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
- STDOUT – стандартный поток вывода
- STDERR - стандартный поток ошибок

Потоки ввода-вывода

Перенаправление потоков ввода-вывода

STDOUT

Вывод в файл:

```
$ cat > file
```

Запись в конец файла:

```
$ cat >> 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

`one|two`	-	'one' или 'two'
`(one){1,2}`	-	одна или две подстроки 'one'
`(one){1,}`	-	одна или больше подстроки 'one'
`(one){,2}`	-	от нуля до двух подстрок 'one'
`(one)+`	-	одна или больше подстроки 'one'
`(one)?`	-	ноль или одна подстрока 'one'
`(one)*`	-	сколько угодно подстрок 'one'
`к(и|о)т`	-	'кит' или 'кот'
`ки+т`	-	'кит', 'киииит', ...
`ки?о?т`	-	'кт', 'кит', 'кот', 'киот'
`кии*т`	-	'кит', 'киииит', ...

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.*a` → л**абораторна**я
- ленивые – `a[^a]+` → л**абор**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"
```

Запись вывода команды в переменную:
```
$ DATE=`date`; echo $DATE
$ A=`ls /bin | grep '^bash' | head -n 1`; echo $A
```

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:

`-f /bin/bash`	-	файл существует
`-x /bin/bash`	-	файл есть и может быть выполнен
`-r /bin/bash`	-	файл есть и может быть прочитан
`-w ~/.bashrc`	-	файл есть и может быть записан
`-d /bin`	-	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

- Запуск программ в нужное время обеспечивает демон crond.
- Получить настройки crond для пользователя:

```
$ crontab -l
```

- Изменить настройки crond для пользователя:

```
$ crontab -e
```

Задание на лабораторную работу

- Обновить систему из репозиториев APT;
- 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