если файл существует и является сокетом
–w	Истина, если файл существует и доступен по записи.
–x	Истина, если файл существует и доступен на выполнение.

В приведённой ниже таблице сведены возможные способы указания

использования команды *test* для проверки строк.

Опция	***Описание***
–z строка	Истина, если строка имеет нулевую длину.
–n строка	Истина, если строка имеет ненулевую длину.
строка1 = строка2	Истина, если строки совпадают.
строка1 != строка2	Истина, если строки различны

При использовании числовых сравнений команда *test* имеет следующий синтаксис: *test число1 оператор число2* В приведённой ниже таблице сведены возможные значения операторов при числовых сравнениях.

Оператор	***Описание***
–eq	Истина, если числа равны.
–ne	Истина, если числа не равны.
–lt	Истина, если меньше.
–le	Истина, если меньше или равны.
–gt	Истина, если больше.
–ge	Истина, если больше или равны.

2.8. Организация циклов

Язык оболочки позволяет организовывать циклическое выполнение команд. В распоряжение пользователю предлагается несколько вариантов циклов:

- *while*
- *for*
- *select*

Оператор цикла *while* имеет следующий синтаксис:

while команда
do
    список
done

При выполнении цикла сначала выполняется *команда*. Если результат ненулевой, то происходит выход из цикла, в противном случае выполняется тело цикла *список* и происходит переход на следующую итерацию. Хотя в качестве условия *команда* может использоваться любая допустимая в GNU/Linux команда, обычно это команда *test*. Ниже приведён пример выдачи на терминал десяти последовательных чисел от 0 до 9.

```
#!/bin/sh
#арифметические вычисления
x=0
while [ $x –lt 10 ] #значение переменной x меньше 10 ?
do
        echo $x
        x=`expr $x + 1` #увеличение x на 1
done
```

Цикл *for* выполняет команды из *список* для каждого элемента из списка и имеет следующий синтаксис:

```
for имя in элемент1 элемент2 ... элементN
do
        список
done
```

В качестве элементов можно использовать различные шаблоны. Рассмотрим пример использования цикла *for*:

```
#!/bin/sh
#выполнение операций над группой файлов
for FILE in $HOME/*.bash
do
        cp $FILE ${HOME}/scripts
        chmod a+r ${HOME}/scripts/${FILE}
done
```

В приведённом примере все файлы из домашнего каталога пользователя, которые заканчиваются на *.bash*, копируются в каталог *scripts* и делаются доступными по чтению всем пользователям.

Язык программирования оболочки содержит операторы, нарушающие нормальное выполнение цикла. Эти операторы имеют названия – *break* и *continue*. Они работают в точности так же, как и их аналоги в языке Си.

Цикл *select* позволяет создавать удобные меню. Он полезен, когда необходимо, чтобы пользователь выбрал один элемент из предлагаемого списка. Оператор *select* имеет такой же вид, как и оператор *for*, за исключением ключевого слова.

При выполнении данного оператора цикла все элементы из списка высвечиваются на экране вместе с их порядковыми номерами, после чего появляется специальное приглашение для ввода. Обычно оно имеет вид *#?*. Введённый пользователем номер пункта меню записывается в переменную *REPLY*. Если *$REPLY* содержит номер пункта меню, то в переменную *имя*

заносится значение соответствующего элемента из списка. В противном случае список будет выдан заново.

После того как пользователем будет сделан допустимый выбор, выполнятся команды из *список*, после чего выполнение цикла повторяется с самого начала (высветится приглашение для ввода и т. д.). Для повторного отображения меню в ответ на приглашение ввода следует нажать клавишу **Enter**. Выход из цикла осуществляется теми же средствами, что и для *while* и *for*.

Ниже приводится пример использования оператора *select*. У пользователя запрашивается тип устройства «мышь», и в зависимости от сделанного выбора выполняются определённые действия. В рассматриваемом случае это просто выдача сообщения, подтверждающая сделанный выбор. В приведённом сценарии обратим внимание на вторую строчку. В большинстве случаев действующий по умолчанию вид приглашения не устраивает. Определяя переменную *PS3*, можно задать необходимый текст приглашения.

```
#!/bin/sh
#«конфигурирование» устройства «мышь»
PS3="Выберите тип устройства «мышь» : "
select ITEM in Microsoft Logitech ps2 none
do
        case $ITEM in
                Microsoft) echo "действия по «мыши» Microsoft " ;;
                Logitech) echo "действия по «мыши» Logitech" ;;
                ps2) echo "действия по «мыши» ps2" ;;
                none) echo "выбрано – нет «мыши»" ;;
        esac
        break
done
```

2.9. Использование функций

В сценариях оболочки допустимо определять и использовать *функции*. Под функцией понимается поименованная группа команд. Во всех случаях, когда в сценарии присутствует повторяющийся код с возможно небольшими вариациями, необходимо рассмотреть возможность применения функции. Так же логически завершённую группу команд удобнее представить отдельной функцией, что позволяет упростить чтение и восприятие сценария.

Определение функции имеет следующий синтаксис:

```
имя_функции ()
{
        список команд
}
```

Определение функции обязательно должно предшествовать её первому использованию. В отличие от языка C++ в языке оболочки bash отсутствуют средства предварительного объявления функции (аналоги прототипов функций). Вызов функции осуществляется путём указания её имени в качестве команды.

Ниже приведён пример сценария, в котором определяется функция *about*, предназначенная для выдачи информации о сценарии, и иллюстрируется её использование:

```
#!/bin/sh
#определение функции
about ()
{
        echo пример сценария с функцией
        echo  (C) Сидоров С.Б., 2007.
}
about          #вызов функции
# ... продолжение сценария
```

Функции при вызове можно передавать аргументы и при своём завершении функция может возвращать результат выполнения. Функция ссылается к переданным параметрам через позиционные переменные - $1, $2 и так далее. Результат работы возвращается с помощью оператора *return*. Ниже приведён пример сценария, в котором иллюстрируется доступ функции к переданным в неё параметрам и вызов функции с передачей параметров.

```
#!/bin/sh
#функция с параметром
 double_echo()
{
        if  [  -n "$1"  ]
        then
                echo $1
```

```
        echo  $1
    fi
    return 0
}
double_echo  параметр      #вызов функции с одним параметром
double_echo                #вызов функции без параметров
# ... продолжение сценария
```

2.10. Отладка сценариев

Один из подходов к отладке сценариев состоит в следующем. Разработчик запускает сценарий на выполнение, просматривает результаты его работы и на основе их анализа принимает решение о работе сценария.

Однако в ряде случаев, например для больших сценариев, или таких, которые изменяют конфигурацию системы, попытка найти источник проблемы по выводу сценария неэффективна. Отладка становится более эффективной при использовании специальных средств. Оболочка обеспечивает несколько встроенных команд для разрешения различных режимов поддержки отладки. Можно выделить два режима отладки:

- проверка синтаксиса;
- трассировка оболочки.

Для разрешения отладки сценарий должен быть запущен специальным образом. Для этого первая строка сценария должна иметь вид:

#!/bin/sh опция

Для отладки не всего сценария целиком, а только некоторого фрагмента, этот фрагмент помечается двумя вызовами команды *set* с указанием у неё требуемой опции из таблицы. При этом для включения режима отладки перед буквой опции указывается знак «–», а для выключения знак «+».

В приведённой ниже таблице сведены возможные значения опции:

Опция	***Описание***
–n	Читать все команды, но не выполнять их.
–v	Отображать все строки по мере их чтения.
–x	Отображать все команды и их аргументы по мере их выполнения

```
#!/bin/sh
#пример отладки сценария
set –x
```

```
if [ –z "$1" ] ; then
        echo "Ошибка: мало аргументов."
        exit 1
fi
set +x
```

Общая методика отладки сценария состоит в том, что прежде, чем запустить его на выполнение, необходимо проверить его синтаксис с помощью опции *–n*. А для большей детализации рекомендуется использовать совокупность ключей *–nv*. На следующем этапе, после устранения синтаксических ошибок, проводится отладка с трассировкой с помощью опции *–x*.

2.11. Основные команды и утилиты

Ниже в таблице сведены основные команды и утилиты, которые могут быть использованы в создаваемых сценариях для решения некоторых стандартных типов подзадач. Для каждой команды приводится её синтаксис, пояснение выполняемого ей действия и перечисляются наиболее важные её опции.

Синтаксис команды	***Краткое описание (опции)***
ls список_каталогов	Выдать содержимое указанных в списке каталогов *–1* расположить каждый файл на отдельной строке *–a* выдать все файлы, включая невидимые *–F* после имени файла указывать символ типа файла (ничего для обычных, / для каталогов и т. д.) *–R* выдать все файлы, включая в подкаталогах (рекурсивно)
cat список_файлов	Выдать содержимое указанных в списке файлов *–n* нумеровать строки *–b* нумеровать строки, кроме пустых
wc список_файлов	Подсчёт числа строк, слов и символов для каждого файла в списке *–l* подсчёт только строк *–w* подсчёт только слов *–c* подсчёт только символов
chmod режим файл	Изменяет права доступа к указанному файлу в соответствии с указанным режимом. Режим – комбинация трёх полей (категория пользователя, операция установки/снятия, вид атрибута) категория пользователя: *u* – владелец, *g* – группа, *o* – все остальные, *a* – все категории операция: + установка атрибута – снятие атрибута

Синтаксис команды	***Краткое описание (опции)***
	вид атрибута: *r* – чтение, *w* – запись, *x* – выполнение
ps	Выдать список запущенных процессов (программ) *–a* отобразить информацию обо всех пользователях *–x* показать информацию о процессах без терминалов *–u* показать дополнительную информацию
grep шаблон файл(ы)	Поиск строк, содержащих заданный шаблон, в указанных файлах *–l* отобразить список файлов, в которых обнаружено совпадение *–n* выдавать в начале номер строки *–r* рекурсивный поиск во всех указанных каталогах *–w* поиск только целых слов (групп символов из букв, цифр и подчёркивания), согласующихся с шаблоном.
tar файл(ы)	Утилита архивирования и сжатия файлов *–c* создание нового архива *–f архив* использовать указанный архив *–j* использовать сжатие через программу *bzip2* *–t* получить список содержимого архива *–x* извлечь файлы из архива *–z* использовать сжатие через программу *gzip*
head файл	Выдать начальную часть файла *– –bytes=N* выдать первые *N* байт *– –lines=N* выдать первые *N* строк *– –lines=–N* выдать все строки, кроме *N* последних
tail файл	Выдать последнюю часть файла *– –bytes=N* выдать последние *N* байт *–f* выдавать добавляемые в файл данные по мере их появления *– –lines=N* выдать последние *N* строк *– –lines=+N* выдать все строки, кроме *N* первых
cut файл	Построчное выделение необходимых полей из файла *–d символ* символ-разделитель полей *–fномера* номера выделяемых полей, перечисленные через запятую
mktemp	Создаёт временный файл с уникальным именем. Имя созданного файла выдаётся на стандартное устройство вывода.
touch имя	Изменение времени доступа и модификации заданного файла или создание нового файла.
sort	Сортировка строк текстовых данных в алфавитном порядке *–n* сортировка в числовом порядке (если в начале строк — числа) *–r* сортировка в обратном порядке
file имя	Определение типа файла (исходный текст программы, исполняемый образ, данные и т.д.)

2.12. Задания и порядок их выполнения

1. Разработайте алгоритм решения задачи, поставленной преподавателем.
2. При старте сценарий должен выдавать информацию об авторе в виде: *(C) имя фамилия, группа, год.*
3. Используя любой текстовый редактор, запишите текст сценария на языке оболочки.
4. Установите необходимые для выполнения сценария атрибуты доступа к созданному файлу.
5. Проведите отладку сценария.
6. Продемонстрируйте преподавателю работу отлаженного варианта сценария.
7. Внесите в сценарий изменения, предложенные преподавателем.
8. Ответьте на контрольные вопросы.

3. Контрольные вопросы

1. Что такое сценарий оболочки?
2. Каково назначение строки *#!/bin/sh* в сценарии оболочки?
3. Перечислите управляющие конструкции языка программирования оболочки bash.
4. Каково назначение команды *test*?
5. Как изменится поведения сценария, иллюстрирующего использование цикла *for* (стр. 12), если в строке 3 в списке элементов убрать символ *$* ?
6. Какие средства отладки сценариев предоставляет разработчику оболочка?
7. Можно ли для отладки сценариев оболочки использовать стандартный отладчик *gdb*. Дайте обоснование ответа.

4. Список литературы

1. Костромин В.А. Основы работы в ОС Linux. – http://www.intuit.ru
2. Курячий Г.В., Маслинский К.А. Операционная система Linux – ИНТУИТ.ру – 2005.
3. Карпов В.Е., Коньков К.А. Основы операционных систем – М.: ИНТУИТ.ру, 2005. – 536 с.
4. Уэлш М., Далхаймер М.К., Кауфман Л. Запускаем Linux. – Пер. с англ. – Спб: Символ-Плюс, 2000.