

Отчёт по лабораторной работе №10

Программирование в командном процессоре ОС UNIX. Командные
файлы

Ганина Таисия Сергеевна, НКАбд-01-22

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Листинги	12
6	Контрольные вопросы	14
7	Выводы	23
	Список литературы	24

Список иллюстраций

4.1	Текст программы 1	8
4.2	Результат	8
4.3	Текст программы 2	9
4.4	Результат	9
4.5	Текст программы 3	10
4.6	Результат	10
4.7	Текст программы 4	11
4.8	Результат	11

Список таблиц

1 Цель работы

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

2 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.

3 Теоретическое введение

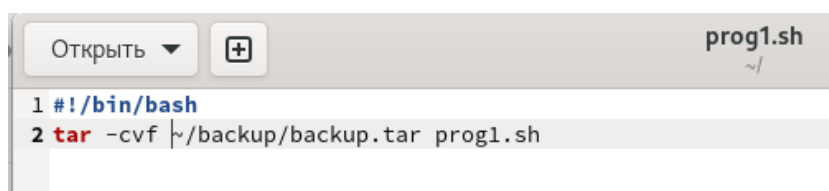
Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: - оболочка Борна (Bourne shell 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.

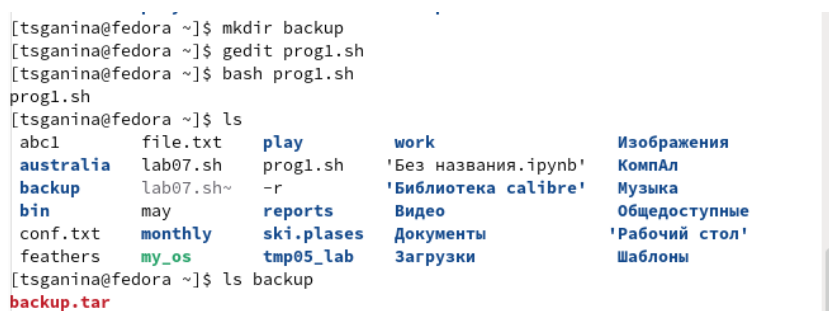
4 Выполнение лабораторной работы

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 (см. рис. 4.1, 4.2).



```
1 #!/bin/bash
2 tar -cvf ~/backup/backup.tar prog1.sh
```

Рис. 4.1: Текст программы 1

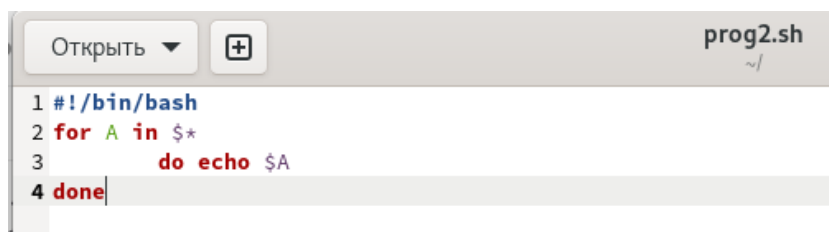


```
[tsganina@fedora ~]$ mkdir backup
[tsganina@fedora ~]$ gedit prog1.sh
[tsganina@fedora ~]$ bash prog1.sh
prog1.sh
[tsganina@fedora ~]$ ls
abc1      file.txt  play      work      Изображения
australia lab07.sh  prog1.sh  'Без названия.ipynb'  КомпАл
backup    lab07.sh~ -r        'Библиотека calibre'  Музыка
bin       may       reports    Видео     Общедоступные
conf.txt  monthly  ski.places Документы 'Рабочий стол'
feathers  my_os    tmp05_lab  Загрузки  Шаблоны
[tsganina@fedora ~]$ ls backup
backup.tar
```

Рис. 4.2: Результат

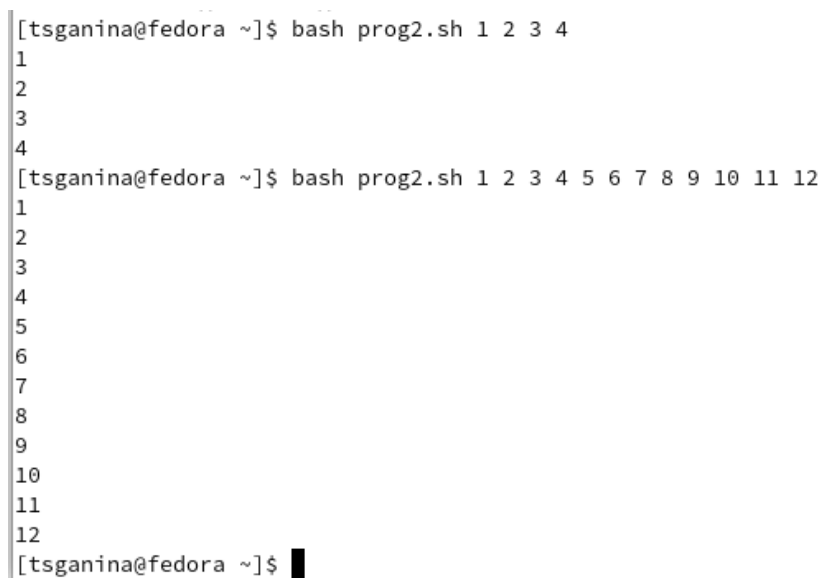
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять.

Например, скрипт может последовательно распечатывать значения всех переданных аргументов (см. рис. 4.3, 4.4).



```
1 #!/bin/bash
2 for A in $*
3 do echo $A
4 done
```

Рис. 4.3: Текст программы 2



```
[tsganina@fedora ~]$ bash prog2.sh 1 2 3 4
1
2
3
4
[tsganina@fedora ~]$ bash prog2.sh 1 2 3 4 5 6 7 8 9 10 11 12
1
2
3
4
5
6
7
8
9
10
11
12
[tsganina@fedora ~]$
```

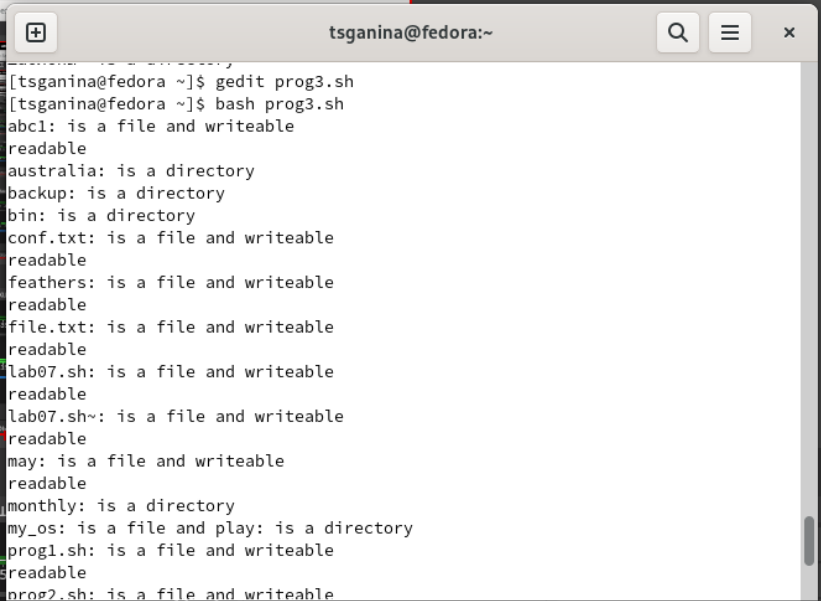
Рис. 4.4: Результат

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 (см. рис. 4.5, 4.6).



```
1 #!/bin/bash
2 for A in *
3 do
4     if test -d "$A"
5     then
6         echo "$A: is a directory"
7     else
8         echo -n "$A: is a file and "
9         if test -w $A
10        then
11            echo writeable
12        if test -r $A
13        then
14            echo "readable"
15        else
16            echo "neither readable or writeable"
17        fi
18    fi
19 fi
20 done
```

Рис. 4.5: Текст программы 3



```
[tsganina@fedora ~]$ gedit prog3.sh
[tsganina@fedora ~]$ bash prog3.sh
abc1: is a file and writeable
readable
australia: is a directory
backup: is a directory
bin: is a directory
conf.txt: is a file and writeable
readable
feathers: is a file and writeable
readable
file.txt: is a file and writeable
readable
lab07.sh: is a file and writeable
readable
lab07.sh~: is a file and writeable
readable
may: is a file and writeable
readable
monthly: is a directory
my_os: is a file and play: is a directory
prog1.sh: is a file and writeable
readable
prog2.sh: is a file and writeable
```

Рис. 4.6: Результат

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 (см. рис. 4.7, 4.8).



```
1 #!/bin/bash
2 format=""
3 directory=""
4 echo "Напишите формат файла"
5 read format
6 echo "Напишите директорию"
7 read directory
8 find "${directory}" -name "*${format}" -type f | wc -l
9
```

Рис. 4.7: Текст программы 4

```
[tsganina@fedora ~]$ gedit prog4.sh
[tsganina@fedora ~]$ bash prog4.sh
Напишите формат файла
.txt
Напишите директорию
Документы
8
```

Рис. 4.8: Результат

5 ЛИСТИНГИ

1. Программа 1

```
#!/bin/bash
tar -cvf ~/backup/backup.tar prog1.sh
```

2. Программа 2

```
#!/bin/bash
for A in $*
do echo $A
done
```

3. Программа 3

```
#!/bin/bash
for A in *
do
    if test -d "$A"
    then
        echo "$A: is a directory"
    else
        echo -n "$A: is a file and "
        if test -w $A
        then
            echo writeable
```

```
        if test -r $A
        then
            echo "readable"
        else
            echo "neither readable or writeable"
        fi
    fi
fi
done
```

4. Программа 4

```
#!/bin/bash
format=""
directory=""
echo "Напишите формат файла"
read format
echo "Напишите директорию"
read directory
find "${directory}" -name "*${format}" -type f | wc -l
```

6 Контрольные вопросы

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

Командные процессоры или оболочки - это программы, позволяющие пользователю взаимодействовать с компьютером. Их можно рассматривать как настоящие интерпретируемые языки, которые воспринимают команды пользователя и обрабатывают их. Поэтому командные процессоры также называют интерпретаторами команд. На языках оболочек можно писать программы и выполнять их подобно любым другим программам. UNIX обладает большим количеством оболочек. Наиболее популярными являются следующие четыре оболочки:

- оболочка Борна (Bourne) - первоначальная командная оболочка UNIX: базовый, но полный набор функций;
- C-оболочка - добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя C-подобный синтаксис команд, и сохраняет историю выполненных команд;
- оболочка Корна - 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. Что такое POSIX?

POSIX (Portable Operating System Interface for Computer Environments) - интерфейс переносимой операционной системы для компьютерных сред. Пред-

ставляет собой набор стандартов, подготовленных институтом инженеров по электронике и радиотехнике (IEEE), который определяет различные аспекты построения операционной системы. POSIX включает такие темы, как программный интерфейс, безопасность, работа с сетями и графический интерфейс. POSIX-совместимые оболочки являются будущим поколением оболочек UNIX и других ОС. Windows NT рекламируется как система, удовлетворяющая POSIX-стандартам. POSIX-совместимые оболочки разработаны на базе оболочки Корна; фонд бесплатного программного обеспечения (Free Software Foundation) работает над тем, чтобы и оболочку BASH сделать POSIX-совместимой.

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 мета символ.

Использование команд `b=/tmp/andy-ls -l myfile >pblls/tmp/andy - ls, ls - l >bls` приведет к подстановке в командную строку значения переменной `bls`. Если переменной `bls` не было предварительно присвоено никакого значения, то ее значением является символ пробел. Оболочка bash позволяет создание массивов. Для создания массива используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделенных пробелом. 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?

Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение - это единичный терм (term), обычно целочисленный. Целые числа можно записывать как последовательность цифр или в любом базовом формате.

Этот формат — radix#number, где radix (основание системы счисления) - любое число не более 26. Для большинства команд основания систем счисления это - 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток (%). Команда let берет два операнда и присваивает их переменной.

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

Оператор Синтаксис Результат !expr Если expr равно 0, возвращает 1; иначе 0 != expr1 !=expr2 Если expr1 не равно expr2, возвращает 1; иначе 0 % expr1%expr2 Возвращает остаток от деления expr1 на expr2 %= var=%expr Присваивает остаток от деления var на expr переменной var & expr1&expr2 Возвращает побитовое AND выражений expr1 и expr2 && expr1&&expr2 Если и expr1 и expr2 не равны нулю, возвращает 1; иначе 0 &= var &= expr Присваивает var побитовое AND переменных var и выражения expr * expr1 * expr2 Умножает expr1 на expr2 = var = expr Умножает expr на значение var и присваивает результат переменной var + expr1 + expr2 Складывает expr1 и expr2 += var += expr Складывает expr со значением var и результат присваивает var - -expr Операция отрицания expr (называется унарный минус) - expr1 - expr2 Вычитает expr2 из expr1 -= var -= expr Вычитает expr из значения var и присваивает результат var / expr / expr2 Делит expr1 на expr2 /= var /= expr Делит var на expr и присваивает результат var < expr1 < expr2 Если expr1 меньше, чем expr2, возвращает 1, иначе возвращает 0 « expr1« expr2 Сдвигает expr1 влево на expr2 бит «= var «= expr Побитовый сдвиг влево значения var на expr <= expr1 <= expr2 Если

exp1 меньше, или равно exp2 , возвращает 1; иначе возвращает 0 $= \text{var} = \text{exp}$
 Присваивает значение exp переменной var $== \text{exp1} == \text{exp2}$ Если exp1 равно exp2 .
 Возвращает 1; иначе возвращает 0 $> \text{exp1} > \text{exp2}$ 1 если exp1 больше, чем exp2 ;
 иначе 0 $>= \text{exp1} >= \text{exp2}$ 1 если exp1 больше, или равно exp2 ; иначе 0 $\gg \text{exp} \gg \text{exp2}$
 Сдвигает exp1 вправо на exp2 бит $\gg \text{var} \gg \text{exp}$ Побитовый сдвиг вправо значения
 var на exp $\wedge \text{exp1} \wedge \text{exp2}$ Исключающее OR выражений exp1 и exp2 $\wedge \text{var} \wedge \text{exp}$
 Присваивает var побитовое исключающее OR var и exp $| \text{exp1} | \text{exp2}$ Побитовое
 OR выражений exp1 и exp2 $| \text{var} | \text{exp}$ Присваивает var «исключающее OR» пе-
 ременной var и выражения exp $|| \text{exp1} || \text{exp2}$ 1 если или exp1 или exp2 являются
 нену- левыми значениями; иначе 0 $\sim \sim \text{exp}$ Побитовое дополнение до exp .

6. Что означает операция (())?

Условия оболочки bash.

7. Какие стандартные имена переменных Вам известны?

Имя переменной (идентификатор) — это строка символов, которая отличает
 эту переменную от других объектов программы (идентифицирует переменную
 в програм- ме). При задании имен переменным нужно соблюдать следующие
 правила: § первым символом имени должна быть буква. Остальные символы —
 буквы и цифры (прописные и строчные буквы различаются). Можно использо-
 вать символ «_»; § в имени нельзя использовать символ «.»; § число символов
 в имени не должно превышать 255; § имя переменной не должно совпадать с
 зарезервированными (служебными) словами языка. Var1, PATH, trash, mon, day,
 PS1, PS2 Другие стандартные переменные: —HOME — имя домашне- го каталога
 пользователя. Если команда cd вводится без аргументов, то происходит переход
 в каталог, указан- ный в этой переменной . —IFS — последовательность символов,
 являющихся разделителями в командной строке. Это символы пробел, табуляция
 и перевод строки(new line). —MAIL — командный процессор каждый раз перед
 выводом на экран промптера про- веряет содержимое файла, имя которого ука-
 зано 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. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set.

8. Что такое метасимволы?

Такие символы, как ' < > * ? | " & являются мета- символами 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, ее нужно заключить в одинарные кавычки. Строка, заключенная в двойные кавычки, экранирует все метасимволы, кроме \$, ', , ". Например, echo ab'|cd выведет на экран символ, echo ab'|cd выдаст 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 осуществит ее 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, заключенных в фигурные скобки. Удалить функцию можно с помощью команды `unset` с флагом `-f`. Команда `typeset` имеет четыре опции для работы с функциями:

- `-f` — перечисляет определенные на текущий момент функции;
- `-ft` — при последующем вызове функции иницирует ее трассировку;
- `-fx` — экспортирует все перечисленные функции в любые дочерние программы оболочек;
- `-fu` — обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную `FPATH`, отыскивая файл с одноименными именами функций, загружает его и вызывает эти функции.

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

`ls -lrf` Если есть `d`, то является файл каталогом

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

Используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделенных пробелом. Например, `set -A states Delaware Michigan "New Jersey"`. Далее можно сделать добавление в массив, например, `states[49]=Alaska`. Индексация массивов начинается с нулевого элемента. В командном процессоре `Си` имеется еще несколько стандартных переменных.

Значение всех переменных можно просмотреть с помощью команды `set`. Наиболее распространенным является сокращение, избавляющееся от слова `let` в про- граммах оболочек. Если объявить переменные целыми значениями, любое присвоение автоматически трактуется как арифметическое. Используйте `typeset -i` для объявления и присвоения переменной, и при последующем использовании она становится целой. Или можете использовать ключевое слово `integer` (псевдоним для `typeset -l`) и объявлять переменные целыми. Таким образом, выражения типа `x=y+z` воспринимаются как арифметические. Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключенных в фигурные скобки. Удалить функцию можно с помощью команды `unset` с флагом `-f`. Команда `typeset` имеет четыре опции для работы с функциями: `-f` — перечисляет определенные на текущий момент функции; `-ft` — при последующем вызове функции инициализирует ее трассировку; `-fx` — экспортирует все перечисленные функции в любые дочерние программы оболочек; `-fu` — обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную `FPATH`, отыскивая файл с одноименными именами функций, загружает его и вызывает эти функции. В переменные `mon` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введенную информацию и игнорировать ее. Изъять переменную из программы можно с помощью команды `unset`.

14. Как передаются параметры в командные файлы?

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.

Использование комбинации символов `$0` приводит к подстановке вместо нее имени данного командного файла. Примере: пусть к командному файлу `where`

имеется доступ по выполнению и этот командный файл содержит следующий конвейер: `who | grep $1` Если Вы введете с терминала команду: `where andy`, то в случае, если пользователь, зарегистрированный в ОС UNIX под именем `andy`, в данный момент работает в ОС UNIX, на терминале будет выведена строка, содержащая номер терминала, используемого указанным пользователем. Если же в данный момент этот пользователь не работает в ОС UNIX, то на терминале не будет выведено ничего. Команда `grep` производит контекстный поиск в тексте, поступающем со стандартного ввода, для нахождения в этом тексте строк, содержащих последовательности символов, переданные ей в качестве аргументов, и выводит результаты своей работы на стандартный вывод. В этом примере команда `grep` используется как фильтр, обеспечивающий ввод со стандартного ввода и вывод всех строк, содержащих последовательность символов `andy`, на стандартный вывод. В ходе интерпретации этого файла командным процессором вместо комбинации символов `$1` осуществляется подстановка значения первого и единственного параметра `andy`. Если предположить, что пользователь, зарегистрированный в ОС UNIX под именем `andy`, в данный момент работает в ОС UNIX, то на терминале Вы увидите примерно следующее: `$ where andy andy ttyG Jan 14 09:12 $` Определим функцию, которая изменяет каталог и печатает список файлов: `$ function clist { > cd $1 > ls > }`. Теперь при вызове команды `clist` каталог будет изменен каталог и выведено его содержимое.

15. Назовите специальные переменные языка `bash` и их назначение.

`$*` — 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`-ному элементу массива;

`${name[*]}` — перечисляет все элементы массива, разделенные 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; это выражение работает противоположно `{name-value}`. Если переменная определена, то подставляется `value`;

`${name#pattern}` — представляет значение переменной `name` с удаленным самым коротким левым образцом (`pattern`);

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

`$#` вместо нее будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполне

7 Выводы

В процессе выполнения лабораторной работы я изучила основы программирования в оболочке ОС UNIX/Linux. Научилась писать небольшие командные файлы.

Список литературы

1. Руководство к выполнению лабораторной работы