UNIX and Shell From the Start

Paddy Corr
<paddy@netsoc.tcd.ie>

November 27, 2013

Connecting

`ssh` (secure shell) to one of our servers using:

- PuTTY on windows with login.netsoc.tcd.ie
- using a terminal on unix $ssh name@login.netsoc.tcd.ie

The Basics

If you have never worked with a UNIX shell then these next few slides are the slides to get you started.
We will go through:

- ls
- cd
- rm
- rmdir
- mkdir
- mv
- cp

But first; What are we looking at?

What is this?

```
paddy@cube:~$
```

This is your prompt. It shows your username (paddy), the hostname (cube), and the directory (~: tilde means home).

ls

ls lists the contents of a directory.

```
$ ls -al
```

in this example the -al are the arguments passed to ls.

- -a means all and will list hidden files in a directory
- -l means long listing format and will list more info about each file

output:
`-rw-r----- 1 paddy paddy 3047 Aug 19 22:51 .bashrc`
shows the privileges, number of files in a folder, user etc.

cd

`cd` will change directory.
If we do `ls -a` we can se that there are folders called . and ..
"." means the current directory and ".." means the directory that is above this one. to change into a different directory:

```
$cd .. or $cd <foldername from ls>
```

rm

`rm` removes a file forever.
WARNING: UNIX is not like windows and does not have a recycling bin. If you rm something it will be gone forever.
Usage:

```
$rm [-rvf] <filename>
```

- -r means recursively and will delete all file in a folder and the folder
- -v means verbal and will list files as it deletes
- -f means force

mkdir

Creates one or more new folders

```
$mkdir <filename> <filename2> <etc>
```

mv

Moves or renames files or directorys.
The argument -f can be used to force.
To rename a file or directory:

```
$mv file1 file2
```

```
$mv directory1 directory2
```

Move a file to a directory:

```
$mv file1 Directory
```

If the folder you are moving a file to has a file with the same name it will be deleted first. See WARNING about rm.

cp

Copy a file

```
$cp file1 file2
```

Copy to a different directory:

```
$cp file directory
```

man

`man` is your friend.
man is the manual for commands in UNIX.
If you want to know how to use ls for example then type :

```
$man ls
```

This will show you a list of arguments for ls along with things like the Author and usage etc.
`man` should be your first port of call. Yes, even before Google.

history

`history` will output the previous commands executed.
With this we can do things like repeat command 157 by typing
`$!157`
The exclamation point here is an original feature of c shell and existed before you could count on having arrow keys on your terminal. There are many more uses of !:

- `!!` - repeats last command
- `!python` - repeats last command that starts with a keyword
- `!?grep` - repeats last command that contains a keyword
- `^python^vim` - replaces word in previous command and inputs command again

autocompletion

After typing a letter or more of a command or directory/file we can press tab once to complete the command if there is only one possible auto-completion.
Otherwise if we press tab twice the output will be all the possible completions of the command.
After pressing tab twice:

```
paddy@cube:~$ wi
widget              winpopup-send     wish              wish-
winicontoppm        wipe              wish8.4           withs
winpopup-install    wireshark         wish8.5
```

cal

Print calender for certain month and year

```
$cal <month> <year>
```

WC

Word Count counts the newlines, words and bytes in a file. We will use this in examples later.

```
$wc [-wcl] <filename>
```

Where:

- w - words
- l - lines
- c - bytes

whoami

Print userid to output.

which

Follows PATH to find executables that have the same name as the arguments.
Used to loacate programs such as:

```
$which bash
```

Will return something like `\bin\bash`

alias

Set a variable or command to be the same command as something else.
Eg to set ls to always print ls -a

```
$alias "ls"="ls -a"
```

and to unalias something:

```
$unalias ls
```

tar

Compress and zip files together.

Main uses:

- compress

```
$tar -cvf <folder> <output file>
```

- decompress

```
$tar -xvf <compressed file> <output folder>
```

ps

List processes currently running on the system.

`$ps aux` prints all processes currently running

Let's start

Now that we know a lot of the basic UNIX commands and how to use them we can use them together to do really cool and helpful things.
To do this we will use:

- piping - |
- redirection of stdin - $<$
- redirection of stdout - $>$

redirection of stdout

Redirection of stdout is used to take the out put of a command and do some thing with it.
This is the begining of your scripting.
when we output to a file the output file will overwrite any file with that name that already exists.

- `grep searchword file1 > file2`
- `grep searchword file1 >> file2` will append to a file if it exists

redirection of stdin

Redirection of stdin is used for example to input a file to something like grep or cat.
eg:

- `grep searchword < file`
- cat < file < grep searchword < file

piping

- `find .* -name *.py | xargs wc -l`
- `ps -ef | grep emacs | grep -v grep | wc -l`
- `ls /home | sort | head -n 5`

Piping will take the output directly from a program and use this output to do something else.
In the case of xargs shown the program xargs takes each line of output piped to it and does something with that output. In this case counding the lines in each file given.

First Script

Lets take the previous `find` command and put it in a file. Open a program such as nano/vim (maybe nano for starting) and type the following into the file:

```
#!/bin/bash
clear
echo "hello $user"
echo "calendar"
echo "today is "; date
cal
exit 0
```

And save it. Try running this file now by typeing `./<name>.sh` What happens?

Permissions

Your file will not have executed because you do not have permissions to execute that file. To fix this we use `chmod`

```
chmod u+x scriptname.sh
```

conditionals

We use && for AND and —— for exclusive OR (XOR)

```
$ true && echo "Hi $USER"
  Hi paddy
$ false && echo "Hi $USER"
$ true || echo "Hi $USER"
$ false || echo "Hi $USER"
  Hi paddy
```

backticks

enclose a string in backticks and that string will be executed as a command. Eg in a script:

```
echo "the date is ‘date‘"
```

special variables

```
$RANDOM: random integer
$$: current PID
$?: exit status of last process exited
$!: PID of most recent child process
$@: argv
$0$9: 0th to 9th argument
$#: number of arguments
$SHELL: current shell
$USER: current user
$HOME: Current users home directory-thanks to Ben xox
```

If and exits

In the script that we have written we had at the end "exit 0" this means that the scripts exits nicely. If this was an "exit 1" then the script would not have exited nicely. We can use this with if statements:

```
$ if (exit 0); then echo "cool"; fi
  yay
$ if (exit 1); then echo "cool"; fi
```

While

A while loop in a shell script

```
#!/bin/bash
COUNTER=0
while [  $COUNTER -lt 10 ]; do
 echo The counter is $COUNTER
 let COUNTER=COUNTER+1
done
```

for

We can use for to iterate over arguments.
eg:

```
$ for i in $(ls); do
> echo $i
> du -sh $i
> done
```

until

```
#!/bin/bash
COUNTER=20
until [  $COUNTER -lt 10 ]; do
 echo COUNTER $COUNTER
 let COUNTER-=1
done
```

functions

functions have the syntax: function name code

```
#!/bin/bash
    function quit {
        exit
    }
    function hello {
        echo Hello!
    }
    hello
    quit
    echo foo
```

functions with params

```
#!/bin/bash
    function quit {
        exit
    }
    function a {
        echo $1
    }
    a hello
    a world
    echo foo
```