IPSL BootCamp: Unix commands

Institute Pierre Simone Laplace, IPSL BootCamp

March 23, 2016

Contents

1 Introduction

Unix is an operating system available on many computers, from PCs to supercomputers.[1]

When you log onto a Unix system, your main interface to the system is called the Unix shell that is a command-line interface allowing you to interact with a computer program via successive lines of text (command-lines). Each command-line starts you with the dollar sign (`$`). One can change this: in some others the prompt ends with `>`. This prompt means that the shell is ready to accept your typed commands. Every user has a unique username. Bash is the default Unix shell on may GNU/Linux distributions (e.g., Debian, Fedora, Ubuntu).

When you logon to the system, you are placed in a home directory, which is a portion of the disk space reserved just for them.

Unix commands are strings of characters typed in at the keyboard. To run a command, you just type it in at the keyboard and press the ENTER key. We will look at several of the most common commands in the following sections.

Unix extends the power of commands by using special flags that are usually preceded by a dash (`-`) and preceed any filenames or other arguments on the command line. Unlike the DOS (or Windows) command line, Unix systems are **case sensitive** (upper and lower case characters are considered different). Nearly all command names and most of their command line switches will be in lowercase.

1.1 Unix architecture

All versions of Unix shares a common architecture that is composed by the following four basics:

- **Kernel**: The kernel is the heart of the operating system. It interacts with hardware and most of the tasks like memory management, tash scheduling and file management.
- **Shell**: The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. *C Shell* (`csh`), *Bourne again Shell* (`bash`) and *Korn Shell* (`ksh`) are the most famous shells which are available with most of the Unix variants.
- **Commands and Utilities**: There are various command and utilities which you would use in your day to day activities. `cp`, `mv`, `cat` and `grep` etc... are a few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various optional options.
- **Users**: Unix system is based on users and privileges. The main user with which one manages the system is called `root` (or *super user* `su`). Then each person who will work in the system has a user with less privileges than `root`. Doing this, any user can damage the system (so, do not panic!). On top of that there are *groups* and *all users* sets of users. With that one can manage who has access to certain files and directories, rights to modify files, etc... `root` is *'the one to rule them all'*
- **Files and Directories**: All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem. Files and directories have assigned `modes`, with which one manages the rights to *execute*, *read* and *modify* of the owner (`user`) of the file, `group` and `all users` in the computer (see further on for a detailed explanation).

[1] *UNIX* (capitalised) is the operating system created by AT&T at Bell Labs in the 1970s. Derivative versions are denoted by *Unix*, but here we'll use that term for any *Unix-like* operating system. Among these Unix-like operating systems one of the most popular is *GNU/Linux*. GNU stands for *GNU's Not Unix*: indeed, it is not Unix as it does not use any of the original codebase of the AT&T UNIX code. It is rather written from scratch with the goal of providing an operating system consisting of only free software. On purpose the GNU system is similar to UNIX and henceforth in this document referred to as *Unix*.

2 Before starting

2.1 Common vocabulary

Common language and basic knowledge to navigate through Unix commands:

- **Command-line**: Where you actually type your commands.
- **Command-prompt**: At the beginning of the command-line, the command-prompt looks like: `user@computer:` It indicates that the computer is ready to accept commands and provides useful information when working with multiple remote computers at the same time.
- **Hidden files**: Files whose first character is the dot or period character (.). Unix programs (including the shell) use most of these files to store configuration information. Common hidden files:

  `.bash_profile` the Bash initialization script (login shells);

  `.bashrc` the per-interactive-shell startup script.
- **Meta characters**: Meta characters have special meaning in Unix. For example * and ? are metacharacters.

  `*` matches 0 or more characters;

  `?` matches a single character.
- **Command structure**: `command {-option} target`

  Many commands require neither flags nor target.
- **Command manual**: `man command1` displays on-line reference manual pages, in the example for `command1`:

  `man -k keyword` → List the manual page subjects that have keyword in their headings. This is useful if you do not yet the name of a command you are looking for.

2.2 Useful keyboard short-cuts

Keyboard short-cuts to move around in your command-line:

Control-a → Move to beginning of line;

Control-e → Move to end of line;

Alt-f → Move forward one word;

Alt-b → Move backwards one word;

Control-l → Clear, leaving current line;

Tab → Try to automatically complete path;

Control-u → Cancel the whole (e.g., if you made a typo);

Up & down arrows → Move back (↑) and forward (↓) in the history of commands you typed, or use the command `history` that lists previously executed commands.

3 Get to know your (digital) self and your environment

3.1 Basic information

Basic commands to better know your environment and answer some important questions:

- `whoami` → This returns your `username`. You may need to you have logged out your `username`;
- `hostname` → Name of the machine you are working on;
- `pwd` → Current directory, where you are typing the command;
- `uname` → Get complete information on your Operating System, use option `-a`: `uname -a`;
- `passwd` → Change your password, which you should do regularly (at least once a year).

3.2 Estimate your quota

- `df` → Summarize free space on disk filesystems

  `df -h` → Print free space in human readable format (e.g., 1K, 234M, 2G, etc.) (flag `-h`).

- `du` → Show disk space used by files or directories (without argument the current directory is analyzed);

  `du -hs dirname` → Show disk space used by `dirname` in human readable format and without specifying disk space used by all sub-directories (flag `-s`).

- `quota` → Show what your disk quota is (i.e. how much space you have to store files), how much you're actually using, and in case you've exceeded your quota (which you'll be given an automatic warning about by the system) how much time you have left to sort them out (by deleting or compressing some, or moving them to your own computer).

4 Directories and files

4.1 Special directories

`/` Root directory;

`.` Current directory (where you are actually working/typing);

.. Parent directory;

∼ Home directory (as `$HOME`);

∼**user** User home directory.

4.2 Handling directories and files

4.2.1 Directories

Directories, like folders on Windows, are used to group files together in a hierarchical structure.

- `mkdir dirname` → Make a new directory;

- `cd dirname` → Change directory and move in `dirname`. `dirname` may be either the full pathname of the directory, or its pathname relative to the current directory.

    `cd` or `cd ~` → Go straight to your home directory;

    `cd ..` → Get one level up from your current directory.

- `ls` → List name of files and directories in your current directory:

    `ls -F` → Indicate sub-directories by appending a slash (`/`) to their name (flag `-F`);

    `ls -t` → Sort the list of files by modification time;

    `ls -h` → Print file/directory sizes in human readable format (e.g., 1K, 234M, 2G, etc.).

    `ls dirname` → List name of the files and directories in the directory `dirname`, excluding hidden files whose names begin with `.`

    `ls -a dirname` → List the contents of `dirname`, including files whose names begin with `.` (flag `-a`);

    `ls -l dirname` → Give details of the access permissions for the directory called `dirname` (flag `-l`) (see next section);

    `ls -l filename` → Same as before, to get details on a file called `filename`;

4.2.2 File permission setup

File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes:

- **Owner permissions**: The owner's permissions determine what actions the owner of the file can perform on the file.

- **Group permissions**: The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.

- **Other (world) permissions**: The permissions for others indicate what action all other users can perform on the file.

Using `ls -l` command, various information related to file permission are shown:

```
$ ls -l /home/amrood
-rwxr-xr- 1 amrood users 1024 Nov 2 00:10 myfile
drwxr-xr-- 1 amrood users 1024 Nov 2 00:10 mydir
```

Here first column represents different access mode, i.e. permission associated with a file (`-` as first character in the first column) or directory (`d` as first character in the first column). The permissions are broken into triples, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x):

- **Owner** The first three characters (2-4) represent the permissions for the file's owner. For example, `-rwxr-xr--` represents that onwer has read (r), write (w) and execute (x) permission.

- **Group** The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, `-rwxr-xr--` represents that group has read (r) and execute (x) permission but no write permission.

- **Other** The last group of three characters (8-10) represents the permissions for everyone else. For example, `-rwxr-xr--` represents that other world has read (r) only permission.

File Access Modes: The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which are described below:

1. Read: Grants the capability to view the contents of files.
2. Write: Grants the capability to modify, or remove the content of the file.
3. Execute: Grants the capability to run files as programs.

Directory Access Modes:

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned:

1. Read: The user can read the contents and can look at the filenames inside the directory.
2. Write: The user can add or delete files to the contents of the directory.
3. Execute: The user can enter the directory (`cd`).

The execute bit clearly has a different meaning than that of a normal file. Try this:

```
$ mkdir -p test0/test1                        create test0, and test1 within test0
$ chmod 000 test0                             lock down test0 for all users
$ cd test0/test1/                             try to get into test1
cd:  test0/test1/:  Permission denied
$ chmod u+x test0                             set only executable bit (for owner)
$ ls test0                                    try to read the content of test0
ls:  can't open test0:  Permission denied
$ cd test0/test1/                             try to get into test0/test1: success!
$ pwd                                         see in which directory you are
/my/path/to/test0/test1
```

Change permission

`chmod {options} filename` → Change the read, write, and execute permissions on your files.

`chmod o+r filename` → Make the file readable (flag `r`) for everyone (flag `o`). The *plus* symbol adds the named right.

`chmod o-r filename` → Make it unreadable for others again. The *minus* symbol removes the named right.

`chmod 755 filename` → Using numerical format the three number give permissions to, in order: the owner, the group, and others. Number 7 corresponds to all permission (read, write and execute), while 5 provides only the rights to read and execute (not to write). See at the following link, for more information: https://en.wikipedia.org/wiki/Chmod.

4.2.3 Files

About filenames in Unix: A filename in Unix can consist of any combination of characters on the keyboard except for the null character and slash (`/`) as these are control characters. Moreover, on modern Unices typically any Unicode character is allowed. You are adviced not to use the following characters in filenames: `* ? ! | \ / ’ " { } < > ; , ˆ ( ) $ ˜`. These characters can be used in filenames, but only by escaping (prefixing with a backslash) or quoting them because they have special meaning to the shell. On Unix systems, the extension of a filename (e.g.: `*.pdf`, `*.png`, `*.txt`) are part of the filename. This does not need to to define the type of file; for that use file(1):

```
$ ls -l
total 105632
-rw-r-r- 1 mvhulten lsce 1040 Mar 23 13:17 README
-rw-r-r- 1 mvhulten lsce 108092785 Mar 23 13:13 rms_gplv3_launch.ogg
-rw-r-r- 1 mvhulten lsce 40541 Mar 23 13:12 sound.ogg
-rw-r-r- 1 mvhulten lsce 26398 Mar 23 13:18 unix.tex
$ file *
README: ASCII text
rms_gplv3_launch.ogg:  Ogg data, Theora video
sound.ogg:  Ogg data, Vorbis audio, stereo, 48000 Hz,  192000 bps
unix.tex:  LaTeX 2e document, UTF-8 Unicode text, with very long lines
```

When using `ls -l` you only see the permission, size and date of change of the files. The extension does not help you much in identifying the filetype: `README` does not have an extension, and, while in this case both `.ogg` files contain Ogg data, you cannot see if they contain audio or video streams. Only when using `file FILES` you see the type of file. Only now you are comfortable executing these commands:

`$ cat README`	show content of the text file with cat(1)
`$ mplayer rms_gplv3_launch.ogg`	play the video with mplayer(1)
`$ mplayer sound.ogg`	play the audio with mplayer(1)
`$ ogg123 sound.ogg`	play the audio with ogg123(1)
`$ vim unix.tex`	edit the text file, which is in fact a LaTeX file

For text (e.g. ASCII or Unicode) files, to get a quick view of the file content

- `cat filename` → Display the content of a file

  `cat -b filename` → Flag `-b`, display line numbers in the file;

- `more filename` → Show the first part of a file, just as much as will fit on one screen. To see more, just hit the space bar. To left, type `q`. To search for a pattern, use `/pattern`;

- `head filename` or `tail filename` → show the first or last ten lines;

- `wc filename` → Count how many lines, words, and characters there are in a file. The command output lists, in order, the number of lines, words, and characters. To restrict the count to: (1) lines, use flag `-l`, (2) words, use flag `-w`, or (3) characters, use flag `-c`.

To handle files:

- `cp filename1 filename2` → Copy the content of `filename1` into `filename2`. `cp` cannot copy a file onto itself.

`cp filename1 filename2 dirname1` → Create copies of `filename1` and `filename2` (with the same names), within the directory `dirname1`, which must already exist for the copying to succeed.

`cp -r dirname1 dirname2` → Recursively copy the directory `dirname1`, together with its contents and subdirectories, to the directory `dirname2` (flag `-r`). If `dirname2` does not already exist, it is created by `cp`, and the contents and subdirectories of `dirname1` are recreated within it. If `dirname2` does exist, a subdirectory called `dirname1` is created within it, containing a copy of all the contents of the original `dirname1`. `dirname1` and `dirname2` can be either directories or full pathnames of directories.

- `mv filename dirname/` → Move a file into a specified directory. `dirname` can be either a directory or a full pathname of the directory

  `mv filename1 filename2` → Rename a file.

- `rm -i filename` → Remove a file. It is wise to use the flag `-i`, which will ask you for confirmation before actually deleting anything.

  `rm -ri dirname` → Remove a whole directory;

  `rm -f filename` → The flag `- f` force the remove: there's no possibility to go back!

- `diff filename1 filename2` → Compare files and show where they differ

  `diff filename1 filename2 > diff.txt` → Print differences between the two files in a text file, named `diff.txt`. The symbol `>` redirects the output of a command in a file, instead of showing results on screen.

4.3 Finding things

- `whereis filename` → Tries to locate binaries (and on GNU systems also their sources and man pages).
- `find` → Search for files in a named directory and all its subdirectories:

  `find . -name *.f -print` → Look for all files ending with `.f` in the current directory and all its subdirectories, and write their names to the standard output (i.e., on screen);

  `find /local -name core -user user1 -print` → Search the directory `/local` and its subdirectories for files called `core` belonging to the user `user1` and writes their full file names to the standard output.

- `grep` → Search for lines in files containing a specified pattern/string and, by default, writes them to the standard output. `grep` is CASE sensitive. This can be useful: e.g. finding the right file among many, figuring out which is the right version of something. `grep` has a lot of very flexible options.

  `grep "motif" filename1` → Search for the string `motif` in `filename1`;

  `grep -i "motif" *` → Search for the string `motif` in all files available in the current directory; using the flag `i`, `grep` will look for the specified pattern in lower and upper case.

5 Compress: zip, gzip and tar

Common tools to compress/uncompress files:

- **zip**

  - `zip filename` → Compress and produce files with the suffix `.zip` appended to the original filename;
  - `unzip filename.zip` → Uncompress a file `.zip`.

- **gzip** usually gives a higher compression rate than zip
  - `gzip filename` → Compress and produce files with the suffix `.gz` appended to the original filename;
  - `gunzip filename.gz` → Uncompress a file `.gz`.

Compression works better if files are combined and then compressed together, rather than compressing them individually, since this allows the compression program to spot repeated patterns between the files. For this need, you should use the Unix tool for packing and unpacking files and create archives: **`tar`**.

Pack into a single `tar` file, all files ending with the suffix `.txt`:

```
tar -cvf pack.tar *.txt
```

Flags: `c` means create, `v` shows on screen files that are packed, `f` means that the filename (`pack.tar`) to write to is specified. Then you may use gzip(1) to compress the *tar ball*: `gzip pack.tar` To pack and gzip sequences simultaneously, add flag `z` to the `tar` command:

```
tar -zcvf pack.tgz *.txt
```

`.tgz` is just short for `.tar.gz`. To extract use flag `x`:

```
tar -xf pack.tar
tar -zxf pack.tar.gz
```

To list the contents without extracting, use flag `t`:

```
tar -ztf sequences.tgz
```

To extract a single file from your `tar`:

```
tar -xf pack.tar filename
```

or, using long-opts:

```
tar --extract --file={pack.tar} {filename}
```

6 Working on remote computers

6.1 ssh - Secure SHell

Secure SHell is an encrypted network protocol allowing secure remote login and other network services even over an unsecured network. To connect to a host (i.e., remote machine):

```
ssh user@anothermachine.org
```

You are prompted for a password and then have a command-line that looks like: `user@anothermachine:`
To leave the connection, type: `exit`
To enable X11 window to open, use the `-X` flag:

```
ssh -X user@anothermachine.org
```

This allows you to start a graphical application; the application that appears on your computer is actually running on `anothermachine` not your local computer.

6.2 Getting files from a external machine

6.2.1 scp - Secure CoPy

Copying file `filename` to host `anothermachine.org`, into a specific directory `~user/examples/`.

```
scp filename.txt user@anothermachine.org:~user/examples/ {filename_copy.txt}
```

Paths on the host could also be specified absolutely, like:

```
user@anothermachine.org:/home/user/examples 23
```

Copying file from host to your current directory (`.`):

```
scp user@anothermachine.org:~user/examples/filename_copy.txt .
```

6.2.2 rsync

This command is very handy to synchronize files and directories between two different machines. Its use is very similar to that of `scp`

```
rsync [flags] user@sourcemachine.org:[source files] user@destinymachine.org:[location]
```

`rsync` does not change metadata of the files (date of creation or last modification). By default rsync(1) overwrites any existing files in the destination directory. If you supply the `--update` with rsync(1), it only copies those files and directories that have changed if it finds the same files on the destiny machine.

6.2.3 ftp

The *'File Transfer Protocol'* is the basic way to transfer files within the internet. It is based on server/client infrastructure. When one is connected with `ftp` to a server, a session is opened there mainly to navigate within the disk space (commonly a dedicated space into the server), copy (via `get`) and upload (via `mput`) files among a full standard set of file related instructions.

See https://en.wikipedia.org/wiki/List_of_FTP_commands for a full set of commands.

6.2.4 wget

This command is commonly used to get from a file from the internet. It supports `http`, `https` and `ftp`. It can resume a failed file retrieval, and use wildcards among other things. Its syntax is at follows:

```
wget internetsourcelink
```

7 Packages

Unix system distributions are based on a series of packages and applications. These can be installed in the system. Assuming you run a Debian- or Redhat-based GNU distribution, they are managed via one of two main applications: `apt` or `yum`. Commonly only `root` manages the applications installed in the machines. Because in almost all the IPSL laboratories there are dedicated computer manager technicians, we will not extend more on this point.

Some of the most popular applications:

- image treatment/creation: convert (list of options: http://www.imagemagick.org/script/command-line-opt php), gimp, inkscape, xpaint
- image display: display, xpdf/evince
- text: LaTeX, libreoffice
- OfficeSuite: libreoffice
- graphics: ferret, gnuplot, grads, NCL, python

8 A few links

- https://swcarpentry.github.io/shell-novice/reference.html
- http://juliend.github.io/linux-cheatsheet/ (in French)

Unix/Linux Command Reference

FOSSwire.com

File Commands

ls – directory listing

ls -al – formatted listing with hidden files

cd ***dir*** - change directory to *dir*

cd – change to home

pwd – show current directory

mkdir ***dir*** – create a directory *dir*

rm ***file*** – delete *file*

rm -r ***dir*** – delete directory *dir*

rm -f ***file*** – force remove *file*

rm -rf ***dir*** – force remove directory *dir* *

cp ***file1 file2*** – copy *file1* to *file2*

cp -r ***dir1 dir2*** – copy *dir1* to *dir2*; create *dir2* if it doesn't exist

mv ***file1 file2*** – rename or move *file1* to *file2* if *file2* is an existing directory, moves *file1* into directory *file2*

ln -s ***file link*** – create symbolic link *link* to *file*

touch ***file*** – create or update *file*

cat > ***file*** – places standard input into *file*

more ***file*** – output the contents of *file*

head ***file*** – output the first 10 lines of *file*

tail ***file*** – output the last 10 lines of *file*

tail -f ***file*** – output the contents of *file* as it grows, starting with the last 10 lines

Process Management

ps – display your currently active processes

top – display all running processes

kill ***pid*** – kill process id *pid*

killall ***proc*** – kill all processes named *proc* *

bg – lists stopped or background jobs; resume a stopped job in the background

fg – brings the most recent job to foreground

fg ***n*** – brings job *n* to the foreground

File Permissions

chmod ***octal file*** – change the permissions of *file* to *octal*, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:

chmod 777 – read, write, execute for all

chmod 755 – rwx for owner, rx for group and world

For more options, see **man chmod**.

SSH

ssh ***user@host*** – connect to *host* as *user*

ssh -p ***port user@host*** – connect to *host* on port *port* as *user*

ssh-copy-id ***user@host*** – add your key to *host* for *user* to enable a keyed or passwordless login

Searching

grep ***pattern files*** – search for *pattern* in *files*

grep -r ***pattern dir*** – search recursively for *pattern* in *dir*

command **| grep** ***pattern*** – search for *pattern* in the output of *command*

locate ***file*** – find all instances of *file*

System Info

date – show the current date and time

cal – show this month's calendar

uptime – show current uptime

w – display who is online

whoami – who you are logged in as

finger ***user*** – display information about *user*

uname -a – show kernel information

cat /proc/cpuinfo – cpu information

cat /proc/meminfo – memory information

man ***command*** – show the manual for *command*

df – show disk usage

du – show directory space usage

free – show memory and swap usage

whereis ***app*** – show possible locations of *app*

which ***app*** – show which *app* will be run by default

Compression

tar cf ***file.tar files*** – create a tar named *file.tar* containing *files*

tar xf ***file.tar*** – extract the files from *file.tar*

tar czf ***file.tar.gz files*** – create a tar with Gzip compression

tar xzf ***file.tar.gz*** – extract a tar using Gzip

tar cjf ***file.tar.bz2*** – create a tar with Bzip2 compression

tar xjf ***file.tar.bz2*** – extract a tar using Bzip2

gzip ***file*** – compresses *file* and renames it to *file.gz*

gzip -d ***file.gz*** – decompresses *file.gz* back to *file*

Network

ping ***host*** – ping *host* and output results

whois ***domain*** – get whois information for *domain*

dig ***domain*** – get DNS information for *domain*

dig -x ***host*** – reverse lookup *host*

wget ***file*** – download *file*

wget -c ***file*** – continue a stopped download

Installation

Install from source:

./configure

make

make install

dpkg -i ***pkg.deb*** – install a package (Debian)

rpm -Uvh ***pkg.rpm*** – install a package (RPM)

Shortcuts

Ctrl+C – halts the current command

Ctrl+Z – stops the current command, resume with **fg** in the foreground or **bg** in the background

Ctrl+D – log out of current session, similar to **exit**

Ctrl+W – erases one word in the current line

Ctrl+U – erases the whole line

Ctrl+R – type to bring up a recent command

!! - repeats the last command

exit – log out of current session

* use with extreme caution.