Отчет по лабораторной работе №11

Дисциплина

Филиппова Анна Дмитриевна

Содержание

Список иллюстраций

1 Цель работы

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

2 Выполнение лабораторной работы

1. Изучаем команды архивации, используя команды «man zip», «man bzip2», «man tar». (рис. -fig. 2.1) (рис. -fig. 2.2) (рис. -fig. 2.3) (рис. -fig. 2.4)

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] Синтаксис команды bzip2 для архивации файла: bzip2 [опции] [имена файлов] Синтаксис команды tar для архивации файла: tar [опции] [архив.tar] [файлы_для_архивации] Синтаксис команды tar для разархивации/распаковки файла: tar [опции] [архив.tar]

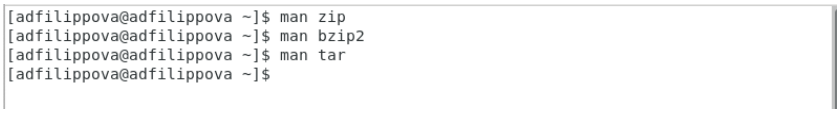

Рис. 2.1: Команда man

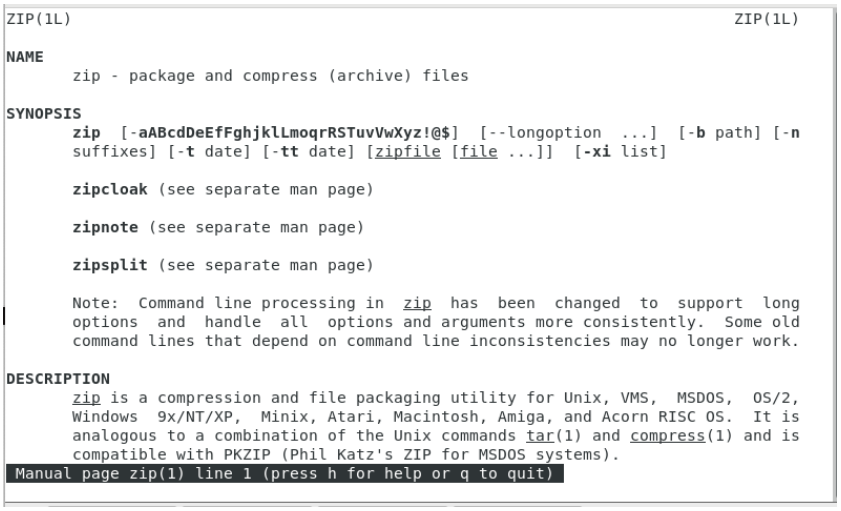

Рис. 2.2: Справка о команде zip

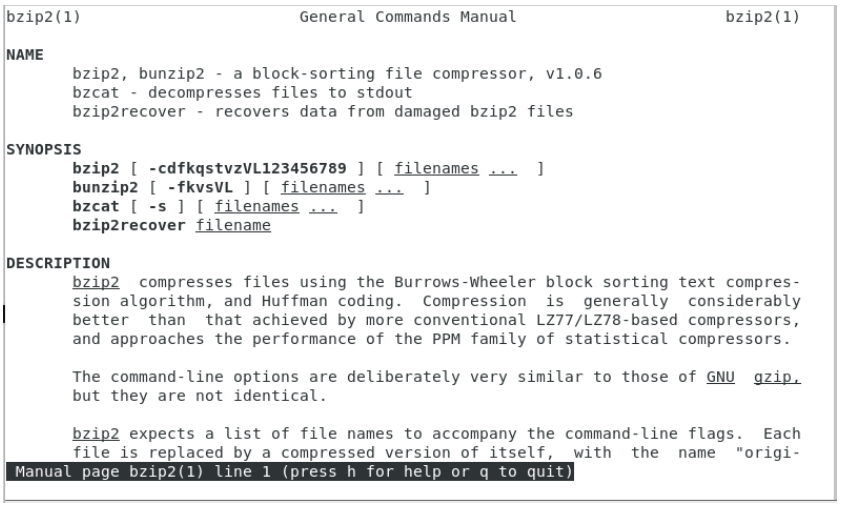

Рис. 2.3: Справка о команде bzip2

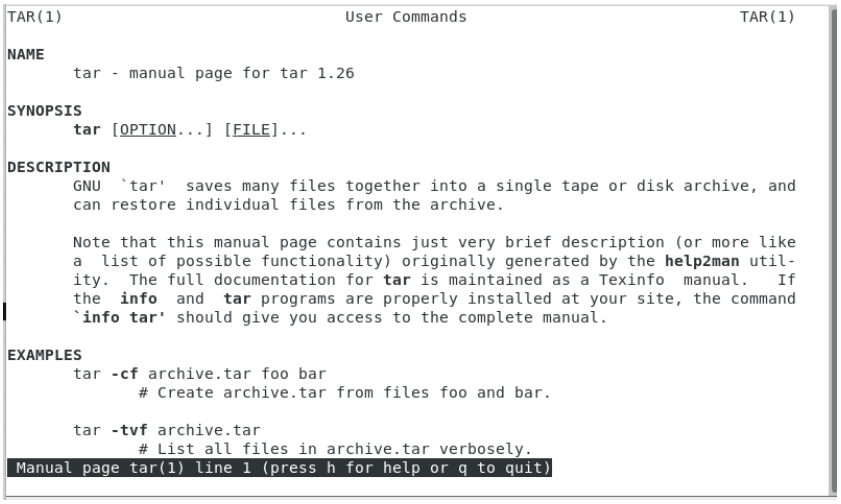

Рис. 2.4: Справка о команде tar

Создаем файл, в котором будем писать первый скрипт, и открываем его в редакторе emacs (команды «touch backup.sh» и «emacs &»).(рис. -fig. 2.5)

Рис. 2.5: Создание файла

Пишем скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в нашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. При написании скрипта используем архиватор bzip2. (рис. -fig. 2.6)

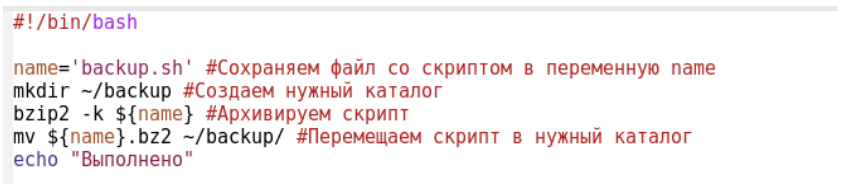

Рис. 2.6: Написание скрипта

Проверяем работу скрипта (команда «./backup.sh»), перед этим добавив для него право на выполнение (команда «chmod +x *.sh»). Проверяем, появился ли

каталог backup/, перейдя в него (команда «cd backup/»), просмотриваем содержимое архива (команда «bunzip2 -c backup.sh.bz2»). Скрипт работает корректно.(рис. -fig. 2.7)

```
[adfilippova@adfilippova ~]$ chmod +x *.sh
[adfilippova@adfilippova ~]$ ./backup.sh
Выполнено
[adfilippova@adfilippova ~]$ cd backup/
[adfilippova@adfilippova backup]$ ls
backup.sh.bz2
[adfilippova@adfilippova backup]$ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh' #Сохраняем файл со скриптом в переменную name
mkdir ~/backup #Создаем нужный каталог
bzip2 -k ${name} #Архивируем скрипт
mv ${name}.bz2 ~/backup/ #Перемещаем скрипт в нужный каталог
echo "Выполнено"
[adfilippova@adfilippova backup]$
```

Рис. 2.7: Проверяем работу скрипта

2. Создаем файл, в котором буду писать второй скрипт, и открываем его в редакторе emacs (команды «touch file.sh» и «emacs &»). (рис. -fig. 2.8)

```
[adfilippova@adfilippova ~]$ touch file.sh
[adfilippova@adfilippova ~]$ emacs &
```

Рис. 2.8: Создание файла

Пишем 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. -fig. 2.9)

```
#!/bin/bash
echo "Аргументы"
for a in $@  # цикл для передвижения по аргументам
do echo $a   # вывод аргумента
done
```

Рис. 2.9: Написание скрипта

Проверила работу написанного скрипта (команды «./file.sh 1 2 3 4 5 6» и «./file.sh 1 2 3 4 5 6 7 8 9 10 11 12»), предварительно добавив для него право на выполнение

(команда «chmod +x *.sh»).Скрипт работает корректно. (рис. -fig. 2.10)

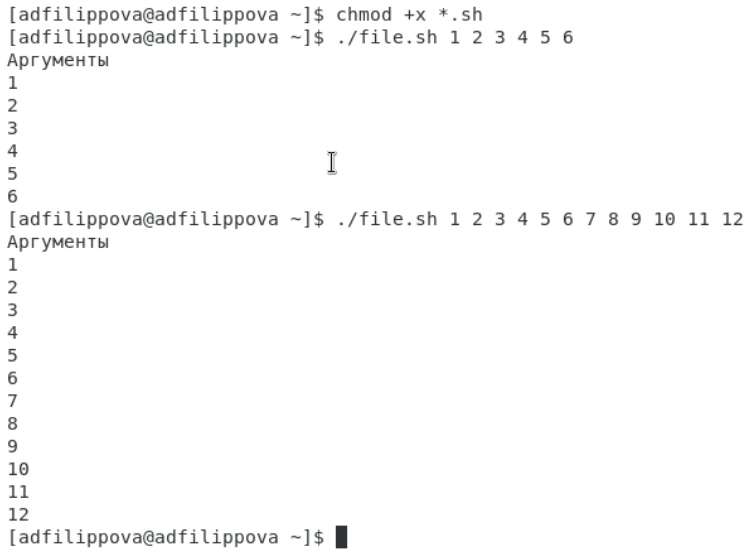

Рис. 2.10: Проверяем работу скрипта

3. Создаем файл, в котором буду писать третий скрипт, и открываем его в редакторе emacs (команды «touch file1.sh» и «emacs &»). (рис. -fig. 2.11)

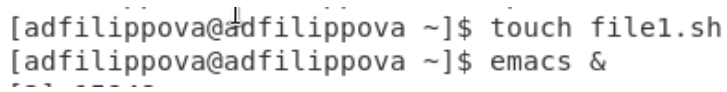

Рис. 2.11: Создаем файл

Пишем командный файл – аналог команды ls (без использования самой этой команды и команды dir). Он должен выдавать информациюо нужном каталоге и выводить информацию о возможностях доступа к файлам этого каталога (рис. -fig. 2.12) (рис. -fig. 2.13)

```
#!/bin/bash
a="$1"                              # В переменную сохраняем путь до заданного ката
алога
for i in ${a}/*                     # Цикл который будет проходить по всем каталога
ам и файлам в каталоге
do
    echo "$i"                       # Выводим название каталога

    if test -f $i                   # Проверяем является ли обычным файлом
    then echo "Обычный файл"        # Если да то выводим данное сообщение
    fi

    if test -d $i                   # Проверяем является ли каталогом
    then echo "Каталог"             # Если да то выводим данное сообщение
    fi

    if test -r $i                   # Проверяем право на чтение каталога или файла
    then echo "Чтение разрешено"    # Если да то выводим данное сообщение
    fi

    if test -w $i                   # Проверяем право на изменение/запись каталога
```

Рис. 2.12: Написание скрипта

```
    if test -w $i                   # Проверяем право на изменение/запись каталога
 или файла
    then echo "Запись разрешена"    # Если да то выводим данное сообщение
    fi
```

Рис. 2.13: Написание скрипта

Далее проверяем работу скрипта (команда «./file1.sh ~»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»). Скрипт работает корректно. (рис. -fig. 2.14)

```
[adfilippova@adfilippova ~]$ chmod +x *.sh
[4]+  Done                    emacs
[adfilippova@adfilippova ~]$ .file1.sh ~
bash: .file1.sh: команда не найдена...
[adfilippova@adfilippova ~]$ ./file1.sh ~
/home/adfilippova/123.cpp
Обычный файл
Чтение разрешено
Запись разрешена
/home/adfilippova/1.txt
Обычный файл
Чтение разрешено
Запись разрешена
/home/adfilippova/abc1
Обычный файл
Чтение разрешено
Запись разрешена
/home/adfilippova/academic-laboratory-report-template
Каталог
Чтение разрешено
Запись разрешена
Выполнение разрешено
/home/adfilippova/academic-presentation-markdown-template
Каталог
Чтение разрешено
Запись разрешена
```

Рис. 2.14: Проверяем работу скрипта

4. Создаем файл, в котором буду писать третий скрипт, и открываем его в редакторе emacs (команды «touch file1.sh» и «emacs &»).(рис. -fig. 2.15)

```
[adfilippova@adfilippova ~]$ touch file2.sh
[adfilippova@adfilippova ~]$ emacs &
```

Рис. 2.15: Создаем файл

Пишем 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.(рис. -fig. 2.16)

```
#!/bin/bash
b="$1"          # Сохраняем в переменную путь заданного каталога
shift           # Удаляем первый аргумент со сдвигом влево
for a in $@     # Цикл проходящий по заданным аргументам
do

    k=0         # обнуление счетчика который будет подсчитывать файлы
    for i in ${b}/*.${a}  # цикл проходящий по файлам имеющи нужное расщирение
    do
        if test -f "$i" # если данный путь указывает на файл то
        then
            let k=k+1   # увеличиваем счетчик на 1
        fi
    done
    echo "$k файлов $b содержится в каталоге  с расширением $a " # вывод сообщения на
 экране
 done
```

Рис. 2.16: Написание скрипта

Проверяем работу написанного скрипта (команда «./file.sh ~ doc pdf txt sh »), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»), а также создав дополнительные файлы с разными расширениями. Скрипт работает корректно. (рис. -fig. 2.17) (рис. -fig. 2.18)

```
[adfilippova@adfilippova ~]$ chmod +x *.sh
[adfilippova@adfilippova ~]$ touch 1.doc
[adfilippova@adfilippova ~]$ touch 2.pdf
[adfilippova@adfilippova ~]$ touch 3.doc
[adfilippova@adfilippova ~]$ touch 4.pdf
```

Рис. 2.17: Проверяем работу скрипта

```
[adfilippova@adfilippova ~]$ ./file2.sh ~ doc pdf txt sh
2 файлов /home/adfilippova содержится в каталоге  с расширением doc
2 файлов /home/adfilippova содержится в каталоге  с расширением pdf
8 файлов /home/adfilippova содержится в каталоге  с расширением txt
6 файлов /home/adfilippova содержится в каталоге  с расширением sh
[adfilippova@adfilippova ~]$
```

Рис. 2.18: Проверяем работу скрипта

3 Выводы

Я изучила основы программирования в оболочке ОС UNIX/Linux и научилась писать небольшие командные файлы.

4 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 C, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH – сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C 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/Linux подобных 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, введённые с клавиатуры, а переменная trash нужна для того, чтобы отобрать всю избыточно введённую 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 вэтой 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 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 еёинтерпретацию.

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 позиционными. Символ $ файла является эти параметры метасимволом являются 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 подстановкевместо 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.