Be in shell

Compound commands

Pavel Fibich

pavel.fibich@prf.jcu.cz
dep. Botany - Na Zlaté stoce 1

03-2015

Přírodovědecká
fakulta
Faculty
of Science

Warm up Exercise

Exercise

- write command that sets variable to the number of lines of some file minus 1 and divided by 2
- write script that count all `txt` files in the folder given as 1st argument and store the value in the variable TXT (use export to make variable visible)
- write script that cheks if in the current folder is more files than number you give as 1st argument of the script

loops

loops

loops

Several commands can repeat actions:

`while` *condition*; `do` ... `done`, `for` ... `in` ... ; `do` ... `done`

```
$ ls -l | while read FILE; do echo $FILE; done
$ for i in `seq 1 10`; do echo $i; done
$ while read -p "Company: " COMPANY; do
  if [ -f "orders_$COMPANY.txt" ]; then
    echo "There is order from this company."
  else
    echo "There are no orders from this company!"
  fi
done
```

Use CTRL+c to get out from the loop.

Exercise

Improve the script to react on `quit`, e.g. by `break` command, jump out of the loop. Add printing of the order, if there is any. Write command that will print file names in the folder without .* suffix.

loops

`for` reads sequence of values into a variable and repeats the enclosed commands one for each value

```
$ for file in *.csv; do wc -l $file; done # lines of csvs
$ for i in $( ls ); do echo "file: $i"; done
$ for FILE_PREF in order invoice purchase_order; do
  if test -f "$FILE_PREF""_vendor.txt"; then
    printf "%s\n" "There is a $FILE_PREF file for vendor"
  fi
done
```

Embedded `let`

```
$ for (( CO=5; CO<50; CO=CO+5 )); do # c style
  printf "The counter is %d\n" $CO
done
```

Exercise

Use `for` loop to create file with numbers from 1 to 999, all in one line and each on separate line.

functions and aliases

For faster execution, we can prepare functions to do the serie of commands.

```
$ function whoson() { date; echo "Hi $1, currtely logged
    in users"; who; }
$ whoson Pavel  # run function with parameter
...
```

We can also create aliases for the commands

```
$ ll
bash: ll: command not found
$ alias ll='ls -l'  # create alias ll
$ ll
...
$ alias  # list of all aliases
alias ll='ls -l'
alias ls='ls --color=auto'
```

To load aliases or functions automaticaly, we must store them in the `.bash...` files (e.g. in `.bashrc`).

`getopts` script options

For single minus options we can use `getopts`, but more standart is to handle also double minus options by `getopt`.

```
$ cat ./opt.sh
while getopts ":sp:" o; do
  case "${o}" in
   s) s="set";;
   p) p=${OPTARG};;
   *) echo "usage: ./opt.sh -s -p arg";;
  esac
done
if [ -n "$s" ]; then echo "s = ${s}"; fi
echo "p = ${p}"
$ ./opt.sh -s -p 4
s = set
p = 4
```

First argument of `getopts` defines if option has argument by adding `:`

Combining commands

Grouping of commands by curly brackets (current shell)

```
$ MX=4
$ { sleep 5; MX=2; echo "Slept for 5s"; }
Slept for 5s
$ echo $MX
2
```

by round brackets (new shell)

```
( sleep 5; MX=3; echo "Slept for 5s" )
Slept for 5s
$ echo $MX
2
```

Conditional sequences

```
$ cat file && wc -l file # run wc if cat succeed
cat: file: No such file or directory
$ cat file || wc -l file # run wc even if cat not succeed
cat: file: No such file or directory
wc: file: No such file or directory
```

jobs and signals *

Running jobs can be watched by `top` command. If you plan to run command and then log out, it is good to run it by `nohup`

```
$ nohup myscript &
[1] 16858 # PID
```

By pressing CTRL+c we are sending signal SIGTERM. We can manually send also other signals

```
$ {sleep 60; echo "DONE";} &
[1] 16863
$ kill -SIGSTOP 16863
[1]+  Stopped                 { sleep 60; echo "DONE"; }
$ kill -SIGCONT 16863
DONE
```

We can adjust our script to be able to react to the signals (e.g. storing results when job is killed). Write `trap` at the beginning of the script

```
trap "cp outputs /storage/" TERM EXIT
```

will do `cp` if signal for cancel or exit is catched.

finding files

To find a file, you have many options

- recursive `ls` and combine with string matching (see man to `ls`)
- more powerful is `find`

```
$ find . -name "*.txt"  # all files .txt from the current dir
$ find . -type d -or -name "*.txt"  # ... or directories
$ find . ! -name "*.txt"  # files not having suffix *.txt
```

You can check file size (`-size`), time of modification (`mtime`), ... and run some command on the found files

```
$ find . -name "*.txt" -exec cat {} \;
...
$ find . -name "*.txt" -ls  # predefined command
...
```

`xargs` takes input and executes your chosen command on it, so we can avoid loops sometimes. It is often use instead of `-exec` in `find`.

```
$ echo Hello | xargs echo  # simple ilustration
Hello
$ find . -name "*.bak" -type f -print | xargs /bin/rm
```

Exercise on loops and signals

Exercise

- Write a script that for the file (first script argument) print the second part of the file (i.e. for file with 10 lines, it prints the last 5).
- Create a script to print every x line of the file (e.g. use variable in loop).
- Add check that the argument is the file and have more than x (second argument) lines. If not, print corresponding error message.
- Run 100 `sleep 30` commands in background and store their PIDS in file (eg. by `ps`).
- Read the file and let all `sleep` commands to suspend (for each line get PID as substring of variable where is stored whole line, e.g. `${MYV:9:8}`).
- Write general function for it (file with PIDS and signal to be send as two arguments of function).

Homework 1 - ppsaver

Homework1

- Write script that will change all wired interpreters of python or perl in the folder.
- In shortcut, just change in all files 1st line (if there is such line)

```
#!/bin/bin/perl
#!/bin/bin/python
```

  change for (respectively)

```
#!/usr/bin/env perl
#!/usr/bin/env python
```

- Script will take 2 or 3 arguments: 1st is folder, where it will be recursivelly looking for files, 2nd is interpreted name (only `perl` or `python` will be specified) and if 3rd argument will be `make`, than changes will be applied, if 3rd is not specified, than just print file names with wired interpreter.

Homework 1 - ppsaver

Homework1

- Script will go through all files in the folder (1st arg) and his subfolder (and so on), check if files contain interpreter (2nd arg) and according 3rd arg will make change in the file or just print the file name.
- Example run: no 3rd arg, just list of files with wired header

```
$ ./ppsaver ttest/ perl
FILE: ttest/myfile1
FILE: ttest/ttest2/myfile1
```

- Example run: changes to be done

```
$ ./ppsaver ttest/ perl make
FILE: ttest/myfile1
MODIFIED!
FILE: ttest/ttest2/myfile2
MODIFIED!
```

Homework 1 - ppsaver

Homework1

- Check the number of arguments.

```
$ ./ppsaver ttest/
Not enough args: ./ppsaver FOLDER INTERPRETER {make}
```

- Leave all files without wired interpreter on 1st line untouched.
- Follow exact format of the input and output.
- Read the homework and test it again before sending it.
- Homework is individual, not team, work!
- Send me the solution (script name according your surname: `ppsaver.SURNAME`) by email before 18:00 28 March
- Score: 0..10

By not presenting at least some efort, you can not go for the exam. Final score from the course is mostly dependent on the scores of homeworks.