Linux Command Line Interface

December 27, 2017

Foreword

- It is supposed to be a refresher (?!)
- If you are familiar with UNIX/Linux/MacOS X CLI, this is going to be boring...
- I will not talk about editors (vi, emacs...)

Basics

- **(Computer) Terminal**: Hardware device for data entry and display;
- **Terminal Emulator** (aka tty): An application program replacing a computer terminal. Many of those, OS dependent (cmd for DOS/Win; Terminal for OS X; xterm for Linux; ...);
- The Terminal provides user access to the computer through the **Command Line Interface** (CLI) where the user issues commands.
- The CLI "dialect" is the **Shell**. In UNIX-like systems a lot of dialects exists: sh, bash, csh, tcsh, ksh, zsh...
- I will only use the `bash` shell.

Shell

- How to access a shell
  - Logging in to your own Linux (virtual) box (CTRL-ALT-F1/CTRL-ALT-F7);
  - Opening a graphic terminal (xterm...);
  - SSHing into a server.
- To know which shell you use: `echo $SHELL`
  I use `this font` to denote commands
- SHELL is an environment variable
- `echo` is a command

Commands

- `apropos`: search the manual page names and descriptions
- `man`: manual page (try `man apropos`, `man man`)
  Almost every command, system program, or API has a man page
  `man apropos`, `man fread`, `man pthreads`, `man 1 open`, `man 2 open`
  Reading man pages is a very worthwhile activity
- Not everything is a command... `type <cmd>`
  `type man` (command), `type echo` (shell built-in), `type ls` (alias (well... on my system))
- Commands take arguments (optional or mandatory); take input from the standard input (aka `stdin`); produce output on the standard output (`stdout`) or the standard error output (`stderr`).

- UNIX philosophy: Have a lot of programs; Each of them does only one thing but does it well.

First Commands

- Print the current working directory: `pwd`;
- List the files in the current directory: `ls`;
- Create a directory: `mkdir <directoryName>`;
- Change the current directory to `<directoryName>`: `cd <directoryName>`;
- Create an empty file: `touch <fileName>`;

Wildcards

- List the files whose names start with a `c`: `ls c*`;
- List the files whose names end with a `c`: `ls *c`;
- List the files whose names contain a `c`: `ls *c*`;
- We could spend a whole session (and maybe more) on bash and the use of regular expressions (Look up *bash regular expressions* with your favorite search engine)

UNIX Philosophy (continued) - Pipes

Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input

- `grep`: find a string in a file or a set of files;
  ```
  prompt> grep Prince TheLittlePrince.txt
  prompt> grep -i Prince TheLittlePrince.txt
  ... [A lot]
  ```
- `wc`: count lines, words, characters
  ```
  prompt> wc TheLittlePrince.txt
  2582 17297 91651 TheLittlePrince.txt
  ```
  lines words characters

UNIX Philosophy (continued) - Pipes

- |: "pipe" (the output of the program at the left of the pipe becomes the input of the program at the right of the pipe)
  ```
  prompt> grep -i Prince TheLittlePrince.txt | wc -l
  195
  ```
- and more...
  ```
  cat TheLittlePrince.txt | sed 's/becuase/because/g'
  ```

Jobs

- Commands can be executed in the *background* thanks to the & symbol.
  `find / -type f | wc -l > /tmp/find.stdout 2> /tmp/find.stderr &`
- The running command is called a *job*
- `jobs` is the command to look at running jobs
- Jobs can be accessed as %1, %2, ...
- `fg %2` brings the job #2 to the *foreground*
- If a job is already running, hitting CTRL-Z suspends the job and gives it a job id
- `bg %4` resumes the suspended job in the background
- `kill %7` attempts to terminate the job #7 gracefully
- `kill -9 %7` kills the job unconditionally

Environment Variables

- `printenv` to get the list of environment variables;
- Sometimes you'll have to set/modify environment variables (PATH) (and you can mess things up badly);
- Setting a new environment variable (or overwriting another one):
  `export NEWTHING="a:b/c"`
- Adding to an environment variable:
  `export NEWTHING="$NEWTHING hello"`

Shell Customization

- You are supposed to always know what you are doing
  `cp`, `rsync` can be useful for backups
- `chsh`: Change the shell
- Edit `.bashrc` and/or `.bash_profile` to add/modify aliases, environment variables...
  `export PATH=$PATH:$HOME/bin`
  — *Tip: Always add the old path (unless you know what you are doing)*
  (Setting `export PATH=foo` is a bad idea... Why?)
- `source $HOME/.bashrc` is faster than logging out and back in.

Tricks and Tips

- Tab completion
- Up-arrow
- `history` and `!<something>`, `!ls`, `!!`, `!23`
- CTRL-R
- ... and others ...

sudo / System Updates/System Upgrades

- `sudo` is dangerous (With great power comes great responsibility);
- Every now and then the system will tell you something like:

```
224 packages can be updated.
130 updates are security updates.

New release '16.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
```

- DO NOT `do-release-upgrade` your Virtual Machine during the semester

Conclusion. Limits

- We've barely scratched the surface...
- The CLI is very powerful
  - But there are things it can't do (e.g., floating-point numbers)
- Knowing a scripting language is a good idea for your future (bash, Python, Ruby?, Perl?...)
  - Allows you to avoid tedious by-hand work (e.g., see the first "programming assignment")
    - There is a Scripting Languages course (ICS215)