Отчет по лабораторной работе 10

Лабораторная работа 10

Куркина Евгения Вячеславовна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Здесь приводится формулировка цели лабораторной работы. Формулировки цели для каждой лабораторной работы приведены в методических указаниях.

Цель данной лабораторной работы — Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы. # Задание

Здесь приводится описание задания в соответствии с рекомендациями методического пособия и выданным вариантом.

2 Выполнение лабораторной работы

1) Получила справки по командам zip bzip2 tar, с помощью команды man.(рис. 2.1)(рис. 2.2)(рис. 2.3)(рис. 2.4)

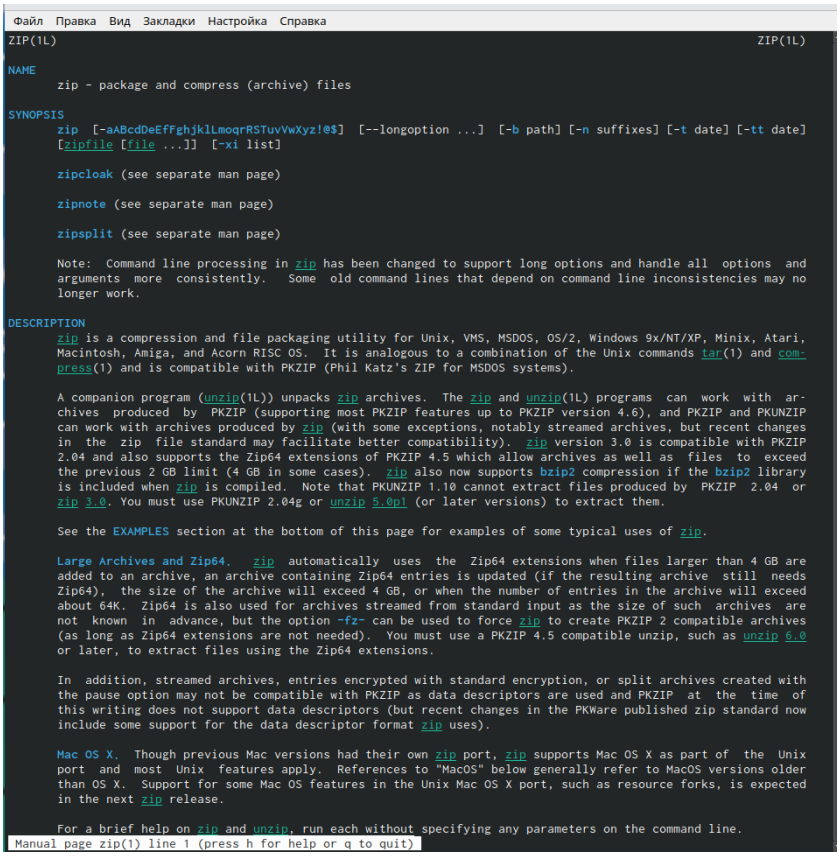

Рис. 2.1: Справка zip

Рис. 2.2: Справка bzip

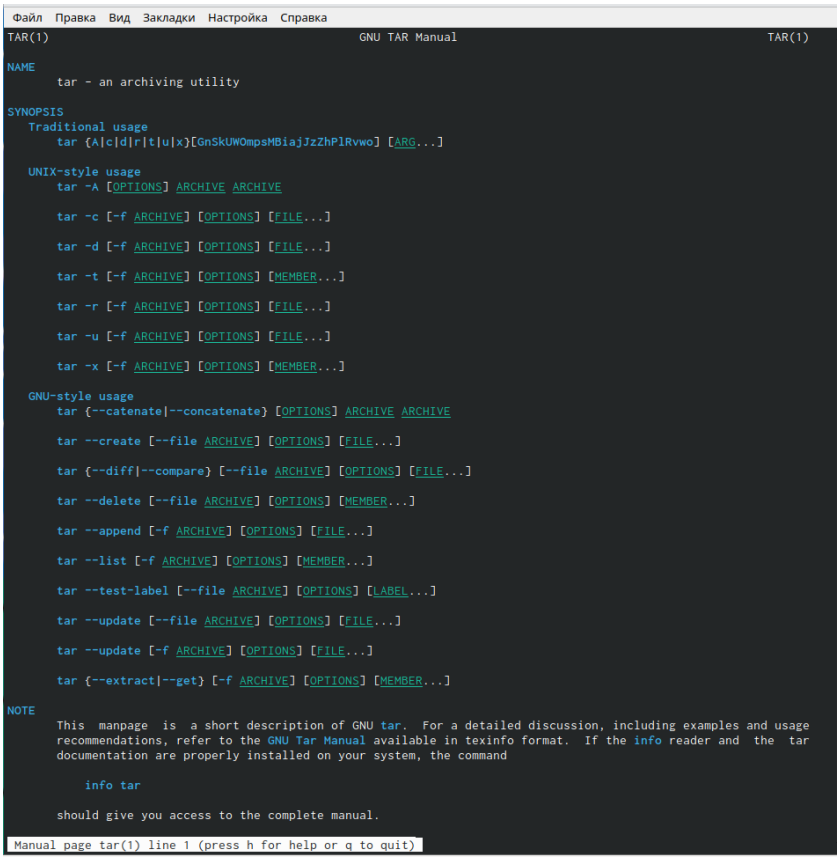

Рис. 2.3: Справка tar

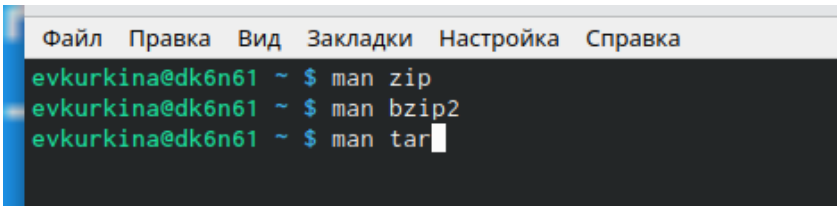

Рис. 2.4: Команда man

2) Создала файлы, в котором буду писать первый скрипт, назвала его backup.sh. Затем перешла в emacs(рис. 2.5), перешла в только что созданный файл(рис. 2.6). Написала необходимый скрипт(рис. 2.7), затем сохранила его(рис. 2.8). Перешла в консоль, дала право на выполнение, а после проверила работу скрипта. Убедились в том, что скрипт работает коректно.(рис. 2.9)(рис. 2.10)(рис. 2.11)(рис. 2.12)

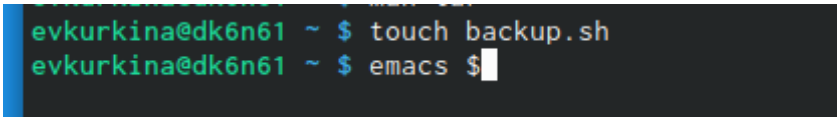

Рис. 2.5: Создание файла

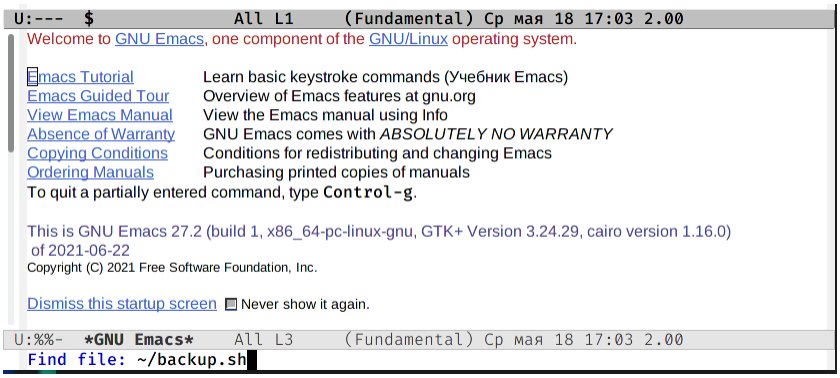

Рис. 2.6: переход в файл в emacs

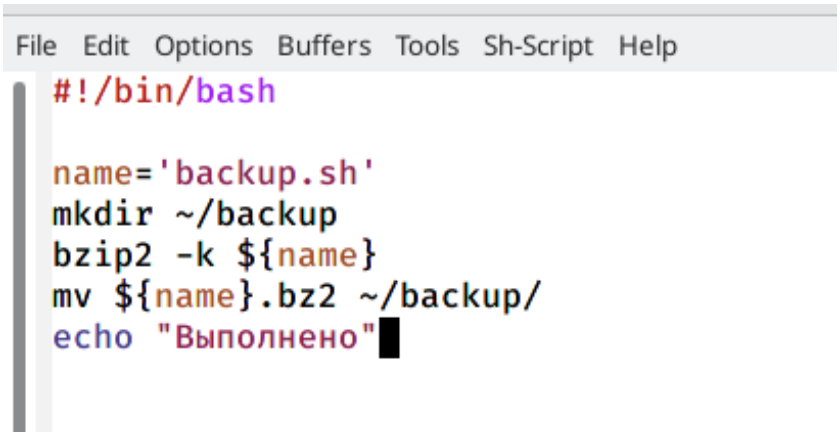

Рис. 2.7: Текст скрипта

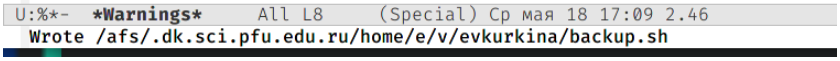

Рис. 2.8: Сохранение файла

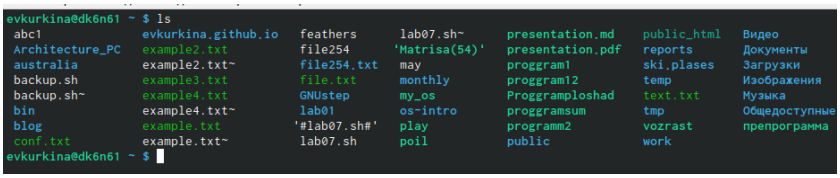

Рис. 2.9: Наличие сохраненного файла

```
evkurkina@dk6n61 ~ $ chmod +x *.sh
evkurkina@dk6n61 ~ $ ./backup.sh
Выполнено
evkurkina@dk6n61 ~ $
```

Рис. 2.10: Права для выполнения

```
evkurkina@dk6n61 ~ $ cd backup/
evkurkina@dk6n61 ~/backup $ ls
backup.sh.bz2
evkurkina@dk6n61 ~/backup $
```

Рис. 2.11: Проверка наличия

```
evkurkina@dk6n61 ~/backup $ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Выполнено"
evkurkina@dk6n61 ~/backup $
```

Рис. 2.12: Корктный результат

3) Создала новый файл с названием prog2.sh, для второго скрипта, запустила emacs (рис. 2.13). Перешла в созданный файл (рис. 2.14), ввела текст скрипта (рис. 2.15), сохранила файл (рис. 2.16),затем проверила его наличие и дала права на выполнение (рис. 2.17), проверила работу скрипта(рис. 2.18)(рис. 2.19).

```
evkurkina@dk6n61 ~/backup $ cd
evkurkina@dk6n61 ~ $ touch prog2.sh
evkurkina@dk6n61 ~ $ emacs
```

Рис. 2.13: Создание нового файла

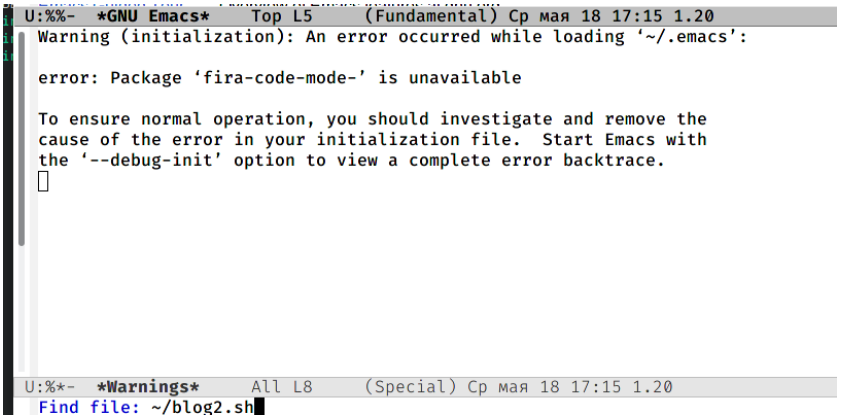

Рис. 2.14: Переход в файл в emacs

Рис. 2.15: Текст скрипта

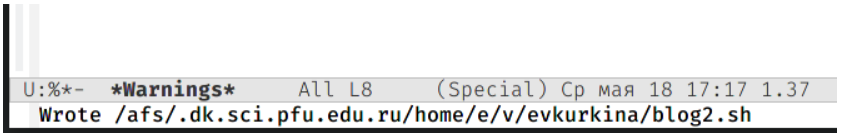

Рис. 2.16: Сохранение файла

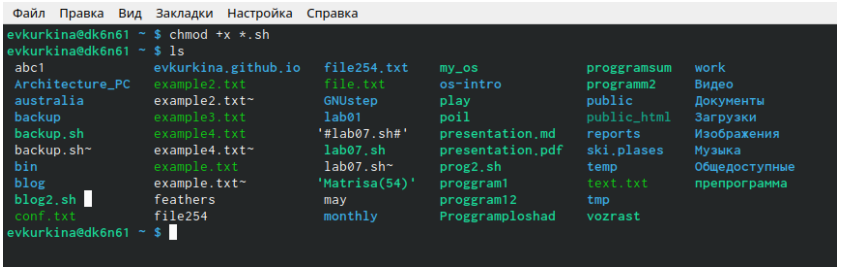

Рис. 2.17: Права выполнения и наличие файла

Рис. 2.18: Команда проверки скрипта

Рис. 2.19: Результат

4) Создала файл с названием progls.sh для третьего скрипта, открыла его в emacs(рис. 2.20). Написала командный файл, являющимся аналогом для команды ls, для вывода необходимой информации о нужном каталоге и возможностях доступа к файлам этого каталога(рис. 2.21), сохранила файл.Далее перешла в консоль, проверила наличие файла, дала права выполнения (рис. 2.22), убедилась в коректной работе скрипта(рис. 2.23).

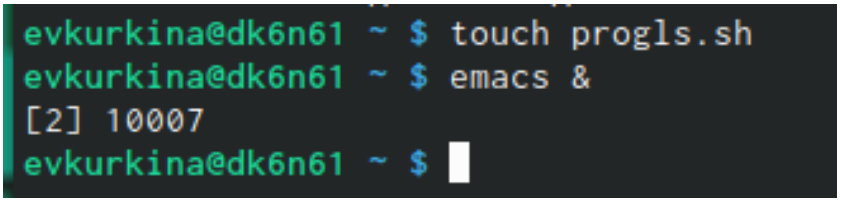

Рис. 2.20: Создание файла и переход в emacs

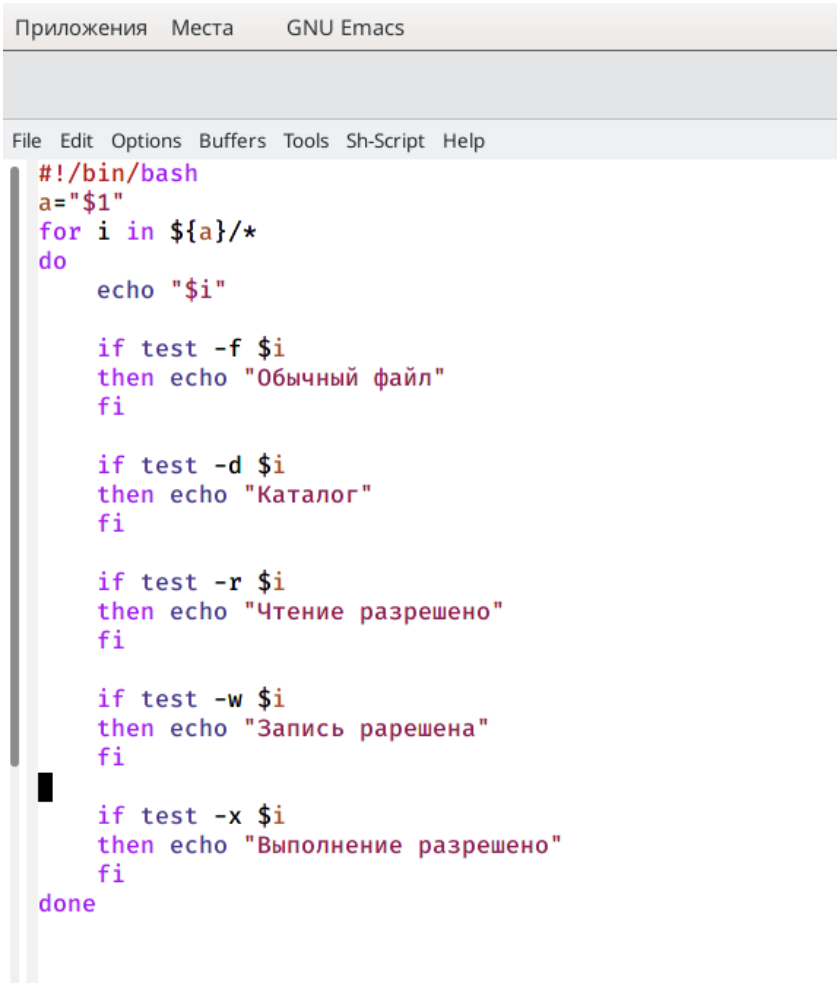

Рис. 2.21: Скрипт командного файла

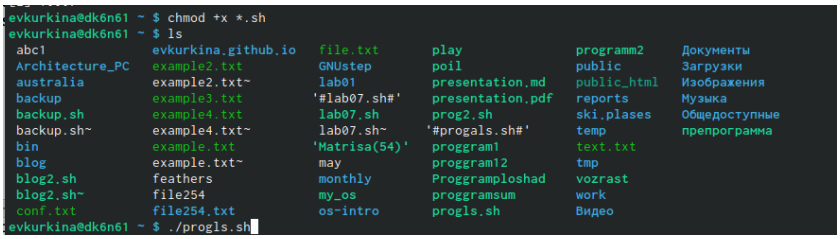

Рис. 2.22: Проверка на наличие и предоставление прав выполнения

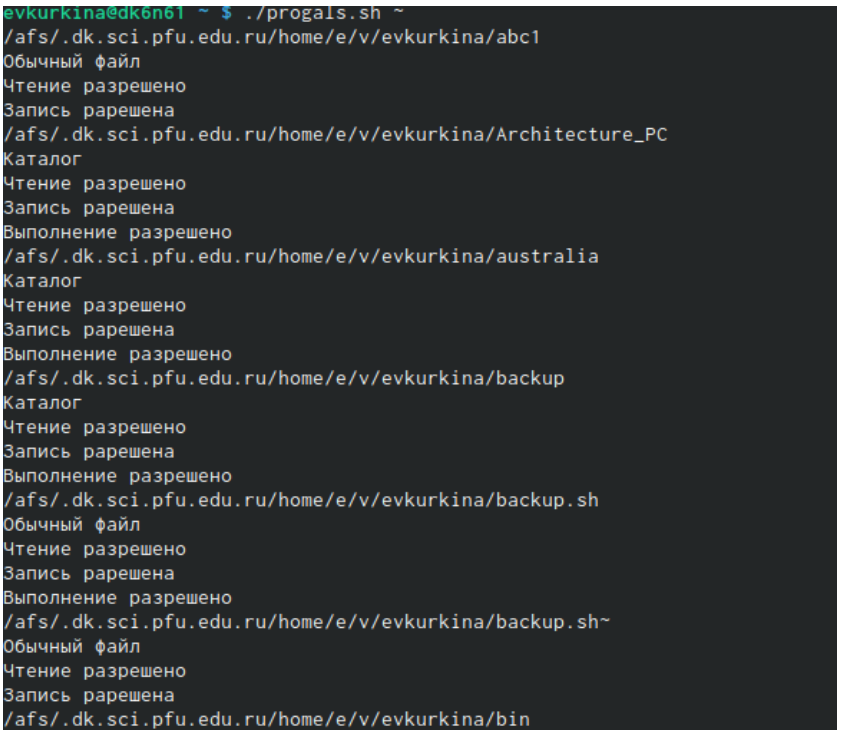

Рис. 2.23: Результат проверки работы скрипта

5) Для последнего скрипта создала файл с названием format.sh(рис. 2.24). Перешла в emax, в созданный файл,написала скрипт, который получает на входе формт файла, а затем считает количество таких файлов в указанной директории (рис. 2.25). Затем вернулась в консоль,дала права выполнения и проверила наличие, затем проверила исправную работу скрипта (рис. 2.26).

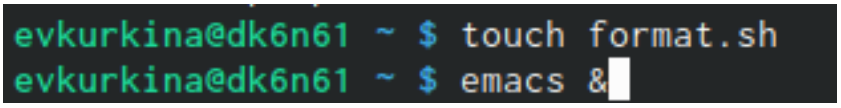

Рис. 2.24: Создание файла format.sh

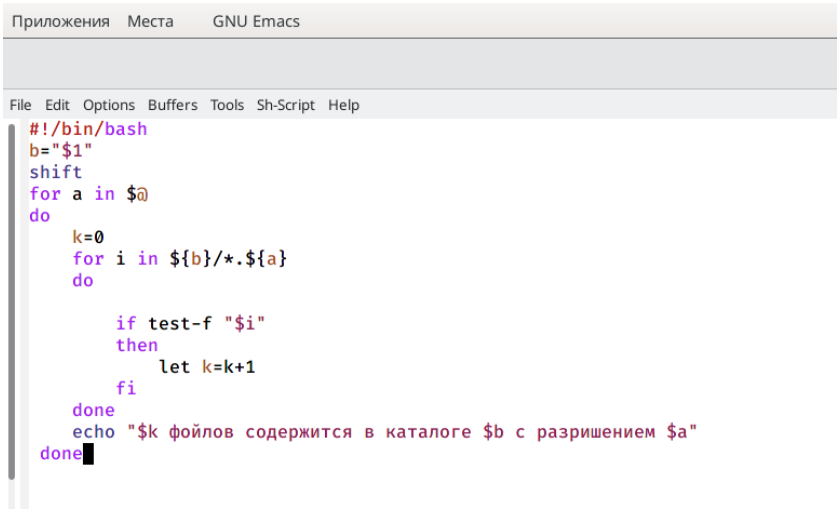

Рис. 2.25: Введенный текст скрипта

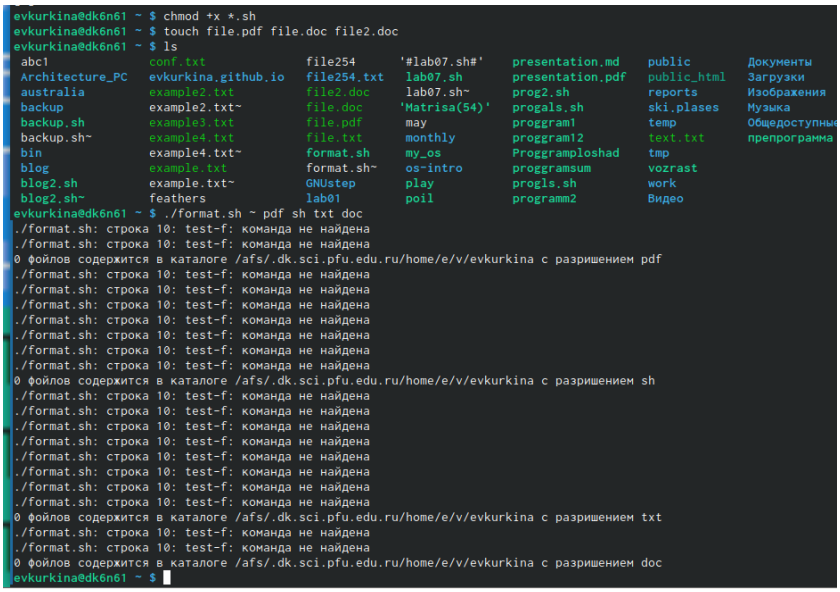

Рис. 2.26: Права доступа,результат проверки работы скрипта

6)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 (Bourneshellили sh) – стандартная командная оболочка UNIX/Linux, с

C-оболочка (или csh) –надстройка на оболочкой Борна, использующая Сподобный синта
Оболочка Корна (или ksh) – напоминает оболочку C, но операторы управления програм
BASH – сокращение от BourneAgainShell(опять оболочка Борна), в основе своей совме

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 Electricaland 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 ., «$mvafile${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 -Astates 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 monи 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

PS1 и PS2: эти переменные предназначены для отображения промптера командного проц
то интерактивная программа, запущенная командным процессором, требует ввода, то и

HOME: имя домашнего каталога пользователя. Если команда cdвводится без аргументов

IFS:последовательность символов, являющихся разделителями в командной строке, наг

MAIL:командный процессор каждый раз перед выводом на экран промптера проверяет со

TERM: тип используемого терминала.

LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автом

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

9). Снятие специального смысла с метасимвола называется экранированием мета символа. Экранирование может быть осуществлено с помощью предшеству-

ющего мета символу символа , который, в свою очередь, является мета символом. 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’|’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 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 осуществить 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 Debia nкоманда «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 пр
$! –номер процесса, в рамках которого выполняется последняя вызванная на выполнен
$--значение флагов командного процессора;
${#} –возвращает целое число –количествослов, 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 заменен
${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 выв
```

${name+value} –это выражение работает противоположно ${name-value}. Если переменн

${name#pattern} –представляет значение переменной name с удалённым самым коротким

${#name[*]} и ${#name[@]}–эти выражения возвращают количество элементов в массиве

3 Выводы

Во время выполнения данной лабораторной работы, я изучила основы программирования в оболчке ОС UNIX/Linux, найчилась писать небольшие командные файлы.

Список литературы