

Отчёт по лабораторной работе №10

**Программирование в командном процессоре ОС UNIX. Командные
файлы**

Аскеров Александр Эдуардович

Содержание

1	Цель работы	4
2	Теоретическое введение	5
3	Выполнение лабораторной работы	6
4	Контрольные вопросы	12
5	Выводы	19

Список иллюстраций

3.1	Просмотр справки по команде tar	6
3.2	Создание директории для резервной копии	6
3.3	Открытие редактора	7
3.4	Скрипт для lab101	7
3.5	Добавление права на исполнение	7
3.6	Запуск скрипта и результат его работы	7
3.7	Открытие редактора	7
3.8	Скрипт для lab102	8
3.9	Добавление права на исполнение	8
3.10	Запуск скрипта	8
3.11	Открытие редактора	9
3.12	Скрипт для lab103	9
3.13	Добавление права на исполнение	9
3.14	Запуск скрипта	10
3.15	Открытие редактора	10
3.16	Скрипт для lab104	10
3.17	Добавление права на исполнение	10
3.18	Запуск скрипта	11

1 Цель работы

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

2 Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера.

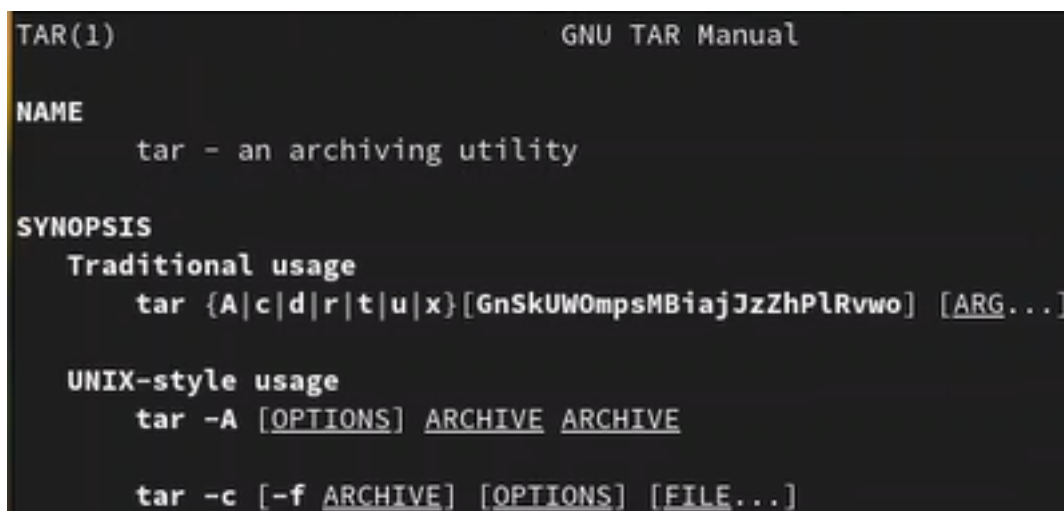
POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

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 Выполнение лабораторной работы

1. Напишем скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации узнаем, изучив справку.

Посмотрим справку по команде tar.



```
TAR(1)                                GNU TAR Manual

NAME
    tar - an archiving utility

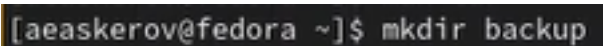
SYNOPSIS
    Traditional usage
        tar {A|c|d|r|t|u|x}[GnSkUW0mpsMBiajJzZhPlRvwo] [ARG...]

    UNIX-style usage
        tar -A [OPTIONS] ARCHIVE ARCHIVE

        tar -c [-f ARCHIVE] [OPTIONS] [FILE...]
```

Рис. 3.1: Просмотр справки по команде tar

Создадим директорию для резервной копии.



```
[aeaskerov@fedora ~]$ mkdir backup
```

Рис. 3.2: Создание директории для резервной копии

Откроем редактор.

```
[aeaskerov@fedora ~]$ vi lab101
```

Рис. 3.3: Открытие редактора

Напишем скрипт.

```
tar -cf lab101.tar lab101
mv lab101.tar backup
```

Рис. 3.4: Скрипт для lab101

Сделаем файл lab101 исполняемым.

```
[aeaskerov@fedora ~]$ chmod +x lab101
```

Рис. 3.5: Добавление права на исполнение

Запустим файл lab101 и посмотрим содержимое каталога backup.

```
[aeaskerov@fedora ~]$ ./lab101
[aeaskerov@fedora ~]$ ls backup
lab101.tar
```

Рис. 3.6: Запуск скрипта и результат его работы

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.

Откроем редактор.

```
[aeaskerov@fedora ~]$ emacs lab102
```

Рис. 3.7: Открытие редактора

Напишем скрипт.

```
count=1
for i
do
    echo "$count: $i"
    count=$((count+1))
done
```

Рис. 3.8: Скрипт для lab102

Сделаем файл lab102 исполняемым.

```
[aeaskerov@fedora ~]$ chmod +x lab102
```

Рис. 3.9: Добавление права на исполнение

Запустим файл lab102.

```
[aeaskerov@fedora ~]$ ./lab102 10 20 30 40 50 60 70 80 90 100 110
1: 10
2: 20
3: 30
4: 40
5: 50
6: 60
7: 70
8: 80
9: 90
10: 100
11: 110
```

Рис. 3.10: Запуск скрипта

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.

Откроем редактор.

```
[aeaskerov@fedora ~]$ emacs lab103
```

Рис. 3.11: Открытие редактора

Напишем скрипт.

```
for A in *
do if test -d $A
  then echo $A: is a directory
  else echo -n $A: "is a file and "
    if test -x $A
    then echo executable
    elif test -w $A
    then echo writeable
    elif test -r $A
    then echo readable
    else echo neither readable nor writeable
    fi
  fi
done
```

Рис. 3.12: Скрипт для lab103

Сделаем файл lab103 исполняемым.

```
[aeaskerov@fedora ~]$ chmod +x lab103
```

Рис. 3.13: Добавление права на исполнение

Запустим файл lab103.

```
[aeaskerov@fedora ~]$ ./lab103
abc1: is a file and writeable
australia: is a directory
backup: is a directory
bin: is a directory
catalog1: is a directory
conf.txt: is a file and writeable
```

Рис. 3.14: Запуск скрипта

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.

Откроем редактор.

```
[aeaskerov@fedora ~]$ emacs lab104
```

Рис. 3.15: Открытие редактора

Напишем скрипт.

```
echo Input a directory
read dir
echo Input a file format
read format
find $dir -maxdepth 1 -name "$format" -type f | wc -l
```

Рис. 3.16: Скрипт для lab104

Сделаем файл lab104 исполняемым.

```
[aeaskerov@fedora ~]$ chmod +x lab104
```

Рис. 3.17: Добавление права на исполнение

Запустим файл lab104.

```

[aeaskerov@fedora ~]$ ./lab104
Input a directory
/home/aeaskerov
Input a file format
.txt
3
[aeaskerov@fedora ~]$ ls
abcl      feathers  lab07.sh~  lab104~  ski.places  Загрузки
australia file.txt  lab101     may      text.txt    Изображения
backup    keys      lab102     monthly  tmp         Музыка
bin       keys.pub  lab103     my_os    work        Общедоступные
catalogi  '#lab07.sh#' lab103~    play     Видео      'Рабочий стол'
conf.txt  lab07.sh  lab104     reports  Документы  Шаблоны

```

Рис. 3.18: Запуск скрипта

4 Контрольные вопросы

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?

Командная оболочка – это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: 1. Оболочка Борна (Bourne shell или sh) – это стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;

2. C оболочка (или csh) – это надстройка над оболочкой Борна, использующая C подобный синтаксис команд с возможностью сохранения истории выполнения команд;
3. Оболочка Корна (или ksh) напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
4. BASH сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).
5. Что такое POSIX?

POSIX (Portable Operating System Interface for Computer Environments) – это 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. Как определяются переменные и массивы в языке программирования bash?

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. Каково назначение операторов `let` и `read`?

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. Какие арифметические операции можно применять в языке программирования `bash`?

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

6. Что означает операция (())?

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

7. Какие стандартные имена переменных Вам известны?

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. Что такое метасимволы?

Такие символы, как `' < > * ? | " &`, являются метасимволами и имеют для командного процессора специальный смысл.

9. Как экранировать метасимволы?

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. Как создавать и запускать командные файлы?

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 осуществить её интерпретацию.

11. Как определяются функции в языке программирования `bash`?

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. Каким образом можно выяснить, является файл каталогом или обычным файлом?

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. Каково назначение команд `set`, `typeset` и `unset`?

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. Команда

«typeset» предназначена для наложения ограничений на переменные. Команду «unset» следует использовать для удаления переменной из окружения командной оболочки.

14. Как передаются параметры в командные файлы?

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. Назовите специальные переменные языка bash и их назначение.

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`.

5 Выводы

Изучены основы программирования в оболочке ОС UNIX/Linux. Приобретён навык написания небольших командных файлов.