

Отчет по лабораторной работе №10

Дисциплина: Операционные системы

Кашкин Иван Евгеньевич

Содержание

Цель работы	5
Задание	6
Теоретическое введение	7
Выполнение лабораторной работы	8
Выводы	16
Список литературы	22

Список иллюстраций

0.1	Команда <code>man</code>	8
0.2	Ввод программы	8
0.3	Ввод программы	9
0.4	Ввод программы	9
0.5	Создание файла	10
0.6	Ввод скрипта	10
0.7	<code>chmod</code> и запуск	11
0.8	Второй скрипт	12
0.9	<code>chmod</code> и запуск	12
0.10	Третий скрипт	13
0.11	<code>chmod</code> и запуск	14
0.12	Третий скрипт	15
0.13	<code>chmod</code> и запуск	15

Список таблиц

Цель работы

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

Задание

1. Написать 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`. Способ использования команд архивации необходимо узнать, изучив справку.
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
3. Написать 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.
4. Написать 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.

Теоретическое введение

- 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;
- С-оболочка (или csh) — надстройка на оболочкой Борна, использующая С-подобный 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). 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. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

Выполнение лабораторной работы

Задание 1

- 1) Сначала с помощью команды `man` изучил информацию о `zip`, `bzip2`, `tar` (рис. [-@fig:001])

```
[ivanekashkin@iekashkin lab10]$ man zip
[ivanekashkin@iekashkin lab10]$ man bzip2
[ivanekashkin@iekashkin lab10]$ man tar
[ivanekashkin@iekashkin lab10]$
```

Рис. 0.1: Команда `man`

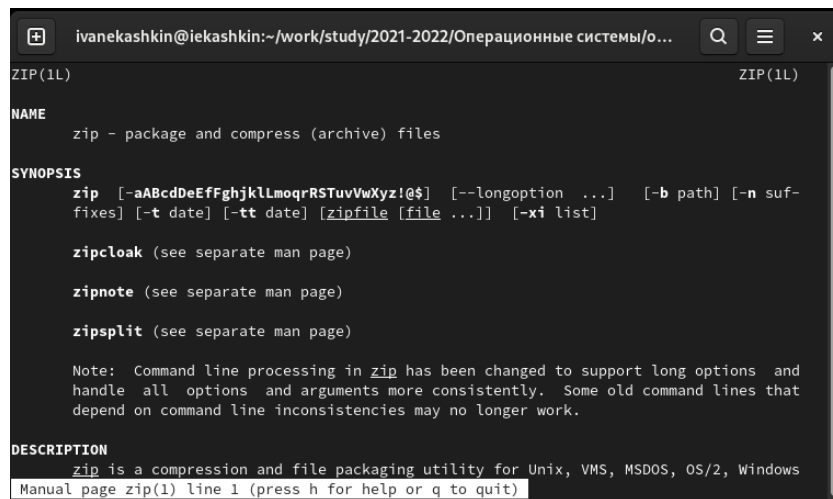


Рис. 0.2: Ввод программы


```
ivanekashkin@iekashkin:~/work/study/2021-2022/Операционные системы/o...
bzip2(1)                                General Commands Manual                                bzip2(1)

NAME
  bzip2, bunzip2 - a block-sorting file compressor, v1.0.8
  bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS
  bzip2 [ -cdfkqstvwVL123456789 ] [ filenames ... ]
  bunzip2 [ -fkvsVL ] [ filenames ... ]
  bzip2recover [ -s ] [ filenames ... ]
  bzip2recover filename

DESCRIPTION
  bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. Compression is generally considerably better than that achieved by more conventional LZ77/LZ78-based compressors, and approaches the performance of the PPM family of statistical compressors.

  The command-line options are deliberately very similar to those of GNU gzip, but they are not identical.

Manual page bzip2(1) line 1 (press h for help or q to quit)
```

Рис. 0.3: Ввод программы

```
ivanekashkin@iekashkin:~/work/study/2021-2022/Операционные системы/o...
TAR(1)                                GNU TAR Manual                                TAR(1)

NAME
  tar - an archiving utility

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

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

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

    tar -d [-f ARCHIVE] [OPTIONS] [FILE...]

    tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]

    tar -r [-f ARCHIVE] [OPTIONS] [FILE...]

    tar -u [-f ARCHIVE] [OPTIONS] [FILE...]

Manual page tar(1) line 1 (press h for help or q to quit)
```

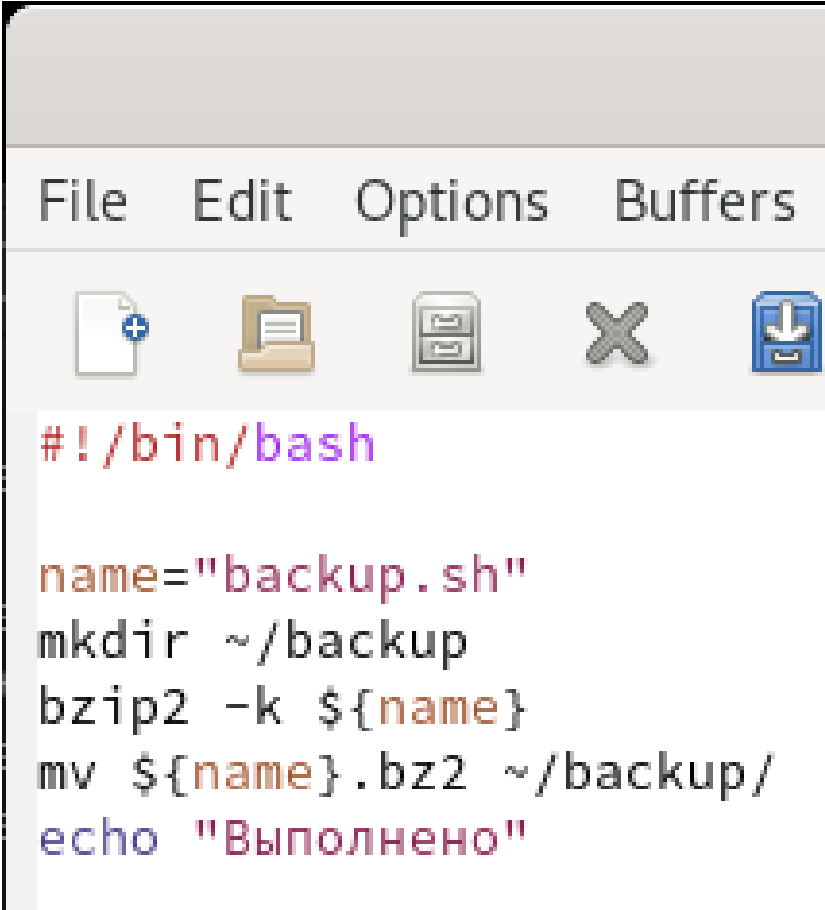
Рис. 0.4: Ввод программы

- 2) Создаем файл превого скрипта “touch backup.sh” и запускаем етас, через него открываем файл(рис. [-@fig:005])

```
[ivanekashkin@iekashkin lab10]$ man cat  
[ivanekashkin@iekashkin lab10]$ touch backup.sh  
[ivanekashkin@iekashkin lab10]$ emacs &  
[1] 37310  
[ivanekashkin@iekashkin lab10]$
```

Рис. 0.5: Создание файла

- 3) После написал скрипт, который при запуске будет делать резервную копию файла, в котором содержится его исходный код в другую директорию backup в домашнем каталоге. При этом файл архивируется bzip2. (рис. [-@fig:006])



The screenshot shows the Emacs editor interface with a menu bar (File, Edit, Options, Buffers) and a toolbar with icons for creating a new file, opening a file, saving, closing, and downloading. The main text area contains a shell script with the following content:

```
#!/bin/bash  
  
name="backup.sh"  
mkdir ~/backup  
bzip2 -k ${name}  
mv ${name}.bz2 ~/backup/  
echo "Выполнено"
```

Рис. 0.6: Ввод скрипта

- 4) Делаю я сохраняю файл (C-x C-s) и делаю его исполняющим “chmod +x *.sh” и запускаю его (рис. [-@fig:007])

```
[ivanekashkin@iekashkin lab10]$ chmod +x *.sh
[1]+  Завершён      emacs
[ivanekashkin@iekashkin lab10]$ ./backup.sh
Выполнено
[ivanekashkin@iekashkin lab10]$ cd -/backup
bash: cd: -/: недопустимый параметр
cd: использование: cd [-L|[-P [-e]] [-@]] [каталог]
[ivanekashkin@iekashkin lab10]$ cd ~/backup
[ivanekashkin@iekashkin backup]$ ls
backup.sh.bz2
[ivanekashkin@iekashkin backup]$
```

Рис. 0.7: chmod и запуск

- 5) Также как и с первым, я создаю файл для второго скрипта и открываю его в emacs.
- 6) Написал скрипт, обрабатывающий любое произвольное числом аргументов и последовательно распечатывающий значения всех переданных аргументов (рис. [-@fig:008])

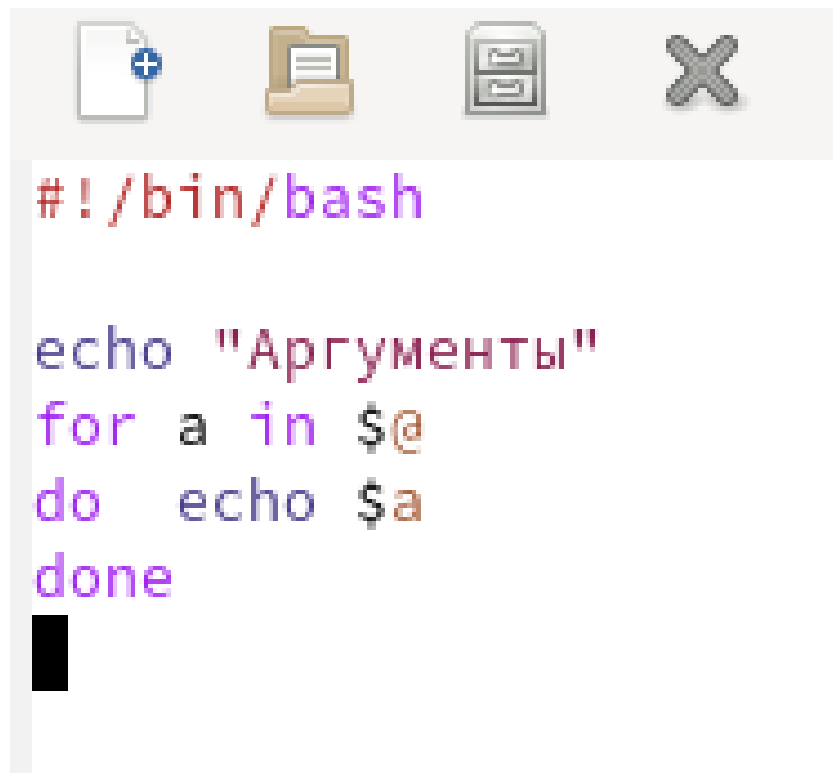


Рис. 0.8: Второй скрипт

- 7) Я сделал файл исполняемым и запустил его также как первый скрипт, и ввел аргументы, которые распечатывались на консоли (рис. [-@fig:009])

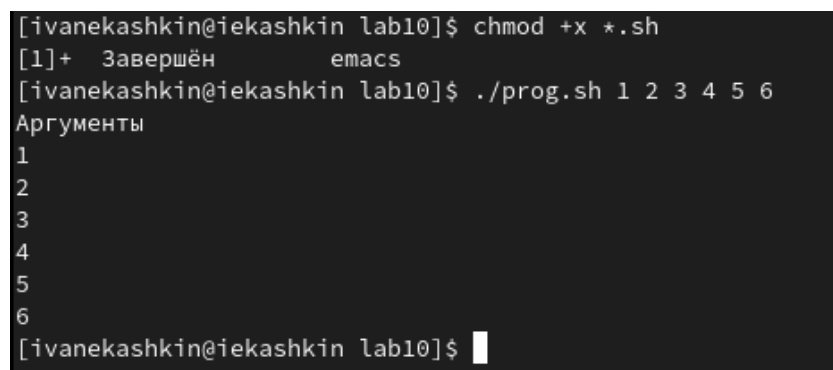


Рис. 0.9: chmod и запуск

- 8) Также как и с предыдущими, я создаю файл для третьего скрипта и открываю его в emacs.

- 9) Написал скрипт - аналог команды ls. Он выдает информацию о нужном каталоге и выводит информацию о возможностях доступа к файлам этого каталога (рис. [-@fig:0010])

for i in \${a}/*
do
 echo "\$i"
 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. 0.10: Третий скрипт

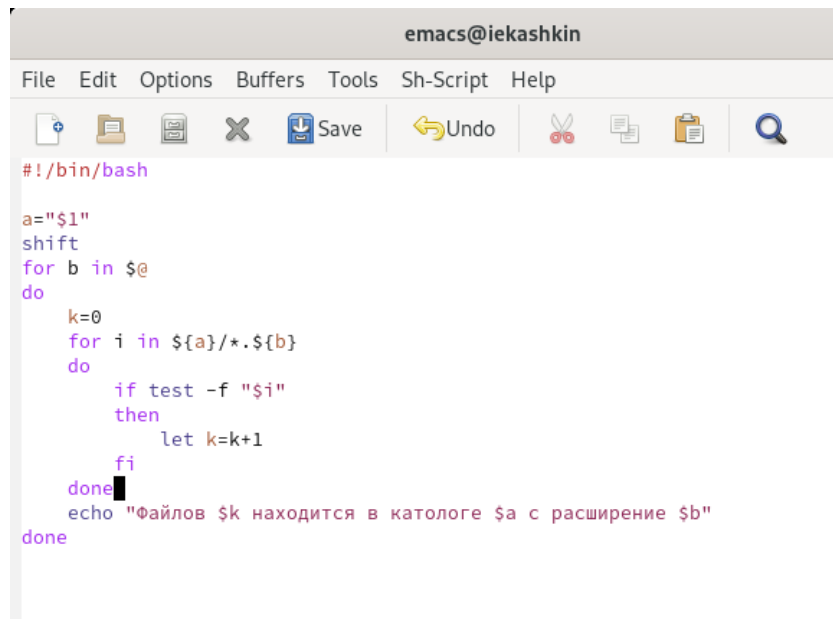
- 10) Я сделал файл исполняемым и запустил его также как первые скрипты, и

ввел каталог для просмотра (рис. [-@fig:0011])

```
[ivanekashkin@iekashkin lab10]$ chmod +x *.sh
[2]+  Завершён      emacs
[ivanekashkin@iekashkin lab10]$ ./prog2.sh ~
/bin
Каталог
Чтение разрешено
Выполнение разрешено
/boot
Каталог
Чтение разрешено
Выполнение разрешено
/dev
Каталог
Чтение разрешено
Выполнение разрешено
/etc
Каталог
Чтение разрешено
Выполнение разрешено
/home
Каталог
```

Рис. 0.11: chmod и запуск

- 11) Также как и с предыдущими, я создаю файл для последнего скрипта и открываю его в emacs.
- 12) Написал командный файл, который получает в качестве аргумента командной строки формат файла и пишет количество таких файлов в указанной директории.(рис. [-@fig:0012])

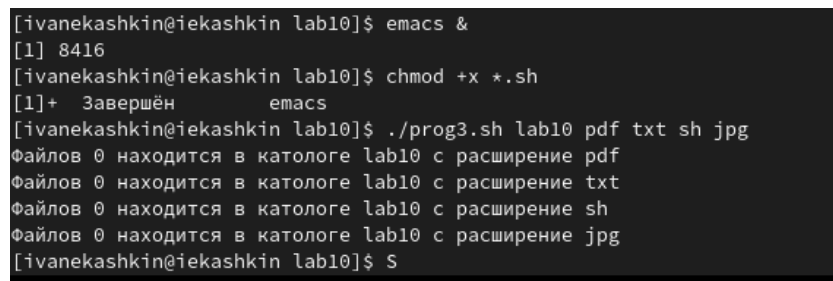


```
#!/bin/bash

a="$1"
shift
for b in $@
do
    k=0
    for i in ${a}/*.${b}
    do
        if test -f "$i"
        then
            let k=k+1
        fi
    done
    echo "Файлов $k находится в каталоге $a с расширение $b"
done
```

Рис. 0.12: Третий скрипт

- 13) Я сделал файл исполняемым и запустил его также как первые скрипты, и ввел каталог для считывания (рис. [-@fig:0013])



```
[ivanekashkin@iekashkin lab10]$ emacs &
[1] 8416
[ivanekashkin@iekashkin lab10]$ chmod +x *.sh
[1]+  Завершён      emacs
[ivanekashkin@iekashkin lab10]$ ./prog3.sh lab10 pdf txt sh jpg
Файлов 0 находится в каталоге lab10 с расширение pdf
Файлов 0 находится в каталоге lab10 с расширение txt
Файлов 0 находится в каталоге lab10 с расширение sh
Файлов 0 находится в каталоге lab10 с расширение jpg
[ivanekashkin@iekashkin lab10]$ S
```

Рис. 0.13: chmod и запуск

Выводы

- Изучил основы программирования в оболочке ОС UNIX/Linux. Научился 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 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 отобразить 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 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 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 файла эти параметры являются позиционными. Символ \$ является метасимволом 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 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 эле-

менты 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`.

Список литературы

::: {#Лабораторная работа No 10. Программирование в командном процессоре ОС UNIX. Командные файлы} :::