

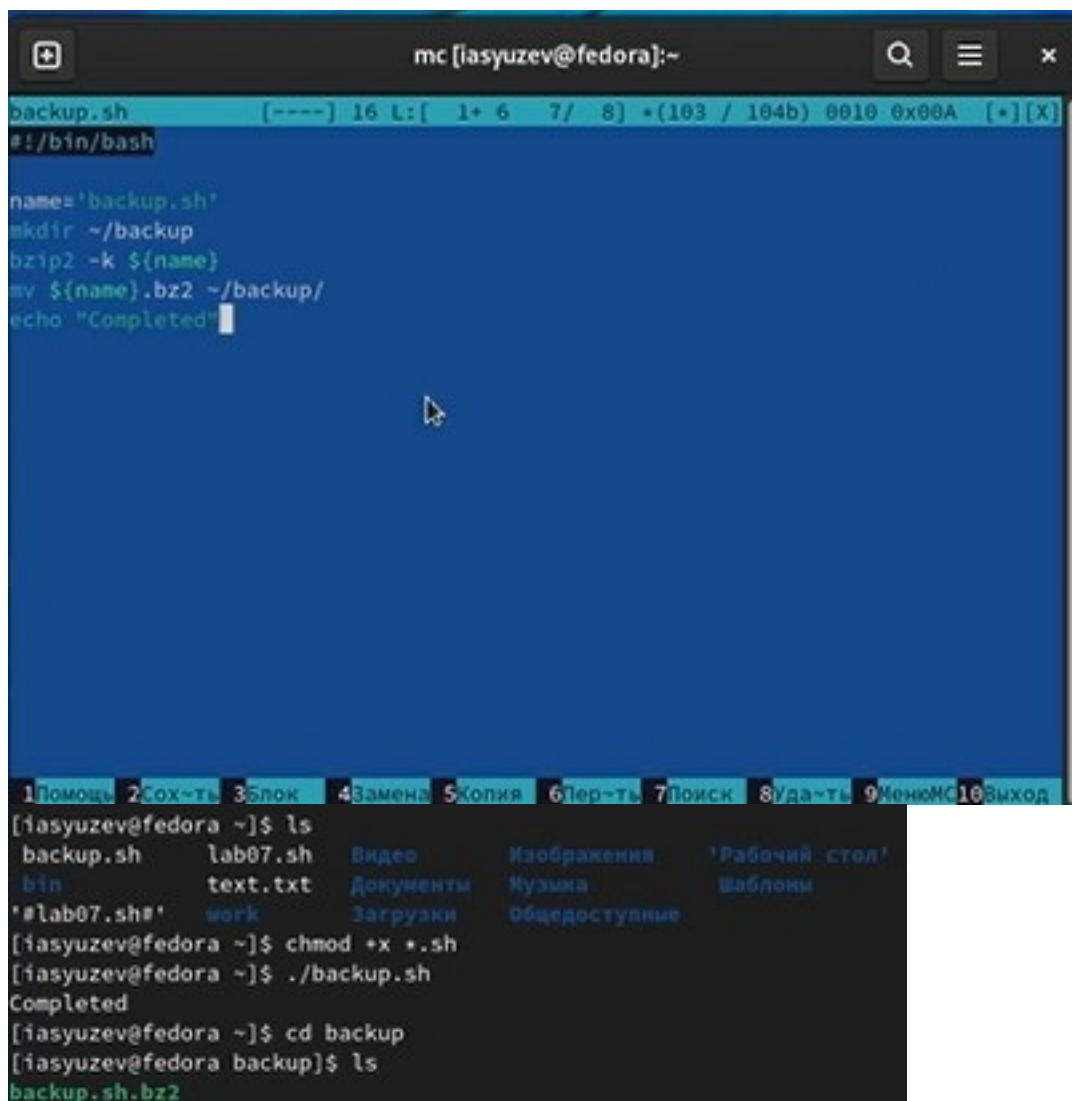
## Отчёт по лабораторной работе №10

### Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux, научиться писать небольшие командные файлы

### Ход выполнения работы

1. Создадим исполняемый файл *backup.sh*, который будет создавать копию самого себя с архивацией



```
mc [iasyuzev@fedora]:~
backup.sh [-----] 16 L: [ 1+ 6 7/ 8] *(103 / 104b) 0010 0x00A [*][X]
#!/bin/bash

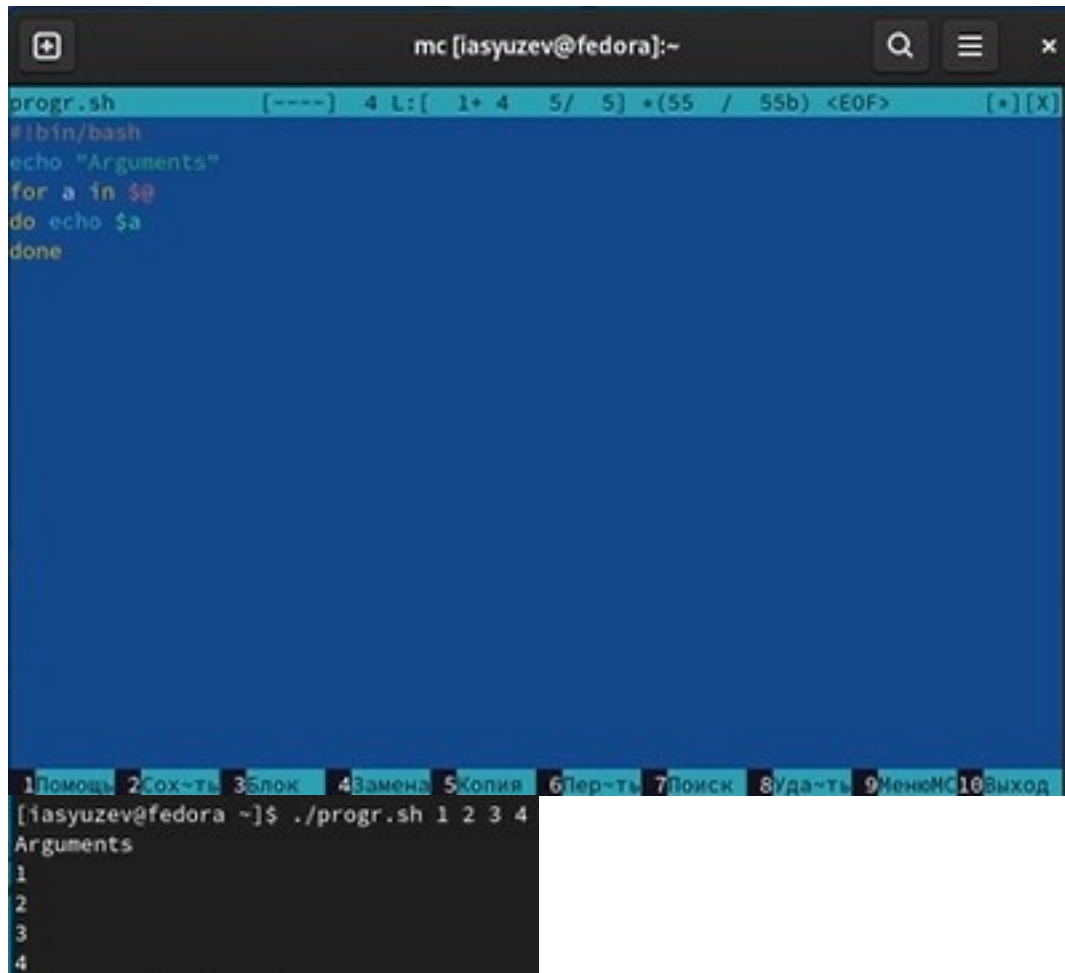
name='backup.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Completed"
```

1Помощь 2Сох-ть 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9МенюMC10Выход

```
[iasyuzev@fedora ~]$ ls
backup.sh  lab07.sh  Видео  Изображения  'Рабочий стол'
bin        text.txt  Документы  Музыка  Шаблоны
'#lab07.sh#' work  Загрузки  Общедоступные

[iasyuzev@fedora ~]$ chmod +x *.sh
[iasyuzev@fedora ~]$ ./backup.sh
Completed
[iasyuzev@fedora ~]$ cd backup
[iasyuzev@fedora backup]$ ls
backup.sh.bz2
```

2. Создадим исполняемый файл `progr.sh`, который будет обрабатывать аргументы из командной строки и последовательно распечатывать их

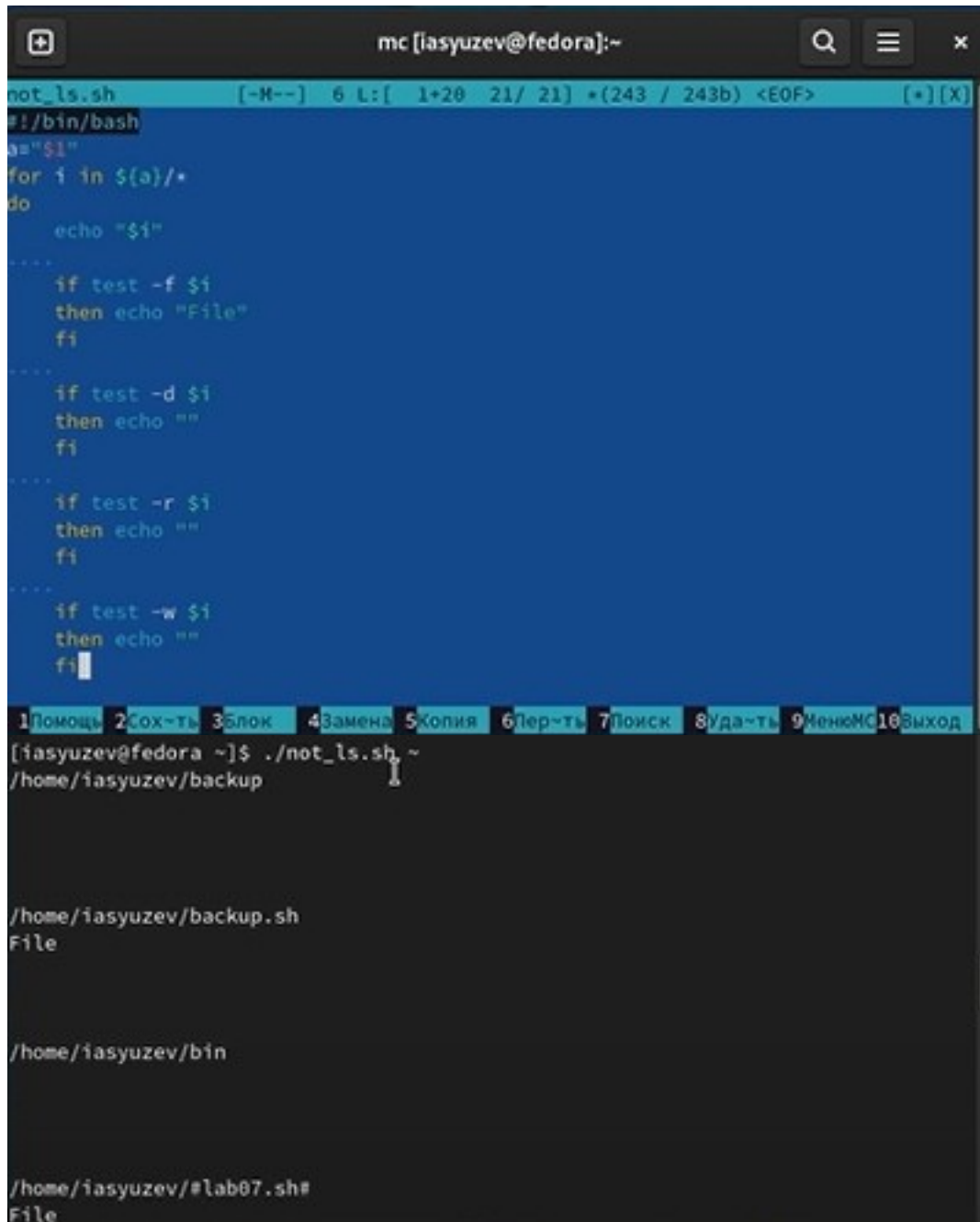


The screenshot shows a terminal window titled `mc [iasyuzev@fedora]:~`. The terminal content is as follows:

```
progr.sh [----] 4 L: [ 1+ 4 5/ 5] *(55 / 55b) <EOF> [*][X]
#bin/bash
echo "Arguments"
for a in $@
do echo $a
done

1Помощь 2Сох-ть 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9МенюМС10Выход
[iasyuzev@fedora ~]$ ./progr.sh 1 2 3 4
Arguments
1
2
3
4
```

3. Создадим исполняемый файл `not_ls.sh`, который будет выполнять задачи команды `ls`, то есть выводить на экран содержимое каталога



```
mc [iasyuzev@fedora]:~
not_ls.sh [-M--] 6 L: [ 1+20 21/ 21] *(243 / 243b) <EOF> [*] [X]
#!/bin/bash
a="$1"
for i in ${a}/*
do
    echo "$i"
    ...
    if test -f $i
    then echo "File"
    fi
    ...
    if test -d $i
    then echo ""
    fi
    ...
    if test -r $i
    then echo ""
    fi
    ...
    if test -w $i
    then echo ""
    fi
done

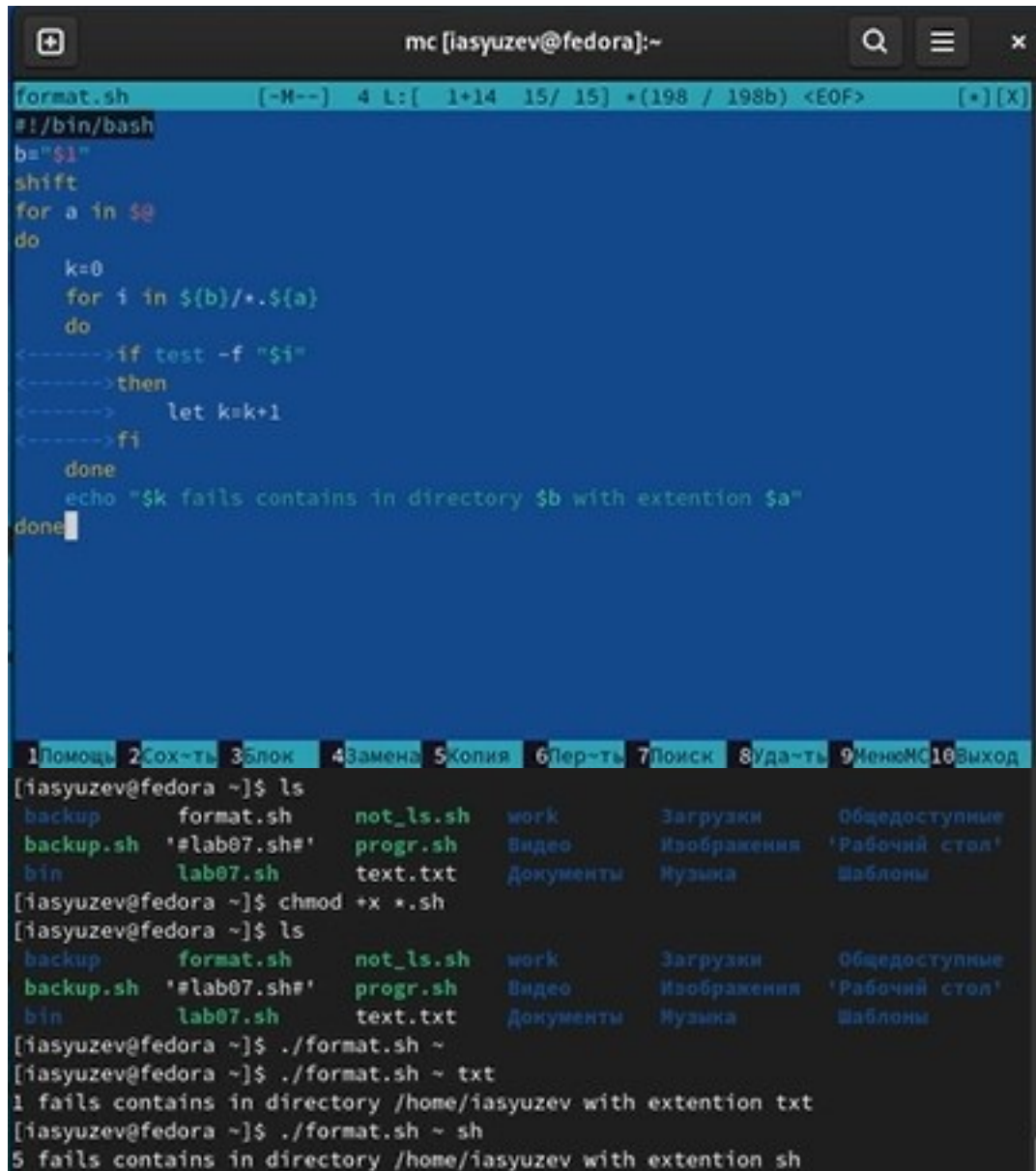
1Помощь 2Сох-ть 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9МенюMC 10Выход
[iasyuzev@fedora ~]$ ./not_ls.sh ~
/home/iasyuzev/backup

/home/iasyuzev/backup.sh
File

/home/iasyuzev/bin

/home/iasyuzev/#lab07.sh#
File
```

4. Создадим исполняемый файл `format.sh`, который будет выводить на экран количество файлов заданного формата в заданном каталоге



```
mc [iasyuzev@fedora]:~
format.sh [-M--] 4 L:[ 1+14 15/ 15] *(198 / 198b) <EOF> [*][X]
#!/bin/bash
b="$1"
shift
for a in $@
do
    k=0
    for i in ${b}/*.${a}
    do
        if test -f "$i"
        then
            let k=k+1
        fi
    done
    echo "$k fails contains in directory $b with extention $a"
done

1Помощь 2Сох-ть 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9Меню10Выход
[iasyuzev@fedora ~]$ ls
backup      format.sh  not_ls.sh  work       Загрузки  Общедоступные
backup.sh   '#lab07.sh#' progr.sh   Видео      Изображения 'Рабочий стол'
bin         lab07.sh   text.txt   Документы  Музыка    Шаблоны
[iasyuzev@fedora ~]$ chmod +x *.sh
[iasyuzev@fedora ~]$ ls
backup      format.sh  not_ls.sh  work       Загрузки  Общедоступные
backup.sh   '#lab07.sh#' progr.sh   Видео      Изображения 'Рабочий стол'
bin         lab07.sh   text.txt   Документы  Музыка    Шаблоны
[iasyuzev@fedora ~]$ ./format.sh ~
[iasyuzev@fedora ~]$ ./format.sh ~ txt
1 fails contains in directory /home/iasyuzev with extention txt
[iasyuzev@fedora ~]$ ./format.sh ~ sh
5 fails contains in directory /home/iasyuzev with extention sh
```

## Выводы

Я изучил основы программирования в оболочке ОС UNIX/Linux, а также научился писать небольшие командные файлы

## Ответы на 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 (BourneShellили 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 BourneAgainShell(опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании FreeSoftwareFoundation). #### 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 Electricaland 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 Напримеркоманда«{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 setc флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, «set -Astates 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, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобразить всю избыточно введенную 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 (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.

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

9.

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.

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.

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 unsetcфлагом -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 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.

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;



`${#}` –возвращает целое число –количествослов, которые были результатом `$`;

`${#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`.