

Лабораторная работа No 10.

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

Боровиков Даниил Александрович

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	11
4	Контрольные вопросы	12

Список иллюстраций

2.1	Создание файла lab10_1.sh	6
2.2	Написание скрипта lab10_1.sh	6
2.3	Право на выполнение и запуск lab10_1.sh	7
2.4	Создание файла lab10_2.sh	7
2.5	Написание скрипта lab10_2.sh	7
2.6	Право на выполнение и запуск lab10_2.sh	8
2.7	Создание файла lab10_3.sh	8
2.8	Написание скрипта lab10_3.sh	8
2.9	Право на выполнение и запуск lab10_3.sh	9
2.10	Создание файла lab10_3.sh	9
2.11	Написание скрипта lab10_3.sh	10
2.12	Право на выполнение и запуск lab10_4.sh	10

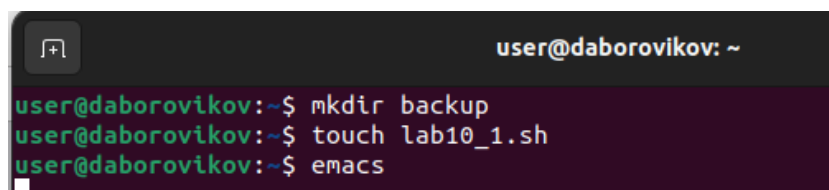
Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

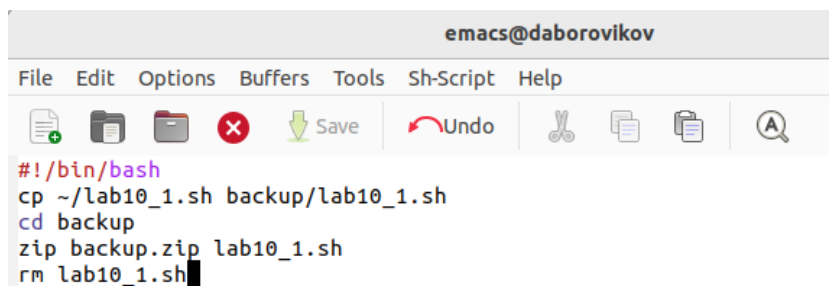
Откроем терминал. Создадим в домашнем каталоге папку backup и файл lab10_1.sh. Откроем emacs.(рис. fig. 2.1).



```
user@daborovikov: ~  
user@daborovikov:~$ mkdir backup  
user@daborovikov:~$ touch lab10_1.sh  
user@daborovikov:~$ emacs
```

Рис. 2.1: Создание файла lab10_1.sh

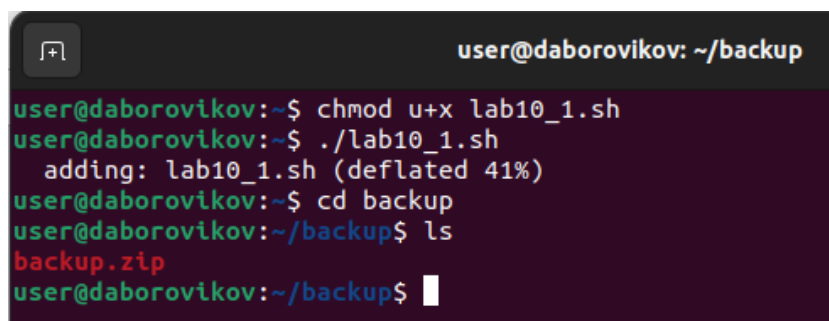
Напишем скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в домашнем каталоге.(рис. fig. 2.2).



```
emacs@daborovikov  
File Edit Options Buffers Tools Sh-Script Help  
[Icons]  
#!/bin/bash  
cp ~/lab10_1.sh backup/lab10_1.sh  
cd backup  
zip backup.zip lab10_1.sh  
rm lab10_1.sh
```

Рис. 2.2: Написание скрипта lab10_1.sh

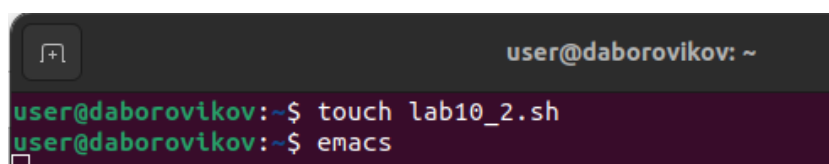
В терминале дадим файлу право на исполнение. Запустим файл и проверим в каталоге backup его копию.(рис. fig. 2.3).



```
user@daborovikov: ~/backup
user@daborovikov:~$ chmod u+x lab10_1.sh
user@daborovikov:~$ ./lab10_1.sh
  adding: lab10_1.sh (deflated 41%)
user@daborovikov:~$ cd backup
user@daborovikov:~/backup$ ls
backup.zip
user@daborovikov:~/backup$
```

Рис. 2.3: Право на выполнение и запуск lab10_1.sh

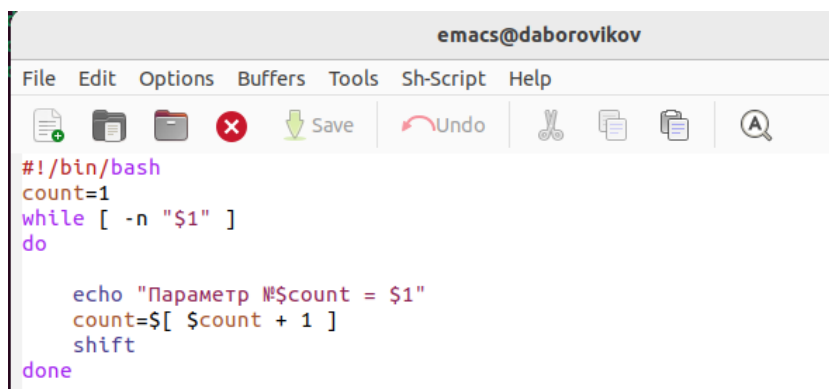
Откроем терминал. Создадим в домашнем и файл lab10_2.sh. Откроем emacs.(рис. fig. 2.4).



```
user@daborovikov: ~
user@daborovikov:~$ touch lab10_2.sh
user@daborovikov:~$ emacs
```

Рис. 2.4: Создание файла lab10_2.sh

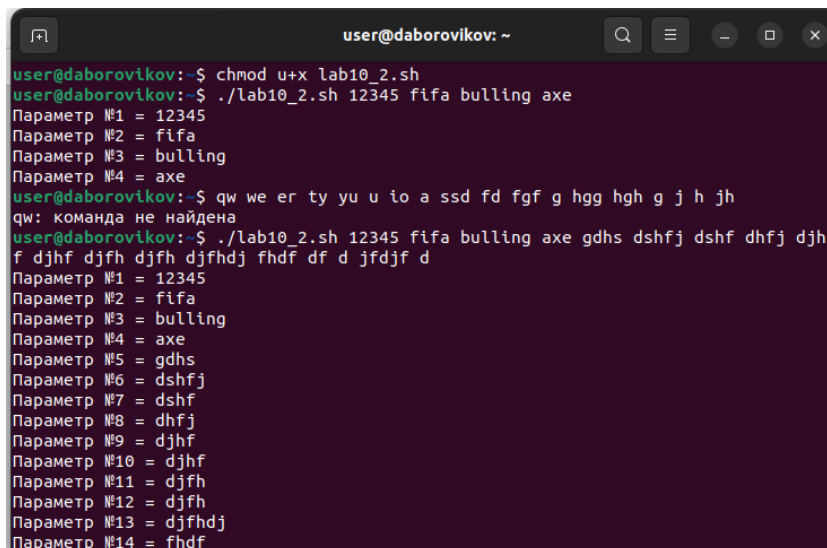
Напишем пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять(рис. fig. 2.5).



```
emacs@daborovikov
File Edit Options Buffers Tools Sh-Script Help
[Icons]
#!/bin/bash
count=1
while [ -n "$1" ]
do
    echo "Параметр №$count = $1"
    count=$((count + 1))
    shift
done
```

Рис. 2.5: Написание скрипта lab10_2.sh

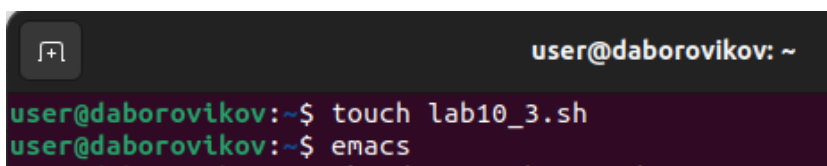
В терминале дадим файлу право на исполнение. Запускаем файл.(рис. fig. 2.6).



```
user@daborovikov: ~  
user@daborovikov:~$ chmod u+x lab10_2.sh  
user@daborovikov:~$ ./lab10_2.sh 12345 fifa bulling axe  
Параметр №1 = 12345  
Параметр №2 = fifa  
Параметр №3 = bulling  
Параметр №4 = axe  
user@daborovikov:~$ qw we er ty yu u io a ssd fd fgf g hgg hgh g j h jh  
qw: команда не найдена  
user@daborovikov:~$ ./lab10_2.sh 12345 fifa bulling axe gdhs dshfj dshf dhfj djh  
f djhf djfh djfh djfhdj fhd f df d jfdjf d  
Параметр №1 = 12345  
Параметр №2 = fifa  
Параметр №3 = bulling  
Параметр №4 = axe  
Параметр №5 = gdhs  
Параметр №6 = dshfj  
Параметр №7 = dshf  
Параметр №8 = dhfj  
Параметр №9 = djhf  
Параметр №10 = djhf  
Параметр №11 = djfh  
Параметр №12 = djfh  
Параметр №13 = djfhdj  
Параметр №14 = fhd f
```

Рис. 2.6: Право на выполнение и запуск lab10_2.sh

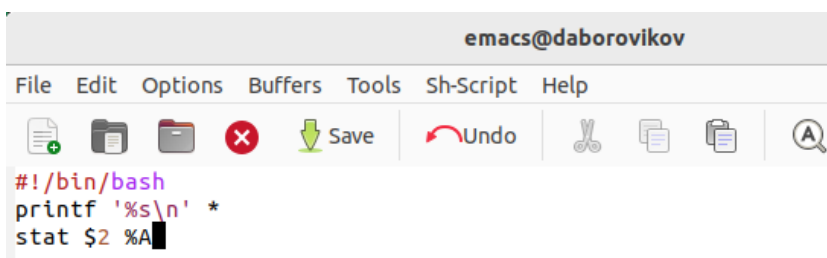
Откроем терминал. Создадим в домашнем и файл lab10_3.sh. Откроем emacs.(рис. fig. 2.7).



```
user@daborovikov: ~  
user@daborovikov:~$ touch lab10_3.sh  
user@daborovikov:~$ emacs
```

Рис. 2.7: Создание файла lab10_3.sh

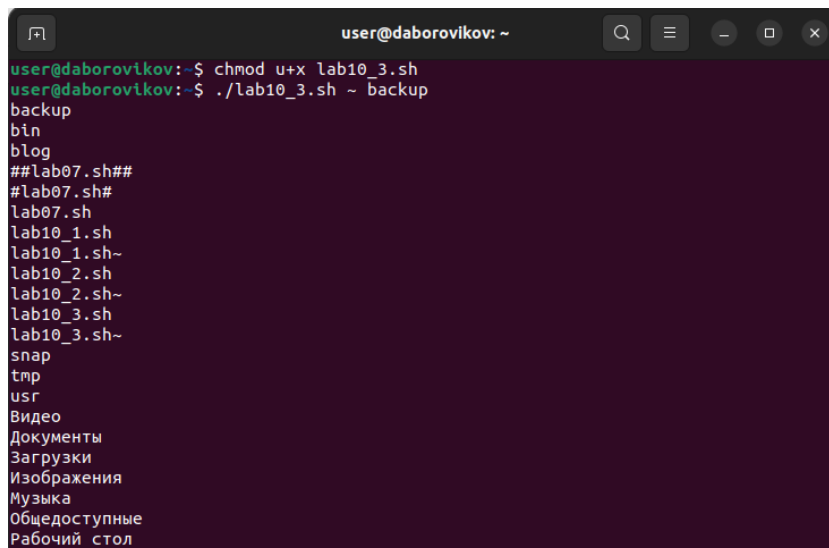
Напишем командный файл — аналог команды ls (без использования самой этой ко- манды и команды dir). (рис. fig. 2.8).



```
emacs@daborovikov  
File Edit Options Buffers Tools Sh-Script Help  
[Icons]  
#!/bin/bash  
printf '%s\n' *  
stat $2 %A
```

Рис. 2.8: Написание скрипта lab10_3.sh

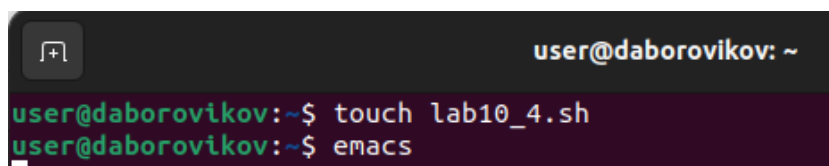
В терминале дадим файлу право на исполнение. Запускаем файл.(рис. fig. 2.9).

A terminal window titled 'user@daborovikov: ~' with search, menu, and window control icons in the title bar. The terminal shows the following commands and output:

```
user@daborovikov:~$ chmod u+x lab10_3.sh
user@daborovikov:~$ ./lab10_3.sh ~ backup
backup
bin
blog
##lab07.sh##
#lab07.sh#
lab07.sh
lab10_1.sh
lab10_1.sh~
lab10_2.sh
lab10_2.sh~
lab10_3.sh
lab10_3.sh~
snap
tmp
usr
Видео
Документы
Загрузки
Изображения
Музыка
Общедоступные
Рабочий стол
```

Рис. 2.9: Право на выполнение и запуск lab10_3.sh

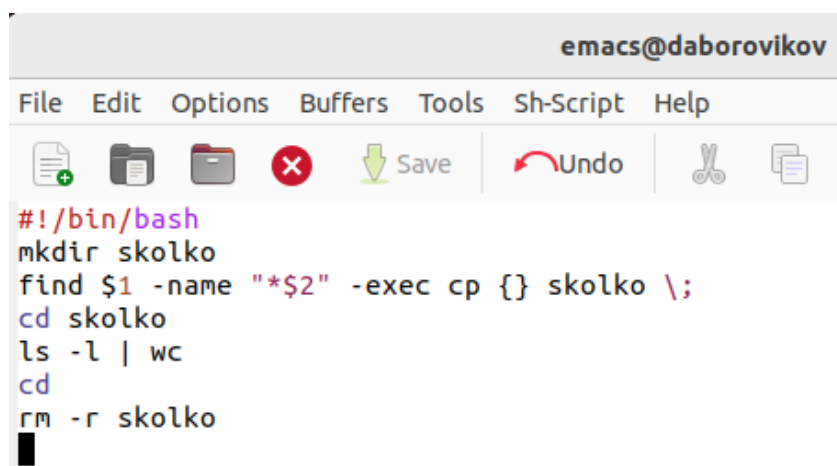
Откроем терминал. Создадим в домашнем и файл lab10_4.sh. Откроем emacs.(рис. fig. 2.10).

A terminal window titled 'user@daborovikov: ~' with search, menu, and window control icons in the title bar. The terminal shows the following commands and output:

```
user@daborovikov:~$ touch lab10_4.sh
user@daborovikov:~$ emacs
```

Рис. 2.10: Создание файла lab10_3.sh

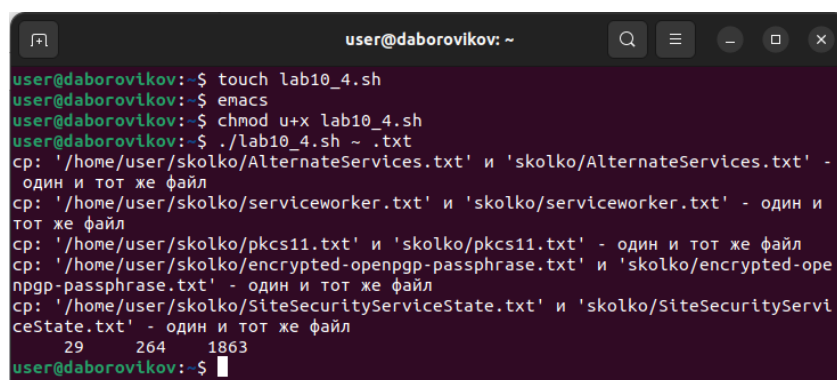
Напишем командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. (рис. fig. 2.11).



```
emacs@daborovikov
File Edit Options Buffers Tools Sh-Script Help
[Icons: New, Open, Save, Undo, Cut, Copy]
#!/bin/bash
mkdir skolko
find $1 -name "$*$2" -exec cp {} skolko \;
cd skolko
ls -l | wc
cd
rm -r skolko
```

Рис. 2.11: Написание скрипта lab10_3.sh

В терминале дадим файлу право на исполнение. Запускаем файл(рис. fig. 2.12).



```
user@daborovikov: ~
user@daborovikov:~$ touch lab10_4.sh
user@daborovikov:~$ emacs
user@daborovikov:~$ chmod u+x lab10_4.sh
user@daborovikov:~$ ./lab10_4.sh ~ .txt
cp: '/home/user/skolko/AlternateServices.txt' и 'skolko/AlternateServices.txt' -
  один и тот же файл
cp: '/home/user/skolko/serviceworker.txt' и 'skolko/serviceworker.txt' - один и
  тот же файл
cp: '/home/user/skolko/pkcs11.txt' и 'skolko/pkcs11.txt' - один и тот же файл
cp: '/home/user/skolko/encrypted-openpgp-passphrase.txt' и 'skolko/encrypted-ope
  npgp-passphrase.txt' - один и тот же файл
cp: '/home/user/skolko/SiteSecurityServiceState.txt' и 'skolko/SiteSecurityServi
  ceState.txt' - один и тот же файл
      29      264     1863
user@daborovikov:~$
```

Рис. 2.12: Право на выполнение и запуск lab10_4.sh

3 Выводы

В ходе лабораторной работы мы изучили основы программирования в оболочке ОС UNIX/Linux. Научились писать небольшие командные файлы.

4 Контрольные вопросы

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

Командный процессор 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 на оболочкой Борна, использующая 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).

2. 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 month day trash`». В переменные `month` 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 (`newline`).

- 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 Youhavemail(y 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 мета символа. Экранирование может быть осуществлено с помощью предшествующего 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. 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 `unsetc`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. 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. 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; - \$# возвращает целое число количеств 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.