OS and Basic Utilities

(Lectures on High-performance Computing for Economists III)

Jesús Fernández-Villaverde[1] and Pablo Guerrón[2]

January 15, 2022

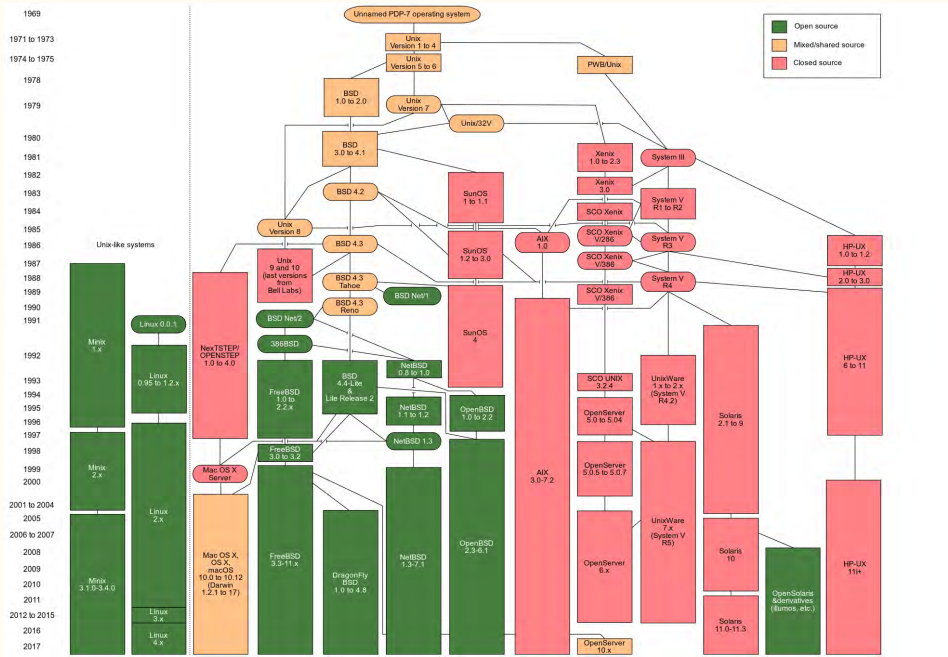
[1]University of Pennsylvania

[2]Boston College

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.


1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2007
2008
2009
2010
2011
2012 to 2015
2016
2017
Open source
Mixed/shared source
Closed source
Unix-like systems
Unnamed PDP-7 operating system
Unix Version 1 to 4
Unix Version 5 to 6
Unix Version 7
PWB/Unix
BSD 1.0 to 2.0
Unix/32V
BSD 3.0 to 4.1
Xenix 1.0 to 2.3
System III
Xenix 3.0
BSD 4.2
SunOS 1 to 1.1
System V R1 to R2
SCO Xenix
Unix Version 8
SCO Xenix V/286
System V R3
AIX 1.0
HP-UX 1.0 to 1.2
BSD 4.3
SunOS 1.2 to 3.0
SCO Xenix V/386
HP-UX 2.0 to 3.0
Unix 9 and 10 (last versions from Bell Labs)
BSD 4.3 Tahoe
BSD Net/1
System V R4
BSD 4.3 Reno
Linux 0.0.1
BSD Net/2
SunOS 4
HP-UX 6 to 11
386BSD
NexTSTEP/ OPENSTEP 1.0 to 4.0
NetBSD 0.8 to 1.0
Minix 1.x
Linux 0.95 to 1.2.x
FreeBSD 1.0 to 2.2.x
BSD 4.4-Lite & Lite Release 2
SCO UNIX 3.2.4
UnixWare 1.x to 2.x (System V R4.2)
NetBSD 1.1 to 1.2
OpenBSD 1.0 to 2.2
OpenServer 5.0 to 5.04
Solaris 2.1 to 9
NetBSD 1.3
FreeBSD 3.0 to 3.2
OpenServer 5.0.5 to 5.0.7
Minix 2.x
Mac OS X Server
AIX 3.0-7.2
Linux 2.x
FreeBSD 3.3-11.x
DragonFly BSD 1.0 to 4.8
NetBSD 1.3-7.1
OpenBSD 2.3-6.1
UnixWare 7.x (System V R5)
OpenServer 6.x
Solaris 10
HP-UX 11i+
Minix 3.1.0-3.4.0
Mac OS X, OS X, macOS 10.0 to 10.12 (Darwin 1.2.1 to 17)
OpenSolaris &derivatives (illumos, etc.)
Linux 3.x
Solaris 11.0-11.3
Linux 4.x
OpenServer 10.x

Why Unix/Linux? I

- Industry-tested for four decades: powerful beyond your imagination.
- Standard OS for scientific computation and high-performance computing → as of November 2021, ALL the Top 500 supercomputers in the world (including those owned by Microsoft!) 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/tc4ROCJYbm0`.

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).

Interaction

- 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

Shell

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

  ```
  In [1]: echo $0
  ```

  on a Mac Terminal, you will probably get:

  ```
  Out[1]: -zsh
  ```

- Easy to change shells.
- Most of them offer similar capabilities, but `bash` and derivatives 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
```

Shebang:

```
$ #!
```

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 (2nd Edition)*, by William Shotts.

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 (quite popular) Homebrew: `https://brew.sh/`.
- Less popular:
  1. MacPorts: `http://www.macports.org/`.
  2. Fink: `https://www.finkproject.org/`.

```
$ brew update
$ brew upgrade
```

Editors

Editors

- By default, you should try to use plain, open files:
  1. Text files (`README`s, `HOWTO`s, ...).
  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. `Visual Studio Code` (my usual choice).
  2. `Sublime`.
  3. `Atom` (not being developed any longer).
- 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.

IDEs

IDEs I

- 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`.

IDEs II

- Specific languages:
    1. `Spyder`.
    2. `RStudio`.
    3. `Matlab IDE`.
    4. `Wolfram Workbench`.

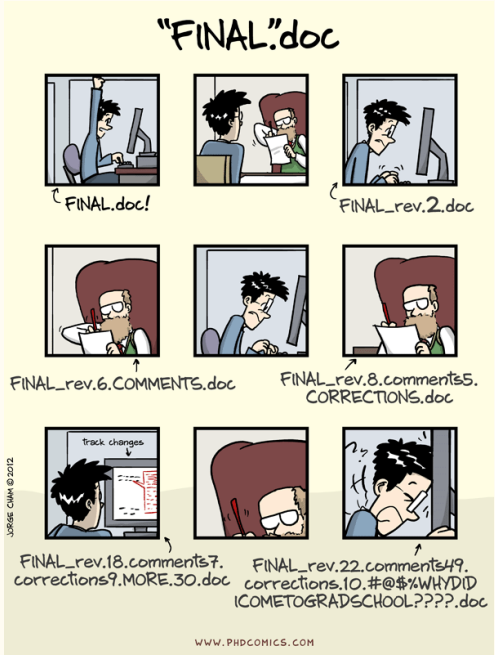
Version Control

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
track changes
JORGE CHAM © 2012
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
WWW.PHDCOMICS.COM

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.

Workflow

- 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.


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Git I

- 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`.

Git II

- Very popular: `http://www.github.com`
- Also, `https://about.gitlab.com/`
- A good reference: *Pro Git (2nd Ed.)* by Scott Chacon, `http://git-scm.com/book`.
- Many tutorials online.
- Integrated with `Juno` and `RStudio` and can easily integrate with `Visual Code Studio` 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.
- Pandoc: http://pandoc.org/

Notebook components