

Отчёт к лабораторной работе №11

**лабораторная работа № 12. Программирование в командном процессоре
ОС UNIX. Ветвления и циклы.**

Кекишева Анастасия Дмитриевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
3.1	Выполнение 1-го пункта задания	7
3.2	Выполнение 2-го пункта задания	9
3.3	Выполнение 3-го пункта задания	11
3.4	Выполнение 4-го пункта задания	13
4	Вывод	16
5	Библиография	17

Список таблиц

Список иллюстраций

3.1	Справка tar	7
3.2	Опция -с - создание	7
3.3	Опция -f - применение файла или устройство АРХИВ	8
3.4	Создание файла, присвоение прав доступа	8
3.5	Написание скрипта	8
3.6	Результат	9
3.7	Командный файл, обрабатывающий любое произвольное число аргументов командной строки №1	9
3.8	Результат работы head	10
3.9	Командный файл, обрабатывающий любое произвольное число аргументов командной строки №2	10
3.10	Результат работы S*	10
3.11	Командный файл, выводящий информацию о правах доступа	11
3.12	Результат выполнения командного файла	12
3.13	Командный файл, который получает в качестве аргумента командной строки формат файла и вычисляет количество таких файлов в указанной директории	13
3.14	Результат выполнения командного файла	13
3.15	Командный файл, который получает в качестве аргумента командной строки формат файла и выводит их в нужной директории	14
3.16	Результат выполнения командного файла	14

1 Цель работы

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

2 Задание

Выполнить данные пункты и ответить на вопросы:

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию `back up` 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.
2. 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.
3. 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.
4. 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.

3 Выполнение лабораторной работы

Перед выполнением лабораторной работы я хорошо ознакомилась с теоритическим материалом для её выполнения Ссылка 1

3.1 Выполнение 1-го пункта задания

```
adkekisheva@dk8n78 ~ $ man tar
adkekisheva@dk8n78 ~ $ tar --help
Использование: tar [ПАРАМЕТР...] [ФАЙЛ]...
GNU 'tar' saves many files together into a single tape or disk archive, and can
restore individual files from the archive.

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.

Имя выбранного локального файла:
```

Рис. 3.1: Справка tar

Во-первых, я изучила справку по способу использования tar. (рис. @fig:001) И нашла необходимые опции для выполнения задания (рис. @fig:002) (рис. @fig:003) :

-c, --create	к архиву создание нового архива
-d, --diff, --compare	поиск различий между архивом и файловой системой

Рис. 3.2: Опция -с - создание

```

Выбор и переключение устройств:

-f, --file=АРХИВ      использовать файл или
                      устройство АРХИВ
--force-local         файл архива является
                      локальным, даже если
                      содержит двоеточие

```

Рис. 3.3: Опция -f - применение файла или устройство АРХИВ

```

adkekisheva@dk8n78 ~ $ touch script01.sh
adkekisheva@dk8n78 ~ $ chmod +x script01.sh
adkekisheva@dk8n78 ~ $ emacs script01.sh
adkekisheva@dk8n78 ~ $ ./script01.sh

```

Рис. 3.4: Создание файла, присвоение прав доступа

Для написания первого скрипта, я создала файл с расширением .sh, так как я буду работать в emacs. Наделила этот файл правом на выполнения и прикрепила к написанию. (рис. @fig:004)

```

File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
cp script01.sh ~/.config/libreoffice/4/user/backup/
cd ~/.config/libreoffice/4/user/backup
tar -cf lab11.tar script01.sh
-:--- script01.sh All L1 (Shell-script[bash]) Чт мая 27 11:36 0.75

```

Рис. 3.5: Написание скрипта

Написала скрипт (рис. @fig:005), который при запуске будет делать резервную копию самого себя в директорию backup. Для этого я нашла путь к каталогу backup. В первой строке мы обязательно пишем #!/bin/bash. Далее применила команду копирования cp и скопировала скрипт в каталог backup. Далее перешла в этот

каталог и после командой архивирования tar с опциями -cf сделала архив файла со скриптом с именем lab11.tar.

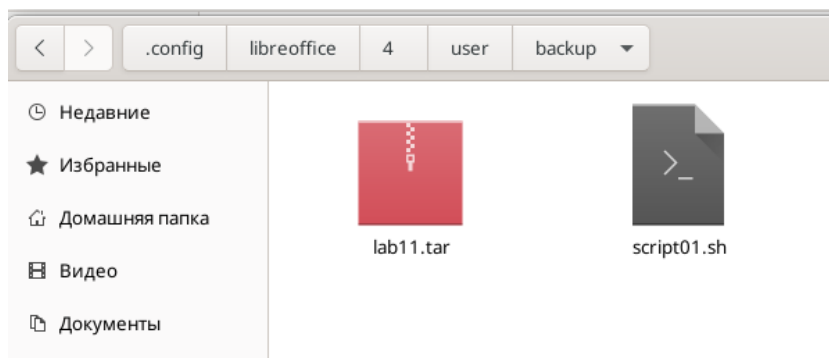


Рис. 3.6: Результат

Запустила скрипт ./ (название файла) (рис. @fig:004) и проверила результат (рис. @fig:006)

3.2 Выполнение 2-го пункта задания

Написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. На мой взгляд, это можно сделать двумя способами:

1. Воспользоваться head - (количество строк для обработки) (рис. @fig:007) (рис. @fig:008).

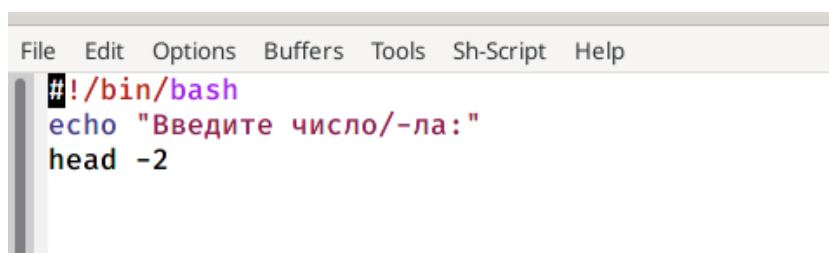


Рис. 3.7: Командный файл, обрабатывающий любое произвольное число аргументов командной строки №1

```

adkekisheva@dk8n78 ~ $ touch script02.sh
adkekisheva@dk8n78 ~ $ chmod +x script02.sh
adkekisheva@dk8n78 ~ $ emacs script02.sh
adkekisheva@dk8n78 ~ $ bash script02.sh
Введите число/-ла:
2 3 10 20
2 3 10 20
adkekisheva@dk8n78 ~ $ emacs script02.sh
adkekisheva@dk8n78 ~ $ bash script02.sh
Введите число/-ла:
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
6 30 31 56
6 30 31 56

```

Рис. 3.8: Результат работы head

2. Ввести \$*, что означает что вызывая командный файл нам не выйдет предложение ввести символы, их мы будем должны ввести сами в командной строке (рис. @fig:009) (рис. @fig:010).

```

#!/bin/bash
echo "Обработанные числа: $*"

```

Рис. 3.9: Командный файл, обрабатывающий любое произвольное число аргументов командной строки №2

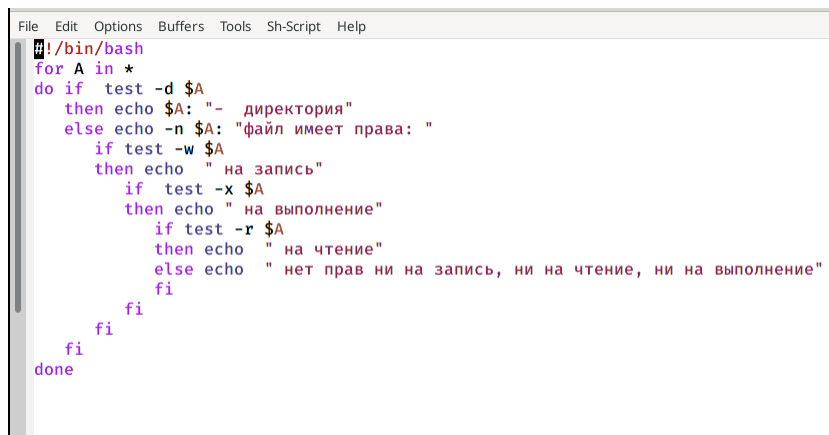
```

^Cadkekisheva@dk8n78 ~ $ emacs 2.sh
^Cadkekisheva@dk8n78 ~ $ bash 2.sh 2 3 10 23 4 5 6 7 8
Обработанные числа: 2 3 10 23 4 5 6 7 8

```

Рис. 3.10: Результат работы S*

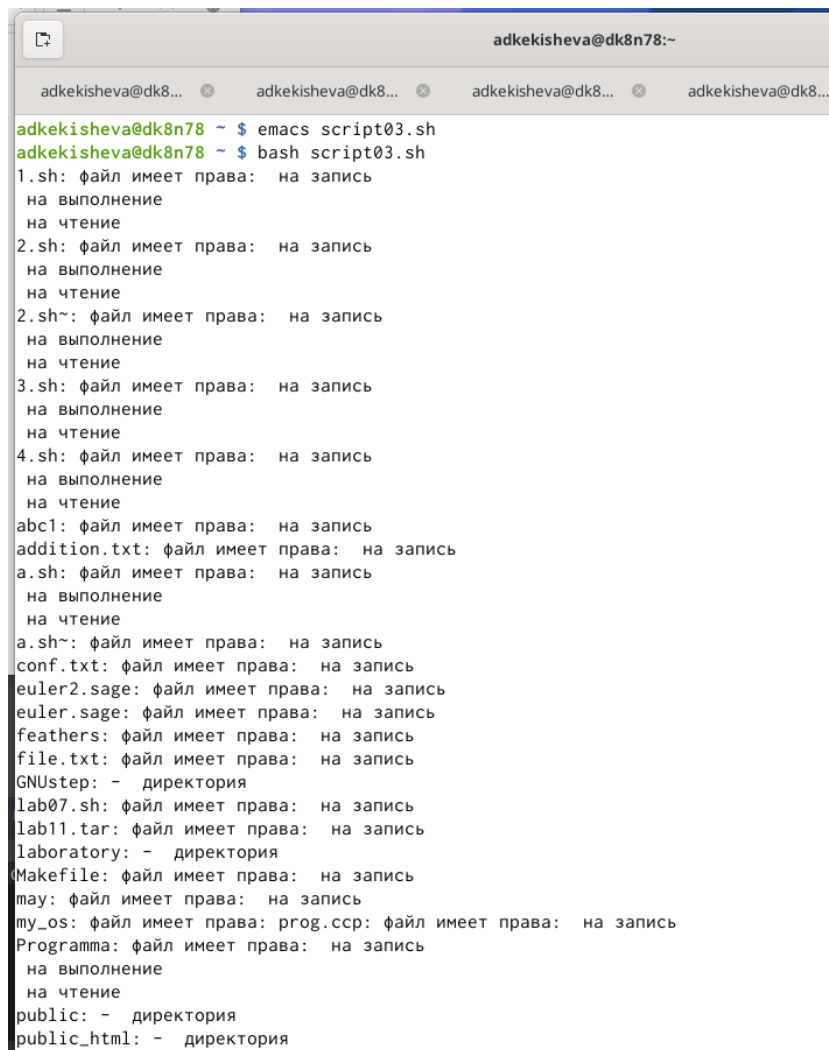
3.3 Выполнение 3-го пункта задания

A screenshot of a terminal window with a menu bar (File, Edit, Options, Buffers, Tools, Sh-Script, Help) and a title bar (/bin/bash). The terminal displays a shell script that iterates over files in the current directory. For each file, it checks if it's a directory. If it is, it prints a message. If it's a regular file, it checks for write, execute, and read permissions, printing messages for each. If no permissions are found, it prints a message stating that. The script ends with 'done'.

```
#!/bin/bash
for A in *
do if test -d $A
then echo $A: "- директория"
else echo -n $A: "файл имеет права: "
if test -w $A
then echo " на запись"
if test -x $A
then echo " на выполнение"
if test -r $A
then echo " на чтение"
else echo " нет прав ни на запись, ни на чтение, ни на выполнение"
fi
fi
fi
done
```

Рис. 3.11: Командный файл, выводящий информацию о правах доступа

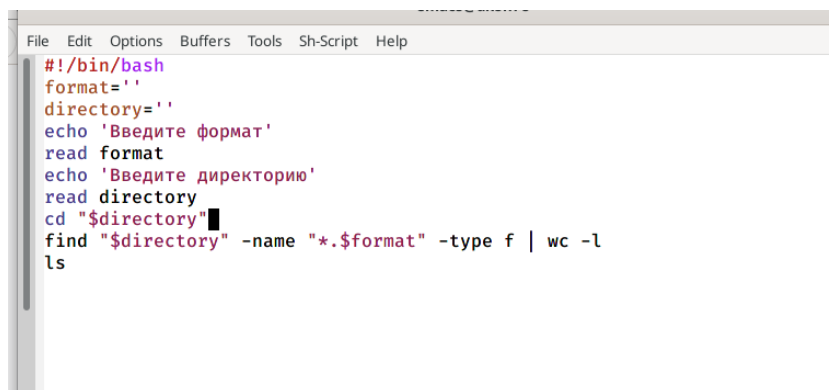
Далее написала командный файл — аналог команды `ls` (без использования самой этой команды и команды `dir`), который выдаёт информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога (рис. @fig:011). Для это я воспользовалась циклом `for` в котором переменная `A` будет принимать значения, равные именам этих файлов. И потом в помощью операторов условия `if` проверяем файл это или директория. Если директория просто выводим, что это директория, если файл, то проверяем его, на наличие прав на чтение `test -r`, на запись `-w`, на выполнение `-x`, в противном случае у файла нет прав. Test проверяет на истину. (рис. @fig:012)



```
adkekisheva@dk8n78:~  
adkekisheva@dk8...  adkekisheva@dk8...  adkekisheva@dk8...  adkekisheva@dk8...  
adkekisheva@dk8n78 ~ $ emacs script03.sh  
adkekisheva@dk8n78 ~ $ bash script03.sh  
1.sh: файл имеет права: на запись  
на выполнение  
на чтение  
2.sh: файл имеет права: на запись  
на выполнение  
на чтение  
2.sh~: файл имеет права: на запись  
на выполнение  
на чтение  
3.sh: файл имеет права: на запись  
на выполнение  
на чтение  
4.sh: файл имеет права: на запись  
на выполнение  
на чтение  
abc1: файл имеет права: на запись  
addition.txt: файл имеет права: на запись  
a.sh: файл имеет права: на запись  
на выполнение  
на чтение  
a.sh~: файл имеет права: на запись  
conf.txt: файл имеет права: на запись  
euler2.sage: файл имеет права: на запись  
euler.sage: файл имеет права: на запись  
feathers: файл имеет права: на запись  
file.txt: файл имеет права: на запись  
GNUstep: - директория  
lab07.sh: файл имеет права: на запись  
lab11.tar: файл имеет права: на запись  
laboratory: - директория  
Makefile: файл имеет права: на запись  
may: файл имеет права: на запись  
my_os: файл имеет права: prog.csr: файл имеет права: на запись  
Programma: файл имеет права: на запись  
на выполнение  
на чтение  
public: - директория  
public_html: - директория
```

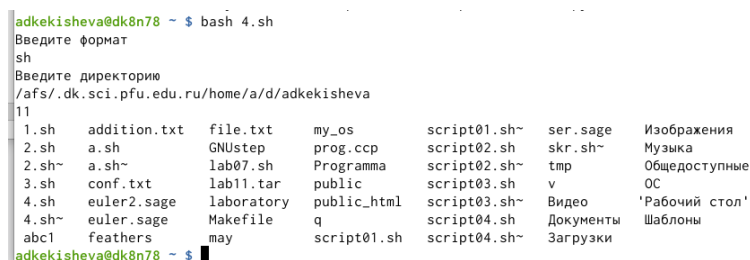
Рис. 3.12: Результат выполнения командного файла

3.4 Выполнение 4-го пункта задания



```
#!/bin/bash
format=''
directory=''
echo 'Введите формат'
read format
echo 'Введите директорию'
read directory
cd "$directory"
find "$directory" -name ".*$format" -type f | wc -l
ls
```

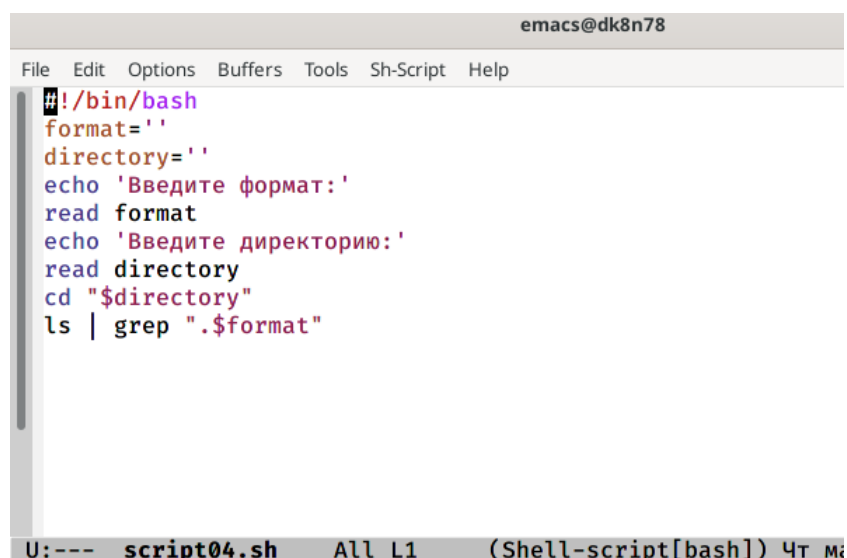
Рис. 3.13: Командный файл, который получает в качестве аргумента командной строки формат файла и вычисляет количество таких файлов в указанной директории



```
adkeki sheva@dk8n78 ~ $ bash 4.sh
Введите формат
sh
Введите директорию
/afs/.dk.sci.pfu.edu.ru/home/a/d/adkeki sheva
11
1.sh      addition.txt  file.txt      my_os        script01.sh~  ser.sage      Изображения
2.sh~     a.sh          GNUstep       prog.ccp     script02.sh~  skr.sh~       Музыка
2.sh~     a.sh~         lab07.sh      Programma    script02.sh~  tmp           Общедоступные
3.sh      conf.txt      lab11.tar     public       script03.sh~  v            ОС
4.sh      euler2.sage   laboratory    public_html  script03.sh~  Видео        'Рабочий стол'
4.sh~     euler.sage    Makefile      q            script04.sh~  Документы    Шаблоны
abc1      feathers      may           script01.sh  script04.sh~  Загрузки
```

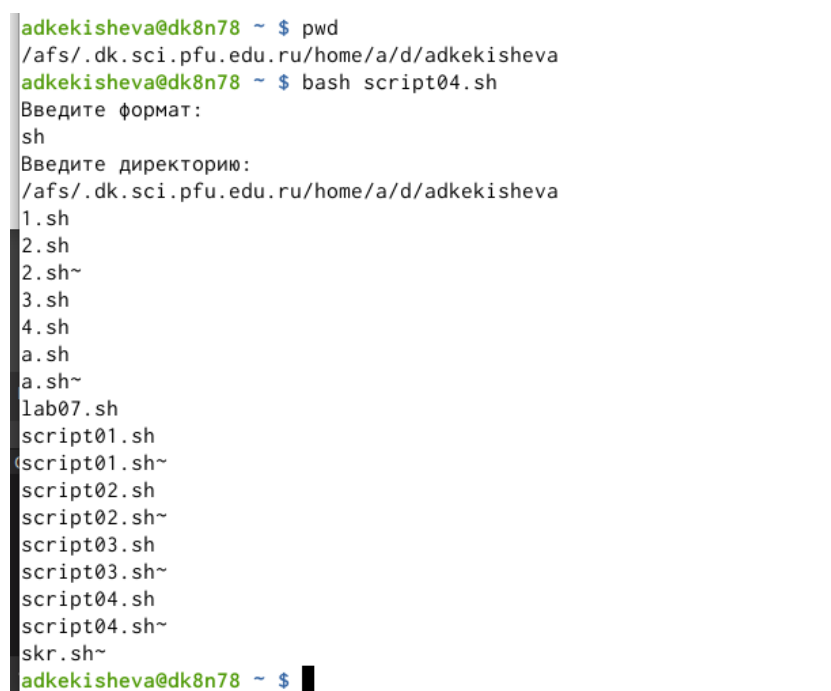
Рис. 3.14: Результат выполнения командного файла

Написала командный файл, который получает в качестве аргумента командной строки формат файла (.txt,.doc,.jpg,.pdfи т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки (рис. @fig:013) (рис. @fig:014) . Здесь, я создала переменные, считывала их командой read и применяла команду find для поиска файлов с определённым расширением, а также конвеер и команду wc с опцией -l. Команда wc -l выводит количество строк в объекте.Ссылка 2



```
emacs@dk8n78
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
format=''
directory=''
echo 'Введите формат:'
read format
echo 'Введите директорию:'
read directory
cd "$directory"
ls | grep ".$format"
U:--- script04.sh All L1 (Shell-script[bash]) Чт м
```

Рис. 3.15: Командный файл, который получает в качестве аргумента командной строки формат файла и выводит их в нужной директории



```
adkekisheva@dk8n78 ~ $ pwd
/afs/.dk.sci.pfu.edu.ru/home/a/d/adkekisheva
adkekisheva@dk8n78 ~ $ bash script04.sh
Введите формат:
sh
Введите директорию:
/afs/.dk.sci.pfu.edu.ru/home/a/d/adkekisheva
1.sh
2.sh
2.sh~
3.sh
4.sh
a.sh
a.sh~
lab07.sh
script01.sh
script01.sh~
script02.sh
script02.sh~
script03.sh
script03.sh~
script04.sh
script04.sh~
skr.sh~
adkekisheva@dk8n78 ~ $
```

Рис. 3.16: Результат выполнения командного файла

Также в начале работы у меня получалось так, что командный файл выводил имена файлов, но он не вычислял количество файлов (рис. @fig:015) (рис.

@fig:016)

4 Вывод

Я изучила основы программирования в оболочке ОС UNIX/Linux, научилась писать командные файлы и скрипты.

5 Библиография

1. Ссылка 1
2. Ссылка 2

Контрольные вопросы:

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

Командная оболочка— 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 на оболочкой Борна, использующая Подобный 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. 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.

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"`.

4. Каково назначение операторов `let` и `read`?

Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. 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;

Команда `read` позволяет читать значения переменных со стандартного ввода.

5. Какие арифметические операции можно применять в языке программирования bash?

Сложение, вычитание, умножение, деление, а также эти же операции с присваиванием значения переменной, также операция отрицания, остаток от деления, побитовый сдвиг, побитовое дополнение и другие.

6. Что означает операция `(())`?

Операция применяется для записи условия. Далее, можно осуществлять присвоение результатов условных выражений переменным, также как и использовать результаты арифметических вычислений в качестве условий.

7. Какие стандартные имена переменных Вам известны?

– PATH (т.е. \$PATH) – 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 /.

– 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. В командном процессоре Си имеется ещё несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set.

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?

Командный файл – это файл, в который помещена последовательность команд. Сначала командный файл можно выполнить по команде: `bash командный_файл [аргументы]` и чтобы не вводить каждый раз последовательности символов `bash`, необходимо обеспечить доступ к этому файлу на выполнение (`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?

Для определения функции в `bash` используется ключевое слово `function`, после которого следует имя функции и список команд, заключённых в фигурные скобки.

12. Каким образом можно выяснить, является файл каталогом или обычным файлом?

13. Каково назначение команд `set`, `typeset` и `unset`?

Команда `set` – создание массива, используется команда `set` с флагом `-A`

Команда `unset` с флагом `-f` – удаление функции.

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

- `f` – перечисляет определённые на текущий момент функции;
- `ft` – при последующем вызове функции инициализирует 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. 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.

15. Назовите специальные переменные языка `bash` и их назначение.

Специальных 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;
- `${#}` — *возвращает целое число — количество слов, которые были результатом* `$`;
- `${#name}` — возвращает целое значение длины строки в переменной `name`;
- `${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`.