

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №8

Дисциплина: Операционные системы

Обрезкова Анастасия Владимировна

Содержание

1	Цель работы	5
1.1	Теоретическое введение	5
2	Выполнение лабораторной работы	7
2.1	Создание нового файла с использованием vi	7
3	Выводы	14
	Список литературы	22

Список иллюстраций

2.1	Создание исполняемого файла prog1	7
2.2	Текст программы	8
2.3	Результат работы	8
2.4	Создание исполняемого файла prog2	8
2.5	Текст программы	9
2.6	Результат работы	9
2.7	Создание исполняемого файла prog3	10
2.8	Текст программы	10
2.9	Результат работы	11
2.10	Создание исполняемого файла prog4	12
2.11	Текст программы	12
2.12	Результат работы	12

Список таблиц

1 Цель работы

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

1.1 Теоретическое введение

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 на оболочкой Борна, использующая C-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).

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/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Керна.

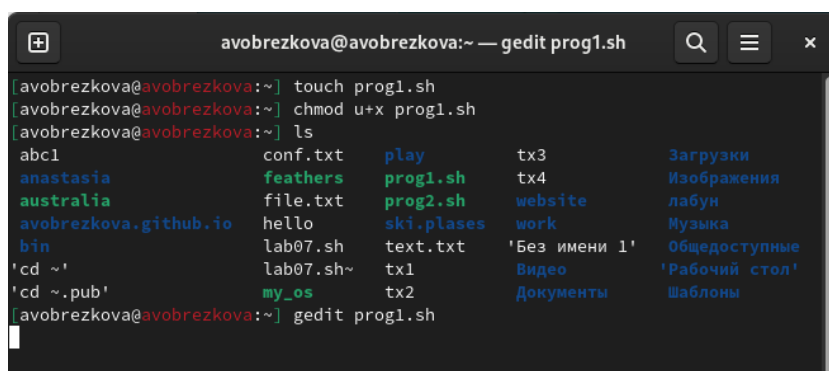
Рассмотрим основные элементы программирования в оболочке `bash`. В других оболочках большинство команд будет совпадать с описанными ниже.

2 Выполнение лабораторной работы

2.1 Создание нового файла с использованием vi

1. Написала скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в моем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar.

Кроме этого я создала директорию backup, в которую потом сохраняется резервная копия файла, после этого запустила программу и проверила ее работу. (рис. [2.1]; рис. [2.2]; рис. [2.3])



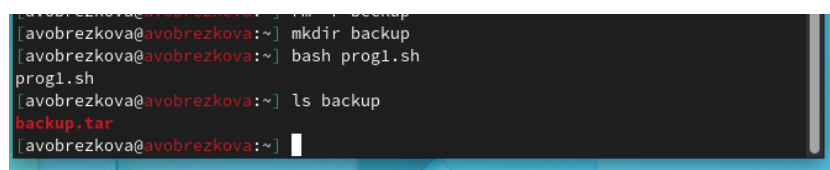
```
avobrezkova@avobrezkova:~ — gedit prog1.sh
[avobrezkova@avobrezkova:~] touch prog1.sh
[avobrezkova@avobrezkova:~] chmod u+x prog1.sh
[avobrezkova@avobrezkova:~] ls
abc1          conf.txt      play          tx3           Загрузки
anastasia     feathers     prog1.sh      tx4           Изображения
australia     file.txt     prog2.sh      website       лабун
avobrezkova.github.io hello        ski.plases    work          Музыка
bin           lab07.sh     text.txt      'Без имени 1' Общедоступные
'cd ~'        lab07.sh~   tx1           Видео         'Рабочий стол'
'cd ~/.pub'   my_os       tx2           Документы     Шаблоны
[avobrezkova@avobrezkova:~] gedit prog1.sh
```

Рис. 2.1: Создание исполняемого файла prog1



```
Открыть  + prog1.sh
1 #!/bin/bash
2 tar -cvf ~/backup/backup.tar prog1.sh
```

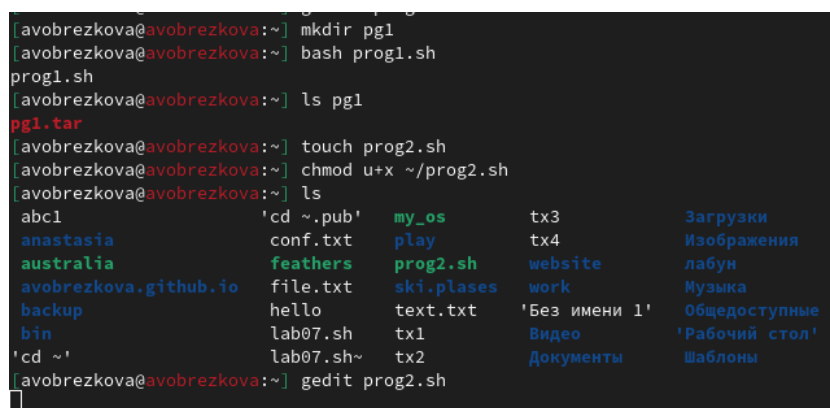
Рис. 2.2: Текст программы



```
[avobrezkova@avobrezkova:~] mkdir backup
[avobrezkova@avobrezkova:~] bash prog1.sh
prog1.sh
[avobrezkova@avobrezkova:~] ls backup
backup.tar
[avobrezkova@avobrezkova:~]
```

Рис. 2.3: Результат работы

2. Написала 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. [2.4]; рис. [2.5]; рис. [2.6])



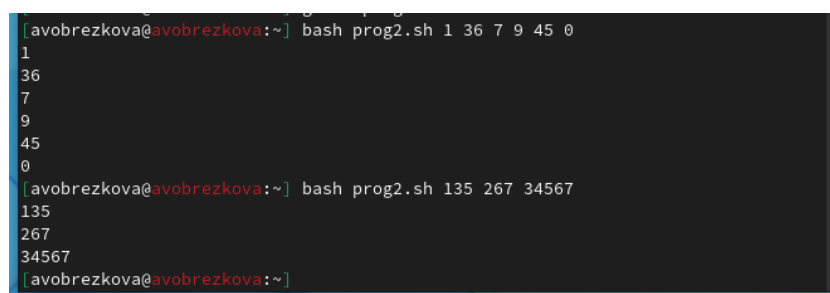
```
[avobrezkova@avobrezkova:~] mkdir pg1
[avobrezkova@avobrezkova:~] bash prog1.sh
prog1.sh
[avobrezkova@avobrezkova:~] ls pg1
pg1.tar
[avobrezkova@avobrezkova:~] touch prog2.sh
[avobrezkova@avobrezkova:~] chmod u+x ~/prog2.sh
[avobrezkova@avobrezkova:~] ls
abc1          'cd ~/.pub'  my_os        tx3           Загрузки
anastasia     conf.txt     play         tx4           Изображения
australia     feathers     prog2.sh     website      лабун
avobrezkova.github.io file.txt     ski.plases   work          Музыка
backup        hello       text.txt     'Без имени 1' Общедоступные
bin           lab07.sh    tx1          Видео        'Рабочий стол'
'cd ~'        lab07.sh~   tx2          Документы    Шаблоны
[avobrezkova@avobrezkova:~] gedit prog2.sh
```

Рис. 2.4: Создание исполняемого файла prog2



```
1 #!/bin/bash
2 # echo 'Введите число:'
3 # read n
4 for N in $*
5 do echo $N
6 done
```

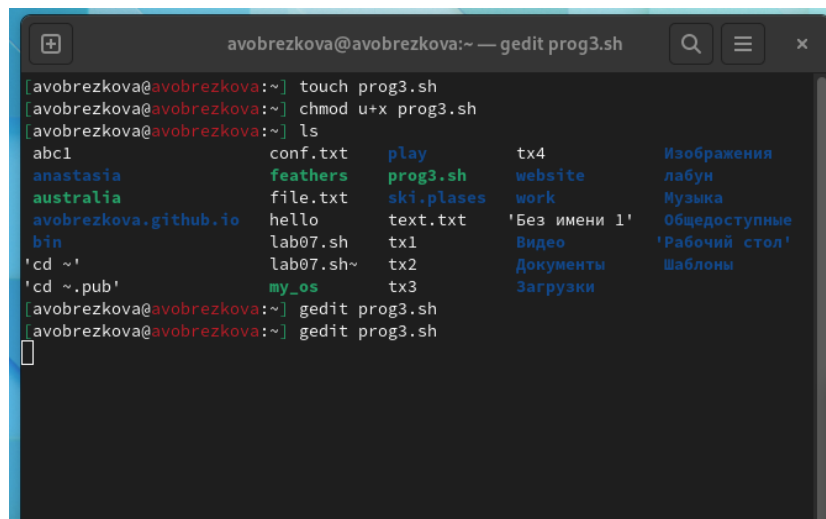
Рис. 2.5: Текст программы



```
[avobrezkova@avobrezkova:~] bash prog2.sh 1 36 7 9 45 0
1
36
7
9
45
0
[avobrezkova@avobrezkova:~] bash prog2.sh 135 267 34567
135
267
34567
[avobrezkova@avobrezkova:~]
```

Рис. 2.6: Результат работы

3. Написала 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. [2.7]; рис. [2.8]; рис. [2.9])



```
avobrezkova@avobrezkova:~ — gedit prog3.sh
[avobrezkova@avobrezkova:~] touch prog3.sh
[avobrezkova@avobrezkova:~] chmod u+x prog3.sh
[avobrezkova@avobrezkova:~] ls
abcl1      conf.txt  play      tx4        Изображения
anastasia  feathers prog3.sh   website    лабун
australia  file.txt  ski.plases work        Музыка
avobrezkova.github.io hello      text.txt  'Без имени 1' Общедоступные
bin        lab07.sh  tx1       Видео      'Рабочий стол'
'cd ~'     lab07.sh~ tx2       Документы  Шаблоны
'cd ~/.pub' my_os     tx3       Загрузки

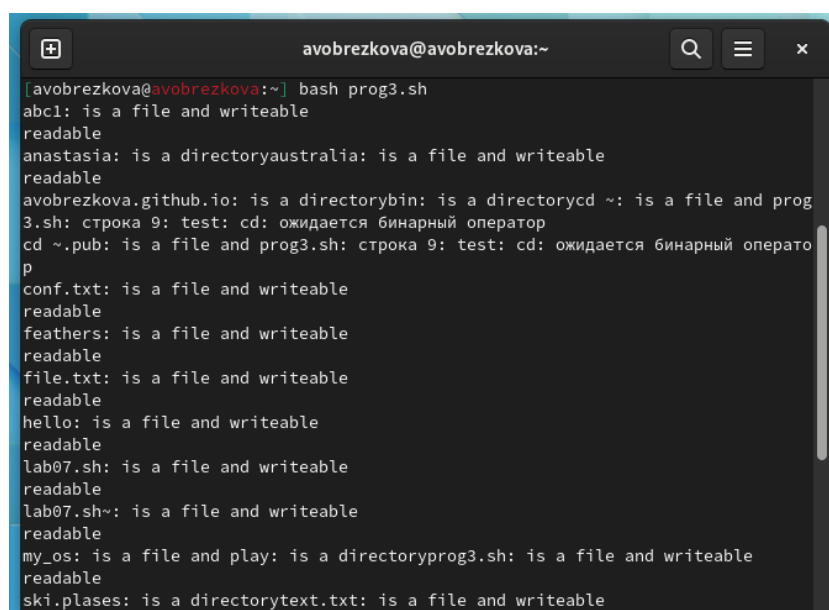
[avobrezkova@avobrezkova:~] gedit prog3.sh
[avobrezkova@avobrezkova:~] gedit prog3.sh
```

Рис. 2.7: Создание исполняемого файла prog3



```
Открыть ▾ + prog3.sh ~/
1 #!/bin/bash
2 for N in *
3 do
4 if test -d "$N"
5 then
6 echo -n "$N: is a directory"
7 else
8 echo -n "$N: is a file and "
9 if test -w $N
10 then
11 echo writeable
12 if test -r $N
13 then
14 echo "readable"
15 else
16 echo "neither readable nor writeable"
17 fi
18 fi
19 fi
20 done
```

Рис. 2.8: Текст программы

A terminal window titled 'avobrezkova@avobrezkova:~' with search, menu, and close icons. The terminal shows the execution of a script 'prog3.sh'. The output lists various files and directories with their permissions and readability status. Some lines contain Russian text indicating an error: 'строка 9: test: cd: ожидается бинарный оператор' (line 9: test: cd: binary operator expected).

```
avobrezkova@avobrezkova:~  
[avobrezkova@avobrezkova:~] bash prog3.sh  
abc1: is a file and writeable  
readable  
anastasia: is a directoryaustralia: is a file and writeable  
readable  
avobrezkova.github.io: is a directorybin: is a directorycd ~: is a file and prog  
3.sh: строка 9: test: cd: ожидается бинарный оператор  
cd ~/.pub: is a file and prog3.sh: строка 9: test: cd: ожидается бинарный операто  
p  
conf.txt: is a file and writeable  
readable  
feathers: is a file and writeable  
readable  
file.txt: is a file and writeable  
readable  
hello: is a file and writeable  
readable  
lab07.sh: is a file and writeable  
readable  
lab07.sh~: is a file and writeable  
readable  
my_os: is a file and play: is a directoryprog3.sh: is a file and writeable  
readable  
ski.places: is a directorytext.txt: is a file and writeable
```

Рис. 2.9: Результат работы

4. Написала 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. [2.10]; рис. [2.11]; рис. [2.12])

```
avobrezkova@avobrezkova:~ — gedit prog4.sh
[avobrezkova@avobrezkova:~] touch prog4.sh
[avobrezkova@avobrezkova:~] chmod u+x prog4.sh
[avobrezkova@avobrezkova:~] ls
abcl      conf.txt  play      tx4      Изображения
anastasia feathers  prog4.sh  website  лабун
australia file.txt  ski.places work      Музыка
avobrezkova.github.io hello    text.txt  'Без имени 1' Общедоступные
bin       lab07.sh tx1       Видео     'Рабочий стол'
'cd ~'    lab07.sh~ tx2      Документы  Шаблоны
'cd ~/.pub' my_os    tx3      Загрузки
[avobrezkova@avobrezkova:~] gedit prog4.sh
```

Рис. 2.10: Создание исполняемого файла prog4

```
Открыть + *prog4.sh
1 #!/bin/bash
2 format=""
3 directory=""
4 echo "Введите формат файла: "
5 read format
6 echo "Введите директорию: "
7 read directory
8 find "${directory}" -name "*.${format}" -type f
9 | wc -l
10 ls
```

Рис. 2.11: Текст программы

```
[avobrezkova@avobrezkova:~] bash prog4.sh
Введите формат файла:
txt
Введите директорию:
/home/avobrezkova
40
abcl      conf.txt  play      tx4      Изображения
anastasia feathers  prog4.sh  website  лабун
australia file.txt  ski.places work      Музыка
avobrezkova.github.io hello    text.txt  'Без имени 1' Общедоступные
bin       lab07.sh tx1       Видео     'Рабочий стол'
'cd ~'    lab07.sh~ tx2      Документы  Шаблоны
'cd ~/.pub' my_os    tx3      Загрузки
[avobrezkova@avobrezkova:~]
```

Рис. 2.12: Результат работы

Данные изменения можно проверить по ссылке: https://github.com/avobrezkova/study_2022-2023_os-intro/tree/master/labs/lab10

3 Выводы

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

#Контрольные вопросы

1. 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. 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 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. Что такое POSIX?

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/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

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

```
«echo “Please enter Month and Day of Birth ?”»
```

```
«read mon day trash»
```

В переменные `mon` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введенную информацию и игнорировать её.

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

В языке программирования `bash` можно применять такие арифметические операции как сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%).

6. Что означает операция (())?

В (()) можно записывать условия оболочки `bash`, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.

7. Какие стандартные имена переменных Вам известны?

Стандартные переменные:

- `PATH`: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /. Если имя команды содержит

хотя бы один символ /, то последовательность поиска, предписываемая значением переменной PATH, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневого или текущего каталога.

- PS1 и PS2: эти переменные предназначены для отображения промптера командного процессора. PS1 – это промптер командного процессора, по умолчанию его значение равно символу \$ или #. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, используется промптер PS2. Он по умолчанию имеет значение символа >.
- HOME: имя домашнего каталога пользователя. Если команда cd вводится без аргументов, то происходит переход в каталог, указанный в этой переменной.
- IFS: последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (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. 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`.

Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: `«bash командный_файл [аргументы]»` Чтобы не вводить каждый раз последовательности символов `bash`, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды `«chmod +x имя_файла»` Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как будто он является выполняемой программой. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществит её интерпретацию.

10. Как создавать и запускать командные файлы?

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`.

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

Чтобы выяснить, является ли файл каталогом или обычным файлом, необходимо воспользоваться командами «test -f [путь до файла]» (для проверки, является ли обычным файлом) и «test -d [путь до файла]» (для проверки, является ли каталогом).

12. Каково назначение команд set, typeset и unset?

Команду «set» можно использовать для вывода списка переменных окружения. В системах Ubuntu и Debian команда «set» также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду «set | more». Команда «typeset» предназначена для наложения ограничений на переменные. Команду «unset» следует использовать для удаления переменной из окружения командной оболочки.

13. 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.

14. Назовите специальные переменные языка bash и их назначение.

Специальные переменные:

- `$*` – отображается вся командная строка или параметры оболочки;
- `$?` – код завершения последней выполненной команды;
- `$$` – уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
- `#!` – номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
- `$-` – значение флагов командного процессора;
- `${#}` – *возвращает целое число – 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`.

Список литературы

1. https://esystem.rudn.ru/pluginfile.php/1976045/mod_resource/content/4/010-lab_shell_prog_1.pdf