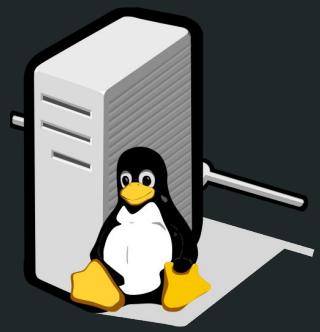
Advanced Linux Commands & Shell Scripting

Advanced Genomics & Bioinformatics Workshop

James Oguya
Nairobi, Kenya
August, 2016



Man pages

- Most Linux commands are shipped with their reference manuals
- To view a command's manual, use '*man*' command & the command as an argument; e.g.

```
$ man ls
$ man cat
```

- Since man itself is a command, we can view also view it's manual

```
$ man man
```

- '`man`' uses the same interface as '`less`';
  - Use '`q`' to quit/close '`man`'

Listing

- Use ‘`ls`’ command to list all files & directories in the current directory
  - doesn’t include *hidden* files & directories, by default
- Common pattern when using the command line is changing directories using ‘`cd`’ & then immediately using ‘`ls`’ to view the contents of the directory.
- ‘`ls`’ has 2 *output formats*:
  - Normal/short list format

    ```
    $ ls
    ```
  - Long list format

    ```
    $ ls -l
    ```

Listing

- To view hidden files & directories, use ‘-a’ option with ‘less’

```
$ ls -a
$ ls -la
```

- ‘`ls`’ command can also be used to check if a file/directory exists

```
$ ls file99.txt
$ ls ~
$ ls ~/earth
```

Copying

- Use ‘cp’ to copy a file or directory to the same or different directory.
  - `cp source destination`
- Copying a file within the same directory

```
$ cp file1.txt file2.txt
```

- Copying a file to a directory

```
$ mkdir files
$ cp file2.txt files/
```

- Copying multiple files to another directory

```
$ cp file1.txt file2.txt files/
```

- Use ‘-r’ option to copy a directory

```
$ cp -r files/ new-files/
```

Renaming

- Use ‘mv’ command to rename files & directories
  - `mv source destination`
- To rename a file

```
$ mv file1.txt file3.txt
```

- To move a file into a different directory

```
$ mv file3.txt new-files/
```

- To rename a directory

```
$ mv new-files/ old-files/
```

- To move a directory into another directory

```
$ mv old-files/ files/
$ ls files/
```

Deleting

- Use ‘rm’ command to delete files & directories
  - `rm file`
- Take care when using ‘`rm`’ command because there is NO Recycle Bin/Trash
- Use ‘`-r`’ to recursively delete directories and their contents
- To delete a file

  ```
  $ rm file2.txt
  ```
- Delete a folder

  ```
  $ rm -rf files/old-files
  ```

Globbing

- Globbing(filename expansion) recognizes and expands wildcard characters(‘*’ and ‘?’), character lists in square brackets & other special characters(e.g. ‘^’ for negating the sense of a match)
  - wildcard characters will not match filenames that start with a dot(‘.’) e.g. `.bashrc`
- A few examples
  - List all files beginning with ‘file’

    ```
    $ ls file*
    ```
  - List all files ending with ‘.txt’

    ```
    $ ls *.txt
    ```

Redirecting and Appending

- Everything you type & read on the command line can be considered as character streams or just streams in general
- 3 types of streams:
  - standard output(stdout): normal command output displayed on the screen
  - standard error(stderr): displays errors from a command or program; similar to stdout
  - Standard input(stdin): standard input stream for a command or program
- To redirect the stdout(output of a command) to a file

```
$ echo “Current date and time is” > today.txt
$ cat today.txt
```

Redirecting and Appending

- To append the stderr(output of a command) to a file
```
$ date >> today.txt
$ cat today.txt
```
- To use a file as stdin stream to a command
```
$ cat < today.txt
```
- To write stderr output to a file
```
$ ls files/ file99.txt 2> stderr
```
- To write stdout & stderr to different files
```
$ ls files/ file99.txt 1> output 2> error
$ cat output
$ cat error
```

Redirecting and Appending

- To combine both stdout & stderr to one file

```
$ ls files/ file99.txt &> streams
$ cat streams
```

Piping

- A form of stream redirection whereby the output of a command is used as an input the other command
  - The pipe operator(‘|’) is placed in between the 2 commands
    - command1 | command2
    - command1 | command2 | command3 | command4
- A few examples:

```
$ cat streams | wc -l
$ ls -l | less
$ cat streams | grep file | wc -l
```

Downloading from the internet

- GNU wget is a powerful non-interactive download manager in Linux
  - non-interactive: can run in the background even when you're logged out
- Basic wget usage
  - `wget URL`
- Downloading a file:

  ```
  $ wget
  http://hpc.ilri.cgiar.org/~joguya/gene-description.txt
  ```
- Download a file & save it with a different name

  ```
  $ wget
  http://hpc.ilri.cgiar.org/~joguya/gene-description.txt -O
  gene-desc.txt
  ```

Grepping

- Use ‘`grep`’ command to search for a substring in a file
  - `grep substring file`
- To ignore case distinctions in both substring & input file, use ‘`-i`’ option
- A few examples:

```
$ grep date today.txt
$ grep protein_coding gene-description.txt | wc -l
```

Making heads and tails of your files

- Two complementary commands for inspecting file contents
  - ‘`head`’: shows the beginning(head) of a file
  - ‘`tail`’: shows the end(tail) of a file
- You can also use stdout stream with ‘`head`’ & ‘`tail`’
  - `commandA | head`
  - `commandB | tail`
- To view the first 5 lines of a file

  ```
  $ head -n5 gene-description.txt
  ```
- To view the last 5 lines of a file

  ```
  $ head -n5 gene-description.txt
  ```

Your first shell script

- A shell script is a text with a list of commands.
- Shell scripts are good for automating tasks you frequently do or for running *batch* jobs
- Using ‘`nano`’(text editor), we’ll create a new file named `script1.sh` with the following contents:

```
echo "Date and time is:"
date
echo "Your current directory is:"
pwd
```

Your first shell script

- Run `script1.sh` shell script
  `$ sh script1.sh`
- It should output something similar this:

```
Date and time is:
Mon Aug  8 12:30:54 EAT 2016
Your current directory is:
/home/user1
```

Your second shell script

- Using ‘`nano`’(text editor), we’ll create another file named `script2.sh` with the following contents:

```
DATE=$(date)
PWD=$(pwd)

echo "Date and time is: $DATE"
echo "Your current directory is: $PWD"
```

Your second shell script

- Two new concepts:
  - variables: a symbolic name for to hold data e.g. numbers, text, e.t.c.
  - command substitution: starts a subshell to run the named command; It's recommended to use `$(command)` instead of `` `command` ``

Loops and sequences

- A loop is a block of code that iterates a list of commands as long as the loop control condition is true.
- Basic looping construct

```
for arg in [list]
do
    command(s)...
done
```

  - During each pass through the loop, *arg* takes on the value of each successive item in the *list*
- So, why do we need loops?

Loops and sequences

- Using ‘`nano`’(text editor), we’ll create another file named `script3.sh` with the following contents:

```
for num in 1 2 3
do
    echo “we are on number: $num”
done
```

- Run `script3.sh` shell script

```
$ sh script3.sh
```

Sequences

- Let’s create another file named `script4.sh` with the following contents:

```
for num in {1..3}
do
    echo “we are on number: $num”
done
```

- Run `script4.sh` shell script

```
$ sh script4.sh
```

More sequences

- Same results but using a command substitution & ‘`seq`’ command
  - Create another file named `script4.sh` with the following contents:

```
for num in $(seq 1 3)
do
    echo “we are on number: $num”
done
```

  - Run `script5.sh` shell script

```
$ sh script5.sh
```

- Several ways to achieve the same thing!!

Questions?