Basics of Shell Scripting

github.com/grapheo12

Introduction

What is a Shell?

- Shell is a program that let's you run shell commands.
- A Shell is a command line interpreter, which runs your shell scripts.
- Shell **is not equal** to a Terminal.
- Examples:
  - tcsh
  - fish
  - Bourne - Again Shell (bash)
  - zsh etc.

What is a Terminal then?

- A terminal emulator is a graphical program that is used to run the shell.
- It emulates, although in a small window and with lots of add-ons, the tty interface.
- Examples:
  - GNOME Terminal
  - Konsole
  - urxvt
  - Alacritty
  - XTerm
  - Termite etc.

Wait! What is tty?

- tty (stands for Teletype) is text based interface to your operating system.
- On most Linux systems, you can use multiple teletype sessions and they can be accessed by hitting `Ctrl + Alt + F<1-12>` .
- What does a tty launch after you login into one? Yes, your default *shell*.

Pipes and redirections i

- Shell commands (we'll see a few in a moment) generally spit there outputs to `stdout` and in some mode take input from `stdin` .
- What this means, is that we can redirect the output of one command to the input of the other without creating an intermediate file.
- To do so we use the pipe (`|`).
- For example, suppose we write out something using `echo` and want to print the number of lines in it using `wc` . The command for that will be:

```
echo "Lorel ipsum dolor sit amet.\nSic mundus creatus es" | wc -l
```

- Operators for redirecting input/output from specific files / streams are:

Pipes and redirections ii

- `< inputfile` Redirects `stdin` to take input from a file.
- `> outputfile` Redirects the output to a new file (Existing file is overwritten).
- `>> outputfile` Appends the output to a file.

- An useful pattern is to redirect `stderr` to `/dev/null` (the black hole of Linux), so that it doesn't pollute your output: `2>/dev/null` .

Some Basic Commands

ls i

- This is used to list files.
- Usage: `ls -[Options] [path]`
- If a path is not given, current directory is assumed.
- Path can also contain wildcards. Example: `ls *.pdf` will list all the pdf files in the current directory.
- Options consists of one or a combination of character flags that invoke special functions:
  - `l` : List files with additional metadata
  - `a` : Show hidden files also. (Name begins with `.`)

- `h` : Show file sizes in MBs and GBs instead of bytes.
- `t` : Sort by date modified.
- ... and many more.

- Multiple options can be combined as: `ls -lah` .

Exercise: Open a shell and create a file with the list of all files (both hidden and visible) sorted by the date modified.

echo and printf i

- These are used to print stuff (Obvious Lol!)
- `printf` supports formatted output like C. `echo` doesn't.
- `echo` is more common in use than `printf` .
- Usage: `echo "some text"` or `printf "format string" "parameters"` .

Single and double quotes in bash

In bash, both single and double quotes are allowed. However there is subtle difference in behaviour. Inside double quoted string, you can use sub-commands enclosed by `$()` . This is not possible with single quotes. Run:

```
echo "$(ls /bin)"
```

and

```
echo '$(ls /bin)'
```

to see the difference.

cat, head and tail i

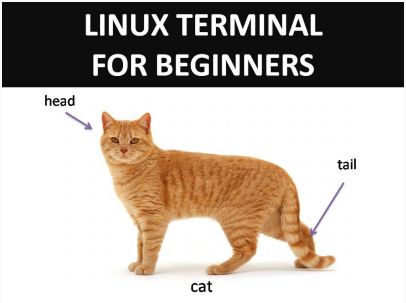

Figure 1: This is what these commands mean, literally!

- `cat` stands for `conCATenate` . `cat file1 file2 file3` will output all the 3 files combined in the given sequence.

cat, head and tail ii

- However, in practice, people use `cat` to print out the 1 full file only.
- `head` prints out the first few lines of a file and `tail` prints out the last few lines of a file.
- Both accept a parameter `-n<Number>` . This limits the output to `Number` number of lines.

Example: to get the first 15 lines of a line, run:

```
head -n15 file
```