

отчёта по лабораторной работе 11

Операционные системы

Саинт Амур Измаэль

Содержание

0.1	Цель работы:	4
-----	------------------------	---

List of Tables

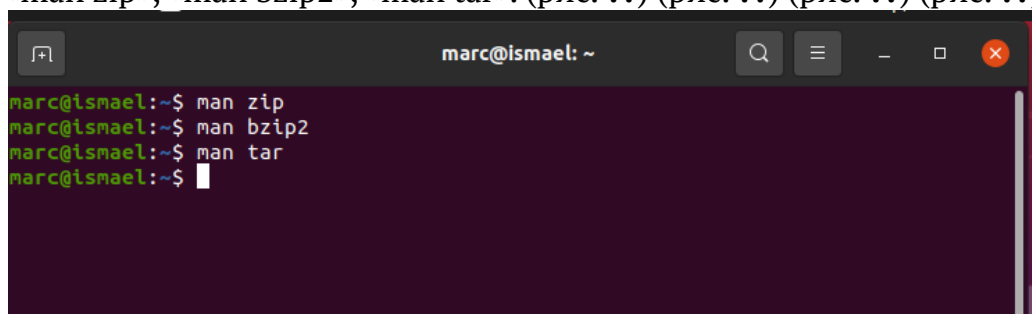
List of Figures

0.1 Цель работы:

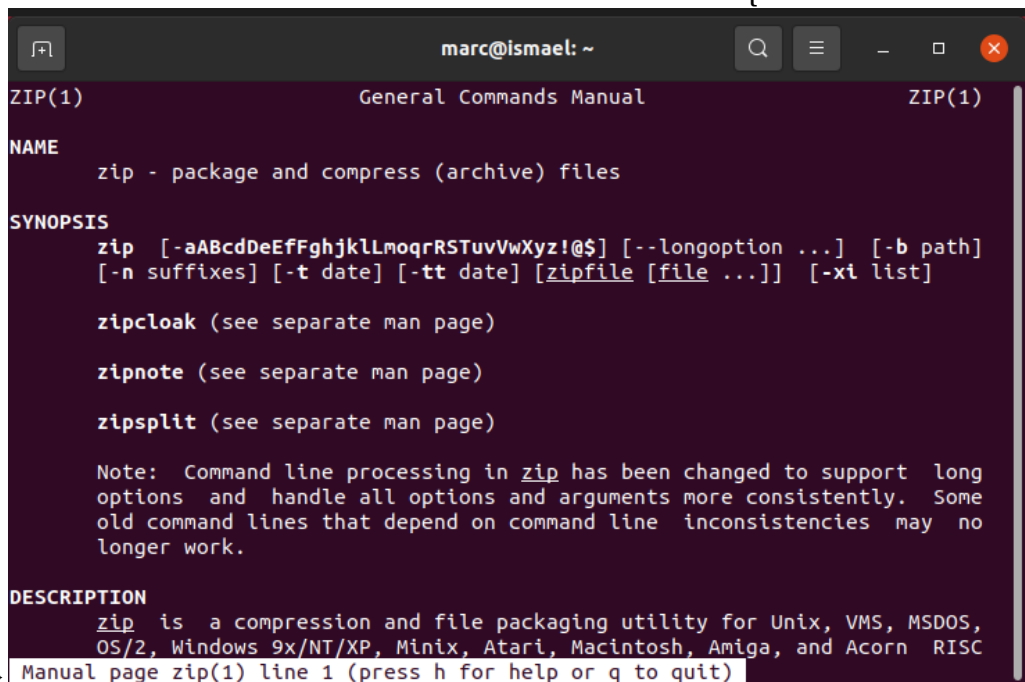
Изучить основы программирования в оболочке ОС UNIX/Linux.

Научиться писать небольшие командные файлы. ## Ход работы:

1. Для начала я изучила команды архивации, используя команды «man zip», «man bzip2», «man tar». (рис. ??) (рис. ??) (рис. ??) (рис. ??)



```
marc@ismael: ~  
marc@ismael:~$ man zip  
marc@ismael:~$ man bzip2  
marc@ismael:~$ man tar  
marc@ismael:~$
```



```
marc@ismael: ~  
ZIP(1)                                General Commands Manual                                ZIP(1)  
  
NAME  
    zip - package and compress (archive) files  
  
SYNOPSIS  
    zip [-aABcdDeEfFghjklLnoqrRSTuvVwXyz!@$] [--longoption ...] [-b path]  
    [-n suffixes] [-t date] [-tt date] [zipfile [file ...]] [-xi list]  
  
    zipcloak (see separate man page)  
    zipnote (see separate man page)  
    zipsplit (see separate man page)  
  
Note: Command line processing in zip has been changed to support long  
options and handle all options and arguments more consistently. Some  
old command lines that depend on command line inconsistencies may no  
longer work.  
  
DESCRIPTION  
    zip is a compression and file packaging utility for Unix, VMS, MSDOS,  
    OS/2, Windows 9x/NT/XP, Minix, Atari, Macintosh, Amiga, and Acorn RISC  
    Manual page zip(1) line 1 (press h for help or q to quit)
```

fig:001 width=70%}

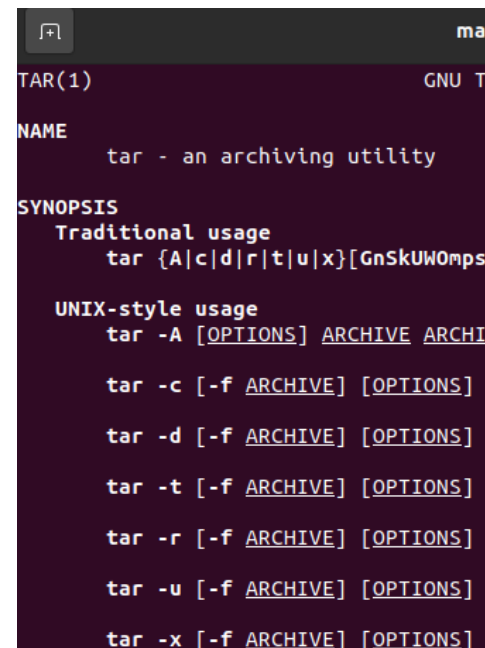


fig:002 width=70%} [3]](image/3.png){ # fig:003 width=70%} Manual page tar(1) line 1 (press h f

fig:004 width=70%} - Далее я создал файл, в котором буду писать первый скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch backup.sh» и «emacs &»)(рис. ??)

```
marc@ismael:~$ touch backup.sh
marc@ismael:~$ emacs &
```

fig:005 width=70%} - После написал скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. (рис. ??)

```
emacs@ismael
File Edit Options Buffers Tools Sh-Script Help
+ Save Undo
#!/bin/bash
name='backup.sh'           # В переменную name файл со скриптом
mkdir ~/backup             #создаем каталог ~/backup
bzip2 -k ${name}           # Архивируем скрипт
mv ${name}.bz2 ~/backup/   # перемещаем Архивированный
echo "выполнено"
```

fig:006 width=70%} - Проверил работу скрипта (команда «./backup.sh»), предварительно добавив для него право на выполнение (команда «chmod +x .sh»). Проверила, появился ли каталог backup/, перейдя в него (команда «cd backup/»), посмотрела его содержимое (команда «ls») и просмотрел содержимое архива (команда «bunzip2 -c backup.sh.bz2»). (рис. ??)

```
marc@ismael:~$ ls
01.sh~      example1.txt~  '#exp4.txt#'  monthly      ski.plases
abc1        example2.txt~  exp4.txt      montly.00    snap
abcd1       example3.txt~  Gmail         Music        Templates
australia   example4.txt~  home          my_os        text.txt
backup.sh   exp1.txt       Lab03         newdir       Videos
backup.sh~  exp1.txt~     Lab05         Pictures     windows
Desktop     '#exp2.txt#'  Lab10         play         work
Documents   exp2.txt      Lab3.pdf      Presentations
Downloads   '#exp3.txt#'  Lab5          Public
'#example1.txt#' exp3.txt      may~         reports
```

??) (рис. ??)

```
marc@ismael:~$ chmod +x *.sh
marc@ismael:~$ ./backup.sh
выполнено
marc@ismael:~$ cd backup/
marc@ismael:~/backup$ ls
backup.sh.bz2
marc@ismael:~/backup$ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'      # В переменную name файл со скриптом
mkdir ~/backup        # создаем каталог ~/backup
bzip2 -k ${name}       # Архивируем скрипт
mv ${name}.bz2 ~/backup/ # перемещаем Архивированный
echo "выполнено"
```

fig:007 width=70%}

fig:008 width=70%} 2. Создал файл, в котором буду писать второй скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch prog2.sh» и «emacs») (рис. ??)

```
marc@ismael:~$ touch prog2.sh
marc@ismael:~$ emacs
```

fig:009 width=70%} - Написал пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов. (рис. ??)

```

emacs@ismael
File Edit Options Buffers Tools Sh-Script Help
[Icons: New, Open, Close, Save, Undo, Cut, Copy, Paste, Find]

#!/bin/bash
echo "Аргументы"
for a in $@ # Цикл для прохода по введенным аргументам
do echo $a # Вывод аргумента
done

```

fig:0010 width=70%} - Проверил работу написанного скрипта (команды «./prog2.sh 0 1 2 3 4 5» и «./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15»), предварительно добавив для него право на выполнение (команда «chmod +x .sh»). Вводил аргументы, количество которых меньше 10 и больше 10. Скрипт работает корректно. (рис. ??) (рис. ??) (рис. ??)

```

marc@ismael:~$ chmod +x *.sh
marc@ismael:~$ ls
01.sh~      '#example1.txt#'  exp3.txt      may~         prog2.sh~
abc1        example1.txt~     '#exp4.txt#'  monthly      Public
abcd1       example2.txt~     exp4.txt      montly.00    reports
australia   example3.txt~     Gmail         Music        ski.plases
backup      example4.txt~     home          my_os        snap
backup.sh   exp1.txt          Lab03         newdir       Templates
backup.sh~  exp1.txt~         Lab05         Pictures     text.txt
Desktop     '#exp2.txt#'      Lab10         play         Videos
Documents   exp2.txt          Lab3.pdf      Presentations
Downloads   '#exp3.txt#'      Lab5          prog2.sh     work

```

```

marc@ismael:~$ ./prog2.sh 0 1 2 3 4 5
Аргументы
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5

```

fig:0011 width=70%}

```

marc@ismael:~$ ./prog2.sh 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Аргументы
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

```

```
# fig:0012width=70%}marc@ismael:~$ █
```

fig:0013 width=70%} 3. Создал файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch prog.sh» и «emacs »). (рис. ??)

```

marc@ismael:~$ touch prog.sh
marc@ismael:~$ emacs █

```

fig:0014width=70%} - Написал командный файл – аналог команды ls (без использования самой этой команды и команды dir). Он должен выдавать информацию о нужном каталоге и выводить информацию о возможностях доступа к файлам этого каталога.(рис. ??) (рис. ??)


```

emacs@ismael
File Edit Options Buffers Tools Sh-Script Help
#1/bin/bash
a="$1"
for i in $(ls $a)/*
do
    echo "$i"
    # В переменную a сохраняем путь до заданного каталога
    # Цикл, который пройдет по всем каталогам и файлам в каталоге $a
    # Вывод названия каталога или файла $i

    if test -f $i
    then echo "Обычный файл"
    fi
    # Проверим, является ли файл $i обычным файлом
    # Если, да то Выводим соответствующее сообщение

    if test -d $i
    then echo "Каталог"
    fi
    # Проверим, является ли файл $i каталогом
    # Если, да то Выводим соответствующее сообщение

    if test -r $i
    then echo "Чтение разрешено"
    fi
    # Проверим право на чтение каталога или файла $i
    # Если, да то Выводим соответствующее сообщение

    if test -w $i
    then echo "Запись разрешена"
    fi
    # Проверим право на изменение/запись каталога или файла $i
    # Если, да то Выводим соответствующее сообщение

    if test -x $i
    then echo "Выполнение разрешено"
    fi
    # Проверим право на выполнение каталога или файла $i
    # Если, да то Выводим соответствующее сообщение

```

```

emacs@ismael
File Edit Options Buffers Tools Sh-Script Help
if test -f $i
then echo "Обычный файл"
fi
# Проверим, является ли файл $i обычным файлом
# Если, да то Выводим соответствующее сообщение

if test -d $i
then echo "Каталог"
fi
# Проверим, является ли файл $i каталогом
# Если, да то Выводим соответствующее сообщение

if test -r $i
then echo "Чтение разрешено"
fi
# Проверим право на чтение каталога или файла $i
# Если, да то Выводим соответствующее сообщение

if test -w $i
then echo "Запись разрешена"
fi
# Проверим право на изменение/запись каталога или файла $i
# Если, да то Выводим соответствующее сообщение

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

```

fig:0015width=70%}

fig:0016width=70%} - Далее проверил работу скрипта (команда «./prog.sh ~»), предварительно добавив для него право на выполнение (команда «chmod +x .sh») Скрипт работает корректно.(рис. ??) (рис. ??)

```

marc@ismael:~$ chmod +x *.sh
marc@ismael:~$ ls
01.sh~          example1.txt~  exp4.txt       Music           reports
abc1            example2.txt~  Gmail          my_os          ski.plases
abcd1          example3.txt~  home           newdir         snap
australia      example4.txt~  Lab03         Pictures       Templates
backup         exp1.txt       Lab05         play          text.txt
backup.sh      exp1.txt~     Lab10        Presentations  Videos
backup.sh~    '#exp2.txt#'  Lab3.pdf     prog2.sh      windows
Desktop       exp2.txt      Lab5         prog2.sh~     work
Documents     '#exp3.txt#'  may~         prog.sh
Downloads     exp3.txt      monthly      prog.sh~
'#example1.txt#' '#exp4.txt#'  montly.00    Public

```

(рис. ??)

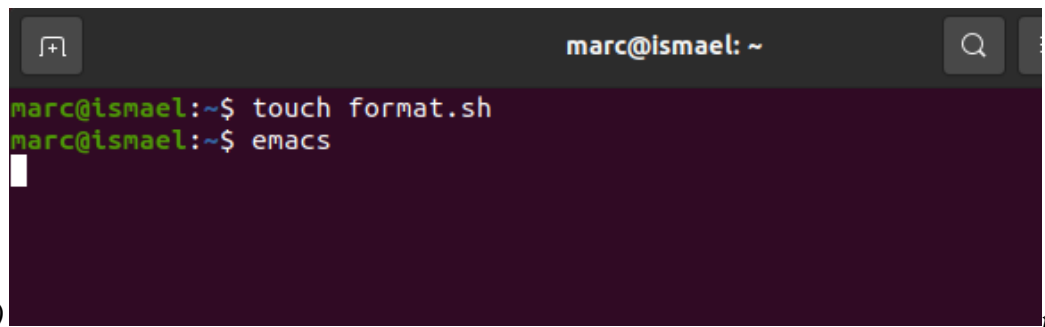
fig:0017width=70%}

```
marc@ismael: ~  
marc@ismael:~$ ./prog.sh ~  
/home/marc/01.sh~  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/marc/abc1  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/marc/abcd1  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/marc/australia  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/marc/backup  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/marc/backup.sh
```

fig:0018width=70%}

```
marc@ismael: ~  
/home/marc/backup.sh  
Обычный файл  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/marc/backup.sh~  
Обычный файл  
Чтение разрешено  
Запись разрешена  
/home/marc/Desktop  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/marc/Documents  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/marc/Downloads  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено
```

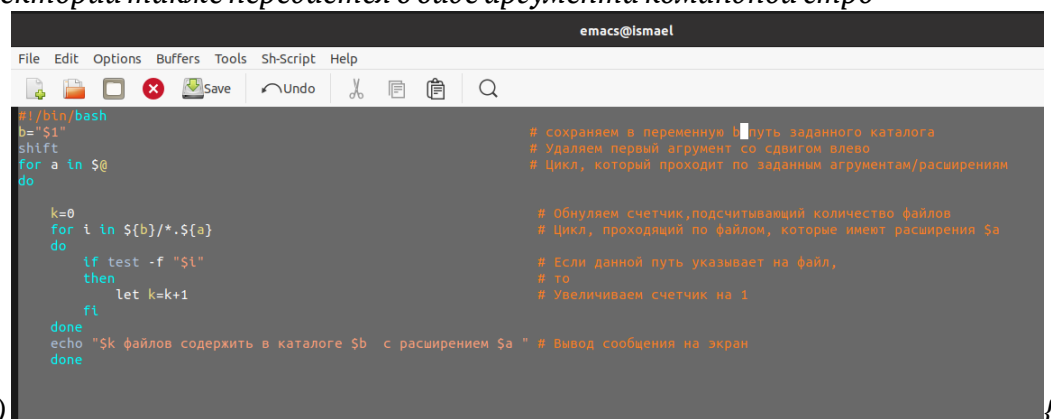
fig:0019width=70%} 4. создал файл (команда «touch format.sh») и от-
крыл его в редакторе emacs, используя клавиши «Ctrlx» и «Ctrl-f» (команда



```
marc@ismael: ~  
marc@ismael:~$ touch format.sh  
marc@ismael:~$ emacs
```

«emacs») (рис. ??)

fig:0020width=70%} - Написал 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 стро-



```
emacs@ismael  
File Edit Options Buffers Tools Sh-Script Help  
[Icons]  
#!/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 содержат в каталоге $b с расширением $a" # Вывод сообщения на экран  
done
```

ки.(рис. ??)

fig:0021width=70%} - Проверил 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 file.pdf file1.doc file2.doc») Скрипт работает корректно.(рис. ??)(рис. ??)

```
marc@ismael:~$ chmod +x *.sh
marc@ismael:~$ ls
01.sh~          example1.txt~  exp4.txt      monthly      prog.sh~
abc1            example2.txt~  format.sh     montly.00    Public
abcd1          example3.txt~  format.sh~    Music        reports
australia      example4.txt~  Gmail         my_os        ski.plases
backup         exp1.txt       home          newdir       snap
backup.sh      exp1.txt~     Lab03         Pictures     Templates
backup.sh~    '#exp2.txt#'  Lab05         play         text.txt
Desktop        exp2.txt      Lab10         Presentations Videos
Documents      '#exp3.txt#'  Lab3.pdf      prog2.sh     windows
Downloads      exp3.txt      Lab5          prog2.sh~    work
'#example1.txt#' '#exp4.txt#'  may~         prog.sh
```

```
marc@ismael:~$ ./format.sh ~ txt doc pdf
5 файлов содержат в каталоге /home/marc с расширением txt
0 файлов содержат в каталоге /home/marc с расширением doc
1 файлов содержат в каталоге /home/marc с расширением pdf
marc@ismael:~$
```

fig:0022width=70%}

fig:0023width=70%} **Вывод:** В ходе выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux и научилась писать небольшие командные файлы. **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 (Bourne shell или sh) – стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; - C-оболочка (или csh) – надстройка на оболочкой Борна, использующая Сподобный 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 Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании Free Software Foundation). 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 `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, `set -A states Delaware Michigan "New Jersey"`. 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 `let` берет два операнда и присваивает их переменной. Команда `read` позволяет 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, введенные с клавиатуры, а переменная trash нужна для того, чтобы отобразить всю избыточно введенную информацию и игнорировать её. 5. В языке программирования bash можно 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 корневого или 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 cd вводится 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 (new line). - 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 *You have mail* (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 , – *echo ab'|'cd* выведет 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 будто он 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 *unset* с флагом *-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 *гделибо* 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; - `$!` – номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда; - `$-` – значение флагов командного процессора; - `${#}` – *возвращает целое число – количество слов, которые были результатом `$`*; - `${#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`