

Лабораторная работа № 2

**Командный интерпретатор и основы
программирования на shell**

Основы регулярных выражений

Рассматриваемые темы

- Порядок выполнения команд
- Перенаправление ввода-вывода
- Основы регулярных выражений
- Создание скриптов
- Программирование на shell
- Автоматизация запуска задач

Общий вид POSIX-программы

C:

```
int main(int argc, char* argv[]) {  
    ...  
    get();  
    put();  
  
    return 0;  
}
```

Pascal:

```
program main;  
begin  
    Writeln(...);  
    Readln(...);  
end.
```

Выполнение программ

- Получение кода возврата:

```
$ echo 'Hello, world!'  
$ echo $?
```

- Последовательное выполнение программ:

```
$ cd; ls
```

- Логическое “И”

```
$ cd /tmp/0 && ls
```

- Логическое “ИЛИ”

```
$ cd /tmp/0 || mkdir /tmp/0
```

Выполнение программ

Фоновый режим выполнения команд:

- Перевести команду из переднего плана в фон:

`Ctrl+Z; bg`

- Запустить команду в фоне:

`$ command &`

- Получить список команд в фоновом режиме:

`$ jobs`

- Вывести команду из фона на передний план:

`$ fg`

- Прервать выполнение команды переднего плана:

`Ctrl+C`

Выполнение программ

Получить список процессов:

```
$ ps; ps aux
```

Послать сигнал процессу:

```
$ kill -<signal> <pid>
```

Остановить процесс:

```
$ kill -SIGSTOP <pid>
```

Продолжить выполнение:

```
$ kill -SIGCONT <pid>
```

Список сигналов:

```
$ kill -l
```

SIGKILL – уничтожить
процесс

SIGTERM – завершить
процесс

SIGQUIT - завершить
процесс

SIGSTOP – остановить
процесс

SIGCONT - продолжить
процесс

Потоки ввода-вывода

Стандартные потоки ввода-вывода:

- `STDIN` — стандартный поток ввода, файловый дескриптор: 0, открыт для чтения данных
- `STDOUT` — стандартный поток вывода, файловый дескриптор: 1, открыт для записи данных
- `STDERR` — стандартный поток ошибок, файловый дескриптор: 2, открыт для записи данных

ПОТОКИ ВВОДА-ВЫВОДА

Перенаправление потоков ввода-вывода

STDOUT

Вывод в файл:

```
$ cat > file
```

Запись в конец файла:

```
$ cat >> file
```

STDERR

Перенаправление вывода ошибок

```
$ ls /1 2>file
```

```
$ ls /1 2>&1 >file
```

```
$ ls /1 >file 2>&1
```

```
$ ls /1 &> file
```

STDIN

Ввод из файла:

```
$ cat < file
```

Ввод до разделителя:

```
$ cat <<END
```

```
Hello, world!
```

```
END
```

Ввод из файла и вывод в файл:

```
$ cat <file >file1
```


ПОТОКИ ВВОДА-ВЫВОДА

Конвейеры:

```
$ ls | sort  
$ cat file | head -n 10 | tail -n 5  
$ cat file | grep 'http://'
```

Устройства для перенаправления потоков ввода-вывода:

/dev/null - “пустое” устройство, в которое можно
записывать

/dev/zero - “нулевое” устройство, из которого можно
прочитать нули

Регулярные выражения

- Язык описания шаблонов текста
- Позволяют:
 - проверить наличие заданного шаблона в тексте
 - выделить в соответствии с шаблоном одну или несколько подстрок из текста

Простейшие шаблоны:

шаблон	-	совпадение с подстрокой 'шаблон'
(шаблон)	-	совпадение и выделение совпавшего текста

Регулярные выражения

<code>one two</code>	- 'one' или 'two'
<code>(one) {1, 2}</code>	- одна или две подстроки 'one'
<code>(one) {1, }</code>	- одна или больше подстроки 'one'
<code>(one) {, 2}</code>	- от нуля до двух подстрок 'one'
<code>(one) +</code>	- одна или больше подстроки 'one'
<code>(one) ?</code>	- ноль или одна подстрока 'one'
<code>(one) *</code>	- сколько угодно подстрок 'one'

<code>к (и о) т</code>	- 'КИТ' или 'КОТ'
<code>ки+т</code>	- 'КИТ', 'КИИИТ', ...
<code>ки?о?т</code>	- 'КТ', 'КИТ', 'КОТ', 'КИОТ'
<code>кии*т</code>	- 'КИТ', 'КИИИТ', ...

Регулярные выражения

.	- любой символ
[abc]	- перечисление символов
[^abc]	- символы, кроме перечисленных
[a-d]	- диапазоны символов
^	- начало строки
\$	- конец строки
^[A-Z][a-z]+	- слово с заглавной буквы, в начале строки.
\. \$	- строки, кончающиеся на точку.

Регулярные выражения:

- жадные - $a \cdot^* a$ → **лабораторная**
- ленивые - $a [^a]^+$ → **лабор**аторная

Утилита grep

grep - фильтр текста.

- \$ grep шаблон [файл] - ПОИСК И ВЫВОД СОВПАДАЮЩИХ строк
- \$ grep -v шаблон [файл] - ПОИСК И ВЫВОД НЕ СОВПАДАЮЩИХ строк

Примеры использования:

- \$ ls /bin | grep '^ [a-c] . * a '
- \$ ls /bin | grep '^ [a-b] . * [n-z] \$ '
- \$ grep -v '^ * \ (# \ | \$ \) '
- \$ grep -E -v '^ * (# | \$) '
- \$ egrep -v '^ * (# | \$) '

Утилита sed

sed – строковый редактор

Поиск и замена текста с sed:

```
$ sed 's/шаблон/замена/[ig]'
```

Примеры:

```
$ date
```

```
Пнд Окт 13 09:55:26 MSK 2014
```

```
$ date | sed 's/Окт/Янв/'
```

```
Пнд Янв 13 09:55:56 MSK 2014
```

```
$ date | sed 's/Окт/Янв/' | sed 's/^[^ ]\+ \+ \+//'
```

```
Янв 13 09:56:41 MSK 2014
```

```
$
```

Утилита awk

awk – скриптовый язык обработки текстовой информации

Общий вид программы awk:

```
$ awk '/шаблон/ {действие;} /шаблон/ {действие;} ...'
```

Примеры:

```
$ ls -l /bin | head -4
```

```
total 5596
```

```
lrwxrwxrwx 1 root root          4 Feb 25 05:30 awk -> gawk
-rwxr-xr-x 1 root root    19064 Apr 20  2008 basename
-rwxr-xr-x 1 root root  549368 Mar 27  2008 bash
```

```
$ ls -l /bin | awk '/^-/ {print $9"\t->\t"$3:"$4"\t"$1;}' \
| head -5
```

```
basename      ->          root:root          -rwxr-xr-x
bash          ->          root:root          -rwxr-xr-x
bzip2         ->          root:root          -rwxr-xr-x
bzip2recover  ->          root:root          -rwxr-xr-x
cat           ->          root:root          -rwxr-x
```

Утилита awk

Использование переменных в программах awk:

```
$ ps aux | head -n 5
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	1.0	86996	11400	?	Ss	Mar10	0:55	/lib/..
message+	60	0.0	0.3	8076	3992	?	Ss	Mar10	0:19	/bin/..
root	62	0.0	0.6	13720	7012	?	Ss	Mar10	0:09	/lib/..
root	77	0.0	0.2	3236	2196	?	Ss	Mar10	0:02	/usr/..

```
$ ps aux | awk '/^[^U]/ {n++;CPU=CPU+$3; MEM=MEM+$4;}
> END{print "CPU: ",CPU,"%; MEM=",MEM,"%"}'
```

```
CPU: 0 %; MEM= 14.8 %
```

```
$ ps aux | awk 'BEGIN{N=0; CPU=100.0; MEM=100.0;}
> /^[^U]/ {N++;CPU-=$3; MEM-=$4;}
> END{print "Proc:",N,"; free CPU: ",CPU"%; free MEM:",MEM"%}'
```

```
Proc: 20 ; free CPU: 100%; free MEM: 85.1%
```


Скрипты

- Текстовые файлы с последовательностями команд:
 - необходимо указать, что это программа:
 - право выполнения.
- Могут быть на разных программных языках:
 - необходимо указать shell как интерпретатор:
 - специальный формат первой строки файла

```
$ cat >hello.sh <<END
#!/bin/sh
echo 'Hello, world!'
END
$ chmod a+x hello.sh
$ ./hello.sh
Hello, world!
$
```

Переменные shell

Переменные:

- окружения
- пользователя

Список переменных:

```
$ set
```

Присваивание значений:

```
$ A=10
$ A=A
$ A='Текст с пробелами'
$ A="Текст с переменными"
```

Использование значений:

```
$ B=$A
$ C="B=$B"
$ echo $C, "C=$C"
$ echo $CB, ${C}B
```

Запись вывода команды в переменную:

```
$ DATE=`date`; echo $DATE
$ A=`ls /bin | grep '^bash' | head -n 1`; echo $A
$ A=$(ls /bin)
```

Управляющие структуры shell

Условное выполнение:

```
if ... ; then ....; else ...; fi
```

```
$ if /bin/true; then echo 'True'; else echo 'False'; fi  
True
```

```
$ if /bin/false; then echo 'True'; else echo 'False'; fi  
False
```

```
$ if ls /bin/ | grep -q 'true'; then  
    echo 'True'  
else  
    echo 'False'  
fi
```

```
$ /bin/true && echo 'True'
```

```
$ /bin/false || echo 'False'
```

Управляющие структуры shell

Проверка условий: `/usr/bin/test ; /usr/bin/`

- Для файлов:

- | | | |
|---------------------------|---|---------------------------------|
| <code>-f /bin/bash</code> | - | файл существует |
| <code>-x /bin/bash</code> | - | файл есть и может быть выполнен |
| <code>-r /bin/bash</code> | - | файл есть и может быть прочитан |
| <code>-w ~/.bashrc</code> | - | файл есть и может быть записан |
| <code>-d /bin</code> | - | каталог существует |

Пример:

```
$ [ -w ~/.bashrc ] && echo "yes"
```

```
$ if [ -w ~/.bashrc ]; then echo "yes"; fi
```

Управляющие структуры shell

Проверка условий:

- Для чисел:

'10' -eq '10'	-	равно
'10' -ne '5'	-	не равно
'10' -gt '5'	-	больше
'10' -lt '20'	-	меньше

- Для строк:

-n 'строка'	-	строка не пустая
-z ''	-	строка пустая
'строка' == 'строка'	-	строки совпадают
'Строка' != 'строка'	-	строки не совпадают

Управляющие структуры shell

Циклы:

```
while команда; do список команд; done  
until команда; do список команд; done  
for переменная in список значений; do команды; done
```

Примеры:

```
$ while /bin/true; do echo "Y"; sleep 1s; done  
$ until /bin/false; do echo "N"; sleep 1s; done  
  
$ for i in `ls /bin`; do echo $i; done  
$ for i in `seq 1 10`; do echo $i; done
```

Выполнение команд в заданное время

Стандартный и традиционный вариант: использование демона crond.

- Получить / изменить настройки crond для пользователя:

```
$ crontab -l; crontab -e
```

- Формат файла crontab:

```
SHELL=/bin/sh
MAILTO=user@domain.tld
5 0 * * * $HOME/bin/daily.job >> $HOME/tmp/out 2>&1
15 14 1 * * $HOME/bin/monthly
23 */2 * * * echo "run at 00:23,02:23, ..."
5 4 * * sun echo "run at 04:05 every sunday"
@reboot $HOME/bin/init_after_reboot
```

- Файлы расписаний процессов пользователей: /var/spool/cron/
- Файлы расписаний для системных процессов: /etc/crontab, /etc/cron.d/, /etc/cron.daily/, /etc/cron.hourly/, /etc/cron.weekly/, /etc/cron.monthly/ .

Выполнение команд в заданное время

Вариант использования systemd для запуска команд в заданное время:

```
# mkdir -p /root/bin/
# cat << EOF > /root/bin/test.sh
#!/bin/bash
echo "Test script executing at $(date)" | \
    tee -a /tmp/test.log
EOF
# chmod 755 /root/bin/test.sh
# cat << EOF > /etc/systemd/system/test.service
[Unit]
Description=Execute the test script

[Service]
Type=oneshot
User=user
Group=group
ExecStart=/root/bin/test.sh
EOF
```


Выполнение команд в заданное время

Вариант использования systemd для запуска команд в заданное время:

```
# cat << EOF > /etc/systemd/system/test.timer
[Unit]
Description=Start test.service every 10 seconds

[Timer]
# OnCalendar=DayOfWeek Year-Month-Day Hour:Minute:Second
# OnBootSec=50seconds
OnCalendar=*:*:0/10
Persistent=true

[Install]
WantedBy=timers.target
EOF

# systemctl daemon-reload

# systemctl enable test.timer
```

Задание на лабораторную работу

В лабораторной работе требуется:

- организовать периодическое получение данных о работе определённых систем;
- записывать их в файлы для последующего анализа;
- организовать вывод текущих значений через веб-интерфейс;
- построить графики изменения наблюдаемых величин и предоставить к ним доступ через веб-интерфейс.

Скрипты сбора данных о работе систем:

- число процессов в виртуальном сервере;
- суммарный объем переданных и принятых через сетевой интерфейс сервера данных;
- число переданных и принятых через порт удалённого коммутатора пакетов и байтов данных;
- данные о текущем состоянии работы системы (загрузка процессора, использования ОЗУ, параметры планировщика процессов и т.п.)

Задание на лабораторную работу

Скрипты для сбора данных:

- тексты скриптов получаются из примеров,
- устанавливаются в виртуальный сервер, настраиваются, адаптируются,
- обеспечивается их запуск с заданной периодичностью.

Вывод данных по запросам браузера:

- выполняется скриптами, запускаемыми с помощью веб-сервера `lighttpd`:
- тексты скриптов получаются из примеров,
- устанавливаются в виртуальный сервер, настраиваются, адаптируются,
- настраивается конфигурация `lighttpd` для запуска этих скриптов.

Задание на лабораторную работу

Требуется обеспечить безопасность работы системы и настроить разграничение доступа:

- сбор данных выполнять от отдельного непривилегированного пользователя, который:
 - не должен иметь возможности удалённого входа в систему,
 - не должен иметь возможности изменения скриптов
- псевдопользователь веб-сервера:
 - не должен иметь возможности записи в файлы скриптов,
 - не должен иметь возможность записи в файлы с собираемыми данными,
- остальные пользователи системы:
 - не должны иметь возможности записи или удаления временных файлов веб-сервера,
 - не должны иметь возможности чтения и записи файлов с собираемыми данными.

Задание на лабораторную работу

- Обновить систему из репозитория АРТ;
- Доставить необходимое программное обеспечение;
- Адаптировать скрипты для чтения и записи значений в журнал;
- Обеспечить регулярное выполнение скриптов;
- Адаптировать скрипты для веб-интерфейса;
- Настроить `lighttpd` для работы с ними;
- Обеспечить безопасное выполнение скриптов;
- Разобрать работу скриптов и быть готовым объяснить, что они делают