Лабораторная работа №11

Российский университет дружбы народов

Тимур Андреевич Дарижапов

Содержание

Список таблиц

Список иллюстраций

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

2 Выполнение лабораторной работы

1.Напишем скрипт, который при запуске будет делать резервную копию самого себя в другую директорию backup в нашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Узнаём способ использования командархивации, изучив справку.

Изучаем справку об zip, bzip2 и tar(Рисунок 2.1, 2.2, 2.3, 2.4).

```
tadarizhapov@tadarizhapov-VirtualBox:~$ man zip
tadarizhapov@tadarizhapov-VirtualBox:~$ man bzip2
tadarizhapov@tadarizhapov-VirtualBox:~$ man tar
tadarizhapov@tadarizhapov-VirtualBox:~$
```

Рис. 2.1: Команда man

Синтаксис команды zip для архивации файла: zip [опции] [имя файла.zip] [файлы или папки, которые будем архивировать]

Синтаксис команды zip для разархивации/распаковки файла: unzip [опции] [файл_архива.zip] [файлы] -x [исключить] -d [папка]

```
ZIP(1)                    General Commands Manual                   ZIP(1)

NAME
       zip - package and compress (archive) files

SYNOPSIS
       zip [-aABcdDeEfFghjklLmoqrRSTuvVwXyz!@$] [--longoption ...]  [-b path]
       [-n suffixes] [-t date] [-tt date] [zipfile [file ...]]  [-xi list]
```

Рис. 2.2: zip

Синтаксис команды bzip2 для архивации файла: bzip2 [опции] [имена файлов]

Синтаксис команды bzip2 для разархивации/распаковки файла: bunzip2 [опции] [архивы.bz2]

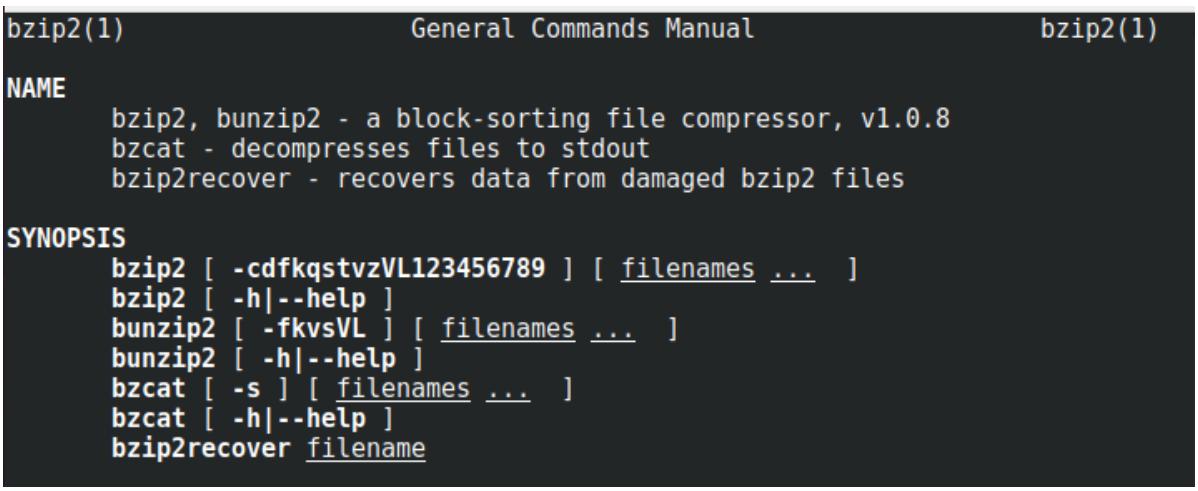

Рис. 2.3: bzip2

Синтаксис команды tar для архивации файла: tar[опции] [архив.tar] [файлы_для_архивации]

Синтаксис команды tar для разархивации/распаковки файла: tar [опции] [архив.tar]

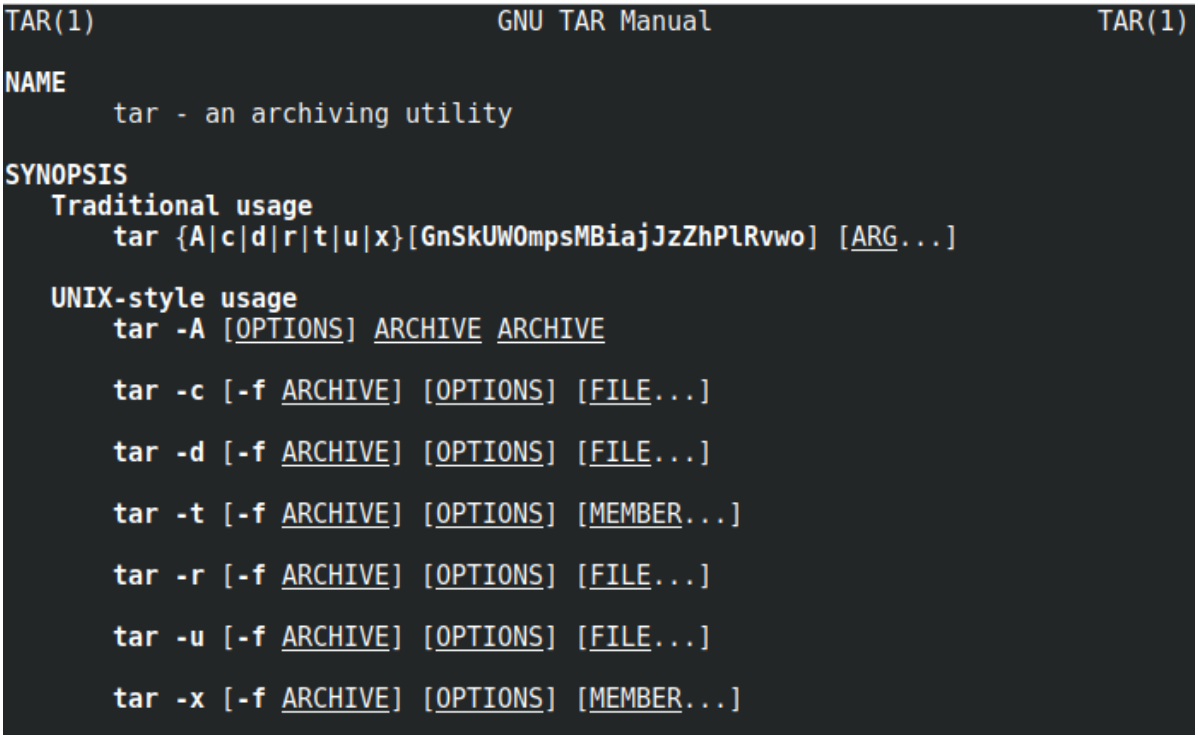

Рис. 2.4: tar

Открывем файл backup.sh через текстовый редактор emacs. Ctrl + x, Ctrl + f(Рисунок 2.5, 2.6).

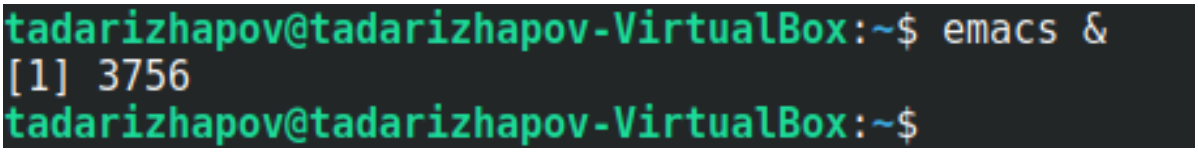

Рис. 2.5: emacs

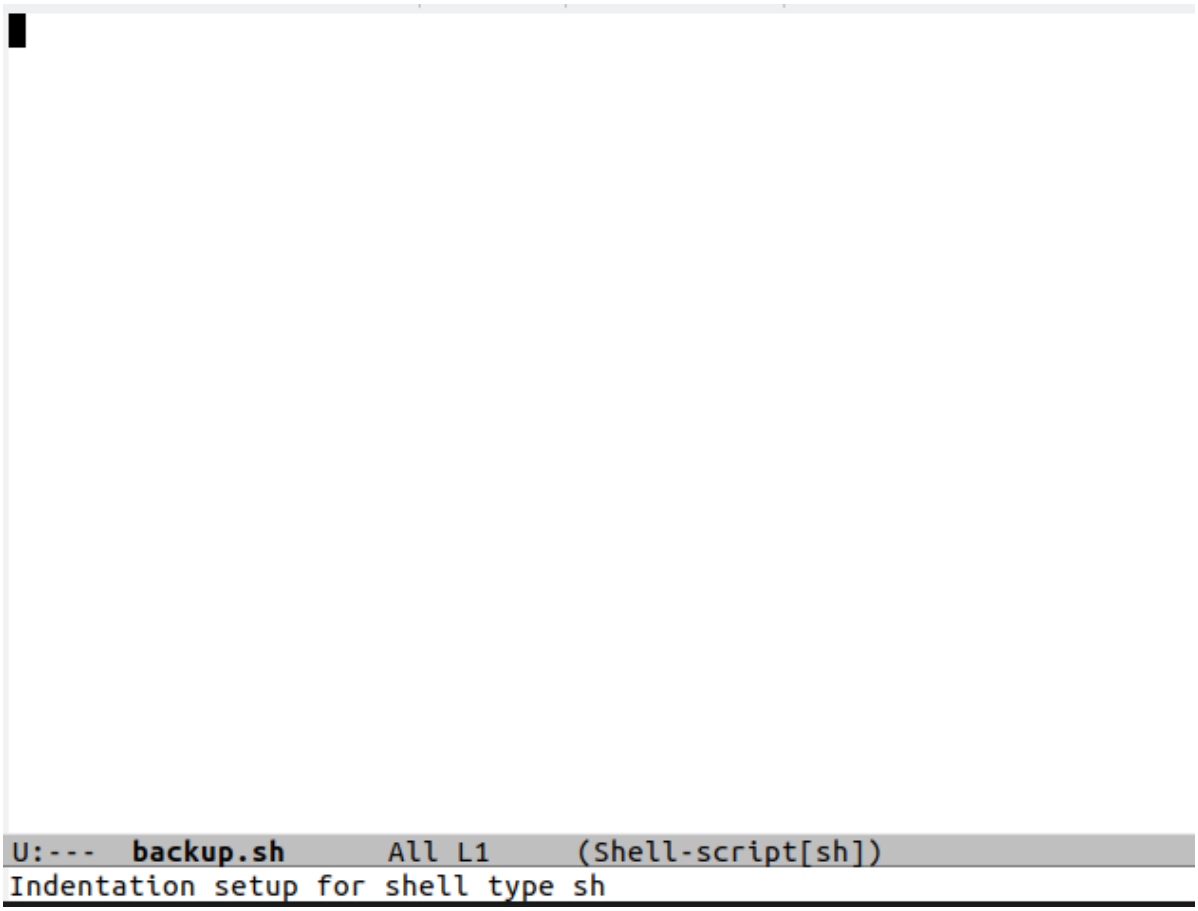

Рис. 2.6: Открытие файла

Пишем скрипт, который при запуске будет делать резервную копию самого себя в другую директорию backup в нашем домашнем каталоге. При написании скрипта я использовал архиватор bzip2(Рисунок 2.7).

```
#!/bin/bash

name='backup.sh'          # name - это файл со скриптом
mkdir ~/backup              # создаём каталог
bzip2 -k ${name}            # архивация
mv ${name}.bz2 ~/backup/    # перемещение в каталог
echo "Выполнено"
```

Рис. 2.7: Скрипт

Проверяем работу скрипта(команда «./backup.sh»), предварительно добавив для него право на выполнение(команда «chmod +x *.sh»). Проверяем, появился ли каталог backup/, перейдя в него(команда «cd backup/»), посмотрим его содержимое(команда «ls») и просмотрим содержимое архива(команда «bunzip2 -c backup.sh.bz2»). Скрипт работает корректно(Рисунок 2.8).

```
tadarizhapov@tadarizhapov-VirtualBox:~$ ls
 1                      pandoc-crossref                work
 backup.sh              pandoc-crossref-Linux.tar.xz   Видео
 dist                   PTSerif-BoldItalic.ttf         Документы
 laboratory             PTSerif-Bold.ttf               Загрузки
 labs                   PTSerif-Italic.ttf             Изображения
 LM_Sans_10             PTSerif-Regular.ttf            Музыка
 lmsans10-regular.otf   q.log                          Общедоступные
 OFL.txt                snap                          'Рабочий стол'
 os.txt                 text.txt                       Шаблоны
tadarizhapov@tadarizhapov-VirtualBox:~$ chmod +x *.sh
tadarizhapov@tadarizhapov-VirtualBox:~$ ./backup.sh
Выполнено
tadarizhapov@tadarizhapov-VirtualBox:~$ cd backup/
tadarizhapov@tadarizhapov-VirtualBox:~/backup$ ls
backup.sh.bz2
tadarizhapov@tadarizhapov-VirtualBox:~/backup$ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'          # name - это файл со скриптом
mkdir ~/backup              # создаём каталог
bzip2 -k ${name}            # архивация
mv ${name}.bz2 ~/backup/    # перемещение в каталог
echo "Выполнено"
tadarizhapov@tadarizhapov-VirtualBox:~/backup$
```

Рис. 2.8: Проверка

2.Напишем пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.

Создаём файл, в котором будем писать второй скрипт, и открываем его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f»(Рисунок 2.9).

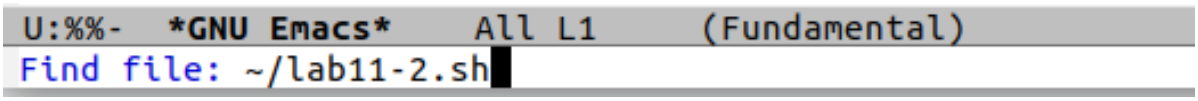

Рис. 2.9: emacs

Пишем пример командного файла(Рисунок 2.10).

```
#!/bin/bash

echo "Аргументы"
for a in $@      # цикл для прохода по введённым аргументам
do echo $a       # вывод аргумента
done
```

Рис. 2.10: Командный файл

Проверяем работу написанного скрипта(команды «./lab11-2.sh 3 2 1 3» и «./lab11-2.sh 1 2 3 4 5 6 7 8 9 10 11»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»). Вводим аргументы, количество которых меньше 10 и больше 10. Скрипт работает корректно(Рисунок 2.11).

```
tadarizhapov@tadarizhapov-VirtualBox:~$ chmod +x *.sh
tadarizhapov@tadarizhapov-VirtualBox:~$ ls
 1            lmsans10-regular.otf          PTSerif-Regular.ttf   Изображения
 backup       OFL.txt                       q.log                 Музыка
 backup.sh    os.txt                        snap                  Общедоступные
 dist         pandoc-crossref               text.txt             'Рабочий стол'
 lab11-2.sh   pandoc-crossref-Linux.tar.xz  work                  Шаблоны
 laboratory   PTSerif-BoldItalic.ttf        Видео
 labs         PTSerif-Bold.ttf              Документы
 LM_Sans_10   PTSerif-Italic.ttf            Загрузки
tadarizhapov@tadarizhapov-VirtualBox:~$ ./lab11-2.sh 3 2 1 4
Аргументы
3
2
1
4
tadarizhapov@tadarizhapov-VirtualBox:~$ ./lab11-2.sh 1 2 3 4 5 6 7 8 9 10 11
Аргументы
1
2
3
4
5
6
7
8
9
10
11
tadarizhapov@tadarizhapov-VirtualBox:~$
```

Рис. 2.11: Проверка

3.Напишем командный файл — аналог команды ls(без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога(Рисунок 2.12).

Создаём файл, в котором будем писать третий скрипт, и открываем его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f».

```
U:%%-  *GNU Emacs*    All L1     (Fundamental)
Find file: ~/lab11-3.sh
```

Рис. 2.12: emacs

Пишем пример командного файла(Рисунок 2.13, 2.14).

```
#!/bin/bash
a="$1"
for i in ${a}/*
do
    echo "$i"

    if test -f $i
    then echo "Обычный файл"
    fi

    if test -d $i
    then echo "Каталог"
    fi

    if test -r $i
    then echo "Чтение разрешено"
    fi
```

Рис. 2.13: Командный файл 1

```
    if test -w $i
    then echo "Запись разрешена"
    fi

    if test -x $i
    then echo "Выполнение разрешено"
    fi

done
```

Рис. 2.14: Командный файл 2

Далее проверяем работу скрипта(команда «./lab11-3.sh ~»), предварительно добавив для него право на выполнение(команда «chmod +x *.sh»). Скрипт работает корректно(Рисунок 2.15).

```
tadarizhapov@tadarizhapov-VirtualBox:~$ chmod +x *.sh
tadarizhapov@tadarizhapov-VirtualBox:~$ ls
 1            LM_Sans_10                  PTSerif-Italic.ttf    Загрузки
 backup       lmsans10-regular.otf        PTSerif-Regular.ttf   Изображения
 backup.sh    OFL.txt                     q.log                 Музыка
 dist         os.txt                      snap                  Общедоступные
 lab11-2.sh   pandoc-crossref            text.txt              'Рабочий стол'
 lab11-3.sh   pandoc-crossref-Linux.tar.xz work                  Шаблоны
 laboratory   PTSerif-BoldItalic.ttf      Видео
 labs         PTSerif-Bold.ttf            Документы
tadarizhapov@tadarizhapov-VirtualBox:~$ ./lab11-3.sh
/bin
Каталог
Чтение разрешено
Выполнение разрешено
/boot
Каталог
Чтение разрешено
Выполнение разрешено
/cdrom
Каталог
Чтение разрешено
Выполнение разрешено
```

Рис. 2.15: Проверка

4.Напишем командный файл, который получает в качестве аргумента командной строки формат файла(.txt,.doc,.jpg,.pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.

Для четвертого скрипта создаём файл и открываем его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f»(Рисунок 2.16).

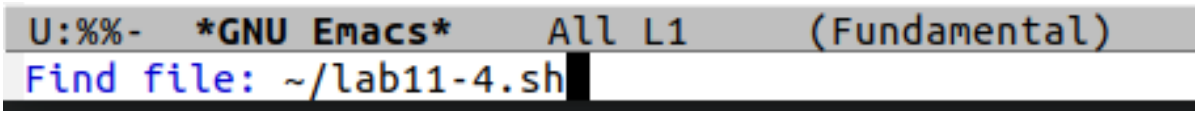

Рис. 2.16: emacs

Пишем пример командного файла(Рисунок 2.17).

```
#!/bin/bash
b="$1"
shift
for a in $@
do
    k=0
    for i in ${b}/*.${a}
    do
        if test -f "$i"
        then
            let k=k+1
        fi
    done
    echo "$k файлов содержится в каталоге $b с расширением $a"
done
```

Рис. 2.17: Командный файл

Проверяем работу написанного скрипта(команда «./format.sh ~ pdf sh txt doc»),предварительно добавив для него право на выполнение (команда «chmod +x *.sh»), а также создав дополнительные файлы с разными расширениями (команда «touch 1.pdf 2.doc»). Скрипт работает корректно(Рисунок 2.18).

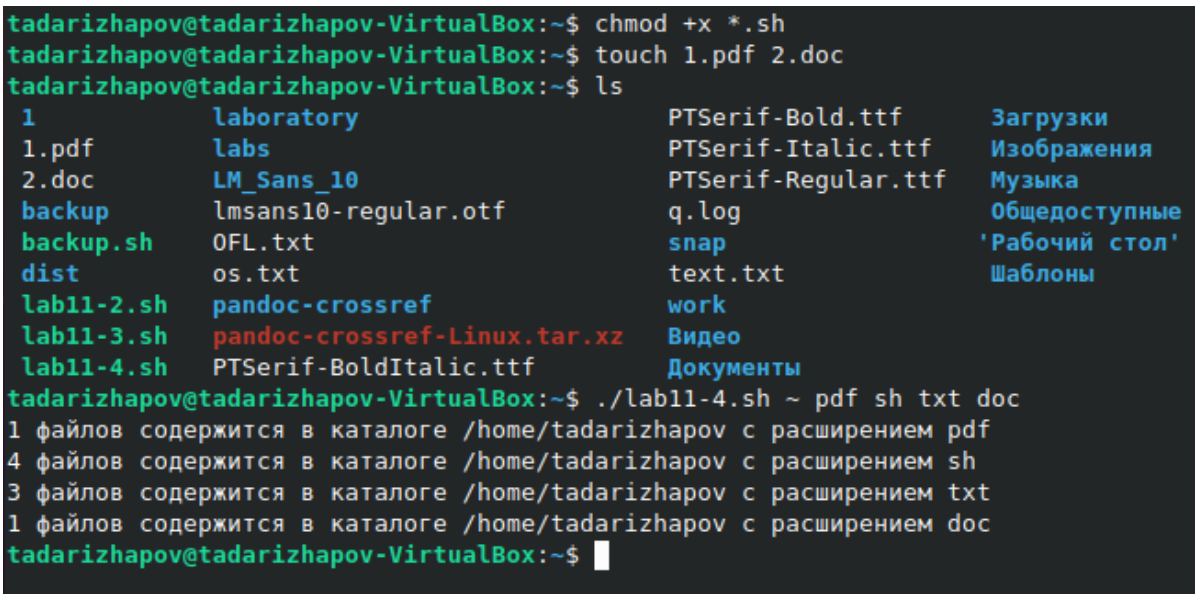

Рис. 2.18: Проверка

3 Выводы

Я изучил основы программирования в оболочке ОС UNIX/Linux. Я научился писать небольшие командные файлы.

4 Ответы на контрольные вопросы

1.Командный процессор (командная оболочка, интерпретатор команд shell) – это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: оболочка Борна (Bourneshell или sh) – стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; C-оболочка (или csh) – надстройка на оболочкой Борна, использующая С подобный синтаксис команд с возможностью сохранения истории выполнения команд; оболочка Корна (или ksh) – напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна; BASH–сокращение от BourneAgainShell(опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании FreeSoftwareFoundation).

2.POSIX(Portable Operating System Interface for Computer Environments) – набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE(Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linuxподобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX -совместимые оболочки разработаны на базе оболочки Корна.

3.Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение

некоторой строки символов. Например, команда «mark=/usr/andy/bin» присваивает значение строки символов /usr/andy/bin переменной mark типа строка символов. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол $. Например, команда «mv afile ${mark}» переместит файл afile из текущего каталога в каталог с абсолютным полным именем /usr/andy/bin. Оболочка bash позволяет работать с массивами. Для создания массива используется команда setc флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, «set -A states Delaware Michigan “NewJersey”» Далее можно сделать добавление в массив, например, states[49]=Alaska. Индексация массивов начинается с нулевого элемента.

4.Оболочка bash поддерживает встроенные арифметические функции. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение – это единичный терм (term), обычно целочисленный.Команда letберет два операнда и присваивает их переменной.Команда readпозволяет читать значения переменных со стандартного ввода: «echo “Please enter Month and Day of Birth ?”»«read mon day trash»В переменные mon и day будут считаны соответствующие значения, введённые с клавиатуры, а переменная trash нужна для того, чтобы отобрать всю избыточно введённую информацию и игнорировать её.

5.В языке программирования bashможно применять такие арифметические операции как сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%).

6.В (( ))можно записывать условия оболочки bash, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.

7.Стандартные переменные: PATH: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное

имя программы или команды не содержит ни одного символа /. Если имя команды содержит хотя бы один символ /, то последовательность поиска, предписываемая значением переменной PATH, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневогоили текущего каталога. PS1 и PS2: эти переменные предназначены для отображения промптера командного процессора. PS1 – это промптер командного процессора, по умолчанию его значение равно символу $ или #. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, то используется промптер PS2. Он по умолчанию имеет значение символа >. HOME: имя домашнего каталога пользователя. Если команда cdвводится без аргументов, то происходит переход в каталог,указанный в этой переменной. IFS:последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (newline). MAIL:командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение Youhavemail(у Вас есть почта). TERM: тип используемого терминала. LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.

8.Такие символы, как ’ < > * ? | ” &, являются метасимволами и имеют для командного процессора специальный смысл.

9.Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа , который, в свою очередь, является метасимволом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме $, ’ , , ”. Например, –echo* выведет на экран символ *, –echoab’|’cdвыведет на экран строку ab*|cd.

10.Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: «bash командный_файл [аргументы]»Чтобы не вводить каждый раз последовательности символов bash, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды «chmod +x имя_файла»Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как будтоон является выполняемой программой. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществит её интерпретацию.

11.Группу команд можно объединить в функцию. Для этого существует ключевое слово function, после которого следует имя функции и список команд, заключённых в фигурные скобки. Удалить функцию можно с помощью команды unsetсфлагом -f.

12.Чтобы выяснить, является ли файл каталогом или обычным файлом, необходимо воспользоваться командами «test -f [путь до файла]» (для проверки, является ли обычным файлом) и «test -d[путь до файла]» (для проверки, является ли каталогом).

13.Команду «set» можно использовать для вывода списка переменных окружения. В системах Ubuntu и Debian команда «set» также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду «set| more». Команда «typeset» предназначена для наложения ограничений на переменные.Команду «unset» следует использовать для удаления переменной из окружения командной оболочки.

14.При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. С точки

зрения командного файла эти параметры являются позиционными. Символ $ является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов $i, где 0 < i < 10, вместо неё будет осуществлена подстановка значения параметра с порядковым номером i, т.е. аргумента командного файла с порядковым номером i. Использование комбинации символов $0 приводит к подстановке вместо неё имени данного командного файла.

15.Специальные переменные: $* –отображается вся командная строка или параметры оболочки; $? –код завершения последней выполненной команды; $$ – уникальный идентификатор процесса, в рамках которого выполняется командный процессор; $! –номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда; — — ;*{#} –возвращает целое число –количество слов, которые были результатом $*; ${#name} –возвращает целое значение длины строки в переменной name; ${name[n]} –обращение к n-му элементу массива; ${name[*]}–перечисляет все элементы массива, разделённые пробелом; ${name[@]}–то же самое, но позволяет учитывать символы пробелы в самих переменных; ${name:-value} – если значение переменной name не определено, то оно будет заменено на указанное value; ${name:value} –проверяется факт существования переменной; ${name=value} –если name не определено, то ему присваивается значение value; ${name?value} –останавливает выполнение, если имя переменной не определено, и выводит value как сообщение об ошибке; ${name+value} – это выражение работает противоположно ${name-value}. Если переменная определена, то подставляется value; ${name#pattern} –представляет значение переменной name с удалённым самым коротким левым образцом (pattern); ${#name[*]} и ${#name[@]} – эти выражения возвращают количество элементов в массиве name.