

Лабораторная работа 14

**Программирование в командном процессоре ОС UNIX. Расширенное
программирование**

Неустроева Ирина Николаевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Вывод	12
5	Ответы на контрольные вопросы	13

Список иллюстраций

3.1	Скрипт	8
3.2	Исполнение	9
3.3	Архивы текстовых файлов	9
3.4	Исполнение	10
3.5	Проверка работы	10
3.6	Проверка работы	10
3.7	Скрипт	11
3.8	Проверка работы	11

Список таблиц

1 Цель работы

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

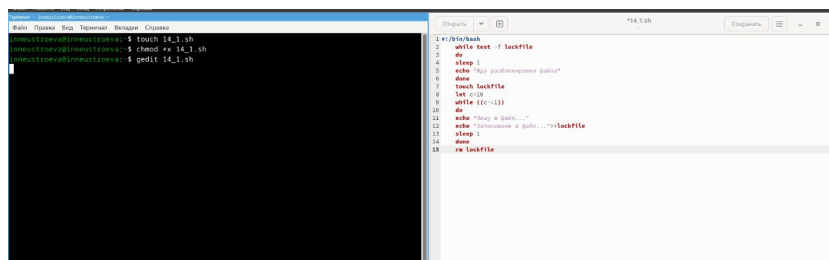
2 Задание

1. Написать 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). Запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (`> /dev/tty#`, где `#` — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.
2. Реализовать команду `man` с помощью командного файла. Изучите 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. 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`.
3. Используя встроенную переменную `$RANDOM`, напишите командный файл, генерирующий случайную последовательность букв латинско-

го алфавита. Учтите, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

3 Выполнение лабораторной работы

1. Написали 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). Запустили командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой ($> /dev/tty\#$, где $\#$ — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработали программу так, чтобы имелась возможность взаимодействия трёх и более процессов. (рис. 3.1).



```
1 #!/bin/bash
2 while test -f lockfile
3 do
4     sleep 1
5     echo "for processname $pid?"
6 done
7 touch lockfile
8 let i=0
9 while [i<12]
10 do
11     echo "loop is going..."
12     echo "processname is $pid...">/dev/tty$1
13     sleep 1
14 done
15 rm lockfile
```

Рис. 3.1: Скрипт

Вызовем файл на исполнение (рис. 3.2).


```
inneustroeva@inneustroeva:~$ bash 14_1.sh
Пишу в файл...
Пишу в файл...
Пишу в файл...
Пишу в файл...
Пишу в файл...
Пишу в файл...
Пишу в файл...
```

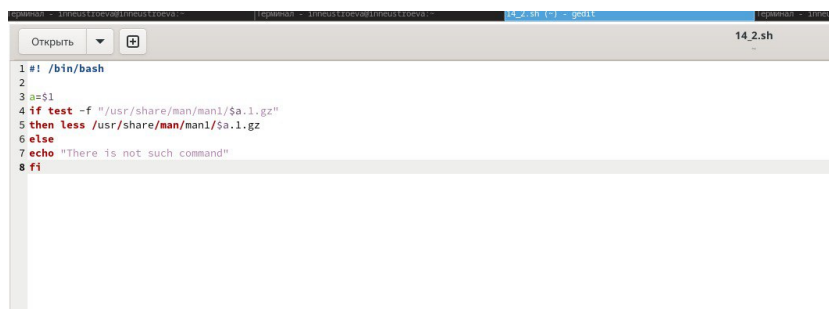
Рис. 3.2: Исполнение

2. Реализовали команду `man` с помощью командного файла. Изучили 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. 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. 3.3).

```
inneustroeva@inneustroeva:~$ ls /usr/share/man/man1
.:1.gz
l1.1.gz
a2ping.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz
abrt-action-analyze-vmcore.1.gz
abrt-action-analyze-vulnerability.1.gz
abrt-action-analyze-xorg.1.gz
abrt-action-check-oops-for-hw-error.1.gz
abrt-action-find-bodhi-update.1.gz
abrt-action-generate-backtrace.1.gz
abrt-action-generate-core-backtrace.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-save-package-data.1.gz
abrt-action-trim-files.1.gz
abrt-applet.1.gz
abrt-auto-reporting.1.gz
abrt-bodhi.1.gz
abrt-cli.1.gz
abrt-dump-journal-core.1.gz
abrt-dump-journal-oops.1.gz
abrt-dump-journal-xorg.1.gz
abrt-dump-oops.1.gz
abrt-dump-xorg.1.gz
abrt-handle-upload.1.gz
abrt-harvest-pstoreoops.1.gz
gnome-keyring-daemon.1.gz
gnroff.1.gz
gold-tool.1.gz
gpasswd.1.gz
gpg.1.gz
gpg2.1.gz
gpg-agent.1.gz
gpg-card.1.gz
gpg-check-pattern.1.gz
gpgconf.1.gz
gpg-connect-agent.1.gz
gpgparsemail.1.gz
gpg-preset-passphrase.1.gz
gpgsm.1.gz
gpgtar.1.gz
gpgv.1.gz
gpgv2.1.gz
gpg-wks-client.1.gz
gpg-wks-server.1.gz
gpics.1.gz
gprof.1.gz
graphviz.1.gz
grep.1.gz
grim.1.gz
groff.1.gz
gropts.1.gz
groty.1.gz
groups.1.gz
grub2-editenv.1.gz
grub2-emu.1.gz
grub2-file.1.gz
ostree-export.1.gz
ostree-find-remotes.1.gz
ostree-fck.1.gz
ostree-gpg-sign.1.gz
ostree-init.1.gz
ostree-log.1.gz
ostree-ls.1.gz
ostree-prepare-root.1.gz
ostree-prune.1.gz
ostree-pull.1.gz
ostree-pull-local.1.gz
ostree-refs.1.gz
ostree-remote.1.gz
ostree-remote.1.gz
ostree-rev-parse.1.gz
ostree-show.1.gz
ostree-sign.1.gz
ostree-static-delta.1.gz
ostree-summary.1.gz
ot2kpx.1.gz
otangle.1.gz
otinfo.1.gz
otftotfm.1.gz
otp2ocp.1.gz
outocp.1.gz
ovf2ovp.1.gz
ovp2ovf.1.gz
pacat.1.gz
pacmd.1.gz
pactl.1.gz
padsp.1.gz
```

Рис. 3.3: Архивы текстовых файлов

Напишем скрипт (рис. 3.4).



```
1 #! /bin/bash
2
3 a=$1
4 if test -f "/usr/share/man/man1/$a.1.gz"
5 then less /usr/share/man/man1/$a.1.gz
6 else
7 echo "There is not such command"
8 fi
```

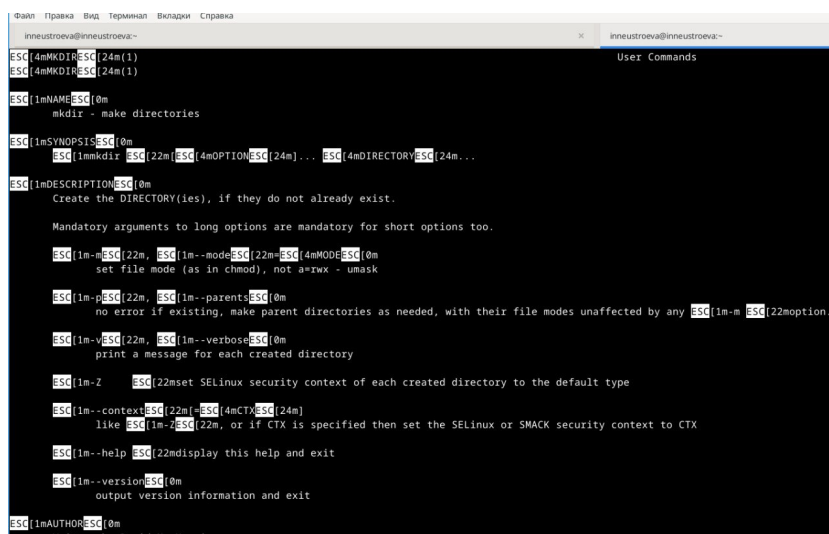
Рис. 3.4: Исполнение

Проверим работу данного файла (рис. 3.5). (рис. 3.6).



```
inneustroeva@inneustroeva:~$ touch 14_2.sh
inneustroeva@inneustroeva:~$ chmod +x 14_2.sh
inneustroeva@inneustroeva:~$ gedit 14_2.sh
inneustroeva@inneustroeva:~$ ./14_2.sh mkdir
```

Рис. 3.5: Проверка работы



```
ESC[4mMKDIRESC[24m(1)
ESC[4mMKDIRESC[24m(1)

ESC[1mNAMEESC[0m
mkdir - make directories

ESC[1mSYNOPSISESC[0m
ESC[1mmkdir ESC[22m[ESC[4mOPTIONESC[24m]... ESC[4mDIRECTORIESESC[24m...

ESC[1mDESCRIPTIONESC[0m
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.

ESC[1m-mESC[22m, ESC[1m--modeESC[22m=ESC[4mMODEESC[0m
set file mode (as in chmod), not a=rwx - umask

ESC[1m-pESC[22m, ESC[1m--parentsESC[0m
no error if existing, make parent directories as needed, with their file modes unaffected by any ESC[1m-m ESC[22moption.

ESC[1m-vESC[22m, ESC[1m--verboseESC[0m
print a message for each created directory

ESC[1m-Z ESC[22mset SELinux security context of each created directory to the default type

ESC[1m--contextESC[22m[=ESC[4mCTXESC[24m]
like ESC[1m-mESC[22m, or if CTX is specified then set the SELinux or SMACK security context to CTX

ESC[1m--help ESC[22mdisplay this help and exit

ESC[1m--versionESC[0m
output version information and exit

ESC[1mAUTHORESC[0m
Written by David MacKenzie
```

Рис. 3.6: Проверка работы

3. Используя встроенную переменную \$RANDOM, написали командный файл, генерирующий случайную последовательность букв латинского алфавита. (рис. 3.7).

```

1 #!/bin/bash
2 declare -a ABC
3 ABC=({a..z})
4 let limit=25
5 let i=10
6 while ((i-=1))
7 do
8 numb=$RANDOM
9 let numb%=limit
10 output=$output${ABC[$numb]}
11 done
12 echo $output

```

Рис. 3.7: Скрипт

Проверим работу данного файла (рис. 3.8).

```

inneustroeva@inneustroeva:~$ touch 14_3.sh
inneustroeva@inneustroeva:~$ chmod +x 14_3.sh
inneustroeva@inneustroeva:~$ gedit 14_3.sh
inneustroeva@inneustroeva:~$ ./14_3.sh
krydccucp
inneustroeva@inneustroeva:~$ ./14_3.sh
fxqheboyh
inneustroeva@inneustroeva:~$ ./14_3.sh

```

Рис. 3.8: Проверка работы

4 Вывод

В данной работе мы изучили основы программирования в оболочке ОС UNIX/Linux. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Ответы на контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `1 while [$1 != "exit"]`

В данной строчке допущены следующие ошибки: не хватает пробелов после первой скобки [и перед второй скобкой] выражение \$1 необходимо взять в "", потому что эта переменная может содержать пробелы. Таким образом, правильный вариант должен выглядеть так:

```
while [ "$1" != "exit" ]
```

2. Как объединить (конкатенацией) несколько строк в одну?

Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: Первый:

```
VAR1="Hello," VAR2=" World" VAR3="$VAR1$VAR2" echo "$VAR3"
```

Результат:

Hello, World.

Второй:

```
VAR1="Hello," VAR1+=" World" echo "$VAR1"
```

Результат:

Hello, World.

3. Найдите информацию об утилите seq. Какими иными способами можно реализовать её функционал при программировании на bash?

Команда seq в Linux используется для генерации чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT. Параметры: seq LAST: если задан только один аргумент, он создает числа от 1 до LAST с шагом шага, равным 1. Если LAST меньше

1, значение `is` не выдает. `seq FIRST LAST`: когда заданы два аргумента, он генерирует числа от `FIRST` до `LAST` с шагом 1, равным 1. Если `LAST` меньше `FIRST`, он не выдает никаких выходных данных. `seq FIRST INCREMENT LAST`: когда заданы три аргумента, он генерирует числа от `FIRST` до `LAST` на шаге `INCREMENT`. Если `LAST` меньше, чем `FIRST`, он не производит вывод. `seq -f «FORMAT» FIRST INCREMENT LAST`: эта команда используется для генерации последовательности в форматированном виде. `FIRST` и `INCREMENT` являются необязательными. `seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО`: Эта команда используется для `STRING` для разделения чисел. По умолчанию это значение равно `/n`. `FIRST` и `INCREMENT` являются необязательными. `seq -w FIRST INCREMENT LAST`: эта команда используется для выравнивания ширины путем заполнения начальными нулями. `FIRST` и `INCREMENT` являются необязательными.

4. Какой результат даст вычисление выражения $\$((10/3))$?

Результатом данного выражения $\$((10/3))$ будет 3, потому что это целочисленное деление без остатка.

5. Укажите кратко основные отличия командной оболочки `zsh` от `bash`.

Отличия командной оболочки `zsh` от `bash`: В `zsh` более быстрое автодополнение для `cd` с помощью `Tab`. В `zsh` существует калькулятор `zcalc`, способный выполнять вычисления внутри терминала. В `zsh` поддерживаются числа с плавающей запятой. В `zsh` поддерживаются структуры данных «хэш». В `zsh` поддерживается раскрытие полного пути на основе неполных данных. В `zsh` поддерживается замена части пути. В `zsh` есть возможность отображать разделенный экран, такой же как разделенный экран `vim`.

6. Проверьте, верен ли синтаксис данной конструкции:

```
for ((a=1; a <= LIMIT; a++))
```

Синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать `$` перед переменными `()`.

7. Сравните язык bash с какими-либо языками программирования. Какие преимущества у bash по сравнению с ними? Какие недостатки?

Преимущества и недостатки скриптового языка bash:

Один из самых распространенных и ставится по умолчанию в большинстве дистрибутивах Linux, MacOS; Удобное перенаправление ввода/вывода; Большое количество команд для работы с файловыми системами Linux; Можно писать собственные скрипты, упрощающие работу в Linux недостатки скриптового языка bash; Дополнительные библиотеки других языков позволяют выполнить больше действий;

Bash не является языком общего назначения; Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта; Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий. :::
{#refs} :::