

OS and Basic Utilities

(Lectures on High-performance Computing for Economists III)

Jesús Fernández-Villaverde,¹ Pablo Guerrón,² and David Zarruk Valencia³

October 22, 2018

¹University of Pennsylvania

²Boston College

³ITAM

Operating Systems

Operating systems

- If you are going to undertake some serious computation, you want to become a skilled user of your OS.
- High rate of return to moderate time investments.
- Two main families of OS:
 1. Unix and Unix-like family (Ken Thompson and collaborators at Bell Labs):
 - 1.1 Commercial versions: AIX, HP-UX, Solaris, ...
 - 1.2 Open source: OpenBSD, Linux, ...
 - 1.3 macOS.
 2. Windows family.

Why Unix/Linux? I

- Industry-tested for four decades: powerful beyond your imagination.
- Standard OS for scientific computation and high-performance computing → as of June 2018, ALL the Top 500 supercomputers in the world run on Linux.
- Particularly important for:
 1. Access to servers.
 2. Web services such as AWS.
 3. Parallelization
- It will be around forever: if you learned to use Unix in 1973, you can open a Mac today and use its terminal without problems.
- Watch <https://youtu.be/tc4R0CJYbm0>.

Why Unix/Linux? II

- Many (Linux) open source implementations. For example, Ubuntu and Fedora. You can check <https://www.distrowatch.com/>
- Much more robust: small kernel.
- Much safer: sandboxing and rich file permission system.
- Easier to port code.
- Plenty of tools.
- For instance, a default macOS installation comes with Emacs, VI, SHH, GCC, Python, Perl,....
- Existence of Windows emulators such as VMWare or Parallels.
- Converse is not true: Cygwin is still not Unix.

Philosophy of Unix/Linux

- “Building blocks” + “glue” (pipes and filters) to build new tools:
 1. Building blocks: programs that do only one thing, but they do it well.
 2. Glue: you can easily combine them.
- Ability to handle Generalized Regular Expressions:

Ken Thompson

A regular expression is a pattern which specifies a set of strings of characters; it is said to match certain strings.

1. `grep` and `awk`: searches plain-text data sets for lines that match a regular expression (although nowadays you can use text processing packages in your favorite programming language).
2. `Flex` (fast lexical analyzer) and `Bison` (general-purpose parser generator).

- Both GUIs and command lines.
- GUIs are useful for routine operations:
 1. X11.
 2. GNOME.
 3. Aqua.
- Command line ends up being much more powerful and adaptable than GUIs.
- The command line works through a shell.

Shell

- Different shells: `bash` (Bourne-again shell, by Brian Fox), `bourne`, `ksh`,...
- For instance, if you type

In [1]: `echo $0`

on a Mac Terminal, you will probably get:

Out[1]: `-bash`

- Easy to change shells.
- Most of them offer similar capabilities, but `bash` is the most popular.
- Basic tutorial: <http://swcarpentry.github.io/shell-novice/>



Some basic instructions I

To check present working directory:

```
$ pwd
```

To list directories and files:

```
$ ls
```

To list all directories and files, including hidden ones:

```
$ ls -all
```

To navigate into directory myDirectory:

```
$ cd myDirectory
```

To go back:

```
$ cd ..
```

Some basic instructions II

To create a directory

```
$ mkdir myDirectory
```

To remove a directory

```
$ rmdir myDirectory
```

To copy myFile

```
$ cp myFile
```

To move myFile to yourFile:

```
$ mv myFile yourFile
```

To remove myFile:

```
$ rm myFile
```

Some basic instructions III

To find myFile:

```
$ find myFile
```

To concatenate and print myFile:

```
$ cat myFile
```

Wild card:

```
$ ls myF*
```

Manual entries

```
$ man
```

Bang

```
$ !!
```

Some basic instructions IV

To check permissions on myFile:

```
$ ls -l myFile
```

To change permissions on myFile:

```
$ chmod 744 myFile
```

Interpretation digit:

- 4: read access.
- 2: write access.
- 1: execute access.

Interpretation position:

- first: user access.
- second: group access.
- third: world access.

Advanced shell interaction

- Customization: `.bash_profile`, `.bash_logout`, and `.bashrc` files.
- `awk` programming language.
- Shell programming:
 1. Automatization.
 2. Aliases.

```
$ alias myproject = ' ~/dropbox/computational_economics_course/  
figures'
```

Some more information

- Good references (among many):
 1. *Unix in a Nutshell (4th Edition)*, by Arnold Robbins.
 2. *Learning Unix for OS X: Going Deep With the Terminal and Shell (2nd Edition)*, by Dave Taylor.
 3. *A Practical Guide to Linux Commands, Editors, and Shell Programming (4th Edition)*, by Mark G. Sobell.
 4. *Learning the bash Shell: Unix Shell Programming (3rd Edition)*, by Cameron Newham and Bill Rosenblatt.

Package Manager

Package manager

- Get yourself a good package manager.
- Update your software in an efficient way.
- Particularly important for open source projects.
- I use:
 1. MacPorts: <http://www.macports.org/>
 2. GUI: Pallet.

Editors

- By default, you should try to use plain, open files:
 1. Text files (READMEs, HOWTOs, ...).
 2. CSV files for data (also as text files).
- .docx and .xlsx files change over time and may not be portable.
- A good editor is an excellent way to write text files.
- A good editor will also help you write source code (with syntax highlight) and tex files.

Alternatives I

Harry J. Paarsch

Choose your editor with more care than you would your spouse because you will spend more time with your editor, even after the spouse is gone.

- Classics:

1. Emacs, originally by Richard Stallman (with many variants: I use Aquamacs).
2. VI.
3. Textwrangler.
4. Notepad++.
5. JEdit.
6. Nano/Pico (simplest, when I am in a real hurry).



Alternatives II

- New generation:

1. Atom (my usual choice).

2. Sublime.

3. Light Table.

- Characteristics:

1. Web-based programming platform targeting customizability.

2. Use modern languages to implement the editor itself.

3. Open-source communities of 3rd party plugins (Atom has over 7,924 packages available).

IDEs

- Integrated Developer Environment: tools to write, compile, debug, run, and version control code.
- Advantages and disadvantages.
- Standard choices:
 1. JetBrains (CLion, PyCharm,...).
 2. Xcode.
 3. VisualStudio.
 4. Eclipse (with Parallel Application Developers package).
 5. NetBeans.

- Specific languages:

1. Spyder.

2. JuliaPro.

3. RStudio.

4. Matlab IDE.

5. Wolfram Workbench.

Version Control

Challenge

- Projects nearly always end up involving many versions of code (even of your tex files).
- Version control is the management of changes to your code or documents.
- This is important:
 1. When you are working yourself, to keep track of changes, and to be able to return to previous versions.
 2. When you are working with coauthors, to coordinate task and ensure that all authors have the right version of the file.
- Hard to emphasize how important this is in real life: when, why, and how you did it?

Simple solution

- Poor man solutions:
 1. Indexing files by version (mytextfile_1, mytextfile_July212018), with major and minor patches (x.y.z), e.g., 0.1.7
 2. Having a VCS folder (for Version Control System).
 3. Dropbox or similar services.
 4. Automatic back-up software (Time Machine for Mac, fwbackups for Linux).
- While 1-4 are useful, they are not good enough to handle complex projects.
- Nevertheless, **SET UP** automatic backups.

"FINAL".doc



FINAL.doc!



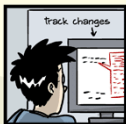
FINAL_rev.2.doc



FINAL_rev.6.COMMENTS.doc



FINAL_rev.8.comments5.
CORRECTIONS.doc



FINAL_rev.18.comments7.
corrections9.MORE.30.doc



FINAL_rev.22.comments49.
corrections.10. #@\$%WHYDID
ICOMETOGRADSCHOOL????.doc

Version control software

- Alternative? version control software (open source):
 1. First generation: RCS.
 2. Second generation: CVS, Subversion.
 3. Third generation: Mercurial, GIT.
- Components:
 1. Repository: place where the files are stored.
 2. Checking out: getting a file from the repository.
 3. Checking in or committing: placing a file back into the repository.

- Standard procedure:
 1. You check out a file from the repository.
 2. You work on it.
 3. You put it back with (optional) some comments about your changes.
 4. The software keeps track of the changes (including different branches) and allows you to recover old versions.
- Version control software is only as good as your own self-discipline.

- Modern, distributed version control system.
- Developed by Linus Torvalds and Junio Hamano.
- Simple and lightweight, yet extremely powerful.
- Easy to learn basic skills.
- Originally designed for command line instructions, but now several good GUIs: Sourcetree.

- Very popular: <http://www.github.com>
- Also, <https://about.gitlab.com/>
- A good reference: *Pro Git* by Scott Chacon, <http://git-scm.com/book>.
- Many tutorials online.
- Integrated with JuliaPro and RStudio and can easily integrate with Atom and standard IDEs.

Dynamic notebooks

- Why?
- Jupyter: <http://jupyter.org/>. Also, JupyterLab.
- Markdown: <https://www.markdownguide.org/>.
- If you work in R:
 1. Knitr package: <https://yihui.name/knitr/>.
 2. *Dynamic Documents with R and knitr (2nd ed.)* by Yihui Xie.
- If you work in Atom: <https://github.com/nteract/hydrogen>.
- Pandoc: <http://pandoc.org/>

Notebook components

