Отчёт по лабораторной работе №10

Операционные системы

Сячинова Ксения Ивановна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

1. Первым делом изучим команды архивации. Для этого будем использовать команды “man zip”, “man bzip2”, “man tar”. (рис. 2.1), (рис. 2.2), (рис. 2.3), (рис. 2.4)

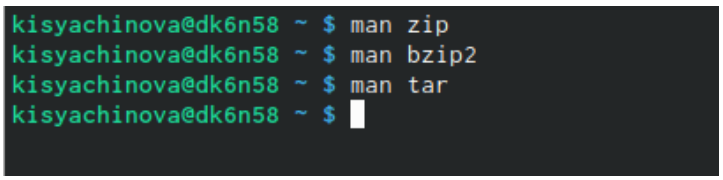


Рис. 2.1: Выполнение команд

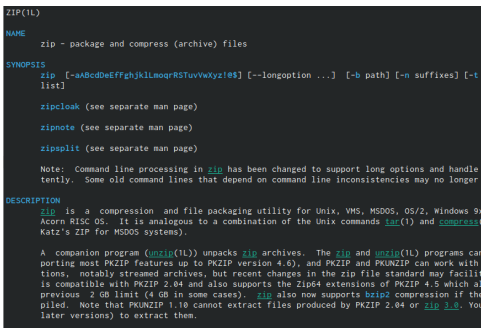


Рис. 2.2: “man zip”

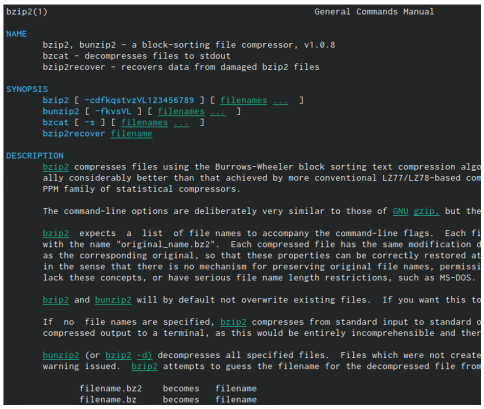


Рис. 2.3: “man bzip2”

Рис. 2.4: “man tar”

После этого создадим файл, в котором будет напиан скрипт, откроем его с помощью редактора “emacs” (сочетания клавиш “ctrl-x”, “ctrl-f”)(рис. 2.5)

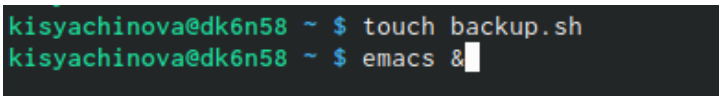


Рис. 2.5: Создание файла

Затем, создадим скрип, который при запуске будет делать резервную копию самого себя (т.е. файла, в котором содержится его исходный код) в другую директорию backup в нашем домашнем каталоге. При написании скрипта я буду использовать архиватор “bzip2”.(рис. 2.6)

```
#!/bin/bash

name='backup.sh'                # Сохранение файла со скриптом в переменную "name"
mkdir ~/backup                  # Создание каталога ~/backup
bzip2 -k ${name}                # Архивирование скрипта
mv ${name}.bz2 ~/backup/        # Перемещение архивированного скрипта в каталог
echo "Выполнено"
```

Рис. 2.6: Создание скрипта

Добавим право на выполнение “chmod +x *.sh” и проверим работу скрипта “./backup.sh”. Также проверим, появился ли каталог backup/, переходим в него, просматриваем его содержимое, и просматриваем содержимое архива “bunzip2 -c backup.sh.bz2”. (рис. 2.7)

```
kisyachinova@dk6n58 ~ $ ls
1.txt   3.txt      backup.sh   Hello                       may      Project      ski.plac
1.txt~  4.txt      backup.sh~  KseniyaSyachinova.github.io monthly  public       text.txt
2.txt   abc1       bin         lab07.sh                    my_os    public_html  tmp
2.txt~  australia  feathers    lab07.sh~                   play     reports      tutorial
[1]+  Завершён     emacs
kisyachinova@dk6n58 ~ $ chmod +x *.sh
kisyachinova@dk6n58 ~ $ ./backup.sh
Выполнено
kisyachinova@dk6n58 ~ $ cd backup/
kisyachinova@dk6n58 ~/backup $ ls
backup.sh.bz2
kisyachinova@dk6n58 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'                # Сохранение файла со скриптом в переменную "name"
mkdir ~/backup                  # Создание каталога ~/backup
bzip2 -k ${name}                # Архивирование скрипта
mv ${name}.bz2 ~/backup/        # Перемещение архивированного скрипта в каталог
echo "Выполнено"
kisyachinova@dk6n58 ~/backup $
```

Рис. 2.7: Проверка

2. Создадим файл для второго скрипта и откроем его в редакторе “emacs” с помощью сочетаний клавиш.(рис. 2.8)

```
kisyachinova@dk6n58 ~/backup $ touch os2.sh
kisyachinova@dk6n58 ~/backup $ emacs $
```

Рис. 2.8: Создание второго скрипта

Нпишем пример командного файла, который обрабатывает любое произвольное число аргументов, в том числе превышающее десять. Этот скрипт может последовательно распечатывать значения всех преданных аргументов.(рис. 2.9)

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

Рис. 2.9: Командный файл

После этого проверим работу написанного скрипта. Для этого спиользуем команду “./os2.sh 0 1 2 3 4 5 6 7 8 9 10 11”. Но для начала добавим право на выполнение “chmod +x *.sh”. Так как у нас файл, который обрабатывает любое произвольное число аргументов, я вводила аргументы, количество которых и меньше 10 и больше 10. Скрип работает верно. (рис. 2.10)

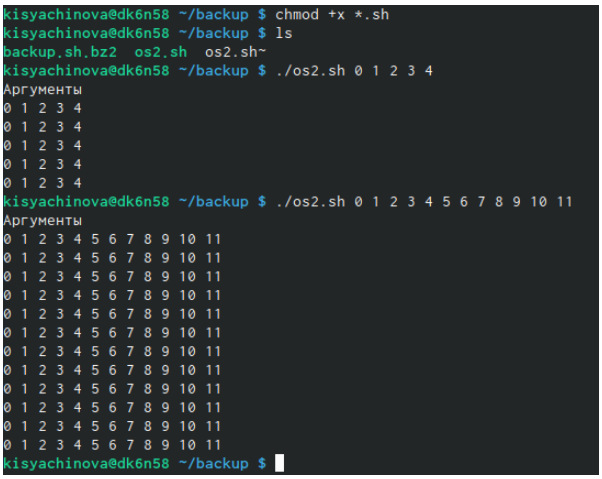


Рис. 2.10: Проверка скрипта

3. Создаём файл для написание третьего скрипта, открываем его в “emacs”.(рис. 2.11)

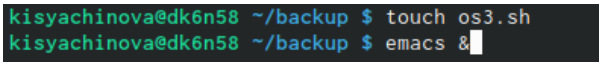
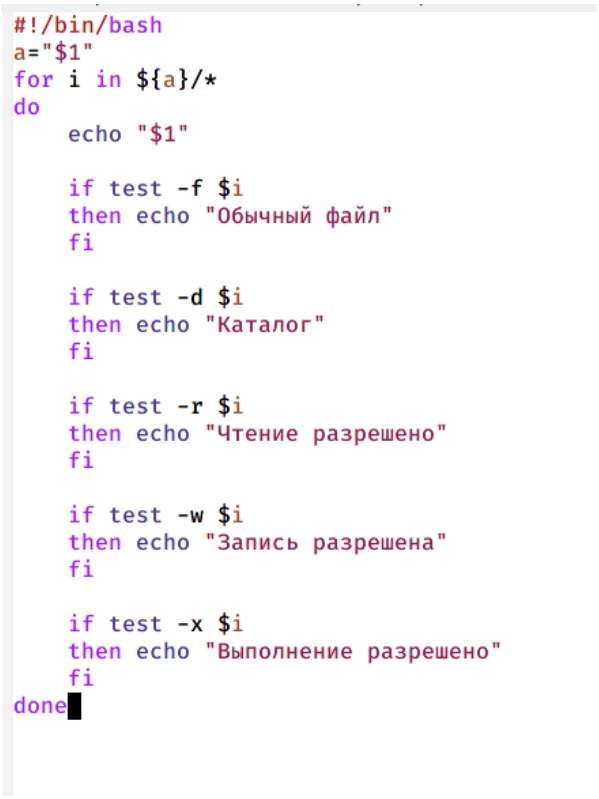


Рис. 2.11: Создание файла

Напишем командный файл, аналог команды “ls”. Он должен будет выдавать информацию о нужном каталоге и выводить информация о возможностях доступа к файлам этого каталога. (рис. 2.12)



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

    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

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

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

Рис. 2.12: Командный файл

Далее даём право на выполнение с помощью команды “chmod +x *.sh” и проверяем работу скрипта “./os3.sh ~”. (рис. 2.13). Работает корректно.

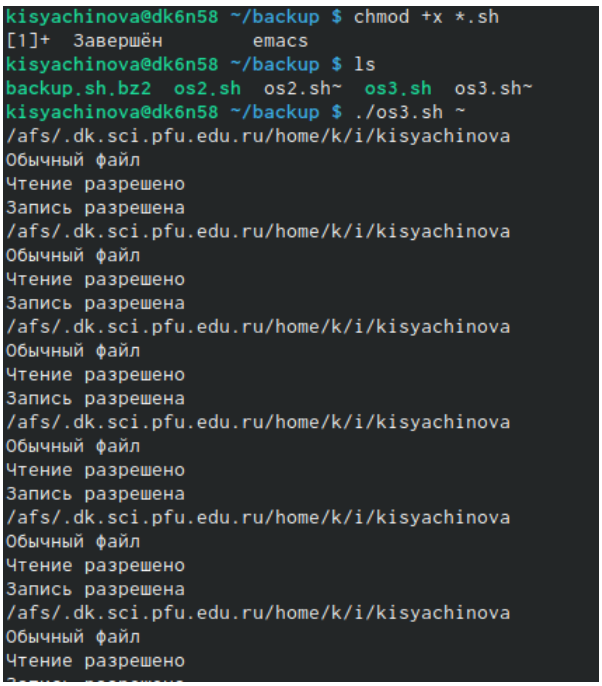


Рис. 2.13: Проверка скрипта

4. Для выполнение третьего скрипта также создаём файл и открываем его в “emacs”.(рис. 2.14)

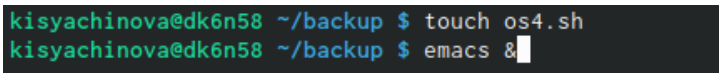


Рис. 2.14: Создание файла

Напишем командный файл для вычисление количества файлов в указанной директории. Файл полуает в качестве аргумента командной строки формат фалйа. Путь к директории также передаётся в виде аргументов командной строки. (рис. 2.15)

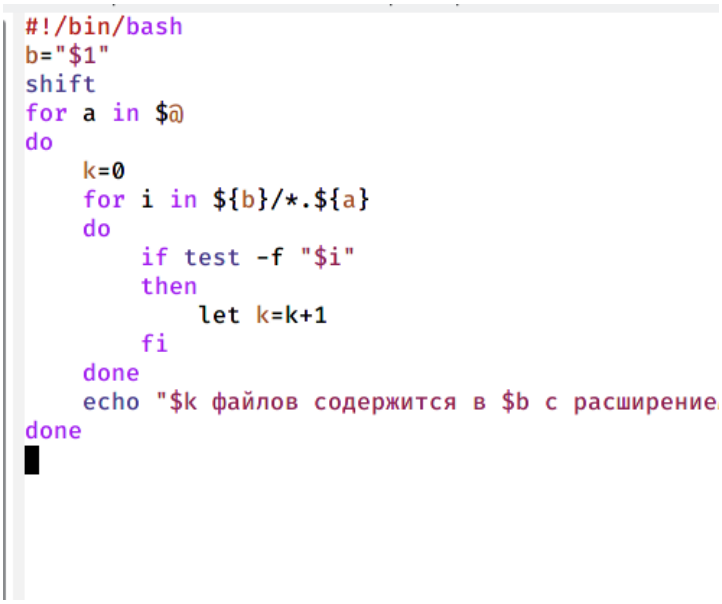


Рис. 2.15: Командный файл

Затем даём право на выполнение с помощью команды “chmod +x *.sh” и проверяем работу скрипта с помощью “./os4.sh~pdf sh txt doc”. Для проверки создадим несколько файлов разного расширения. Видим, что скрипт работает верно.(рис. 2.16)

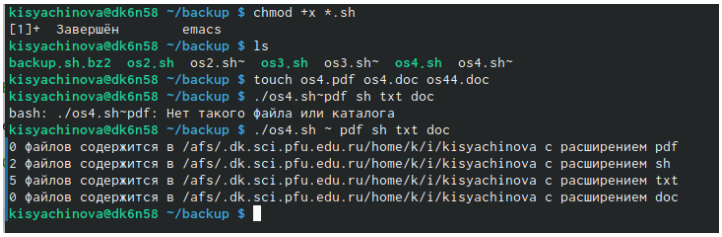


Рис. 2.16: Проверка скрипта

3 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 (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/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, 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, 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. 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 интерпритацию.

10) 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.

11) Чтобы выяснить, является ли файл каталогом или обычным файлом, необ-

ходимо воспользоваться командами «test -f [путь до файла]» (для проверки, является ли обычным файлом) и «test -d [путь до файла]» (для проверки, является ли каталогом).

12) 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.

13) 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.

14) 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.

4 Выводы

В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX/Linux и научилась писать разные командные файлы.