

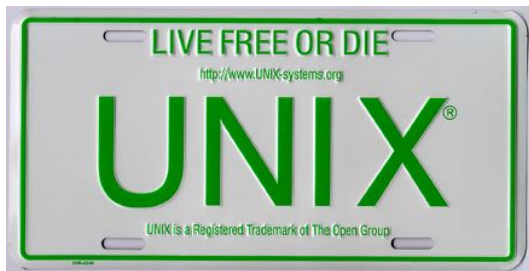
Unix introduction

Mikhail Dozmorov

Summer 2018

What is Unix

- Unix is a family of operating systems and environments that exploits the power of linguistic abstractions to perform tasks
- Unix is not an acronym; it is a pun on “Multics”. Multics was a large multi-user operating system that was being developed at Bell Labs shortly before Unix was created in the early '70s. Brian Kernighan is credited with the name.
- All computational genomics is done in Unix



History of Unix

- Initial file system, command interpreter (shell), and process management started by Ken Thompson
- File system and further development from Dennis Ritchie, as well as Doug McIlroy and Joe Ossanna
- Vast array of simple, dependable tools that each do one simple task



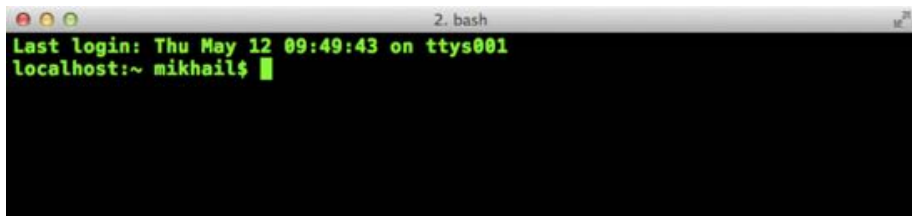
Ken Thompson (sitting) and Dennis Ritchie working together at a PDP-11

Philosophy of Unix

- Vast array of simple, dependable tools
- Each do one simple task, and do it really well
- By combining these tools, one can conduct rather sophisticated analyses
- The Linux help philosophy: “RTFM” (Read the Fine Manual)

Know your Unix

- Unix users spend a lot of time at the **command line**
- In Unix, a word is worth a thousand mouse clicks

A screenshot of a terminal window titled "2. bash". The terminal displays a green login message: "Last login: Thu May 12 09:49:43 on ttys001". Below this, the prompt "localhost:~ mikhail\$" is shown with a green cursor. The terminal background is black, and the text is green.

```
2. bash
Last login: Thu May 12 09:49:43 on ttys001
localhost:~ mikhail$
```

Unix systems

- Three common types of laptop/desktop operating systems: Windows, Mac, Linux.
- Mac and Linux are both Unix-like!
- What that means for us: Unix-like operating systems are equipped with “shells” that provide a command line user interface.



What is shell

- Shell is an interactive environment with a set of commands to initiate and direct computations
- Shell encloses the complexity of OS, hence the name
 - You type in commands
 - Shell executes them

https://en.wikipedia.org/wiki/Unix_shell

Command line, aka shell

- The Bourne shell (`sh`) is a shell, or command-line interpreter, for computer operating systems.
- Developed by Stephen Bourne at Bell Labs, 1976
- `bash` (the Bourne-Again shell) was later developed for the GNU project and incorporates features from the Bourne shell, `csh`, and `ksh`. It is meant to be POSIX-compliant.

https://en.wikipedia.org/wiki/Stephen_R._Bourne

Most popular types of shell

- `bash` - Bourne-Again shell
- `tcsh` - TENEX C shell
- `zsh` - Z shell
- Change shell: `chsh -s /bin/zsh`
- `$SHELL` environmental variable has path to the currently used shell

Getting to the command line

- Remote access, SSH, **PuTTY**
(<http://www.chiark.greenend.org.uk/~sgtatham/putty/>), **MobaXterm**
(<https://mobaxterm.mobatek.net/>)
- **Mac OS X + Xcode development suite** (free, <https://developer.apple.com/xcode/>) + **X11 server** (free, <https://www.xquartz.org/>) + **iTerm2** (optional, <https://iterm2.com/>)
- **Ubuntu Linux** (long-term support LTS version, XX.04, <http://www.ubuntu.com/download/desktop>)

Getting to the command line | Windows users

- **Cygwin**, <http://www.cygwin.com/>
- **Git Bash**, <https://git-for-windows.github.io/>
- Boot from a CD or USB (search for “linux usb”)
- Install the whole Linux systems as a Virtual Machine in **VirtualBox**

<https://www.virtualbox.org/>

Getting to the command line | Mac users

- “Terminal” is already installed, `bash` shell
- Why? Darwin, the system on which Apple’s Mac OS X is built, is a derivative of 4.4 BSD-Lite2 and FreeBSD. In other words, the Mac is a Unix system.
- For X11 (graphics), see XQuartz
- iTerm2 - a better terminal replacement for Mac

<http://xquartz.macosforge.org/landing/>

<https://www.iterm2.com/>

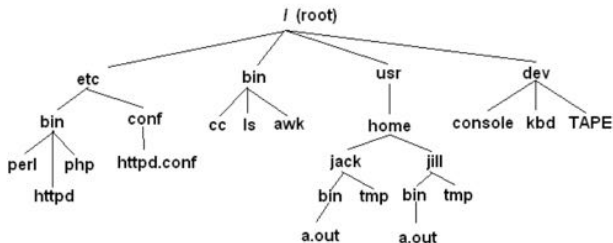
Obtaining new command-line software

- Modern Unix systems have package managers to that download install (free) software for you
- On a Mac, **Homebrew** (<http://brew.sh/>) is a popular package-management system (alternatively, **MacPorts**, <https://www.macports.org/>)
- On Ubuntu, **apt** (https://en.wikipedia.org/wiki/Advanced_Packaging_Tool) is the standard package manager, with both a command-line and graphical interface available
- On Windows, **Cygwin** (<https://cygwin.com/install.html>) installs everything precompiled through its setup file. Do not delete `setup-x86_64.exe` file after installing Cygwin, explore what Linux tools are available (a lot)

Interacting with shell

- Most commands take additional arguments that fine tune their behavior
- If you don't know what a command does, use the command `man <command>`
- Press `q` to quit the `man` page viewer
- Most often, you'll use `<command> -h` or `<command> --help`
- Some commands output help if executed without any arguments

File system: Full vs. relative paths



- `cd /` - go to the root directory
- `cd /usr/home/jack/bin` - go to the user's sub-directory
- `cd ..` - go to the upper level directory
- `cd`, or `cd ~` - go to the user's home directory
- `cd --` - go to the last visited directory

Orienting in the filesystem

- `pwd` - print working directory
- `ls` - list all files in the current directory
- `ls -1` - list files in *one* column
- `ls -lah` - list files in long, human readable format, include all content, user, owner, permissions

Creating, moving, copying, and removing files

- `touch <file>` - creates an empty file
- `nano <file>` - edit it
- `mkdir <dirname>` - creates a directory
- `cp <source_file> <target_file>` - copy a file to another location/file
- `mv <source_file> <target_file>` - move a file
- `rm <file>` - remove a file. If multiple files provided, removes all of them
- `rm -r <dirname>` - recursive removal (deletes a directory)

Permissions: chmod, chown and chgrp

In Unix, every file and directory has an **owner** and a **group**

- **Owner** - is the one who created a file/directory
- **Group** - defines rules of file operations and/or permissions
- **Every** - user on a Unix machine can belong to one or more groups

Every file has **three permission levels**

- what the **u**ser can do
- what the **g**roup can do
- what the **a**ll can do

Permissions: `chmod`, `chown` and `chgrp`

- The first column tells you about the permissions on the file
 - The very first character in the permissions column tells you what kind of file it is. A `-` means it's a regular file. A `d` means it's a directory
 - The next nine characters come in three classes, each has three characters. The three classes are `owner/group/world` permissions
 - Inside a permission class, `r` means that class can **read** the file; `w` means that class can **w**rite the file; `x` means that class can **execute** the file
- The second column has the number of files (inside a directory)
- The third and fourth columns tell you the owner and group

Permissions: chmod, chown and chgrp

```
-rw-r--r--  1 mdozmorov  staff    205B Dec 19 11:01 BIOS692.2018.Rpt
-rw-r--r--  1 mdozmorov  staff    3.5K Dec 18 10:20 BUILD.md
-rw-r--r--  1 mdozmorov  staff    470B Dec 19 08:48 README.md
-rw-r--r--  1 mdozmorov  staff    2.1K Dec 19 07:51 _config.yml
drwxr-xr-x 10 mdozmorov  staff    340B Dec 18 10:20 _includes
drwxr-xr-x 10 mdozmorov  staff    340B Dec 18 10:20 _layouts
drwxr-xr-x  7 mdozmorov  staff    238B Dec 18 10:29 _posts
-rw-r--r--  1 mdozmorov  staff    1.0K Dec 20 15:54 acknowledgement
```

Finding your files

- `find` - lists all files under the working directory (and its subdirectories) based on arbitrary criteria
- `find .` - prints the name of every file or directory, recursively. Starts from the current directory
- `find . -type f` - finds files only
- `find . -type d -maxdepth 1` - finds directories only, at most 1 level down
- `find . -type f -name "*.mp3"` - finds only *.mp3 files
- `find . -type f -name "README.md" -exec wc -l {} \;` - find files and execute a command on them

Wildcards and patterns

- * - matches any character
- ? - matches a single character
- [chars] - matches any character in chars
- [a-zA-Z] - matches any character between a and z, including capital letters
- `ls *.md`
- `ls [Rt]*`

Looking inside files

- `cat <file>` - prints out content of a file. If multiple files, consequently prints out all of them (concatenates)
- `zcat` - prints out content of gzipped files
- `more/less <file>` - shows the content of the file one screen at a time

Keyboard shortcuts for more command

- space - forward
- b - backward
- g - go to the beginning
- G - go to the end
- `/<text>` - starts forward search, enter to find next instance
- q - quit

Chaining commands: pipes

One of the most useful capabilities of Unix is the ability to redirect the STDOUT of one command into the STDIN of another

The | (pipe) character feeds output from the first program (to the left of the |) as input to the second program on the right. Therefore, you can string all sorts of commands together using the pipe

```
find . | wc -l  
cat names.txt | sort | uniq -c
```

Executing one command AFTER another completed successfully:

```
<command> && <command>
```

```
mkdir music && mv *.mp3 music/
```


Chaining commands: redirections

- Nearly every command in Unix makes use of a convention to have a “standard input” (also called `stdin` or `STDIN`, or channel 0) and “standard output” (also called `stdout` or `STDOUT`, or channel 1)
- There is also a “standard error” (`stderr` or `STDERR`, or channel 2) output that is, by convention, reserved for error messages
- `find / 2> error.log` - capture `STDERR` into a file
- `find / 2> /dev/null` - suppress `STDERR` messages
- `find / 2>&1` - add `STDERR` to `STDOUT`

Chaining commands: redirections

- If you want to dump the standard output into a file, use `command > file` (overwrites the file). `command >> file` (appends to the file)
- Redirection example: `ls > README.md` - save file list in the current directory into README.md file
- Redirection works in another direction: `grep CC0 < License.md`
- Or, the content of a command into another command: `join <(sort file1) <(sort file2)`

Other essential commands

head/tail	cut
for	comm
sort	echo
uniq	basename
wc	dirname
tr	history
grep	which
join	who
kill	grep
tar	seq
gzip	paste

Shell conveniences

- Tab completion
- `Ctrl-c` - cancel the command you are writing
- `Ctrl-r` - reverse search through your command line history
- `history` - shows your previous commands
- `!<history number>` - repeats specific command. Or, `!ls` to match the most recent `ls` command
- `!!` - repeats the last command

Processes and job control

- As we interact with Linux, we create numbered instances of running programs called “processes.” You can use the `ps` command to see a listing of your processes (and others!). To see a long listing, for example, of all processes on the system try: `ps -ef`
- To see all the processes owned by you and other members of the class, try: `ps -ef | grep bash`
- To see the biggest consumers of CPU, use the `top` command (which refreshes every few seconds): `top`

Foreground/background

- Thus far, we have run commands at the prompt and waited for them to complete. We call this running in the “foreground.” It is also possible, using the & operator, to run programs in the “background”, with the result that the shell prompts immediately without waiting for the command to complete:

```
$ mycommand &  
[1] 54356    <----- process id  
$
```

Backgrounding a running job with C-z and 'bg'

- Sometimes you start a program, then decide you want to run it in the background. Here's how:
 - `countdown 200 > c.out`
 - Press C-z to suspend the job
 - Type `bg` at the command prompt
 - The job is now running in the background. To bring it back to the foreground, type `fg` at the command prompt

Process control

- To kill the job, use the 'kill' command, either with the five-digit process id: `kill 56894` #for example!

Statistical command line goodies

- **data_hacks**, https://github.com/bitly/data_hacks
 - Command line tools for data analysis
 - `histogram.py`
 - `bar_chart.py`
 - `sample.py`
- **datamash**, <https://www.gnu.org/software/datamash/>
 - summary statistics
 - transposing matrixes

Additional commands

- `tree` - lists the contents of directories in a tree-like format
- `csvkit` - collection of command-line tools to work with CSV data
- `parallel` - a shell tool for executing jobs in parallel using one or more computers

<https://www.cyberciti.biz/faq/linux-show-directory-structure-command-line/>

<https://csvkit.readthedocs.io/en/1.0.2/scripts/csvlook.html>

<https://csvkit.readthedocs.io/en/1.0.3/>

<https://www.datascienceatthecommandline.com/chapter-8-parallel-pipelines.html>

Unix for high-performance cluster computing

- Allow you to submit multiple jobs at once
- Depending on the system, can schedule jobs for you
- Are optimized for high-throughput performance

- VCU Biostatistics cluster information,
<https://wiki.vcu.edu/display/biosit/Home>
- Contact Helen Wang (huwang at vcu.edu) to establish an account
- Google if you run into problems using Unix!

Learn more

- <https://www.tutorialspoint.com/unix/index.htm>
- Heng Li's "A Bioinformatician's UNIX Toolbox", <http://lh3lh3.users.sourceforge.net/biounix.shtml>
- Bioinformatics one-liners by Stephen Turner, <https://github.com/stephenturner/oneliners>
- Collection of bioinformatics-genomics bash one liners, using awk, sed etc. <https://github.com/crazyhottommy/bioinformatics-one-liners>
- Links and references to many genomics and bioinformatics resources, <https://github.com/crazyhottommy/getting-started-with-genomics-tools-and-resources>