Introductory LINUX/command line training

Steinchneider Research Group, Cornell University

Motivation

In sort of priority order:

- **Only** way to interface with HPC resources, either locally (e.g. Hopper) or external (e.g. ACCESS) [HPC training]
- Easiest way to interact with Git repositories, although GUI methods exist [Git training]
- Many spatial geophysical datasets (e.g. reanalysis, gridded hydrology, forecasts, etc) most easily accessed/processed through command line tools [Spatial data training]
- Task automation through shell scripting

Objectives

- Familiarize with basic LINUX terminology and derivations
- Familiarize with basic LINUX shell command structure and commands
- Introduce some LINUX shell operations useful for later training (HPC, git, spatial data)
- Introduce some useful resources, motivate the utility of LINUX proficiency, pave the way for self study

Overview

- Terminology
- Basic command line construction and commands
- Slightly more advanced command line
- Automation example for R
- Shell scripting
- High level (e.g. R) to shell functionality

Terminology

LINUX: The Linux operating system is an extremely versatile Unix-like operating system, and has taken a clear lead in the High Performance Computing (HPC) and scientific computing community.

There are two major components of Linux, the kernel and the shell:

- The **kernel** is the core of the Linux operating system that schedules processes and interfaces directly with the hardware. It manages system and user I/O, processes, devices, files, and memory.
- The **shell** is a text-only interface to the kernel. Users input commands through the shell, and the kernel receives the tasks from the shell and performs them. The shell tends to do four jobs repeatedly: display a prompt, read a command, process the given command, then execute the command. After which it starts the process all over again.

It is important to note that users of a Linux system typically *do not* interact with the kernel directly. Rather, most user interaction is done through the shell or a desktop environment.

Demystifying terminology…maybe

So Linux is basically the newer, open source version of UNIX that allows us to interface somewhat directly with the computer's operating system. But what is the 'shell', 'command line', 'bash', 'bash scripting'?

In a nutshell, these are terms for how you interact with LINUX/UNIX kernel and tell the computer what you want it to do:

- 'shell' is the most basic form; **shells** interpret commands for the operating system to execute
- 'bash' (Bourne again shell) is a **specific form** of shell very commonly used
- 'command line' or 'command line interface' is a more generic descriptor for something that interacts with the computer's operating system
- 'shell scripting' is simply the automation of a bunch of shell tasks, just like one would automate a bunch of high-level language (R, Python) with a script

As a user (ie not a software engineer or computer scientist), these nuanced differences really aren't that important; it's just useful to know that all these terms refer to roughly the same thing in many contexts…I will probably use these terms somewhat interchangeably going forward

Command line operations on non-LINUX OS

For non-LINUX operating systems (Windows PC), there are a number of LINUX-like interfaces to the OS (powershell, Cygwin, R-terminal, etc).

These interfaces often allow command line tasks in line with LINUX/bash conventions. However, they can be fairly limited in their LINUX functionality and are often unable to execute higher level command line executables/packages.

For instance, many command line tools for spatial data processing (CDO, NCO) can be challenging to get to work on PCs, but will work great on Mac/Ubuntu as well as LINUX based HPCs

Construction of shell command (LINUX/bash)

Note: these examples largely follow the Cornell CAC Linux tutorial

Structure of a command:

```
§ <command> <option(s)> <argument(s)>
```

command – the executable to be run

option(s) – (aka 'flags'), optional arguments that alter command behavior (some commands have them, some don't)

arguments – depend on the command; one common example of an argument is 'file.in file.out'

*bash can be case sensitive and is also sensitive to spaces

**common convention is '$' to refer to a bash command and '>' to refer to a higher level language command (R, Python)

Getting info on a bash command

To get more information on a s d
can be used in actual LINUX ma

```
man <program or command>
```

LINUX-like interpreters may use nd:

$ mv –help

Help pages will include the follc

1. **NAME** – a one-line description of what it does.
2. **SYNOPSIS** – basic syntax for the command line.
3. **DESCRIPTION** – describes the program's functic
4. **OPTIONS** – lists command line options that are a
5. **EXAMPLES** – examples of some of the options a
6. **SEE ALSO** – list of related commands.

```
$ mv --help
Usage: mv [OPTION]... [-T] SOURCE DEST
  or:  mv [OPTION]... SOURCE... DIRECTORY
  or:  mv [OPTION]... -t DIRECTORY SOURCE...
Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.

Mandatory arguments to long options are mandatory for short options too.
      --backup[=CONTROL]       make a backup of each existing destination file
  -b                           like --backup but does not accept an argument
  -f, --force                  do not prompt before overwriting
  -i, --interactive            prompt before overwrite
  -n, --no-clobber             do not overwrite an existing file
If you specify more than one of -i, -f, -n, only the final one takes effect.
      --strip-trailing-slashes  remove any trailing slashes from each SOURCE
                                  argument
  -S, --suffix=SUFFIX          override the usual backup suffix
  -t, --target-directory=DIRECTORY  move all SOURCE arguments into DIRECTORY
  -T, --no-target-directory    treat DEST as a normal file
  -u, --update                 move only when the SOURCE file is newer
                                 than the destination file or when the
                                 destination file is missing
  -v, --verbose                explain what is being done
  -Z, --context                set SELinux security context of destination
                                 file to default type
      --help     display this help and exit
      --version  output version information and exit

The backup suffix is '~', unless set with --suffix or SIMPLE_BACKUP_SUFFIX.
The version control method may be selected via the --backup option or through
the VERSION_CONTROL environment variable.  Here are the values:

  none, off       never make backups (even if --backup is given)
  numbered, t     make numbered backups
  existing, nil   numbered if numbered backups exist, simple otherwise
  simple, never   always make simple backups

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation <https://www.gnu.org/software/coreutils/mv>
or available locally via: info '(coreutils) mv invocation'
```

Getting info on a bash command

For the ‘mv’ [move] directory management executable, you get this for instance:

```
$ mv --help
Usage: mv [OPTION]... [-T] SOURCE DEST
  or:  mv [OPTION]... SOURCE... DIRECTORY
  or:  mv [OPTION]... -t DIRECTORY SOURCE...
Rename SOURCE to DEST, or move SOURCE(S) to DIRECTORY.

Mandatory arguments to long options are mandatory for short options too.
      --backup[=CONTROL]       make a backup of each existing destination file
  -b                           like --backup but does not accept an argument
  -f, --force                  do not prompt before overwriting
  -i, --interactive            prompt before overwrite
  -n, --no-clobber             do not overwrite an existing file
If you specify more than one of -i, -f, -n, only the final one takes effect.
      --strip-trailing-slashes  remove any trailing slashes from each SOURCE
                                  argument
  -S, --suffix=SUFFIX          override the usual backup suffix
  -t, --target-directory=DIRECTORY  move all SOURCE arguments into DIRECTORY
  -T, --no-target-directory    treat DEST as a normal file
  -u, --update                 move only when the SOURCE file is newer
                                 than the destination file or when the
                                 destination file is missing
  -v, --verbose                explain what is being done
  -Z, --context                set SELinux security context of destination
                                 file to default type
      --help     display this help and exit
      --version  output version information and exit

The backup suffix is '~', unless set with --suffix or SIMPLE_BACKUP_SUFFIX.
The version control method may be selected via the --backup option or through
the VERSION_CONTROL environment variable.  Here are the values:

  none, off       never make backups (even if --backup is given)
  numbered, t     make numbered backups
  existing, nil   numbered if numbered backups exist, simple otherwise
  simple, never   always make simple backups

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation <https://www.gnu.org/software/coreutils/mv>
or available locally via: info '(coreutils) mv invocation'
```

Other helpful bash tricks

Tab completion: this is a general coding trick, but is very useful in bash.

Command line training exercises

Directory navigation

Resources

- Learning Linux (linkedin.com)
- A very quick intro to Linux (linkedin.com)
- Cornell Virtual Workshop: Overview