

Лабораторная работа №13

Дисциплина: Операционные системы

Колчева Юлия Вячеславовна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	13
4	Контрольные вопросы	14
5	Библиография	19

List of Tables

List of Figures

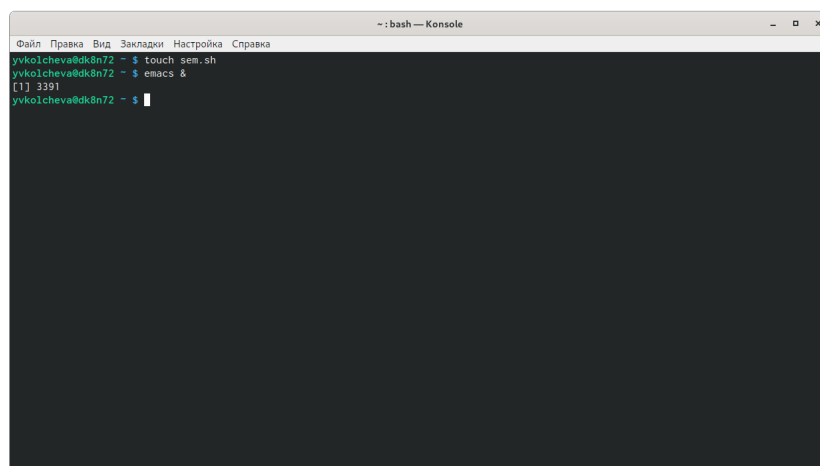
2.1	Создание и запуск	6
2.2	Скрипт1	7
2.3	Проверка первого скрипта	7
2.4	Первая часть скрипта	8
2.5	Вторая часть скрипта	9
2.6	Выполнение	9
2.7	Скрипт3	11
2.8	Проверка работы	11
2.9	Последний скрипт	12
2.10	Проверка работы4	12

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

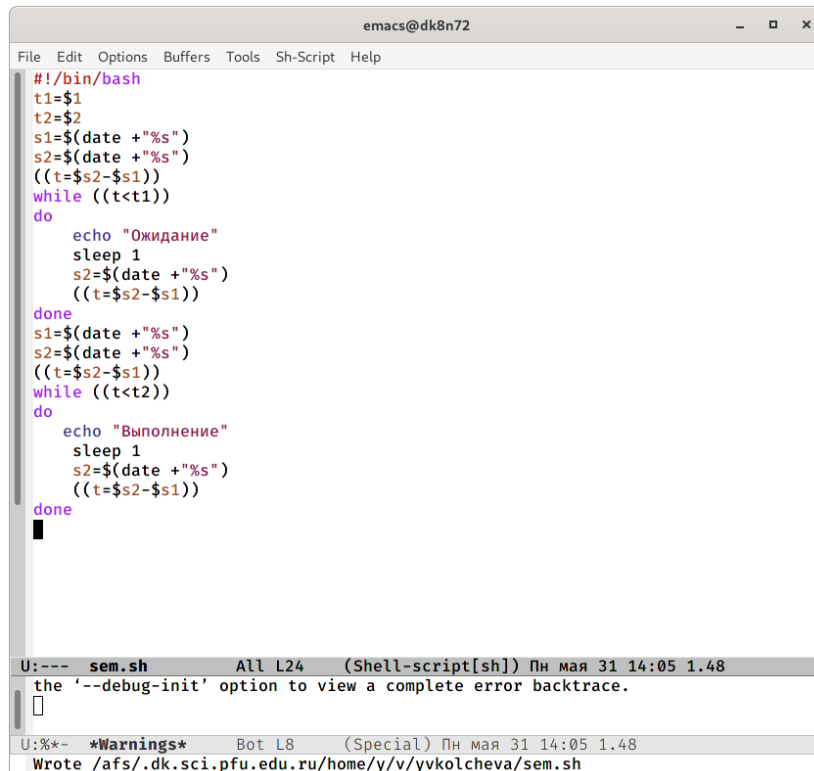
2 Выполнение лабораторной работы

Сначала я написала командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Для данной задачи я создала файл `sem.sh` и написала соответствующий скрипт. (рис. 2.2) (рис. 2.1)



```
~: bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
yvkoicheva@dkn72 ~$ touch sem.sh
yvkoicheva@dkn72 ~$ emacs &
[1] 3391
yvkoicheva@dkn72 ~$
```

Figure 2.1: Создание и запуск



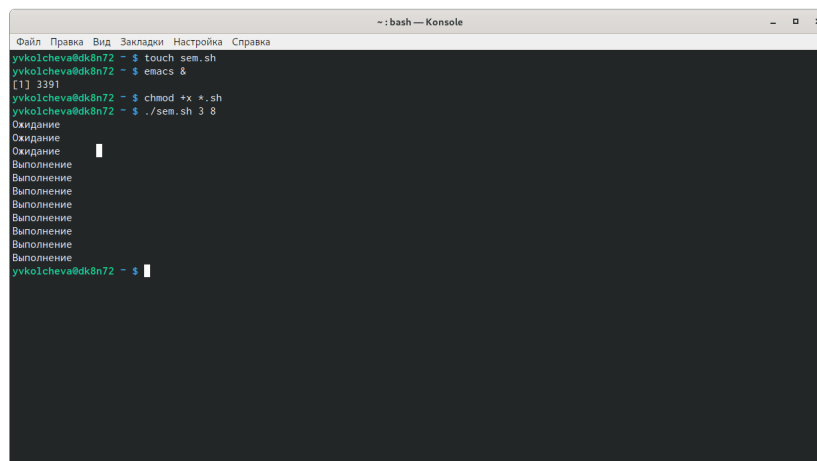
```
#!/bin/bash
t1=$1
t2=$2
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t<t1))
do
    echo "Ожидание"
    sleep 1
    s2=$(date +%s)
    ((t=$s2-$s1))
done
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t<t2))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s)
    ((t=$s2-$s1))
done
```

U:--- sem.sh All L24 (Shell-script[sh]) Пн мая 31 14:05 1.48
the '--debug-init' option to view a complete error backtrace.

U:%*- *Warnings* Bot L8 (Special) Пн мая 31 14:05 1.48
Wrote /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva/sem.sh

Figure 2.2: Скрипт 1

Далее я проверила работу написанного скрипта (./prog1.sh 2 6), добавив право на исполнение файла (chmod +x *.sh). Скрипт работает корректно. (рис. 2.3)



```
yu.kolcheva@dk8n72 ~$ touch sem.sh
yu.kolcheva@dk8n72 ~$ emacs &
[1] 3391
yu.kolcheva@dk8n72 ~$ chmod +x *.sh
yu.kolcheva@dk8n72 ~$ ./sem.sh 3 8
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
yu.kolcheva@dk8n72 ~$
```

Figure 2.3: Проверка первого скрипта

После этого я изменила скрипт так, чтобы его можно было выполнять в несколь-

ких терминалах и проверила его работу (./san.sh 2 5 Выполнение > /dev/tty2 &). Но ни одна команда не работала, так как мне “Отказано в доступе”. При этом скрипт работает корректно (команда «./prog1.sh 3 8»). (рис. 2.4) (рис. 2.5) (рис. 2.6)

```
#!/bin/bash
function a
{
t1=$1
t2=$2
s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))
while ((t<t1))
do
echo "Ожидание"
sleep 1
s2=$(date +%s)
((t=s2-s1))
done
}
function b
{
s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))
while ((t<t2))
do
echo "Выполнение"
sleep 1
s2=$(date +%s)
((t=s2-s1))
done
}
t1=$1
t2=$2
command=$3
while true
do
if [ "$command" == "Выход" ]
then
echo "Выход"
exit 0
fi
if [ "$command" == "Ожидание" ]

```

U:*** sem.sh Top L39 (Shell-script[sh]) Пн мая 31 14:14 1.58
the '--debug-init' option to view a complete error backtrace.
[]

U:*** *Warnings* Bot L8 (Special) Пн мая 31 14:14 1.58

Figure 2.4: Первая часть скрипта

The screenshot shows an Emacs editor window with the title bar "emacs@dk8n72". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". The main text area contains a shell script with the following content:

```

fi
if [ "$command" == "Ожидание" ]
then a
fi
if [ "$command" == "Выполнение" ]
then b
fi
echo "Действие: "
read command
done

```

Below the editor window is a terminal window. The prompt is "U:***- sem.sh Bot L39 (Shell-script[sh]) Пн мая 31 14:14 1.58". The terminal output shows the command "the '--debug-init' option to view a complete error backtrace." followed by a cursor.

At the bottom of the terminal window, there is a status bar with the text "U:***- *Warnings* Bot L8 (Special) Пн мая 31 14:14 1.58".

Figure 2.5: Вторая часть скрипта

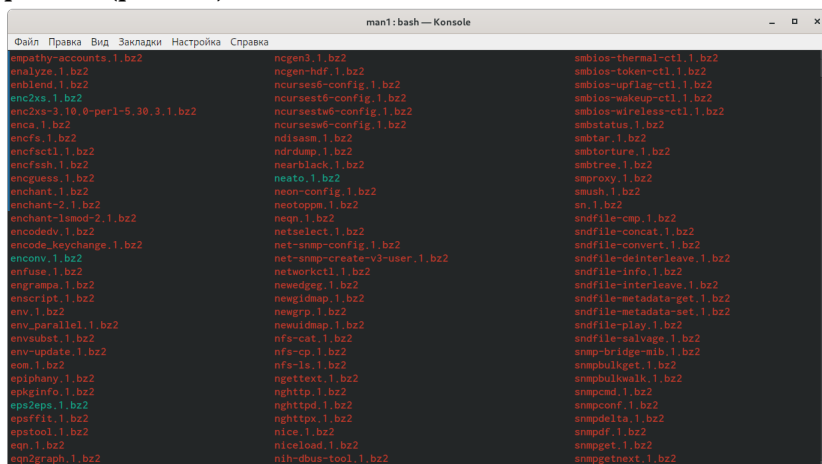
```

- :
Файл Правка Вид Закладки Настройка Справка
Выполнение
Выполнение
Выполнение
Выполнение
Действие:
Выход
Выход
yvkolcheva@dskn72 ~ $ ./sem.sh 3 8 Ожидание > /dev/tty2
bash: /dev/tty2: Отказано в доступе
yvkolcheva@dskn72 ~ $ ./sem.sh 3 8 Ожидание > /dev/tty1
bash: /dev/tty1: Отказано в доступе
yvkolcheva@dskn72 ~ $ ./sem.sh 3 8 Ожидание > /dev/tty1
bash: /dev/tty1: Отказано в доступе
yvkolcheva@dskn72 ~ $ ./sem.sh 3 8 Ожидание > /dev/tty2
bash: /dev/tty2: Отказано в доступе
yvkolcheva@dskn72 ~ $ ./sem.sh 3 8 Ожидание > /dev/tty2 &
[2] 5818
yvkolcheva@dskn72 ~ $ bash: /dev/tty2: Отказано в доступе
./sem.sh 3 8 Ожидание > /dev/tty2 &
[3] 6760
[2] Выход 1
./sem.sh 3 8 Ожидание > /dev/tty2
bash: /dev/tty2: Отказано в доступе
yvkolcheva@dskn72 ~ $ ./sem.sh 3 8 Ожидание > /dev/tty0 &
[4] 6388
[3] Выход 1
./sem.sh 3 8 Ожидание > /dev/tty2
yvkolcheva@dskn72 ~ $ bash: /dev/tty0: Отказано в доступе
./sem.sh 3 8 Ожидание > /dev/tty0 &
[5] 6492
[4] Выход 1
./sem.sh 3 8 Ожидание > /dev/tty0
yvkolcheva@dskn72 ~ $ bash: /dev/tty0: Отказано в доступе

```

Figure 2.6: Выполнение

Реализовала команду `man` с помощью командного файла. Изучила содержимое каталога `/usr/share/man/man1`. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой `less` сразу же просмотрев содержимое справки. (рис. ??)



Командный файл должен получать в виде аргумента командной строки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге `man1`. (рис. 2.7)

```
emacs@dk8n72
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
c=$1
if [ -f /usr/share/man/man1/$c.1.gz ]
then
gunzip -c /usr/share/man/man1/$1.1.gz | less
else
echo "Справки по этой команде нет"
fi
█

U:--- man.sh All L9 (Shell-script[sh]) Пн мая 31 14:54 1.49
Warning (initialization): An error occurred while loading '~/.emacs':

error: Package 'fira-code-mode-' is unavailable

To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
█

U:%*- *Warnings* All L8 (Special) Пн мая 31 14:54 1.49
Wrote /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva/man.sh
```

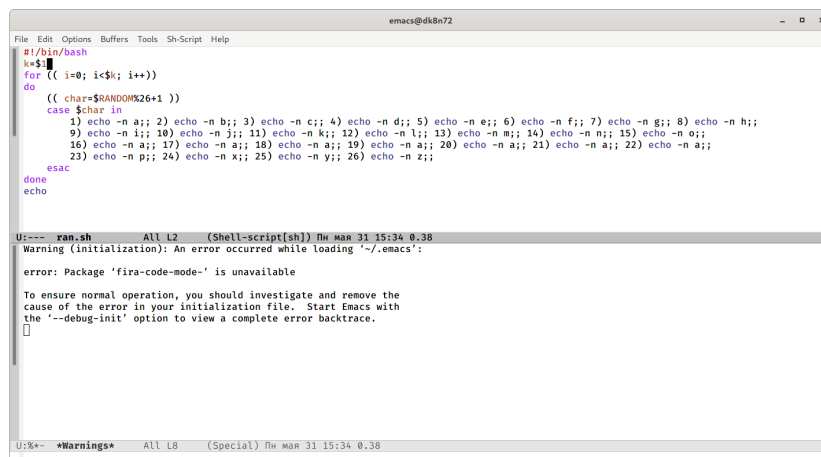
Figure 2.7: Скрипт3

Далее я проверила работу написанного скрипта (команды «./prog2.sh ls», «./prog2.sh mkdir»), предварительно добавив право на исполнение файла (команда «chmod +x *.sh»). Скрипт сработал и вывел, что по данным командам справок нет. (рис. 2.8)

```
~: bash — Konsole
Файл Правка Вид Закладки Настройка Справка
iso-hybrid.1.bz2 pmnagema.1.bz2 zshcompys.1.bz2
iso-info.1.bz2 pmahisteq.1.bz2 zshcompwid.1.bz2
iso-read.1.bz2 pmahistap.1.bz2 zshcontrib.1.bz2
isql.1.bz2 pmindex.1.bz2 zshexpn.1.bz2
isutf8.1.bz2 pminterp.1.bz2 zshmisc.1.bz2
isympy.1.bz2 pminvert.1.bz2 zshmodules.1.bz2
istool.1.bz2 pmmergin.1.bz2 zshoptions.1.bz2
itweb-settings.1.bz2 pmmercator.1.bz2 zshparam.1.bz2
iusql.1 pmnmontage.1.bz2 zshroadmap.1.bz2
iwctl.1.bz2 pmnifilt.1.bz2 zshrcpsys.1.bz2
jimon.1.bz2 pmnoraw.1.bz2 zshzftpsys.1.bz2
jackbufsize.1.bz2 pmnorm.1.bz2 zshzle.1.bz2
jack_connect.1.bz2 pmpad.1.bz2 zsoella.1.bz2
jackd.1.bz2 pmpaste.1.bz2 zstd.1.bz2
jack_disconnect.1 pmpasnr.1.bz2 zstdcat.1.bz2
jack_freewheel.1.bz2 pmquant.1.bz2 zstdgrep.1.bz2
jack_impulse_grabber.1.bz2 pmquantall.1.bz2 zstdless.1.bz2
jack_jockey.1.bz2 pmremap.1.bz2 zvbi-chains.1.bz2
jack_load.1.bz2 pmrotate.1.bz2 zvbi.1.bz2
jack_lsp.1.bz2 pmscale.1.bz2 zvbi-ntsc-cc.1.bz2
yvkolcheva@dk8n72 /usr/share/man/man1 $ cd ~
yvkolcheva@dk8n72 ~ $ touch man.sh
yvkolcheva@dk8n72 ~ $ emacs &
[2] 7542
[1] Завершён emacs
yvkolcheva@dk8n72 ~ $ chmod +x man.sh
yvkolcheva@dk8n72 ~ $ ./man.sh ls
Справки по этой команде нет
yvkolcheva@dk8n72 ~ $ ./man.sh mkdir
Справки по этой команде нет
yvkolcheva@dk8n72 ~ $
```

Figure 2.8: Проверка работы

- 3) Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита.



```
#!/bin/bash
x=$RANDOM
for (( i=0; i<$k; i++))
do
  (( char=$RANDOM%26+1 ))
  case $char in
    1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;; 5) echo -n e;; 6) echo -n f;; 7) echo -n g;; 8) echo -n h;;
    9) echo -n i;; 10) echo -n j;; 11) echo -n k;; 12) echo -n l;; 13) echo -n m;; 14) echo -n n;; 15) echo -n o;;
    16) echo -n a;; 17) echo -n a;; 18) echo -n a;; 19) echo -n a;; 20) echo -n a;; 21) echo -n a;; 22) echo -n a;;
    23) echo -n p;; 24) echo -n x;; 25) echo -n y;; 26) echo -n z;;
  esac
done
echo
```

U:-- ran.sh All L2 (Shell-script[sh]) Пн мая 31 15:34 0.38

Warning (initialization): An error occurred while loading '~/.emacs':

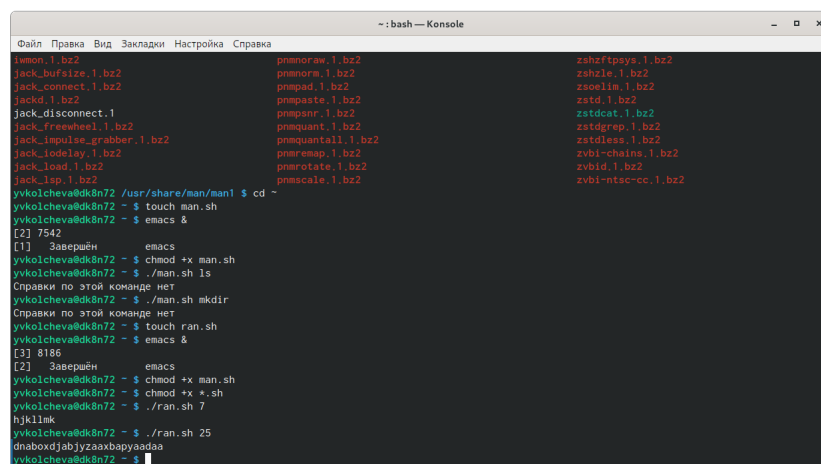
error: Package 'fira-code-mode-' is unavailable

To ensure normal operation, you should investigate and remove the cause of the error in your initialization file. Start Emacs with the '--debug-init' option to view a complete error backtrace.

U:-- *Warnings* All L8 (Special) Пн мая 31 15:34 0.38

Figure 2.9: Последний скрипт

Далее я проверила работу написанного скрипта (./ran.sh N где N - это номер), предварительно добавив право на исполнение файла (команда «chmod +x *.sh»). Скрипт работает корректно.(рис. 2.9)



```
vimon.1.bz2      pmanoraw.1.bz2      zshzfipays.1.bz2
jack_bufsize.1.bz2  pmanorm.1.bz2       zshzle.1.bz2
jack_connect.1.bz2  pmpad.1.bz2         zsoelim.1.bz2
jackd.1.bz2         pmpaste.1.bz2       zstd.1.bz2
jack_disconnect.1  pmpar.1.bz2         zstdcat.1.bz2
jack_freemem.1.bz2 pmpquant.1.bz2      zstdgrep.1.bz2
jack_impulse_grabber.1.bz2 pmpquantall.1.bz2  zstdless.1.bz2
jack_lodelay.1.bz2 pmsemap.1.bz2       zvbi-chains.1.bz2
jack_load.1.bz2    pmsrotate.1.bz2     zvbi.1.bz2
jack_lsp.1.bz2     pmscale.1.bz2       zvbi-ntsc-cc.1.bz2
yukoicheva@dk8n72 /usr/share/man/man1 $ cd -
yukoicheva@dk8n72 ~ $ touch man.sh
yukoicheva@dk8n72 ~ $ emacs &
[2] 7542
[1] Завершён emacs
yukoicheva@dk8n72 ~ $ chmod +x man.sh
yukoicheva@dk8n72 ~ $ ./man.sh ls
Справки по этой команде нет
yukoicheva@dk8n72 ~ $ ./man.sh mkdir
Справки по этой команде нет
yukoicheva@dk8n72 ~ $ touch ran.sh
yukoicheva@dk8n72 ~ $ emacs &
[3] 8186
[2] Завершён emacs
yukoicheva@dk8n72 ~ $ chmod +x man.sh
yukoicheva@dk8n72 ~ $ chmod +x *.sh
yukoicheva@dk8n72 ~ $ ./ran.sh 7
hjkllmk
yukoicheva@dk8n72 ~ $ ./ran.sh 25
dnaboxdjabjyzaaxbapyaadaa
yukoicheva@dk8n72 ~ $
```

U:-- *Warnings* All L8 (Special) Пн мая 31 15:34 0.38

Figure 2.10: Проверка работы4

3 Выводы

Я изучила основы программирования в оболочке ОС UNIX и научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

4 Контрольные вопросы

1) 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 оболочке 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) 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) 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 `.`, «*`mvafile{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` флагом `-A`. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, «`set-AstatesDelawareMichigan"NewJersey"`» Далее можно сделать добавление в массив, например, `states[49]=Alaska`. Индексация массивов начинается с нулевого элемента. 4) 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 month day`» В переменные `month` и `day` будут считаны соответствующие значения, введённые с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введённую информацию и игнорировать её. 5) В языке программирования `bash` можно применять такие арифметические операции как сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%). 6) В (()) можно записывать условия оболочки `bash`, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат. 7) 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 последо-

тельность 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, то 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 (*newline*). *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) Такие символы, как ' < > ? | " &, являются метасимволами и имеют для командного процессора специальный смысл. 9) 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 , *-echoab'|'cd* выведет на экран строку *ab|cd*.

10) 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+хмия_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.

11) 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*.

12) 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).

13) 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.

14) 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. 15) 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*

5 Библиография

Лекция Кудрявцева <https://esystem.rudn.ru/mod/url/view.php?id=718562>

Лабораторная работа №13 <https://esystem.rudn.ru/mod/resource/view.php?id=718610>