

Отчёт по лабораторной работе №12

Дисциплина: Операционные системы

Верниковская Екатерина Андреевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Ответы на контрольные вопросы	14
5	Выводы	23
6	Список литературы	24

Список иллюстраций

3.1	Создание файла task1.sh и добавление прав на исполнение	7
3.2	Открытие файла task1.sh	7
3.3	Написанный скрипт для task1.sh	7
3.4	Проверка работы скрипта task1.sh (1)	8
3.5	Проверка работы скрипта task1.sh (2)	8
3.6	Проверка работы скрипта task1.sh (3)	8
3.7	Создание файла task2.sh и добавление прав на исполнение	8
3.8	Открытие файла task2.sh	9
3.9	Написанный скрипт для task2.sh	9
3.10	Проверка работы скрипта task2.sh	9
3.11	Создание файла task3.sh и добавление прав на исполнение	10
3.12	Открытие файла task3.sh	10
3.13	Написанный скрипт для task3.sh	10
3.14	Проверка работы скрипта task3.sh	12
3.15	Создание файла task4.sh и добавление прав на исполнение	12
3.16	Открытие файла task4.sh	12
3.17	Написанный скрипт для task4.sh	12
3.18	Проверка работы скрипта task4.sh (1)	13
3.19	Проверка работы скрипта task4.sh (2)	13

Список таблиц

1 Цель работы

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

2 Задание

1. 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.
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 также передаётся в виде аргумента командной строки.

3 Выполнение лабораторной работы

Создаю файл для первого задания с расширением sh и делаю его исполняемым (рис. 3.1)

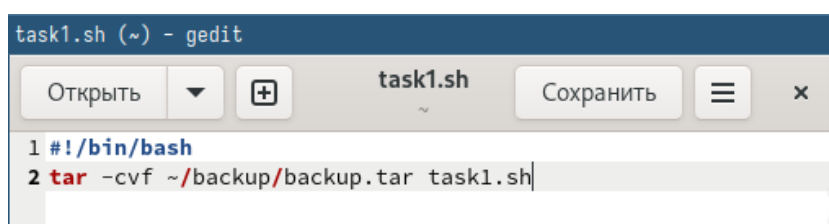
```
[eavernikovskaya@eavernikovskaya ~]$ touch task1.sh  
[eavernikovskaya@eavernikovskaya ~]$ chmod +x task1.sh
```

Рис. 3.1: Создание файла task1.sh и добавление прав на исполнение

Открываю файл task1.sh в текстовом редакторе gedit и пишу 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. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar (рис. 3.2), (рис. 3.3)

```
[eavernikovskaya@eavernikovskaya ~]$ gedit task1.sh  
█
```

Рис. 3.2: Открытие файла task1.sh



The screenshot shows the gedit text editor window titled "task1.sh (~) - gedit". The window has a toolbar with buttons for "Открыть" (Open), a dropdown menu, a "+" icon, "task1.sh", "Сохранить" (Save), a hamburger menu icon, and a close "x" icon. The editor content shows two lines of code: line 1 is "#!/bin/bash" and line 2 is "tar -cvf ~/backup/backup.tar task1.sh".

Рис. 3.3: Написанный скрипт для task1.sh

Программа для задания №1:

```
#!/bin/bash  
tar -cvf ~/backup/backup.tar task1.sh
```

Далее запускаю файл с помощью `bash` и проверяю работу скрипта (рис. 3.4), (рис. 3.5), (рис. 3.6)

```
[eavernikovskaya@eavernikovskaya ~]$ mkdir backup  
[eavernikovskaya@eavernikovskaya ~]$ bash task1.sh  
task1.sh  
[eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.4: Проверка работы скрипта `task1.sh` (1)

```
[eavernikovskaya@eavernikovskaya ~]$ ls backup/  
backup.tar  
[eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.5: Проверка работы скрипта `task1.sh` (2)

The image shows a file manager window with a top status bar and two main panes. The top bar includes a terminal icon, the path 'mc [eavernikovskaya@eavernikovskaya]:~/backup/backup.tar/utar://', a progress indicator at 40%, a date '10.0.2 15/24', a storage indicator at 1%, a percentage '29%', a 'mc' icon, a signal strength indicator at 87%, and a clock showing '17:53'. The left pane is titled 'Левая панель' and shows a directory tree with '~/backup' selected. The right pane is titled 'Правая панель' and shows the contents of 'backup.tar/utar://'. Both panes have a table-like view with columns for file name, size, and modification time. The left pane shows '..' and 'backup.tar'. The right pane shows '..' and '*task1.sh'.

Левая панель	Файл	Команда	Настройки	Правая панель		
~/backup	Имя	Размер	Время правки	Имя	Размер	Время правки
..	-BBERX-	18240	апр 25 17:47	..	-BBERX-	50
backup.tar				*task1.sh		

Рис. 3.6: Проверка работы скрипта `task1.sh` (3)

Создаю файл для второго задания с расширением `sh` и делаю его исполняемым (рис. 3.7)

```
[eavernikovskaya@eavernikovskaya ~]$ touch task2.sh  
[eavernikovskaya@eavernikovskaya ~]$ chmod +x task2.sh  
[eavernikovskaya@eavernikovskaya ~]$
```

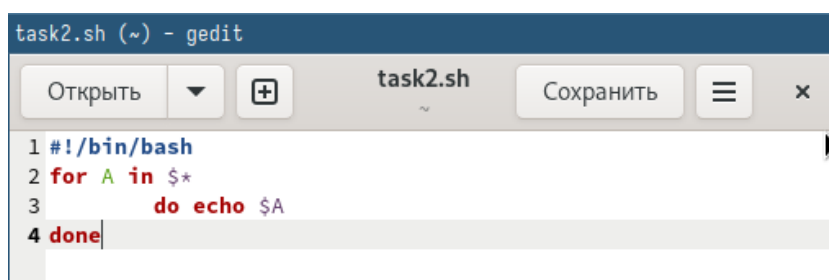
Рис. 3.7: Создание файла `task2.sh` и добавление прав на исполнение

Открываю файл `task2.sh` в текстовом редакторе `gedit` и пишу 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. 3.8), (рис. 3.9)

```
[eavernikovskaya@eavernikovskaya ~]$ gedit task2.sh
```

Рис. 3.8: Открытие файла task2.sh



The screenshot shows a gedit window titled 'task2.sh (~) - gedit'. The window has a toolbar with buttons for 'Открыть', a dropdown menu, a '+' icon, 'task2.sh ~', 'Сохранить', a hamburger menu, and a close button 'x'. The script content is as follows:

```
1 #!/bin/bash
2 for A in $*
3     do echo $A
4 done
```

Рис. 3.9: Написанный скрипт для task2.sh

Программа для задания №2:

```
#!/bin/bash
for A in $*
do echo $A
done
```

Далее запускаю файл с помощью bash и проверяю его работу (рис. 3.10)

```
[eavernikovskaya@eavernikovskaya ~]$ bash task2.sh 13 katya 333 18
13
katya
333
18
[eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.10: Проверка работы скрипта task2.sh

Создаю файл для третьего задания с расширением sh и делаю его исполняемым (рис. 3.11)

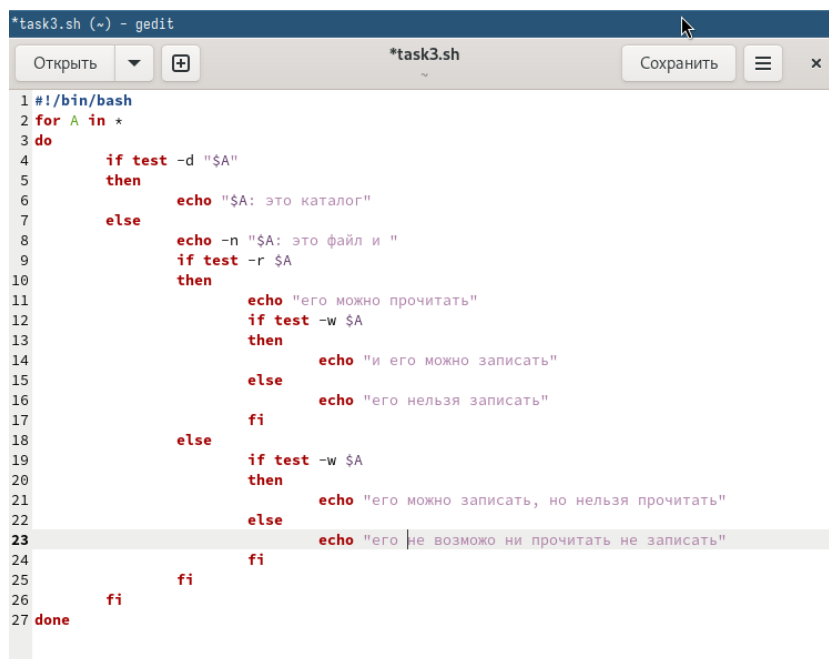
```
[eavernikovskaya@eavernikovskaya ~]$ touch task3.sh
[eavernikovskaya@eavernikovskaya ~]$ chmod +x task3.sh
[eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.11: Создание файла task3.sh и добавление прав на исполнение

Открываю файл task3.sh в текстовом редакторе gedit и пишу командный файл — аналог команды ls (без использования самой этой команды и команды dir). Он будет выдавать информацию о нужном каталоге и выводить информацию о возможностях доступа к файлам этого каталога. (рис. 3.12), (рис. 3.13)

```
[eavernikovskaya@eavernikovskaya ~]$ gedit task3.sh
```

Рис. 3.12: Открытие файла task3.sh



```
*task3.sh (~) - gedit
Открыть  +  *task3.sh  Сохранить  ☰  ✕

1 #!/bin/bash
2 for A in *
3 do
4     if test -d "$A"
5     then
6         echo "$A: это каталог"
7     else
8         echo -n "$A: это файл и "
9         if test -r $A
10        then
11            echo "его можно прочитать"
12            if test -w $A
13            then
14                echo "и его можно записать"
15            else
16                echo "его нельзя записать"
17            fi
18        else
19            if test -w $A
20            then
21                echo "его можно записать, но нельзя прочитать"
22            else
23                echo "его не возможно ни прочитать не записать"
24            fi
25        fi
26    fi
27 done
```

Рис. 3.13: Написанный скрипт для task3.sh

Программа для задания №3:

```
#!/bin/bash
for A in *
```

```

do
    if test -d "$A"
    then
        echo "$A: это каталог"
    else
        echo -n "$A: это файл и "
        if test -r $A
        then
            echo "его можно прочитать"
            if test -w $A
            then
                echo "и его можно записать"
            else
                echo "его нельзя записать"
            fi
        else
            if test -w $A
            then
                echo "его можно записать, но нельзя прочитать"
            else
                echo "его не возможно ни прочитать не записать"
            fi
        fi
    fi
done

```

Далее запускаю файл с помощью bash и проверяю его работу (рис. 3.14)

```
[eavernikovskaya@eavernikovskaya ~]$ bash task3.sh
abc1: это файл и его можно записать, но нельзя прочитать
australia: это каталог
backup: это каталог
conf.txt: это файл и его можно прочитать
и его можно записать
feathers: это файл и его можно прочитать
и его можно записать
```

Рис. 3.14: Проверка работы скрипта task3.sh

Создаю файл для третьего задания с расширением sh и делаю его исполняемым (рис. 3.15)

```
[eavernikovskaya@eavernikovskaya ~]$ touch task4.sh
[eavernikovskaya@eavernikovskaya ~]$ chmod +x task4.sh
[eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.15: Создание файла task4.sh и добавление прав на исполнение

Открываю файл task4.sh в текстовом редакторе gedit и пишу 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. 3.16), (рис. 3.17)

```
[eavernikovskaya@eavernikovskaya ~]$ gedit task4.sh
```

Рис. 3.16: Открытие файла task4.sh

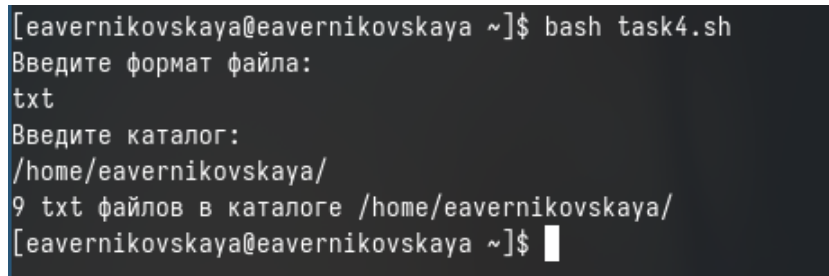
```
task4.sh (~) - gedit
Открыть task4.sh Сохранить
1 #!/bin/bash
2 echo "Введите формат файла: "
3 read extention
4 echo "Введите каталог: "
5 read directory
6 count=$(find "$directory" -name ".*$extention" -type f | wc -l)
7 echo "$count $extention файлов в каталоге $directory"
```

Рис. 3.17: Написанный скрипт для task4.sh

Программа для задания №4:

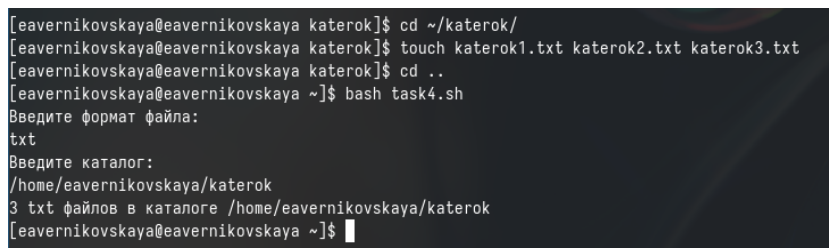
```
#!/bin/bash
echo "Введите формат файла: "
read extention
echo "Введите каталог: "
read directory
count=$(find "$directory" -name ".*$extention" -type f | wc -l)
echo "$count $extention файлов в каталоге $directory"
```

Далее запускаю файл с помощью bash и проверяю его работу (рис. 3.18), (рис. 3.19)



```
[eavernikovskaya@eavernikovskaya ~]$ bash task4.sh
Введите формат файла:
txt
Введите каталог:
/home/eavernikovskaya/
9 txt файлов в каталоге /home/eavernikovskaya/
[eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.18: Проверка работы скрипта task4.sh (1)



```
[eavernikovskaya@eavernikovskaya katerok]$ cd ~/katerok/
[eavernikovskaya@eavernikovskaya katerok]$ touch katerok1.txt katerok2.txt katerok3.txt
[eavernikovskaya@eavernikovskaya katerok]$ cd ..
[eavernikovskaya@eavernikovskaya ~]$ bash task4.sh
Введите формат файла:
txt
Введите каталог:
/home/eavernikovskaya/katerok
3 txt файлов в каталоге /home/eavernikovskaya/katerok
[eavernikovskaya@eavernikovskaya ~]$
```

Рис. 3.19: Проверка работы скрипта task4.sh (2)

4 Ответы на контрольные вопросы

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

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. UNIX обладает большим количеством оболочек. Наиболее популярными являются следующие четыре оболочки: - 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. Что такое POSIX?

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. 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 `{имя переменной}` например, использование команд `b=/tmp/andy-ls -l myfile > blsls/tmp/andy - ls, 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 мас-

сив, например, `states[49]=Alaska`. Индексация массивов начинается с нулевого элемента.

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

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.

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

- Сложение (+)
- Вычитание (-)
- Умножение (*)
- Деление (/)
- Модуль (остаток от деления) (%)

Примеры:

- `echo $((1 + 2))` # Выведет 3
- `echo $((5 - 3))` # Выведет 2
- `echo $((4 * 5))` # Выведет 20
- `echo $((10 / 2))` # Выведет 5
- `echo $((13 % 5))` # Выведет 3

6. Что означает операция (())?

Условия оболочки bash.

7. Какие стандартные имена переменных Вам известны?

Имя переменной (идентификатор) — это строка символов, которая отличает эту переменную от других объектов программы (идентифицирует переменную в программе). При задании имен переменным нужно соблюдать следующие правила: 1) первым символом имени должна быть буква. Остальные символы — буквы и цифры (прописные и строчные буквы различаются). Можно использовать символ «_»; 2) в имени нельзя использовать символ «.»; \$ число символов в имени не должно превышать 255; \$ имя переменной не должно совпадать с зарезервированными (служебными) словами языка. Var1, PATH, trash, mon, day, PS1, PS2 Другие стандартные переменные: - 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. Это символы 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 еще 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, ее нужно заключить в одинарные кавычки. Строка, заключенная в двойные кавычки, экранирует все метасимволы, кроме \$, ' , , ". Например, `-echo` выведет на экран символ, `-echo ab'|'cd` выдаст 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. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществит ее интерпретацию.

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, 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.

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

ls -lrt Если есть d, то является файл каталогом

13. Каково назначение команд set, typeset и unset?

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. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set. Наиболее распространенным является сокращение, избавляющееся от слова let в программах оболочек. Если объявить переменные целыми значениями, любое присвоение автоматически трактуется как арифметическое. Используйте typeset -i для объявления и присвоения переменной, и при последующем использовании она становится целой. Или можете использовать ключевое слово integer (псевдоним для typeset -l) и объявлять переменные целыми. Таким образом, выражения типа $x = y + z$ воспринимаются как арифметические. 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. 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 ее. Изъять переменную из программы можно с помощью команды unset.

14. Как передаются параметры в командные файлы?

Символ \$ является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов \$i, где $0 < i < 10$, вместо нее будет осуществлена подстановка значения параметра с порядковым номером i, т.е. аргумента командного файла с порядковым номером i. Использование комбинации символов \$0 приводит к подстановке вместо нее имени данного командного файла. 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 стандарт-

ного ввода и вывод всех строк, содержащих последовательность символов `andy`, на стандартный вывод. В ходе интерпретации этого файла командным процессором вместо комбинации символов `$1` осуществляется подстановка значения первого и единственного параметра `andy`. Если предположить, что пользователь, зарегистрированный в ОС UNIX под именем `andy`, в данный момент работает в ОС UNIX, то на терминале Вы увидите примерно следующее: `$ where andy andy ttyG Jan 14 09:12 $` Определим функцию, которая изменяет каталог и печатает список файлов: `$ function clist { > cd $1 > ls > }`. Теперь при вызове команды `clist` каталог будет изменен каталог и выведено его содержимое.

15. 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`;
- `${name[n]}` — обращение к `n`-ному элементу массива;
- `${name[*]}` — перечисляет все элементы массива, разделенные 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`;

- `${name?value}` — останавливает выполнение, если имя переменной не определено, и выводит `value`, как сообщение об ошибке; это выражение работает противоположно `{name-value}`. Если переменная определена, то подставляется `value`;
- `${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.

5 Выводы

В ходе выполнения лабораторной работы мы изучили основы программирования в оболочке ОС UNIX/Linux, а также научились писать небольшие командные файлы.

6 Список литературы

Не пользовалась сайтами.