

Лабораторная работа № 10

**Программирование в командном процессоре ОС UNIX. Командные
файлы**

Сильвен Макс Грегор Филс , НКАбд-03-22

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
5	Выводы	13
6	Контрольные вопросы	14
	Список литературы	24

Список иллюстраций

4.1	Текст первой программы	8
4.2	Проверка работы программы	8
4.3	Создание файла для второй программы, проверка содержимого домашнего каталога	9
4.4	Текст второй программы	9
4.5	Проверка работы второй программы	10
4.6	Создание файла для третьей программы	10
4.7	Текст третьей программы	11
4.8	Проверка работы третьей программы	11
4.9	Текст четвертой программы	12
4.10	Проверка работы четвертой программы	12

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux. 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. Путь к директории также передаётся в виде аргумента командной строки.

3 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;

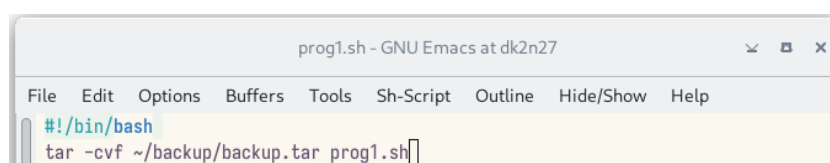
- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH — сокращение от 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 Korn. Рассмотрим основные элементы программирования в оболочке `bash`. В других оболочках большинство команд будет совпадать с описанными ниже. **[[Prog:b?]]**[ash]

4 Выполнение лабораторной работы

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. рис. ([4.1])



```
prog1.sh - GNU Emacs at dk2n27
File Edit Options Buffers Tools Sh-Script Outline Hide/Show Help
#!/bin/bash
tar -cvf ~/backup/backup.tar prog1.sh
```

Рис. 4.1: Текст первой программы

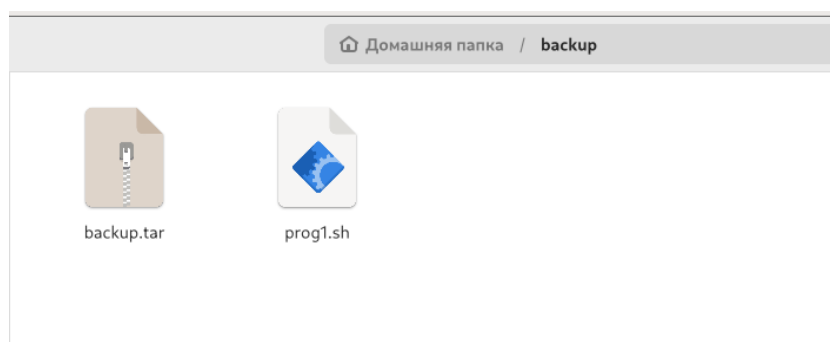


Рис. 4.2: Проверка работы программы

A terminal window showing a series of commands and their outputs. The user is at the prompt 'fsmaksgregor@dk2n27:~'. They run 'ls' and see a list of files and directories. Then they run 'bash prog1.sh' and 'touch prog2.sh'. Finally, they run 'ls' again, showing the updated directory listing.

```
fsmaksgregor@dk2n27:~ $ ls
'2023-04-15 13-51-34.mkv' lab07.sh      prog1.sh      test-2.asm    Музыка
abc                     lab10-2.asm   prog1.sh~     text.txt      Общедоступные
abc1                    lab5          prog1.tar.sh  tmp           'Рабочий стол'
abs                     lab6-1        programming.odt work          Шаблоны
backup                  lab6-1.asm   public        Видео
bin                     may           public_html   Документы
dir1                    monthly      reports       Загрузки
GNUstep                 parentdir    test-2        Изображения

fsmaksgregor@dk2n27:~ $ bash prog1.sh
prog1.sh
fsmaksgregor@dk2n27:~ $ touch prog2.sh
fsmaksgregor@dk2n27:~ $ ls
'2023-04-15 13-51-34.mkv' lab07.sh      prog1.sh      test-2        Изображения
abc                     lab10-2.asm   prog1.sh~     test-2.asm    Музыка
abc1                    lab5          prog1.tar.sh  text.txt      Общедоступные
abs                     lab6-1        prog2.sh      tmp           'Рабочий стол'
backup                  lab6-1.asm   programming.odt work          Шаблоны
bin                     may           public        Видео
dir1                    monthly      public_html   Документы
GNUstep                 parentdir    reports       Загрузки

fsmaksgregor@dk2n27:~ $
```

Рис. 4.3: Создание файла для второй программы, проверка содержимого домашнего каталога

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. ([4.4])

A screenshot of a text editor window titled 'prog2.sh'. The editor contains a shell script that reads a line of input and prints each word on a new line.

```
Открыть  +  prog2.sh  Сохранить  ^  v  x
1 #!/bin/bash
2 # echo 'Введите любые числа'
3 # read n
4 for A in $*
5 do echo $A
6 done
```

Рис. 4.4: Текст второй программы

```

Терминал - fsmaksgregor@dk2n27:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
fsmaksgregor@dk2n27 ~ $ touch prog1.sh
fsmaksgregor@dk2n27 ~ $ bash prog2.sh
fsmaksgregor@dk2n27 ~ $ bash prog2.sh 1 2 3
1
2
3
fsmaksgregor@dk2n27 ~ $ bash prog2.sh 1 2 3 4 5 6 3 2 3 4 5 7 1
1
2
3
4
5
6
3
2
3
4
5
7
1
fsmaksgregor@dk2n27 ~ $ touch prog3.sh
fsmaksgregor@dk2n27 ~ $ ls
'2023-04-15 13-51-34.mkv'  lab07.sh      prog1.sh      reports      Загрузки
abc                      lab10-2.asm   prog1.sh~     test-2       Изображения
abc1                     lab5          prog1.tar.sh  test-2.asm   Музыка
abs                      lab6-1        prog2.sh      text.txt     Общедоступные
backup                  lab6-1.asm    prog3.sh      tmp          'Рабочий стол'
bin                     may           programmation.odt  work        Шаблоны
dir1                   monthly      public        public_html  Видео
GNUstep                parentdir     public_html    Документы
fsmaksgregor@dk2n27 ~ $

```

Рис. 4.5: Проверка работы второй программы

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. ([4.6])

```

fsmaksgregor@dk2n27 ~ $ touch prog3.sh
fsmaksgregor@dk2n27 ~ $ ls
'2023-04-15 13-51-34.mkv'  lab07.sh      prog1.sh      reports      Загрузки
abc                      lab10-2.asm   prog1.sh~     test-2       Изображения
abc1                     lab5          prog1.tar.sh  test-2.asm   Музыка
abs                      lab6-1        prog2.sh      text.txt     Общедоступные
backup                  lab6-1.asm    prog3.sh      tmp          'Рабочий стол'
bin                     may           programmation.odt  work        Шаблоны
dir1                   monthly      public        public_html  Видео
GNUstep                parentdir     public_html    Документы
fsmaksgregor@dk2n27 ~ $

```

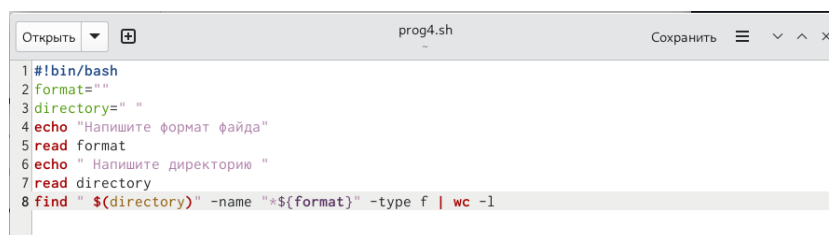
Рис. 4.6: Создание файла для третьей программы

```

dir1: is a directory
GNUstep: is a directory
lab07.sh: is a file andwriteable
readable
lab10-2.asm: is a file andwriteable
readable
lab5: is a directory
lab6-1: is a file andwriteable
readable
lab6-1.asm: is a file andwriteable
readable
may: is a file andwriteable
readable
monthly: is a directory
parentdir: is a directory
prog1.sh: is a file andwriteable
readable
prog1.sh~: is a file andwriteable
readable
prog1.tar.sh: is a file andwriteable
readable
prog2.sh: is a file andwriteable
readable
prog3.sh: is a file andwriteable
readable
programmation.odt: is a file andwriteable
readable
public: is a directory
public_html: is a directory
reports: is a directory
test-2: is a file andwriteable
readable

```

Рис. 4.7: Текст третьей программы



```

Открыть  +  prog4.sh
Сохранить  ^  x
1 #!/bin/bash
2 format=""
3 directory=""
4 echo "Напишите формат файла"
5 read format
6 echo " Напишите директорию "
7 read directory
8 find "$directory" -name "${format}" -type f | wc -l

```

Рис. 4.8: Проверка работы третьей программы

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. ([4.9])

```
Терминал - fsmaksgregor@dk2n27:~
Файл  Правка  Вид  Терминал  Вкладки  Справка
fsmaksgregor@dk2n27:~
fsmaksgregor@dk2n27 ~ $ touch prog4.sh
fsmaksgregor@dk2n27 ~ $ ls
'2023-04-15 13-51-34.mkv'  lab07.sh      prog1.sh      public_html   Документы
abc                      lab10-2.asm   prog1.sh~     reports       Загрузки
abc1                     lab5          prog1.tar.sh  test-2        Изображения
abs                      lab6-1        prog2.sh      test-2.asm    Музыка
backup                  lab6-1.asm    prog3.sh      text.txt      Общедоступные
bin                     may           prog4.sh      tmp           'Рабочий стол'
dir1                    monthly       programming.odt work          Шаблоны
GNUstep                parentdir     public         Видео
fsmaksgregor@dk2n27 ~ $ bash prog4.sh
Напишите формат файла
txt
Напишите директорию
/work
prog4.sh: строка 8: directory: команда не найдена
find: ' ': Нет такого файла или каталога
0
fsmaksgregor@dk2n27 ~ $ bash prog4.sh
Напишите формат файла
txt
Напишите директорию
/fsmaksgregor
prog4.sh: строка 8: directory: команда не найдена
find: ' ': Нет такого файла или каталога
0
fsmaksgregor@dk2n27 ~ $
```

Рис. 4.9: Текст четвертой программы

```
prog1.sh - GNU Emacs at dk2n27
File Edit Options Buffers Tools Sh-Script Outline Hide/Show Help
#!/bin/bash
tar -cvf prog1.tar.sh /path/to/backup
```

Рис. 4.10: Проверка работы четвертой программы

5 Выводы

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

6 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; –С-оболочка - добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя С-подобный синтаксис команд, и сохраняет историю выполненных команд; –оболочка Корна - напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна; –BASH - сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании Free Software Foundation).

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 радиотехники (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 `mv afile $mark` переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того, чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: `${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 массив, 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`?

Какие арифметические операции можно применять в языке программирования `bash`? Оператор Синтаксис Результат `!` `!expr` Если `expr` равно 0, возвращает 1; иначе 0 `!= expr1 !=expr2` Если `expr1` не равно `expr2`, возвращает 1; иначе 0 `% expr1%expr2` Возвращает остаток от деления `expr1` на `expr2` `%= var=%expr` Присваивает остаток от деления `var` на `expr` переменной `var` `& expr1&expr2` Возвращает побитовое AND выражений `expr1` и `expr2` `&& expr1&&expr2` Если и `expr1` и `expr2` не равны нулю, возвращает 1; иначе 0 `&= var &= expr` Присваивает `var` побитовое AND переменных `var` и выражения `expr` `* expr1 * expr2` Умножает `expr1` на `expr2` `= var = expr` Умножает `expr` на значение `var` и присваивает результат переменной `var` `+ expr1 + expr2` Складывает `expr1` и `expr2` `+= var += expr` Складывает `expr` со значением `var` и результат присваивает `var` `- -expr` Операция отрицания `expr` (называется унарный минус) `- expr1 - expr2` Вычитает `expr2` из `expr1` `-- var -- expr` Вычитает `expr` из значения `var` и

присваивает результат $var / expr / expr2$ Делит $expr1$ на $expr2$ $var /= expr$ Делит var на $expr$ и присваивает результат $var < expr1 < expr2$

Если $expr1$ меньше, чем $expr2$, возвращает 1, иначе возвращает 0 « $expr1$ « $expr2$ Сдвигает $expr1$ влево на $expr2$ бит «= var «= $expr$ Побитовый сдвиг влево значения var на $expr$ $<= expr1 <= expr2$ Если $expr1$ меньше, или равно $expr2$, возвращает 1; иначе возвращает 0 $= var = expr$ Присваивает значение $expr$ переменной var $== expr1 == expr2$ Если $expr1$ равно $expr2$. Возвращает 1; иначе возвращает 0 $> expr1 > expr2$ 1 если $expr1$ больше, чем $expr2$; иначе 0 $>= expr1 >= expr2$ 1 если $expr1$ больше, или равно $expr2$; иначе 0 » $expr$ » $expr2$ Сдвигает $expr1$ вправо на $expr2$ бит »= var »= $expr$ Побитовый сдвиг вправо значения var на $expr$ $^ expr1 ^ expr2$ Исключающее OR

выражений $expr1$ и $expr2$ $^= var ^= expr$ Присваивает var побитовое исключающее OR var и $expr$ $| expr1 | expr2$ Побитовое OR выражений $expr1$ и $expr2$ $|= var |= expr$ Присваивает var «исключающее OR» переменной var и выражения $expr$ $|| expr1 || expr2$ 1 если или $expr1$ или $expr2$ являются ненулевыми значениями; иначе 0 $\sim expr$ Побитовое дополнение до $expr$.

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: § первым символом имени должна быть буква. Остальные символы — буквы и цифры (прописные и строчные буквы различаются). Можно использовать символ «_»; § 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,

указан- ный 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 -i`) 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 инициализирует 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

$0 < \text{№} < 10$, 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 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 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;
- `${name+value}` — это выражение работает противоположно `${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.

Список литературы