

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 11

дисциплина: Операционные системы

Студент:

Афтаева Ксения Васильевна

Преподаватель:

Велиева Т.В.

Группа: НПИбд-01-20

МОСКВА 2021 г.

Цель работы:

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

Задачи:

1. Изучить теоретический материал
2. 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.
3. 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*.
4. 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*.
5. 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.

6. Ответить на контрольные вопросы

Объект и предмет исследования:

Программирование в оболочке ОС UNIX/Linux

Техническое оснащение:

Ноутбук, на котором установлена виртуальная машина с линукс

Теоретические вводные данные [1] :

bash (сокр. от «Bourne-Again shell») — это командная оболочка (или «интерпретатор командной строки»), используемая по умолчанию в операционных системах на базе Unix и Linux, созданная в 1989 году Брайаном Фоксом с целью усовершенствования командной оболочки sh.

bash позволяет автоматизировать различные задачи, устанавливать программное обеспечение, настраивать конфигурации для своего рабочего окружения и многое другое.

Основные преимущества:

- Позволяет работать со структурами «[[» (в sh доступна только «[» с ограничениями)
- Поддерживает работу с массивами в Линуксе
- Доступно множество расширений, выполненных по стандартам C, включая циклы с тремя аргументами «for((i=0;i<=3;i++))», возможность присваивать инкремент «+=» и многое другое
- Поддерживает синтаксис «<<<'here strings'»
- Работает с расширениями «.{png,jpg}»
- Доступны алиасы для перенаправления, подобно «Csh», подобно «&|» для «2>&1|» и «&>» для «> ... 2>&1»
- Поддерживает сопроцессы с перенаправлением «<>»
- Огромный комплект расширений нестандартных конфигураций, включая изменение регистра
- Существенно увеличены возможности арифметики (правда, нет поддержки чисел с плавающей точкой)

- Переменные «*RANDOM*», «*SECONDS*», «*\$PIPESTATUS[@]*» и «*\$FUNCNAME*» в Bash являются расширениями
- Доступно огромное количество функций, обеспечивающих работу в интерактивном режиме. Хотя на поведение скриптов они не влияют

Условные обозначения и символы:

- *\$** — 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*.
-

Выполнение работы:

Задание 1

1. Предварительно прочитав информацию о команде `tar` (с помощью команды `man tar`), изучила структуру архивации файлов, которая понадобится в дальнейшем (Рис.1).

A screenshot of a terminal window displaying the manual page for the 'tar' command. The window title is 'User Commands'. The content includes sections for NAME, SYNOPSIS, DESCRIPTION, EXAMPLES, and DEFAULTS. The DESCRIPTION section explains that GNU 'tar' saves files into a single tape or disk archive. The EXAMPLES section shows commands for creating, listing, and extracting archives. The DEFAULTS section lists various options like --format=gnu, --quoting-style=escape, etc. At the bottom, a prompt asks to press 'h' for help or 'q' to quit.

```
TAR(1)                                User Commands                                TAR(1)

NAME
  tar - manual page for tar 1.26

SYNOPSIS
  tar [OPTION...] [FILE]...

DESCRIPTION
  GNU 'tar' saves many files together into a single tape or disk archive,
  and can restore individual files from the archive.

  Note that this manual page contains just very brief description (or
  more like a list of possible functionality) originally generated by the
  help2man utility. The full documentation for tar is maintained as a
  Texinfo manual. If the info and tar programs are properly installed at
  your site, the command 'info tar' should give you access to the com-
  plete manual.

EXAMPLES
  tar -cf archive.tar foo bar
      # Create archive.tar from files foo and bar.

  tar -tvf archive.tar
      # List all files in archive.tar verbosely.

  tar -xf archive.tar
      # Extract all files from archive.tar.

DEFAULTS
  *This* tar installation defaults to:

  --format=gnu -f- -b20 --quoting-style=escape --rmt-command=/sbin/rmt
  --rsh-command=/usr/bin/rsh
Manual page tar(1) line 1 (press h for help or q to quit)
```

Рис.1 Информация о `tar`

Создаю директорию **backup** в домашнем каталоге с помощью команды `mkdir backup`. Я буду работать в текстовом редакторе **emacs**. Поэтому с помощью команды `emacs scr1.sh` создаю файл **scr1.sh** и открываю его на редактирование (Рис.2)

A screenshot of a terminal window showing two commands being executed. The first command is 'mkdir backup' and the second is 'emacs scr1.sh'. The prompt is '[kvaftaeva@kvaftaeva ~]\$' and the cursor is at the end of the second command.

```
[kvaftaeva@kvaftaeva ~]$ mkdir backup
[kvaftaeva@kvaftaeva ~]$ emacs scr1.sh
```

Рис.2 Создание каталога и файла

2. Написала программу выполняющую требуемые функции.

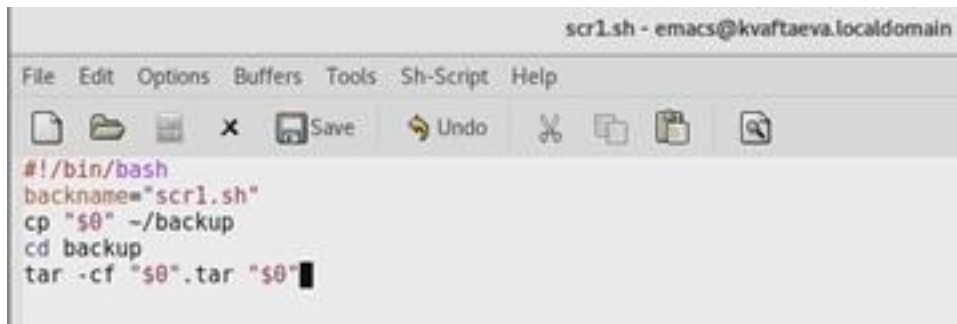


Рис.3 Программа к заданию 1

О ее структуре: в первой строке мы вызываем интерпритатор, в нашем случае `bash`. Далее мы объявляем переменную **backname**, куда помещаем название. Далее с помощью команды `cp` копируем наш файл в каталог **backup**, обращаясь к файлу с помощью указателя `.`. Перехожу в каталог с помощью команды `cd`. Далее с помощью команды `tar -cf`, которая позволяет создать архив, имеющий такое название, какое имеет командный файл, но в формате `.tar`, при этом создаю архив и сразу же помещаю в него наш командный файл.

После написания программы сохраняю его с помощью комбинации клавиш `C-x C-s` и закрываю текстовый редактор.

3. Запускаю программу на выполнение, введя в командной строке `bash scr1.sh` (Рис.4)



Рис.4 Запуск на выполнение 1 программы

4. Проверяю корректность выполнения: открываю директорию **backup** (Рис. 5)

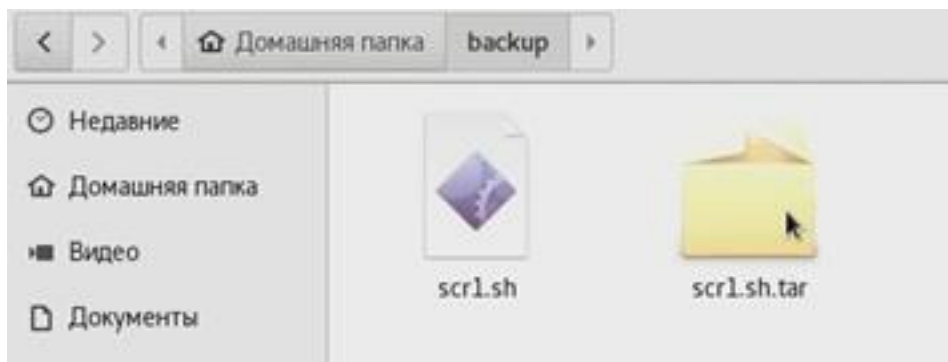


Рис.5 Директория backup

Видим, что файл скопирован, и так же у нас есть архив в нужном формате

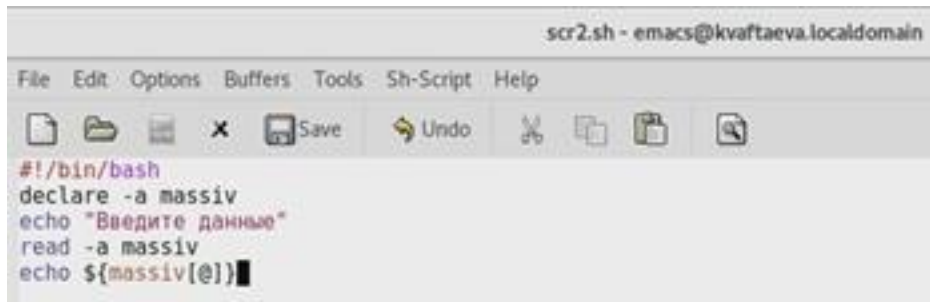
Задание 2

1. Создаю файл для этого задания (scr2.sh) и открываю его на редактирование с помощью команды `emacs scr2.sh` (Рис.6)

```
[kvaftaeva@kvaftaeva ~]$ emacs scr2.sh
```

Рис.6 Создание файла для второго задания

2. Написала программу, выполняющую требуемые функции (Рис.7)



```
scr2.sh - emacs@kvaftaeva.localdomain
File Edit Options Buffers Tools Sh-Script Help
[Icons] Save Undo [Icons]
#!/bin/bash
declare -a massiv
echo "Введите данные"
read -a massiv
echo ${massiv[@]}
```

Рис.7 Программа к заданию 2

Ее структура: в первой строке мы вызываем интерпритатор, в нашем случае `bash`. Далее мы объявляем массив **massiv**. С помощью `echo` выводжу надпись запроса. Далее считываем массив с клавиатуры. Последняя строка выводит все элементы массива: `{massiv[@]}` обращается к каждому элементу массива, а `echo` выводит их.

3. Запускаю программу с помощью команды `bash scr2.sh`. Видим, что у нас появился запрос. Ввожу произвольные числа через пробел и в конце ввода нажимаю ввод (Рис.8)

```
[kvaftaeva@kvaftaeva ~]$ bash scr2.sh
Введите данные
1 5 10 15 20 25
1 5 10 15 20 25
```

Рис.8 Результат работы программы 2

Видим, что программа вывела наши элементы

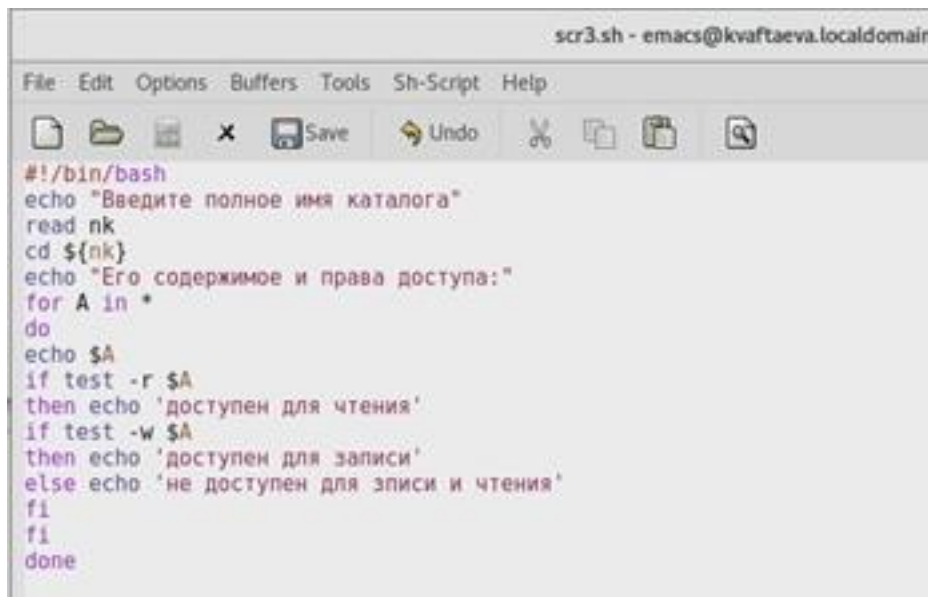
Задание 3

1. Создаю файл для этого задания (scr3.sh) и открываю его на редактирование с помощью команды `emacs scr3.sh` (Рис.9)

```
[kvaftaeva@kvaftaeva ~]$ emacs scr3.sh
```

Рис.9 Создание файла для третьего задания

2. Написала программу, выполняющую требуемые функции (Рис.10)



```
#!/bin/bash
echo "Введите полное имя каталога"
read nk
cd ${nk}
echo "Его содержимое и права доступа:"
for A in *
do
echo $A
if test -r $A
then echo 'доступен для чтения'
if test -w $A
then echo 'доступен для записи'
else echo 'не доступен для записи и чтения'
fi
fi
done
```

Рис.10 Программа к заданию 3

Ее структура: в первой строке мы вызываем интерпритатор, в нашем случае `bash`. Далее мы выводим запрос на ввод полного имени каталога с помощью `echo`. Затем считываем имя каталога с клавиатуры в переменную `nk`. Далее переходим в него командой `cd`. После этого вывожу фразу для красоты оформления. Далее идет цикл, который выполняется для каждого файла (`A`) в текущем каталоге (`*`). В цикле мы выводим название файла командой `echo $A`. Затем условием `if test -r $A` проверяется есть ли право на чтение этого файла. Если это условие истинно, то `echo 'доступен для чтения'` выводит эту информацию на экран. Аналогично проверяется право на запись. Если нет этих прав, то мы выводим эту информацию. Завершаем цикл строкой `done`.

3. Запускаю программу на выполнение с помощью команды `bash scr3.sh`. Видим, что у нас появился запрос на ввод директории. Ввожу директорию, которую мы создавали для первого задания (Рис.11)

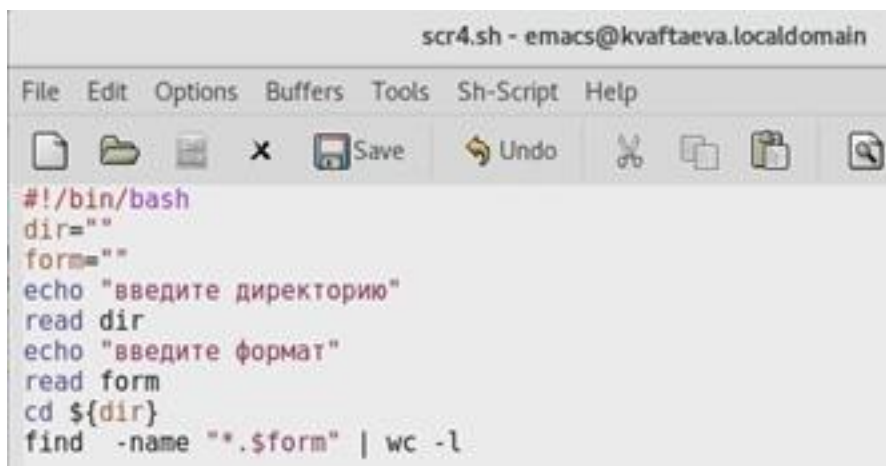

```
[kvaftaeva@kvaftaeva ~]$ bash scr3.sh
Введите полное имя каталога
backup
Его содержимое и права доступа:
scr1.sh
доступен для чтения
доступен для записи
scr1.sh.tar
доступен для чтения
доступен для записи
```

Рис.11 Запуск третьей программы на выполнение

Видим, что у программа вывела нам названия файлов данного каталога и права доступа

Задание 4

1. Создаю файл для этого задания (scr4.sh) и открываю его на редактирование с помощью команды `emacs scr4.sh` (Рис.12)
2. Написала программу, выполняющую требуемые функции (Рис.12)

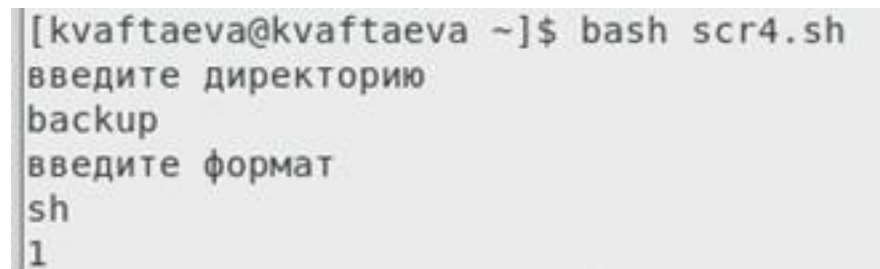


```
scr4.sh - emacs@kvaftaeva.localdomain
File Edit Options Buffers Tools Sh-Script Help
[Icons] Save Undo [Icons]
#!/bin/bash
dir=""
form=""
echo "введите директорию"
read dir
echo "введите формат"
read form
cd ${dir}
find -name ".*$form" | wc -l
```

Рис.12 Программа к заданию 4

Ее структура: в первой строке мы вызываем интерпритатор, в нашем случае `bash`. Объявляем две переменные: для хранения имени директории (`dir`) и искомого формата (`form`). Далее мы выводим фразы запроса и считываем имя директории и искомый формат с клавиатуры с помощью `read`. Далее мы переходим в нужную директорию. И в последней строке мы совершаем поиск: файлы по именам (`-name`), в которых встречается нам введенный формат. Конвейером считываем нереализованный вывод и командой `wc -l` считаем его строки, т.е. - файлы, найденные в данной директории и соответствующие требованиям

3. Запускаю программу на выполнение с помощью команды `bash scr4.sh`. У нас появился запрос на ввод. Ввожу директорию из первого задания и формат **sh** (Рис.13)



```
[kvaftaeva@kvaftaeva ~]$ bash scr4.sh
введите директорию
backup
введите формат
sh
1
```

Рис.13 Запуск четвертой программы на выполнение

Программа выводит 1, именно столько файлов такого формата у нас есть в данной директории

Контрольные вопросы: [2]

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.

Примеры:

Оболочка Борна (Bourne) - первоначальная командная оболочка UNIX: базовый, но полный набор функций;

C - оболочка - добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя C-подобный синтаксис команд, и сохраняет историю выполненных команд;

Оболочка Корна - 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*)- 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 графический

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: `имяпеременнойнапример, использованиекоманд * b =/tmp/andy - ls -l myfile >{b}ls*` приведет к переназначению стандартного вывода команды `ls` с терминала на файл `/tmp/andy-ls`, а использование команды `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** является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение - это единичный терм (`term`), обычно целочисленный. Целые числа можно записывать как последовательность цифр или в любом базовом формате. Этот формат — `radix#number`, где `radix` (основание системы счисления) - любое число не более 26. Для большинства команд основания систем счисления это - 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток (%). Команда `let` берет два операнда и присваивает их переменной.

Команда **read** читает одну строку из стандартного входного потока и записывает ее содержимое в указанные переменные. Если задана единственная переменная, в нее записывается вся строка. В результате ввода команды `read` без параметров строка

помещается в переменную среды `$reply`. При указании нескольких переменных в первую из них записывается первое слово строки, во вторую — второе слово и т. д. Последней переменной присваивается остаток строки.

- "+" сложение
- "-" вычитание
- "*" умножение
- "/" деление
- "**" возведение в степень
- "%" остаток от деления

5. Внутри `(())` вычисляются арифметические выражения и возвращается их результат. Например, в простейшем случае, конструкция `a=$((5 + 3))` присвоит переменной «а» значение выражения «5 + 3», или 8. Кроме того, двойные круглые скобки позволяют работать с переменными в стиле языка C.

- В переменной `$BASH` содержится полный путь до исполняемого файла командной оболочки Bash.
- В переменную `$BASH_VERSION` записывается версия Bash.
- Переменная, которая хранит пути поиска каталога. (используется при вводе команды `cd` имя_каталога без слэша)
- Содержит список каталогов для поиска файлов классов Java и архивов Java.
- Домашний каталог текущего пользователя.
- В переменной `$HOSTNAME` хранится имя компьютера.
- Количество событий, хранимых в истории за 1 сеанс
- Расположение файла истории событий
- Количество событий, хранимых в истории между сеансами
- Переменная хранит символы, являющиеся разделителями команд и параметров. (по умолчанию - пробел, табуляция и новая строка)
- Текущая установка локализации, которая позволяет настроить командную оболочку для использования в различных странах и на различных языках.
- Место, где храниться почта
- В переменной `$OSTYPE` содержится описание операционной системы.

- Список каталогов для поиска команд и приложений, когда полный путь к файлу не задан.
 - PS1 используется как основная строка приглашения. (то самое [root@proxy ~]#)
 - PS2 используется как вторичная строка приглашения.
 - Эта команда должна быть выполнена до отображения строки приглашения Bash.
 - Полный путь к текущему рабочему каталогу.
 - Полный путь к текущей командной оболочке.
 - В переменной \$USER содержится имя текущего пользователя.
6. Такие символы, как ' < > * ? | " & . Они имеют для командного процессора специальный смысл.
7. 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'|'cdвыдаст строку ab|*cd.`
8. Для создания командного файла: Запустить текстовый редактор. Последовательно записать команды, располагая каждую команду на отдельной строке. Сохранить этот файл, сделать его исполняемым, применив команду: `chmod +x имя_файла.`
- Запустить этот файл можно или используя команду `bash имя_файла`
9. 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.
10. `ls -lrt` Если есть `d`, то является файл каталогом
- Команда `set` изменяет значения внутренних переменных сценария.
 - Команда `typeset` задаёт и/или накладывает ограничения на переменные.

- Команда `unset` удаляет переменную, фактически -- устанавливает ее значение в `null`.

11. 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 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 в файле `where` содержится команда `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.

12. `$*` — 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 пробелов 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.

Заключение:

Теоретический материал изучен и пригодится в дальнейшей работе. Все цели и задачи выполнены.

Вывод:

Я изучила основы программирования в оболочке **ОС UNIX/Linux**. Научилась писать небольшие командные файлы.

Библиографический список:

[1]: [Программирование в bash](#)

[2]: [Описание лабораторной работы №11](#)