

Лабораторная работа №10

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

Демидова Екатерина Алексеевна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
5	Контрольные вопросы	12
6	Выводы	21
	Список литературы	22

Список иллюстраций

4.1	Скрипт 1	8
4.2	Вывод скрипта 1	8
4.3	Скрипт 2	9
4.4	Вывод скрипта 2	9
4.5	Скрипт 3	10
4.6	Вывод скрипта 3	10
4.7	Скрипт 4	11
4.8	Вывод скрипта 4	11

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

При интерактивной работе с операционной системой пользователь постоянно сталкивается с необходимостью отдавать ей команды. В UNIX эту работу выполняет программа, которая называется командным процессором (shell). Иногда командный процессор называют шеллом или оболочкой, или интерпретатором команд (последнее неточно, потому что круг задач командного процессора шире, чем интерпретация команд).

Shell действует как посредник между вами и ядром операционной системы. Ядро – это часть операционной системы, которая всегда находится в памяти компьютера, это программа. Командный процессор преобразует ваши команды в инструкции для операционной системы, а операционная система превращает их в инструкции для аппаратных средств компьютера. По сути, именно оболочка придает определенную “персонализацию” системам UNIX.

Командный процессор выполняет в системе следующие задачи:

- интерпретация команд пользователя, в том числе разбор командной строки;
- запуск программ;
- организация перенаправлений потоков между процессами;
- интерпретация языка скриптов и их выполнение;
- управление заданиями;
- интерпретация шаблонов имен файлов;

- подстановка имен файлов в командную строку.

Кроме того, shell является мощным языком программирования.

Любая команда, являющаяся отдельной программой, т.е. не встроенной в интерпретатор, будет выполняться одинаково, независимо от shell'a. Например, если вы хотите что-то напечатать, команда печати всегда работает одинаково.

Некоторые команды встроены в shell, т.е. они являются частью программы оболочки и, как следствие, могут выполняться по-разному в зависимости от оболочки, в которой они запускаются. Есть три вида команд, встроенных в shell:

- общие команды запускаются несколько быстрее, так как они являются частью оболочки;
- команды адаптации позволяют адаптировать оболочку.
- команды программирования образуют язык программирования оболочки.

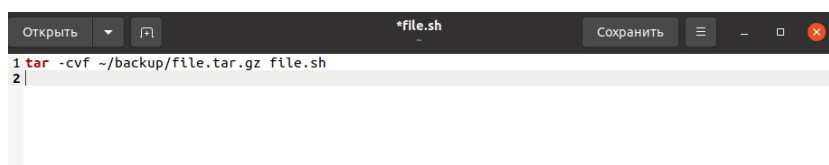
При смене shell'a вы не заметите никакой разницы между общими командами, которые встроены просто для повышения быстродействия. Однако команды адаптации и программирования изменяются.

Примером общих команд, встроенных в оболочку, служат команды `cd`, `echo`, `pwd`, `login`, `umask`.

Адаптация включает в себя создание новых команд или новых имен для старых команд и привлечение новых возможностей. Примером общепринятой адаптации служит изменение стимула (или приглашения системы).[1].

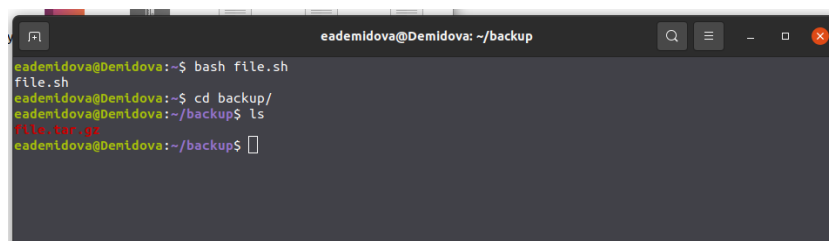
4 Выполнение лабораторной работы

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. (рис. 4.1, 4.2)



```
1 tar -cvf ~/backup/file.tar.gz file.sh
2 |
```

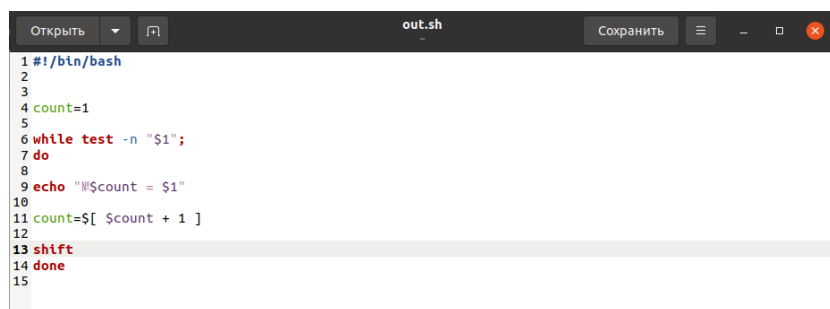
Рис. 4.1: Скрипт 1



```
eademidova@Demidova: ~/backup
eademidova@Demidova:~$ bash file.sh
file.sh
eademidova@Demidova:~$ cd backup/
eademidova@Demidova:~/backup$ ls
file.tar.gz
eademidova@Demidova:~/backup$
```

Рис. 4.2: Вывод скрипта 1

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..(рис. 4.3, 4.4)

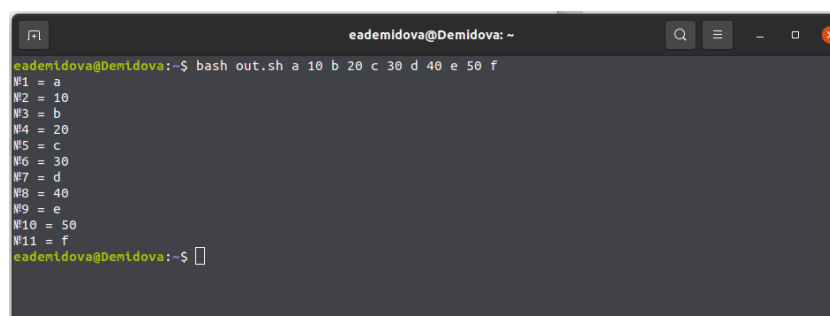


```

1 #!/bin/bash
2
3
4 count=1
5
6 while test -n "$1";
7 do
8
9 echo "$count = $1"
10
11 count=$((count + 1))
12
13 shift
14 done
15

```

Рис. 4.3: Скрипт 2



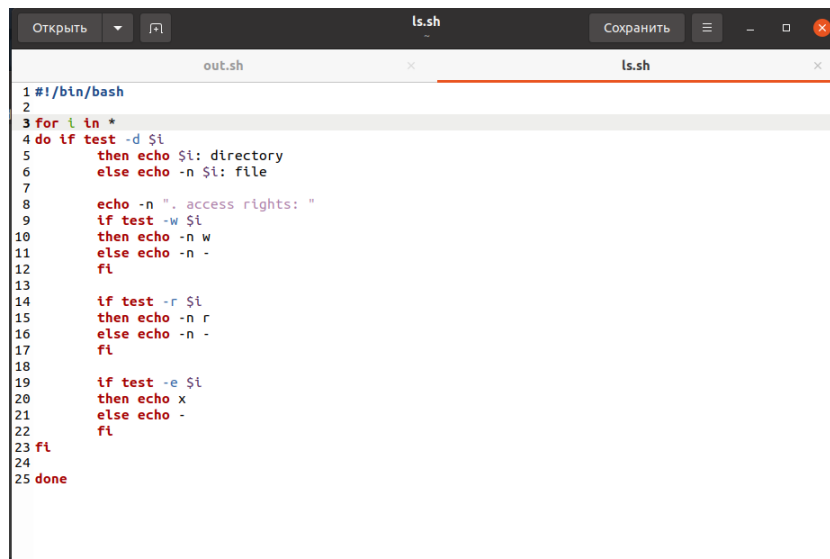
```

eademidova@Demidova: ~
eademidova@Demidova:~$ bash out.sh a 10 b 20 c 30 d 40 e 50 f
#1 = a
#2 = 10
#3 = b
#4 = 20
#5 = c
#6 = 30
#7 = d
#8 = 40
#9 = e
#10 = 50
#11 = f
eademidova@Demidova:~$

```

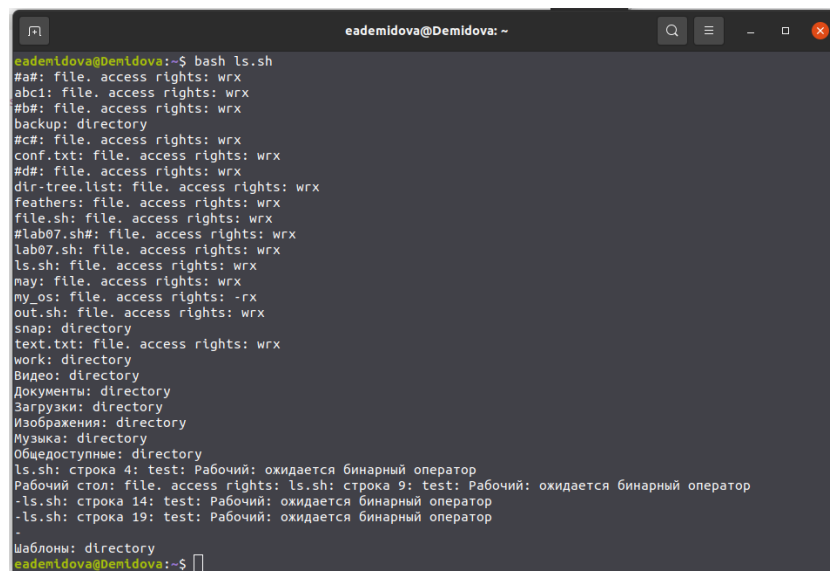
Рис. 4.4: Вывод скрипта 2

3. Написать командный файл — аналог команды `ls` (без использования самой этой ко- манды и команды `dir`). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога. (рис. 4.5, 4.6)

A screenshot of a code editor window titled 'ls.sh'. The editor shows a shell script with 25 lines of code. The script starts with a shebang line, followed by a loop that iterates over all files and directories in the current directory. For each item, it checks if it's a directory, then if it's a file, and then if it has execute permissions. The script uses 'echo' to print the name and permissions of each item. The code is as follows:

```
1#!/bin/bash
2
3for i in *
4do if test -d $i
5then echo $i: directory
6else echo -n $i: file
7
8    echo -n ". access rights: "
9    if test -w $i
10then echo -n w
11else echo -n -
12fi
13
14    if test -r $i
15then echo -n r
16else echo -n -
17fi
18
19    if test -e $i
20then echo x
21else echo -
22fi
23fi
24
25done
```

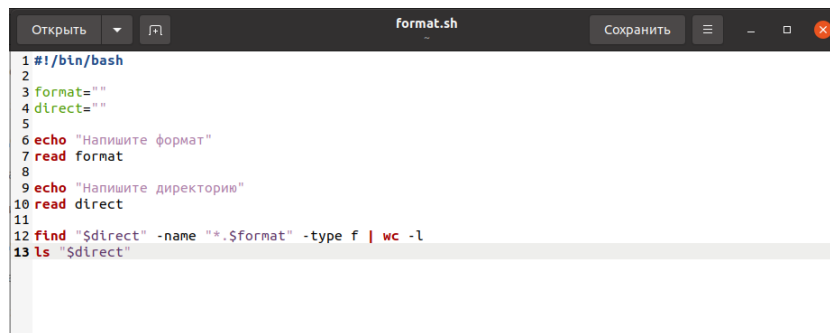
Рис. 4.5: Скрипт 3

A screenshot of a terminal window titled 'eademidova@Demidova: ~'. The terminal shows the output of the 'ls.sh' script. The output lists the names and permissions of all files and directories in the current directory. The output is as follows:

```
eademidova@Demidova:~$ bash ls.sh
##: file. access rights: wrx
abc1: file. access rights: wrx
#b#: file. access rights: wrx
backup: directory
#c#: file. access rights: wrx
conf.txt: file. access rights: wrx
#d#: file. access rights: wrx
dir-tree.list: file. access rights: wrx
feathers: file. access rights: wrx
file.sh: file. access rights: wrx
#lab07.sh#: file. access rights: wrx
lab07.sh: file. access rights: wrx
ls.sh: file. access rights: wrx
may: file. access rights: wrx
my_os: file. access rights: -rx
out.sh: file. access rights: wrx
snap: directory
text.txt: file. access rights: wrx
work: directory
Видео: directory
Документы: directory
Загрузки: directory
Изображения: directory
Музыка: directory
Общедоступные: directory
ls.sh: строка 4: test: Рабочий: ожидается бинарный оператор
Рабочий стол: file. access rights: ls.sh: строка 9: test: Рабочий: ожидается бинарный оператор
-ls.sh: строка 14: test: Рабочий: ожидается бинарный оператор
-ls.sh: строка 19: test: Рабочий: ожидается бинарный оператор
-
Шаблоны: directory
eademidova@Demidova:~$
```

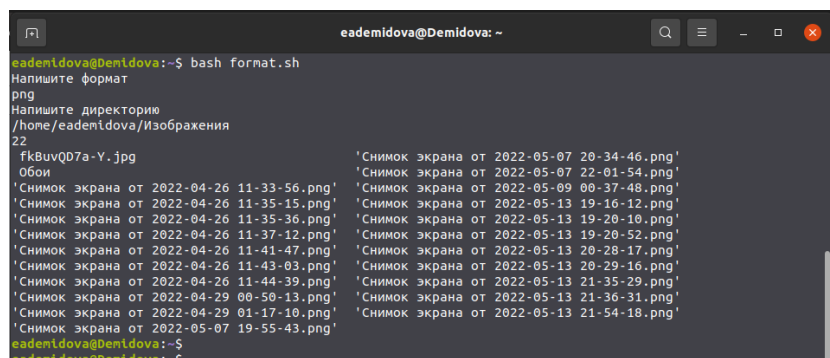
Рис. 4.6: Вывод скрипта 3

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 (рис. 4.7, 4.8)



```
1 #!/bin/bash
2
3 format=""
4 direct=""
5
6 echo "Напишите формат"
7 read format
8
9 echo "Напишите директорию"
10 read direct
11
12 find "$direct" -name ".*$format" -type f | wc -l
13 ls "$direct"
```

Рис. 4.7: Скрипт 4



```
eademidova@Demidova:~$ bash format.sh
Напишите формат
png
Напишите директорию
/home/eademidova/Изображения
22
fkBuvQ07a-Y.jpg 'Снимок экрана от 2022-05-07 20-34-46.png'
обои 'Снимок экрана от 2022-05-07 22-01-54.png'
'Снимок экрана от 2022-04-26 11-33-56.png' 'Снимок экрана от 2022-05-09 00-37-48.png'
'Снимок экрана от 2022-04-26 11-35-15.png' 'Снимок экрана от 2022-05-13 19-16-12.png'
'Снимок экрана от 2022-04-26 11-35-36.png' 'Снимок экрана от 2022-05-13 19-20-10.png'
'Снимок экрана от 2022-04-26 11-37-12.png' 'Снимок экрана от 2022-05-13 19-20-52.png'
'Снимок экрана от 2022-04-26 11-41-47.png' 'Снимок экрана от 2022-05-13 20-28-17.png'
'Снимок экрана от 2022-04-26 11-43-03.png' 'Снимок экрана от 2022-05-13 20-29-16.png'
'Снимок экрана от 2022-04-26 11-44-39.png' 'Снимок экрана от 2022-05-13 21-35-29.png'
'Снимок экрана от 2022-04-29 00-50-13.png' 'Снимок экрана от 2022-05-13 21-36-31.png'
'Снимок экрана от 2022-04-29 01-17-10.png' 'Снимок экрана от 2022-05-13 21-54-18.png'
'Снимок экрана от 2022-05-07 19-55-43.png'
eademidova@Demidova:~$
eademidova@Demidova:~$
```

Рис. 4.8: Вывод скрипта 4

5 Контрольные вопросы

1. 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; –С-оболочка - добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя С-подобный синтаксис команд, и сохраняет историю выполненных команд; –оболочка Корна - 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).
2. POSIX (Portable Operating System Interface for Computer Environments)- интерфейс переносимой операционной системы для компьютерных сред. Представляет 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 гра-

фический 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. 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 $mark` переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того, чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: `${имя переменной}` например, использование команд `b=/tmp/andy-ls -l myfile > blsls/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. Команда `let` также расширяет другие выражения `let`, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения. Команда `let` не ограничена простыми арифметическими выражениями. Табл. 10.1 показывает полный набор `let`-операций.
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 `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 `expr1` меньше, или равно `expr2`, возвращает 1; иначе возвращает 0
`= var = expr` Присваивает значение `expr` переменной `var`
`== expr1==expr2` Если `expr1` равно `expr2`. Возвращает 1; иначе возвращает 0
`> expr1 > expr2` 1 если `expr1` больше, чем `expr2`; иначе 0
`>= expr1 >= expr2` 1 если `expr1` больше, или равно `expr2`; иначе 0
`>> expr >> expr2` Сдвигает `expr1` вправо на `expr2` бит
`>= var >= expr` Побитовый сдвиг вправо значения `var` на `expr`
`^ expr1 ^ expr2` Исключающее OR

выражений `exp1` и `exp2` `^= var ^= exp` Присваивает `var` побитовое исключающее OR `var` и `exp` `| exp1 | exp2` Побитовое OR выражений `exp1` и `exp2` `|= var |= exp` Присваивает `var` «исключающее OR» переменной `var` и выражения `exp` `|| exp1 || exp2` 1 если или `exp1` или `exp2` являются ненулевыми значениями; иначе 0 `~exp` Побитовое дополнение до `exp`.

6. Условия оболочки `bash`, помещаются в двойные скобки `--(())`.
7. Имя переменной (идентификатор) — это строка символов, которая отличает эту переменную от других объектов программы (идентифицирует переменную в программе). 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 использовать 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 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. Это символы 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. В командном процессоре Си имеются еще несколько стандартных переменных. Значение всех переменных

можно просмотреть с помощью команды `set`.

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, ее нужно заключить в одинарные кавычки. Строка, заключенная в двойные кавычки, экранирует все метасимволы, кроме `$, ', , "`. Например, `-echo выведет на экран символ, -echo ab'|cd` выдаст 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 ее интерпретацию.
11. 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. `ls -lrt` Если есть `d`, то является файл каталогом
13. 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 программах 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 иницирует ее трассировку; `-fx` — экспортирует все перечисленные функции в любые дочерние программы оболочек; `-fu` — обозначает указанные функции как автоматически загружаемые. Автома-

тически 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 отобразить всю избыточно введенную информацию и игнорировать ее. Изъять переменную из программы можно с помощью команды `unset`.

14. 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 нее 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 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 терминал будет выведена строка, содержащая номер терминала, используемого указанным пользователем. Если же в данный момент этот пользователь не работает в ОС UNIX, то на терминал 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. 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;

- `${name?value}` — останавливает выполнение, если имя переменной не определено, и выводит `value`, как сообщение об ошибке;
- `${name+value}` — 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 выполнение.

6 Выводы

Изучила основы программирования в оболочке ОС UNIX/Linux. Научилась писать небольшие командные файлы.

Список литературы

1. Командные процессоры ОС UNIX [Электронный ресурс]. life-prog.ru, 2014.
URL: https://life-prog.ru/1_54716_glava--komandnie-protssessori-os-UNIX.html.