

Лабораторная работа №10

Отчёт по лабораторной работе №10

Макарова Анастасия Михайловна

Содержание

Цель работы

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

Выполнение лабораторной работы

1. Изучим справку команд архивации zip, bzip2 и tar с помощью команды man (Рис.1-4).

```
ZIP(1L) ZIP(1L)

NAME
    zip - package and compress (archive) files

SYNOPSIS
    zip [-aABcdDeEfFghjklLmoqrRSTuvVwXyz!@$] [--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
```

Рис.1

```

bzip2(1)                                General Commands Manual                                bzip2(1)

NAME
    bzip2, bunzip2 - a block-sorting file compressor, v1.0.8
    bzipcat - decompresses files to stdout
    bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS
    bzip2 [ -cdfkqstvzVL123456789 ] [ filenames ... ]
    bunzip2 [ -fkvsVL ] [ filenames ... ]
    bzipcat [ -s ] [ filenames ... ]
    bzip2recover filename

DESCRIPTION
    bzip2 compresses files using the Burrows-Wheeler block sorting text
    compression algorithm, and Huffman coding. Compression is generally
    considerably better than that achieved by more conventional
    LZ77/LZ78-based compressors, and approaches the performance of the PPM
    family of statistical compressors.

    The command-line options are deliberately very similar to those of GNU
    gzip, but they are not identical.

```

Puc.2

```

TAR(1)                                GNU TAR Manual

NAME
    tar - an archiving utility

SYNOPSIS
    Traditional usage
        tar {A|c|d|r|t|u|x}[GnSkUW0mpsMBiajJzZhPlRvwo] [ARG...]

    UNIX-style usage
        tar -A [OPTIONS] ARCHIVE ARCHIVE

        tar -c [-f ARCHIVE] [OPTIONS] [FILE...]

        tar -d [-f ARCHIVE] [OPTIONS] [FILE...]

        tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]

        tar -r [-f ARCHIVE] [OPTIONS] [FILE...]

        tar -u [-f ARCHIVE] [OPTIONS] [FILE...]

        tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]

```

Puc.3

```
[ammakarova@10 ~]$ man zip  
[ammakarova@10 ~]$ man bzip2  
[ammakarova@10 ~]$ man tar
```

Рис.4

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 нашем домашнем каталоге. Файл должен архивироваться одним из архиваторов на выбор: zip, bzip2 или tar, я выбрала bzip2 (Рис.5).

```
#!/bin/bash

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

U:--- **backup.sh** All L7 (Shell-script[sh])

Рис.5

Добавим права на выполнение программы backup.sh с помощью команды chmod с опцией +x и проверим работу скрипта с помощью команды ./backup.sh (Рис.6, 7).

```
[ammakarova@10 ~]$ chmod +x backup.sh
[ammakarova@10 ~]$ ./backup.sh
Выполнено
[ammakarova@10 ~]$
```

Рис.6

```
[ammakarova@10 ~]$ ls
'##'      backup.sh~  file3.sh  monthly  sci.plases  Изображения
'$'      catalog    file3.sh~  my_os    text.txt    Музыка
abc1      conf.txt   file4.sh  OS_2022  work        Общедоступные
australia feathers   lab07.sh  play     Видео       'Рабочий стол'
backup    file1.sh   lab07.sh~ prog2.cpp Документы   Шаблоны
backup.sh file2.sh   may       reports  Загрузки
```

Рис.7

Переходим в каталог backup и видим, что файл появился в этом каталоге, затем посмотрим содержимое архива с помощью команды bunzip2 -c backup.sh.bz2 (Рис.8).

```
[ammakarova@10 ~]$ cd backup
[ammakarova@10 backup]$ ls
backup.sh.bz2
[ammakarova@10 backup]$ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh'
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/backup/
echo "Выполнено"
[ammakarova@10 backup]$
```

Рис.8

2. Создадим в редакторе emacs файл с названием command_file.sh, в нем мы будем писать второй скрипт (Рис.9).



Рис.9

Напишем 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.10).

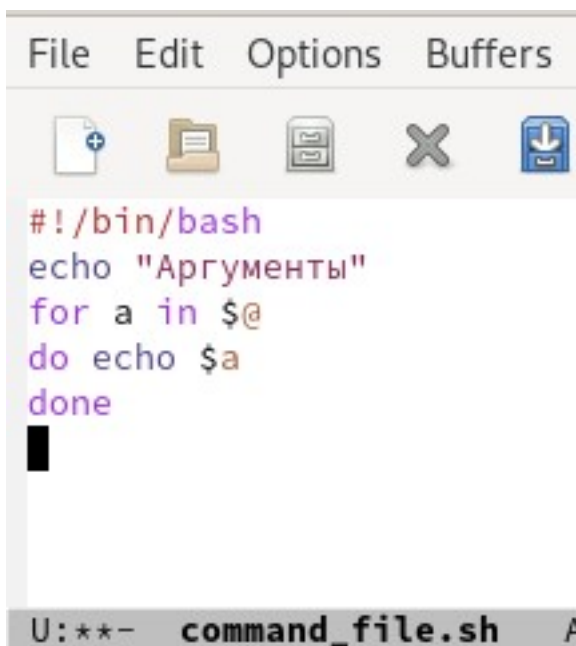


Рис.10

Добавим права на выполнение программы `command_file.sh` с помощью команды `chmod` с опцией `+x` и проверим работу скрипта, вводя различные аргументы (Рис.11, 12).

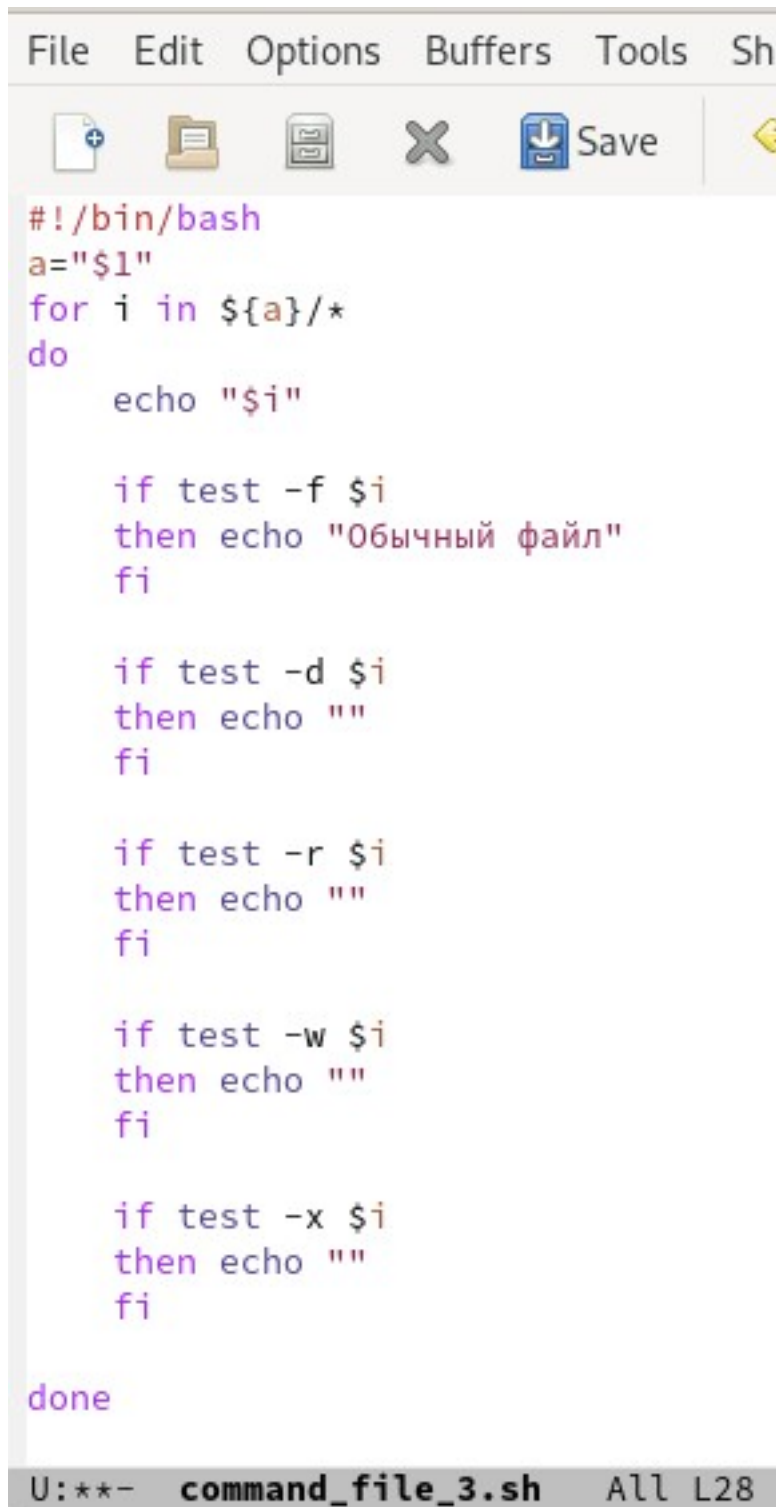
```
[ammakarova@10 ~]$ chmod +x command_file.sh
[ammakarova@10 ~]$ ls
abcl      conf.txt   lab07.sh  prog2.cpp  Загрузки
australia feathers   lab07.sh~ reports     Изображения
backup    file1.sh  may       sci.plases  Музыка
backup.sh file2.sh  monthly   text.txt    Общедоступные
backup.sh~ file3.sh  my_os     work        'Рабочий стол'
catalog   file3.sh~ OS_2022   Видео       Шаблоны
command_file.sh file4.sh  play      Документы
```

Рис.11

```
[ammakarova@10 ~]$ ./command_file.sh 4 19 8 349 0 38 8 2
Аргументы
4
19
8
349
0
38
8
2
[ammakarova@10 ~]$
```

Рис.12

3. Создадим в редакторе emacs файл с названием command_file_3.sh, в нем мы будем писать третий скрипт. Напишем 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.13).



```
#!/bin/bash
a="$1"
for i in ${a}/*
do
    echo "$i"

    if test -f $i
    then echo "Обычный файл"
    fi

    if test -d $i
    then echo ""
    fi

    if test -r $i
    then echo ""
    fi

    if test -w $i
    then echo ""
    fi

    if test -x $i
    then echo ""
    fi

done
```

U:*- **command_file_3.sh** All L28

Рис.13

Добавим права на выполнение программы `command_file_3.sh` с помощью команды `chmod` с опцией `+x` и проверим работу скрипта с помощью команды `./command_file_3.sh ~` (Рис.14).


```

[ammakarova@10 ~]$ chmod +x command_file_3.sh
[ammakarova@10 ~]$ ls
abc1          command_file.sh  lab07.sh      reports       Музыка
australia     conf.txt         lab07.sh~     sci.plases    Общедоступные
backup        feathers         may           text.txt      'Рабочий стол'
backup.sh     file1.sh        monthly       work          Шаблоны
backup.sh~    file2.sh        my_os         Видео
catalog       file3.sh        os_2022       Документы
'#command_file_3#' file3.sh~      play          Загрузки
command_file_3.sh file4.sh       prog2.cpp     Изображения
[ammakarova@10 ~]$ ./command_file_3.sh ~
/home/ammakarova/abc1
Обычный файл

/home/ammakarova/australia

/home/ammakarova/backup

/home/ammakarova/backup.sh
Обычный файл

```

Рис.14

4. Создадим в редакторе emacs файл с названием command_file_4.sh, в нем мы будем писать четвертый скрипт (Рис.15).

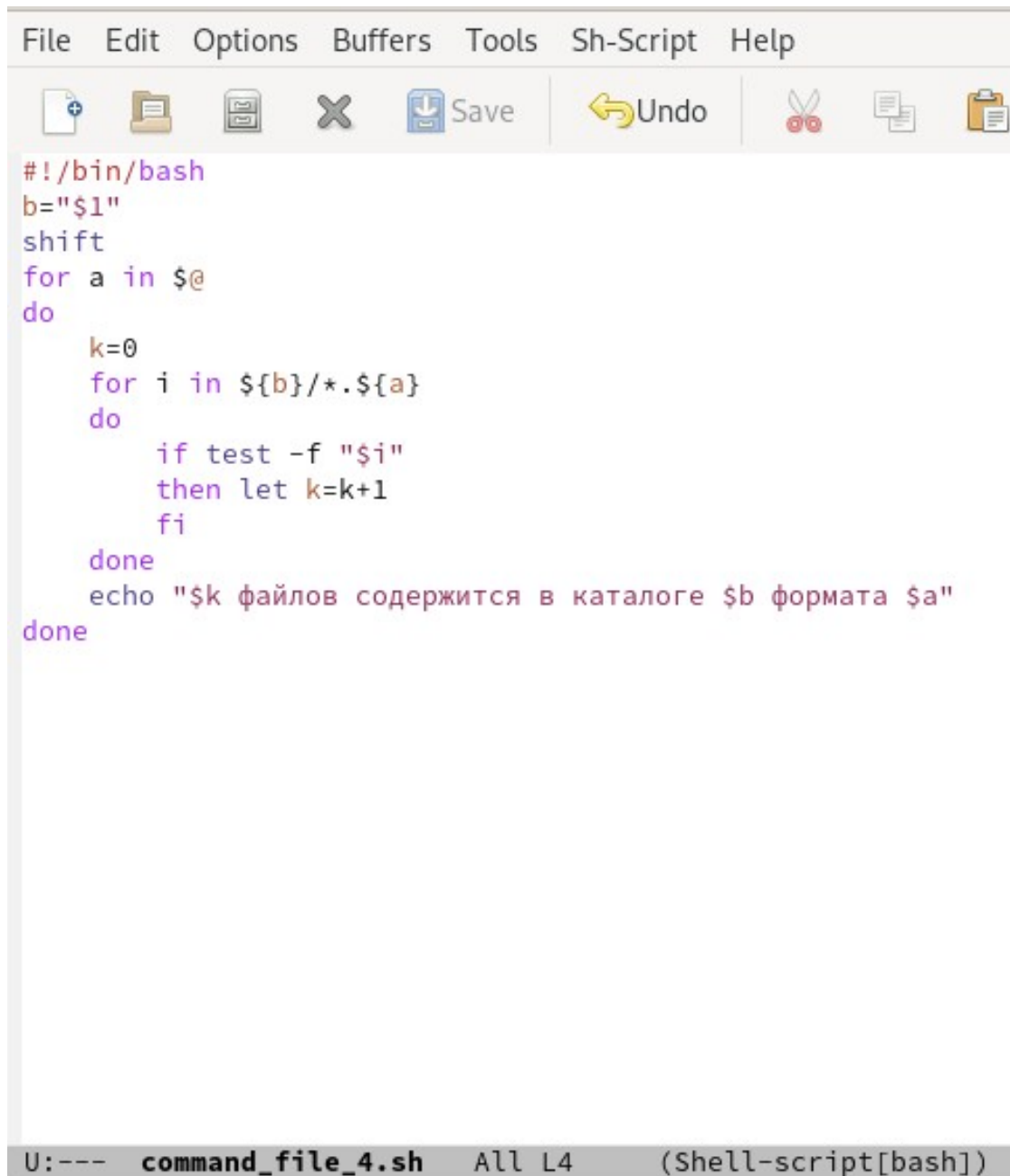
```

U:--- $ All L1
Find file: ~/command_file_4.sh

```

Рис.15

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. Путь к директории также передаётся в виде аргумента командной строки (Рис.16).



```
#!/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 файлов содержится в каталоге $b формата $a"
done
```

U:--- **command_file_4.sh** All L4 (Shell-script[bash])

Рис.16

Добавим права на выполнение программы `command_file_4.sh` с помощью команды `chmod` с опцией `+x` и проверим работу скрипта с помощью команды `./command_file_4.sh`. Я просматриваю файлы с форматом `txt cpp`, поэтому после команды добавляю `~ txt cpp` (Рис.17).

```

[ammakarova@10 ~]$ chmod +x command_file_4.sh
[ammakarova@10 ~]$ ls
'$'                command_file_4.sh~  lab07.sh~         work
abc1               command_file.sh    may               Видео
australia          conf.txt           monthly          Документы
backup             feathers           my_os            Загрузки
backup.sh          file1.sh           OS_2022          Изображения
backup.sh~         file2.sh           play            Музыка
catalog           file3.sh           prog2.cpp        Общедоступные
'#command_file_3#' file3.sh~          reports          'Рабочий стол'
command_file_3.sh  file4.sh           sci.plases       Шаблоны
command_file_4.sh  lab07.sh           text.txt
[ammakarova@10 ~]$ ./command_file_4.sh ~ txt cpp
2 файлов содержится в каталоге /home/ammakarova формата txt
1 файлов содержится в каталоге /home/ammakarova формата cpp

```

Рис.17

Вывод

В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX/Linux и научилась писать небольшие командные файлы.

Контрольные вопросы

- Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются? Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера.
 - 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).
- Что такое POSIX? POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

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. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол `$`.
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. 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 `read` позволяет читать значения переменных со стандартного ввода.
5. Какие арифметические операции можно применять в языке программирования bash?

Арифметические операторы оболочки `bash`

Оператор	Синтаксис	Результат
<code>!</code>	<code>!exp</code>	Если <code>exp</code> равно 0, то возвращает 1; иначе 0
<code>!=</code>	<code>exp1 !=exp2</code>	Если <code>exp1</code> не равно <code>exp2</code> , то возвращает 1; иначе 0
<code>%</code>	<code>exp1%exp2</code>	Возвращает остаток от деления <code>exp1</code> на <code>exp2</code>
<code>%=</code>	<code>var=%exp</code>	Присваивает остаток от деления <code>var</code> на <code>exp</code> переменной <code>var</code>
<code>&</code>	<code>exp1&exp2</code>	Возвращает побитовое AND выражений <code>exp1</code> и <code>exp2</code>
<code>&&</code>	<code>exp1&&exp2</code>	Если и <code>exp1</code> и <code>exp2</code> не равны нулю, то возвращает 1; иначе 0
<code>&=</code>	<code>var &= exp</code>	Присваивает переменной <code>var</code> побитовое AND <code>var</code> и <code>exp</code>
<code>*</code>	<code>exp1 * exp2</code>	Умножает <code>exp1</code> на <code>exp2</code>
<code>*=</code>	<code>var *= exp</code>	Умножает <code>exp</code> на значение переменной <code>var</code> и присваивает результат переменной <code>var</code>
<code>+</code>	<code>exp1 + exp2</code>	Складывает <code>exp1</code> и <code>exp2</code>
<code>+=</code>	<code>var += exp</code>	Складывает <code>exp</code> со значением переменной <code>var</code> и результат присваивает переменной <code>var</code>
<code>-</code>	<code>-exp</code>	Операция отрицания <code>exp</code> (унарный минус)
<code>-</code>	<code>exp1 - exp2</code>	Вычитает <code>exp2</code> из <code>exp1</code>
<code>--</code>	<code>var -= exp</code>	Вычитает <code>exp</code> из значения переменной <code>var</code> и присваивает результат переменной <code>var</code>
<code>/</code>	<code>exp / exp2</code>	Делит <code>exp1</code> на <code>exp2</code>
<code>/=</code>	<code>var /= exp</code>	Делит значение переменной <code>var</code> на <code>exp</code> и присваивает результат переменной <code>var</code>
<code><</code>	<code>exp1 < exp2</code>	Если <code>exp1</code> меньше, чем <code>exp2</code> , то возвращает 1, иначе возвращает 0
<code><<</code>	<code>exp1 << exp2</code>	Сдвигает <code>exp1</code> влево на <code>exp2</code> бит
<code><=<</code>	<code>var <=< exp</code>	Побитовый сдвиг влево значения переменной <code>var</code> на <code>exp</code>
<code><=</code>	<code>exp1 <= exp2</code>	Если <code>exp1</code> меньше или равно <code>exp2</code> , то возвращает 1; иначе возвращает 0
<code>=</code>	<code>var = exp</code>	Присваивает значение <code>exp</code> переменной <code>var</code>
<code>==</code>	<code>exp1==exp2</code>	Если <code>exp1</code> равно <code>exp2</code> , то возвращает 1; иначе возвращает 0
<code>></code>	<code>exp1 > exp2</code>	1, если <code>exp1</code> больше, чем <code>exp2</code> ; иначе 0
<code>>=</code>	<code>exp1 >= exp2</code>	1, если <code>exp1</code> больше или равно <code>exp2</code> ; иначе 0
<code>>></code>	<code>exp >> exp2</code>	Сдвигает <code>exp1</code> вправо на <code>exp2</code> бит
<code>>>=</code>	<code>var >>=exp</code>	Побитовый сдвиг вправо значения переменной <code>var</code> на <code>exp</code>
<code>^</code>	<code>exp1 ^ exp2</code>	Исключающее OR выражений <code>exp1</code> и <code>exp2</code>
<code>^=</code>	<code>var ^= exp</code>	Присваивает переменной <code>var</code> побитовое XOR <code>var</code> и <code>exp</code>
<code> </code>	<code>exp1 exp2</code>	Побитовое OR выражений <code>exp1</code> и <code>exp2</code>
<code> =</code>	<code>var = exp</code>	Присваивает переменной <code>var</code> результат операции XOR <code>var</code> и <code>exp</code>
<code> </code>	<code>exp1 exp2</code>	1, если или <code>exp1</code> или <code>exp2</code> являются ненулевыми значениями; иначе 0
<code>~</code>	<code>~exp</code>	Побитовое дополнение до <code>exp</code>

6. Что означает операция `(())`? В эти скобки можно записывать условия оболочки `bash`.
 7. Какие стандартные имена переменных Вам известны? Имена некоторых переменных имеют для командного процессора специальный смысл.
- Значением переменной `PATH` (т.е. `$PATH`) 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.

- 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.
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 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 \$, ' , , ". Например:
- echo * выведет на экран символ *,
 - echo ab/'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.
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 `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.
 12. Каким образом можно выяснить, является файл каталогом или обычным файлом? Команда `test`, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения. Так например, команда:
 - `test -f file` возвращает нулевой код завершения (истина), если файл `file` существует, и ненулевой код завершения (ложь) в противном случае:
 - `test s` — истина, если аргумент `s` имеет значение истина;
 - `test -f file` — истина, если файл `file` существует;
 - `test -i file` — истина, если файл `file` доступен по чтению;
 - `test -w file` — истина, если файл `file` доступен по записи;
 - `test -e file` — истина, если файл `file` — исполняемая программа;
 - `test -d file` — истина, если файл `file` является каталогом.
 13. Каково назначение команд `set`, `typeset` и `unset`?
 - Вы можете использовать команду `set` для вывода списка переменных окружения. Поэтому для ознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду `set | more`.
 - Следует использовать команду `unset` для удаления переменной из вашего окружения командной оболочки.
 - Команда `typeset` является встроенной и предназначена для наложения ограничений на переменные. Это попытка контроля над типами, которая имеется во многих языках программирования.
 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.

- `$*` — отображается вся командная строка или параметры оболочки;
- `$?` — 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.