

# Цель работы:

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

# Ход работы:

1. Написал 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. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации узнал, изучив справку.(рис. 1 [-@fig:001], [-@fig:002], [-@fig:003] )

```

+
PRoMofficialPRoM
ZIP(1L)
ZIP(1L)
NAME
zip - package and compress (archive) files
SYNOPSIS
zip [-aABcdDeEfFghjklLmoqrRSTuvVwWxyz!@&] [--longoption ...] [-b path] [-n suffixes] [-t date] [-tt date]
[zipfile [file ...]] [-xi list]

zipcloak (see separate man page)

zipnote (see separate man page)

zipsplit (see separate man page)

Note: Command line processing in zip has been changed to support long options and handle all options and
arguments more consistently. Some old command lines that depend on command line inconsistencies may no
longer work.
DESCRIPTION
zip is a compression and file packaging utility for Unix, VMS, MSDOS, OS/2, Windows 9x/NT/XP, Minix, Atari,
Macintosh, Amiga, and Acorn RISC OS. It is analogous to a combination of the Unix commands tar(1) and com-
press(1) and is compatible with PKZIP (Phil Katz's ZIP for MSDOS systems).

A companion program (unzip(1L)) unpacks zip archives. The zip and unzip(1L) programs can work with archives
produced by PKZIP (supporting most PKZIP features up to PKZIP version 4.6), and PKZIP and PKUNZIP can work
with archives produced by zip (with some exceptions, notably streamed archives, but recent changes in the
zip file standard may facilitate better compatibility). zip version 3.0 is compatible with PKZIP 2.04 and
also supports the Zip64 extensions of PKZIP 4.5 which allow archives as well as files to exceed the previous
2 GB limit (4 GB in some cases). zip also now supports bzip2 compression if the bzip2 library is included
when zip is compiled. Note that PKUNZIP 1.10 cannot extract files produced by PKZIP 2.04 or zip 3.0. You
must use PKUNZIP 2.04g or unzip 5.0p1 (or later versions) to extract them.

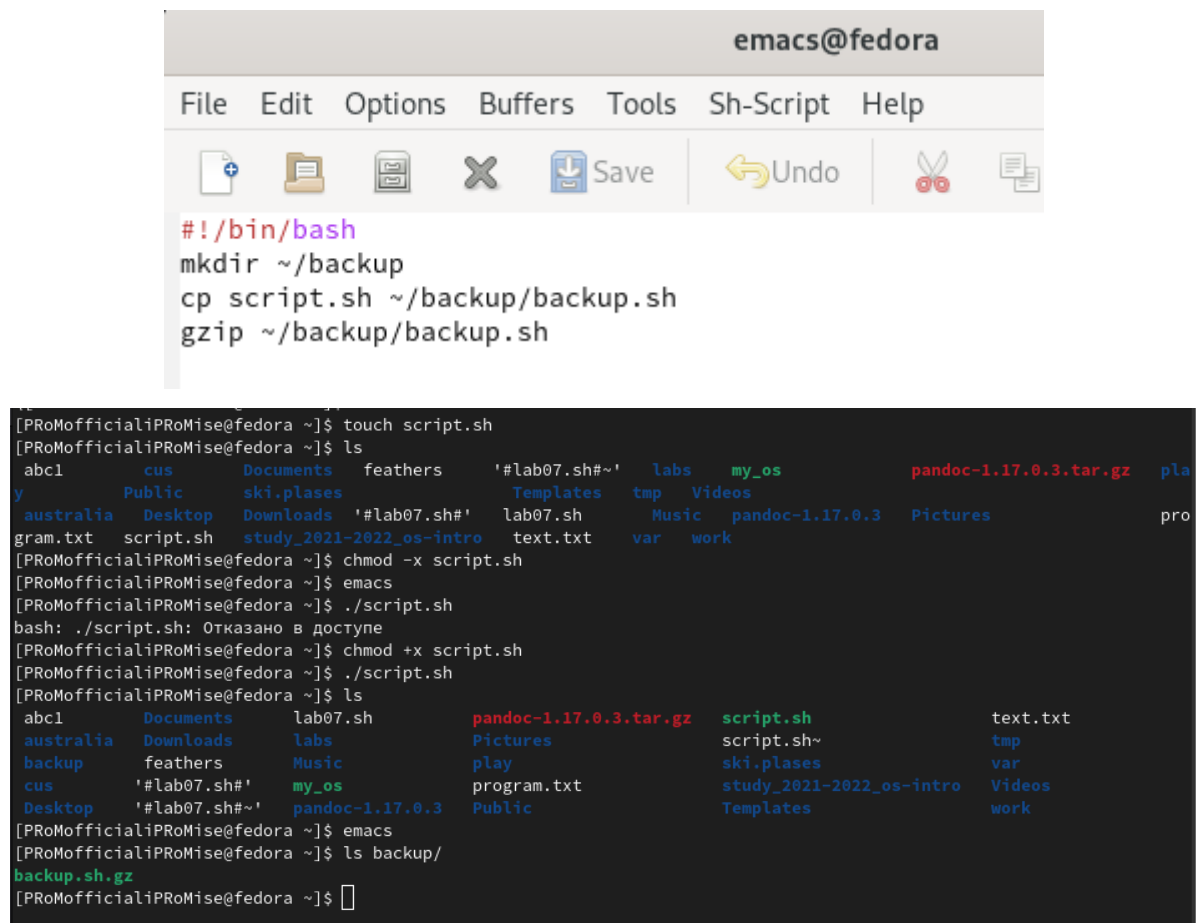
See the EXAMPLES section at the bottom of this page for examples of some typical uses of zip.

Large Archives and Zip64. zip automatically uses the Zip64 extensions when files larger than 4 GB are added
to an archive, an archive containing Zip64 entries is updated (if the resulting archive still needs Zip64),
the size of the archive will exceed 4 GB, or when the number of entries in the archive will exceed about
64K. Zip64 is also used for archives streamed from standard input as the size of such archives are not
known in advance, but the option -fz- can be used to force zip to create PKZIP 2 compatible archives (as
long as Zip64 extensions are not needed). You must use a PKZIP 4.5 compatible unzip, such as unzip 6.0 or
later, to extract files using the Zip64 extensions.

In addition, streamed archives, entries encrypted with standard encryption, or split archives created with
the pause option may not be compatible with PKZIP as data descriptors are used and PKZIP at the time of this
writing does not support data descriptors (but recent changes in the PKWare published zip standard now in-
clude some support for the data descriptor format zip uses).

Mac OS X. Though previous Mac versions had their own zip port, zip supports Mac OS X as part of the Unix
port and most Unix features apply. References to "MacOS" below generally refer to MacOS versions older than
OS X. Support for some Mac OS features in the Unix Mac OS X port, such as resource forks, is expected in
the next zip release.

For a brief help on zip and unzip, run each without specifying any parameters on the command line.
Manual page zip(1) line 1 (press h for help or q to quit)
```



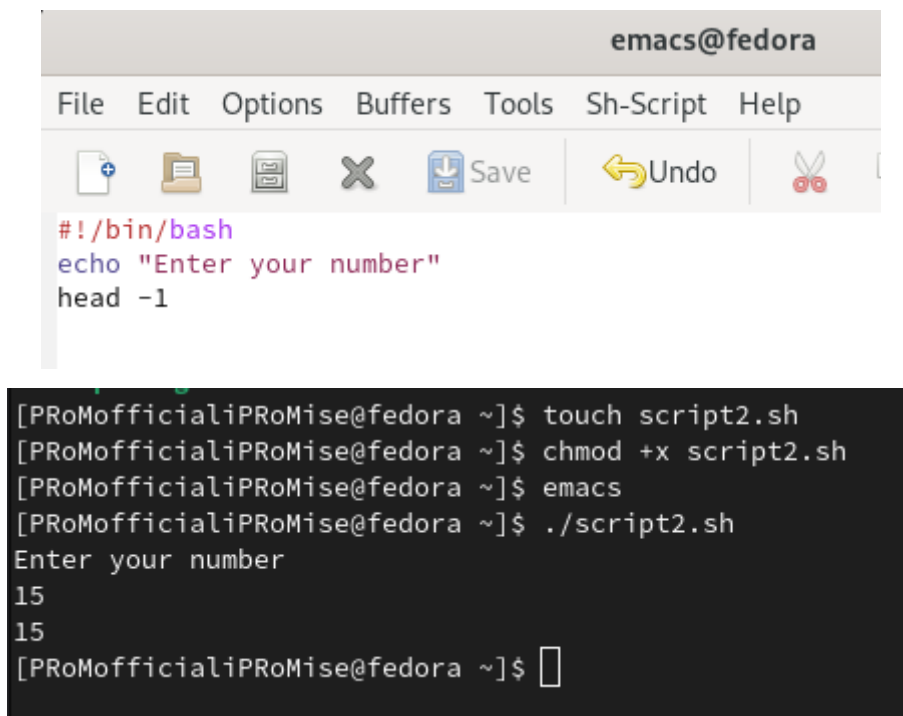
The screenshot shows the Emacs editor interface with the title bar 'emacs@fedora'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The toolbar contains icons for creating a new file, opening a file, saving, closing, and undo. The main text area contains a shell script:

```
#!/bin/bash
mkdir ~/backup
cp script.sh ~/backup/backup.sh
gzip ~/backup/backup.sh
```

Below the editor, a terminal window shows the execution of the script:

```
[PProMofficialiPProMise@fedora ~]$ touch script.sh
[PProMofficialiPProMise@fedora ~]$ ls
abc1      cus      Documents  feathers  '#lab07.sh#~'  labs  my_os  pandoc-1.17.0.3.tar.gz  pla
y         Public   ski.plases Templates  tmp  Videos
australia Desktop Downloads '#lab07.sh#' lab07.sh Music  pandoc-1.17.0.3 Pictures pro
gram.txt  script.sh study_2021-2022_os-intro text.txt var  work
[PProMofficialiPProMise@fedora ~]$ chmod -x script.sh
[PProMofficialiPProMise@fedora ~]$ emacs
[PProMofficialiPProMise@fedora ~]$ ./script.sh
bash: ./script.sh: Отказано в доступе
[PProMofficialiPProMise@fedora ~]$ chmod +x script.sh
[PProMofficialiPProMise@fedora ~]$ ./script.sh
[PProMofficialiPProMise@fedora ~]$ ls
abc1      Documents  lab07.sh  pandoc-1.17.0.3.tar.gz  script.sh  text.txt
australia Downloads  labs      Pictures                script.sh~  tmp
backup    feathers   Music     play                   ski.plases  var
cus       '#lab07.sh#' my_os     program.txt             study_2021-2022_os-intro Videos
Desktop   '#lab07.sh#~' pandoc-1.17.0.3 Public               Templates   work
[PProMofficialiPProMise@fedora ~]$ emacs
[PProMofficialiPProMise@fedora ~]$ ls backup/
backup.sh.gz
[PProMofficialiPProMise@fedora ~]$
```

2. Написал 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. [-@fig:004])(рис. [-@fig:005])



The screenshot shows the Emacs editor interface with the title bar 'emacs@fedora'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help'. The toolbar contains icons for creating a new file, opening a file, saving, closing, and undo. The main text area contains a shell script:

```
#!/bin/bash
echo "Enter your number"
head -1
```

Below the editor, a terminal window shows the execution of the script:

```
[PProMofficialiPProMise@fedora ~]$ touch script2.sh
[PProMofficialiPProMise@fedora ~]$ chmod +x script2.sh
[PProMofficialiPProMise@fedora ~]$ emacs
[PProMofficialiPProMise@fedora ~]$ ./script2.sh
Enter your number
15
15
[PProMofficialiPProMise@fedora ~]$
```

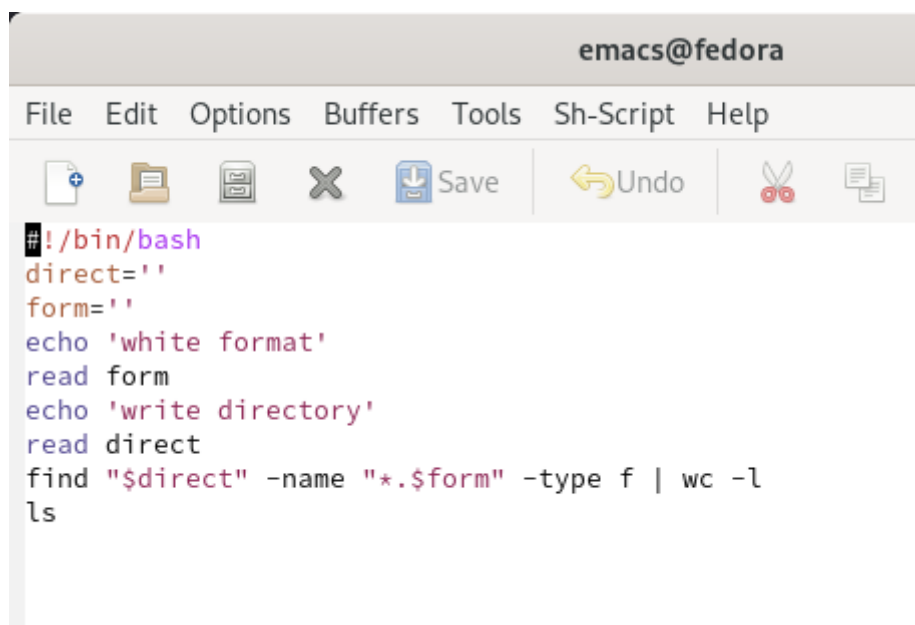
3. Написал командный файл — аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.(рис. [-@fig:006])(рис. [-@fig:007])

```
#!/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 can be written
    elif test -r $A
    then echo readable
    else echo impossible to process
    fi
fi
done
```

```
[PRoMofficialiPRoMise@fedora ~]$ ./file.sh
abc1: is a file andcan be written
australia: is a directory
backup: is a directory
cus: is a directory
Desktop: is a directory
Documents: is a directory
Downloads: is a directory
feathers: is a file andcan be written
file.sh: is a file andcan be written
file.sh~: is a file andcan be written
#lab07.sh#: is a file andcan be written
#lab07.sh#~: is a file andcan be written
lab07.sh: is a file andcan be written
labs: is a directory
Music: is a directory
my_os: is a file and./file.sh: строка 8: teat: команда не найдена
impossible to process
pandoc-1.17.0.3: is a directory
pandoc-1.17.0.3.tar.gz: is a file andcan be written
Pictures: is a directory
play: is a directory
program.txt: is a file andcan be written
Public: is a directory
script2.sh: is a file andcan be written
script2.sh~: is a file andcan be written
script.sh: is a file andcan be written
script.sh~: is a file andcan be written
ski.plases: is a directory
study_2021-2022_os-intro: is a directory
Templates: is a directory
text.txt: is a file andcan be written
tmp: is a directory
var: is a directory
Videos: is a directory
work: is a directory
[PRoMofficialiPRoMise@fedora ~]$
```

4. Написал 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. [-@fig:009])(рис. [-@fig:010])

```
[PROMofficialiPROMise@fedora ~]$ ./file2.sh
white format
sh
write directory
backup
0
abcl Downloads '#lab07.sh#' pandoc-1.17.0.3 script2.sh Templates
australia feathers '#lab07.sh#~' pandoc-1.17.0.3.tar.gz script2.sh~ text.txt
backup file2.sh lab07.sh Pictures script.sh tmp
cus file2.sh~ labs play script.sh~ var
Desktop file.sh Music program.txt ski.plases Videos
Documents file.sh~ my_os Public study_2021-2022_os-intro work
[PROMofficialiPROMise@fedora ~]$
```



## Вывод:

Изучил основы программирования в оболочке ОС UNIX/Linux, 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 (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
  - C-оболочка (или csh) — надстройка над 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/Linuxподобных

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`.

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` и `ls -l myfile > ${b}` приведёт к переназначению стандартного вывода команды `ls` с терминала на файл `/tmp/andy-ls`, а использование команды `ls -l > $bls` приведёт к подстановке в командную строку значения переменной `bls`. Если переменной `bls` не было предварительно присвоено никакого значения, то её значением будет символ пробела.

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, 5, 6. 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.

Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа `let sum=x+7`, и `let` будет искать переменную `x` и добавлять к ней 7.

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.

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 отобрать всю избыточно введённую информацию и игнорировать её.

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

10. Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде:

`bash командныйфайл [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 имяфайла`

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 её 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 её 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. Для создания массива используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Удалить функцию можно с помощью команды `unset` с флагом `-f`.

Команда `typeset` имеет четыре опции для работы с функциями:

– `-f` — перечисляет определённые на текущий момент функции;

– `-ft` — при последующем вызове функции иницирует её 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 одноимёнными именами функций, загружает его и вызывает эти функции.

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. Рассмотрим это на 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. – \$\* — 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 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`.