Shell scripting

Mikhail Dozmorov

Spring 2018

Workflow scripts

- A script is a file with a `.sh` extension. It contains a list of shell commands executed by an interpreter
- Shebang (`#!`) defines the interpreter on the first line
  - `#!/bin/bash` - commands interpreted by bash
  - `#!/usr/bin/python` - interpreted by Python
- A script file should have `x` permissions: `chmod u+x hello_world.sh`
- Running a script: `./hello_world.sh`

Variables

- Set a variable: `count_of_files=3`
- Wrong set a variable: `count_of_files = 3` (spaces)
- Quotes are optional. The following commantd are equivalent:

```
file="/home/mdozmorov/work/README.md"
file=/home/mdozmorov/work/README.md
```

- Use a variable: `echo $file`

Capturing output of a command into a variable using `backticks`

- Wrap a command into backticks - the backwards apostrophes that appear on a US English keyboard at the upper left, under the ~ (tilde)
- Equivalent saying "get the output of the backticked command as a string variable"

```
echo `date`
CURRENT_DIR=`pwd`
file_name=`basename /bin/mkdir`
```

Arguments of a script as variables

- Example of an argument: `./hello_world.sh "Hello World!"`
- Within a script, special variables are reserved:

```
echo $0      - prints the script name
echo $1      - prints the first argument
echo $2      - prints the second argument
echo ${10} - prints the tenth argument
echo $#    - prints the number of arguments
```

Internal variables

Set system's parameters. Can be defined in system's configuration files `.bashrc`, `.bash_profile`

```
DISPLAY - tells X11 on which display to open windows
EDITOR  - default text editor; usually emacs or vim
HOME    - path to user's home directory; same as ~
PATH    - path to executable programs
PWD     - current directory, same as pwd
SHELL   - path to the current shell
TERM    - current terminal type
USER    - account name of current user, same as whoami
```

Use `echo` to see their content

Aliases

To avoid retyping commands - use an alias. Can be defined in system's configuration files `.profile` (Linux), `.bash_profile`, `.bashrc` (Mac)

```
alias lah='ls -lah'
alias ..='cd ..'

# get top process eating memory
alias psmem='ps auxf | sort -nr -k 4'
alias psmem10='ps auxf | sort -nr -k 4 | head -10'

# get top process eating cpu
alias pscpu='ps auxf | sort -nr -k 3'
alias pscpu10='ps auxf | sort -nr -k 3 | head -10'

# Find files eating space in the current directory
alias spacehogs='du -cks * | sort -rn'
```

Conditional execution (if .. then .. else)

```
if [ ! -e $results_dir ]; then
    mkdir $results_dir;
fi
```

Some popular operators for checking a condition include:

```
-e <file>                    - TRUE if a specific file/directory e
-s <file>                    - TRUE if non-empty file
-z <string>                  - TRUE if the given string is empty
<string1> = <string2>  - TRUE if the two strings are equal
```

- `help test` - see all operators

Loops (for .. do .. done)

```
for file in `ls *.txt`; do
    echo $file;
    gzip $file;
done
```

- `while-do-done` construct also available

The PATH environment variable

- Unix executable commands are located in special folders

```
$ which ls
/usr/bin/ls
$ which cat
/usr/bin/cat
$ which head
/usr/bin/head
```

- Executables may be kept in many different places on the Unix system.
- The PATH environmental variable is a colon-delimited list of directories where your shell will look to find exexutable commands

```
$ echo $PATH
/Users/mdozmorov/miniconda2/bin:/Users/mdozmorov/.rvm/gems/rub
```

Exercise:

Expanding the PATH

- Often you need to install software **as a user**, i.e., not as `root` or `sudo` user
- Create user-specific `bin`, `lib` folders, like:

```
$ mkdir ~/.local/bin
$ mkdir ~/.local/lib
```

- `.local` is a hidden folder in your home directory (use `ls -lah` to see it)
- Add these folders to the search path: `export PATH=$PATH:$HOME/.local/bin:$HOME/.local/lib` - now, Unix will look there for executables
- Put the `export ...` command in `bash_profile` to automatically execute it every time you use shell

Installing software as a user

- Read `README` - each software is different
- When installing using `make`, typically:

```
$ ./configure --prefix=$HOME/.local
$ make
$ make install
```

- When using Python `setup.py`, typically:

```
$ python setup.py install --user
```

- When installing Python packages using `pip`

```
$ pip install --user FOOBAR
```

https://unix.stackexchange.com/questions/42567/how-to-install-program-locally-without-sudo-privileges