

Command Line - Part 1

STAT 133

Gaston Sanchez

Department of Statistics, UC–Berkeley

`gastonsanchez.com`

`github.com/gastonstat`

Course web: gastonsanchez.com/teaching/stat133

GUIs

Graphical User Interfaces

- ▶ Windows and Mac use a Graphical User Interface (GUI) for you to interact with the OS.
- ▶ GUIs are easy to learn
- ▶ GUIs rely on visual displays
- ▶ GUIs can be extremely useful
- ▶ GUIs have improved the friendliness and usability of computers

GUIs or Command Line?

- ▶ However, GUIs come with trade-offs
- ▶ They don't allow you to have more control over what your computer can do
- ▶ Some operations are labor intensive and repetitive
- ▶ You organize things by clicking and dragging with the cursor (which reduces reproducibility)

GUI Disadvantages

- ▶ Lack of repeatability
- ▶ Lack of reproducibility
- ▶ Some tasks may be labor intensive using a GUI
- ▶ GUIs limit analyses on a cluster of computers

Command Line

Command Line

- ▶ Instead of using a GUI, we can use a command line program
- ▶ The command line program is known as the **shell**
- ▶ By typing commands we perform tasks on the computer (without using a mouse)

Shell

- ▶ You're working with a program called the **shell**
- ▶ The shell interprets the commands you enter
- ▶ It runs the program you've asked for
- ▶ It coordinates what happens between you and the operating system
- ▶ There are various kinds or flavors of shells: e.g. Bourne (BASH), Korn, C shell

Command Line

- ▶ To interact with the shell we need a **terminal emulator**
- ▶ In Unix-like systems (e.g. Mac) the terminal is usually known as “terminal”
- ▶ Windows does not really provide a terminal; instead it provides the *command prompt*

Command Prompt in Windows

Finding MS Windows command prompt

- ▶ Click the **Start** button
- ▶ Click **All Programs**
- ▶ Click **Accessories**
- ▶ Click **Command Prompt**

Windows command prompt is not a UNIX shell

Shells for Windows

- ▶ Instead of using the command prompt you can use ad-hoc shell environments for Windows
- ▶ e.g. Git-Bash, PowerShell, Cygwin
- ▶ Git for Windows provides a BASH emulation
- ▶ PowerShell is part of Windows Management Framework 4.0
- ▶ Cygwin is large collection of GNU and Open Source tools

Mac Terminal



- ▶ Go to **Applications**
- ▶ Go to **Utilities**
- ▶ Click **Terminal**

Try Some Commands

- ▶ `date` (current time and date)
- ▶ `cal` (calendar of current month)
- ▶ `df` (amount of free space in your disk drives)
- ▶ `who` (logged in users)
- ▶ `echo 'Hello'`

Shell

- ▶ Shells run in terminal emulators, or **terminals**
- ▶ In Mac OS X, the default terminal program is called **Terminal**
- ▶ The command line is displayed within the terminal window
- ▶ The program behind the terminal is the **shell**
- ▶ There are many different shell programs

BASH

The most common type of shell is BASH

- ▶ BASH: Bourne Again SHell
- ▶ BASH is the default shell for Linux
- ▶ BASH is usually the default shell on Mac
- ▶ type `echo $SHELL` to see your shell
- ▶ type `bash` to get a bash shell

BASH

- ▶ A shell does much more than simply run commands
- ▶ It has wildcards for matching filenames
- ▶ It has a command history to recall previous commands quickly
- ▶ It has pipes for making the output of one command become the input of another
- ▶ It has variables for storing values for use by the shell

Command who

- ▶ `who` displays a list of users that are currently logged in
- ▶ `who am i` (`whoami`) tells you the current user name

Shell Command Syntax

```
command -options arg1 arg2
```

- ▶ Blanks and "-" are delimiters
- ▶ The number of arguments may vary
- ▶ An argument comes at the end of the command line
- ▶ It's usually the name of a file or some text
- ▶ Many commands have default arguments

Date and Calendar

- ▶ `date`
- ▶ `cal` (current calendar year)
- ▶ `cal july 2015` (July 2015)
- ▶ `cal jan 2000`
- ▶ `ncal -w july 2015` (week number)

Options

`command -options arg1 arg2`

- ▶ Options come between the command and the arguments
- ▶ They tell the command to do something other than its default
- ▶ They are usually prefaced with one or two hyphens
- ▶ e.g. `ncal -w july 2015`

Some Control Sequences

keys	description
Ctrl + l	clear screen
Ctrl + c	stop current command
Ctrl + z	suspend current command
Ctrl + k	kill to end of line
Ctrl + r	search history
Ctrl + n	next history item
Ctrl + p	previous history item

Manual Documentation

- ▶ To see the help documentation of a command use `man` followed by the name of the command:
 - `man cal`
 - `man date`
 - `man who`
- ▶ `q` quits manual documentation

Logging Out

- ▶ `exit` logs you out
- ▶ `q` quits manual documentation

System Navigation

Filesystem Reminder

- ▶ The nested hierarchy of folders and files on your computer is called the **filesystem**
- ▶ The filesystem follows a tree-like structure
- ▶ The root directory is the most inclusive folder on the system
- ▶ The root directory serves as the container for all other files and folders
- ▶ A Unix-based system (e.g. OS X) has a single root directory
- ▶ Windows users usually have multiple roots (C: , D: , etc)

Paths

- ▶ Each file and directory has a unique name in the filesystem
- ▶ Such unique name is called a **path**
- ▶ A path can be **absolute** or **relative**
- ▶ An **absolute path** is a complete and unambiguous description of where something is in relation to the root
- ▶ A **relative** describes where a folder or file is in relation to another folder

Paths

- ▶ There are two special relative paths: `.` and `..`
- ▶ The single period `.` refers to your current directory
- ▶ The two periods means your parent directory, one level above

Home Directory

- ▶ User's personal files are found in the `/Users` directory
- ▶ A user directory is the **home** directory
- ▶ `cd` (with no other arguments) returns you to your home directory
- ▶ `echo $HOME` prints your home directory
- ▶ `cd ~` takes you to your home directory

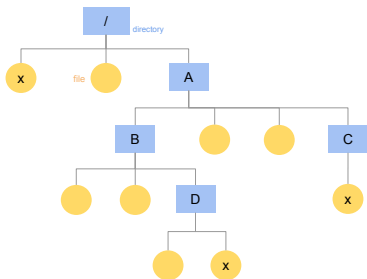
Working Directory

- ▶ Another special type of directory is the so-called **working directory**
- ▶ The working directory is the current directory where you perform any task
- ▶ `pwd` prints the working directory

Changing Directories

- ▶ `cd`
- ▶ `cd ..`
- ▶ `cd /`
- ▶ `cd ~`
- ▶ `cd ~/Documents`

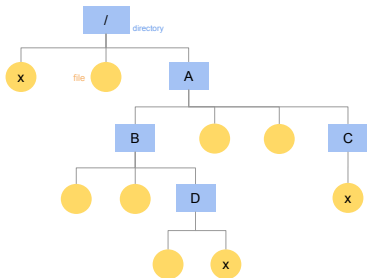
Absolute Path Names



From the root directory to D:

`cd /A/B/D`

Relative Path Names



Changing directories from D to C

`cd ../../C`

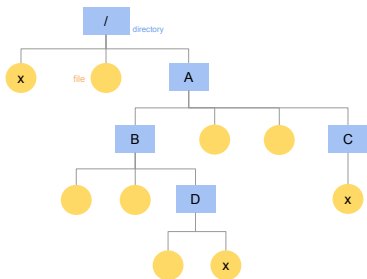
Listing Contents in a Directory

- ▶ `ls`
- ▶ `ls -l` (one entry per line)
- ▶ `ls -l` (list in long format)
- ▶ `ls -a` (show files starting with a dot)
- ▶ `man ls` (manual documentation)

Listing Contents in a Directory

- ▶ `ls /` (specify root directory)
- ▶ `ls /usr` (specifying a directory)
- ▶ `ls ~` (home directory)
- ▶ `ls -lt` (long format, sorted by modification time)

Listing Contents



Show contents in D from C

```
ls ../B/D/
```

Inspecting Files

File Permissions

- ▶ run the command: `ls -l`
- ▶ directories may be displayed as: `drwxr-xr-x`
- ▶ files may be displayed as: `-rw-r--r--`
- ▶ file permissions are the 10 most left characters
- ▶ `r` means reads
- ▶ `w` means write
- ▶ `x` means execute

File Permissions

Read from left to right the permissions mean

position	description
1	File type. A dash - means a plain file and d means a directory. There are other less common options.
2-4	Owner permissions: read, write, and execute permissions for the file's owner.
5-7	Group permissions: read, write, and execute permissions for the file's group.
8-10	World permissions: read, write, and execute permissions for all other users.

Type of File

Determine the type of a file:

`file filename`

Some commands for inspecting text files

- ▶ `wc filename`
- ▶ `cat filename`
- ▶ `head filename`
- ▶ `tail filename`
- ▶ `more filename`
- ▶ `less filename`

Viewing file contents with less

- ▶ There are several commands that display the contents of text files
- ▶ The most commonly used file viewer is `less`
- ▶ `less` presents the contents of that file on the screen one page at a time
- ▶ There are various keyboard shortcuts to navigate in `less`

Viewing file contents with less

key	description
Page Up or b	scroll back one page
Page Down or space	scroll forward one page
Up Arrow	scroll up one line
Down Arrow	scroll down one line
G	move to the end of text file
1G or g	move to the beginning of the text file
/hello	search forward to next occurrence of hello
n	search for the next search occurrence
h	display help screen
q	quit less

Quoting Files

If you want a word to contain whitespace (e.g. a filename with a space in it), surround it with single or double quotes to make the shell treat it as a unit:

```
ls "My file"
```

Exploring a file

- ▶ `cd` into a given directory
- ▶ List the directory contents with `ls -l`
- ▶ Determine the contents of a file with `file`
- ▶ If it looks like it might be text, try viewing it with `less`

Editing text files at the command line

- ▶ Sometimes it is more convenient to create or modify a file right at the command line
- ▶ Although `less` is a convenient file viewer, it does not allow you to edit the contents
- ▶ Depending on your operating system and shell tool, you may have one or more command-line text editors:
- ▶ e.g. `vi`, `nano`, `gedit`

Editing text files at the command line

- ▶ One common text editor is `vi` (there's also `vim`)
- ▶ It should be available in Mac, and also in Git-Bash (Windows)
- ▶ Depending on your operating system and shell tool, you may have one or more command-line text editors:
- ▶ Type `which vi` to find out if you have it

Editing text files with vi

- ▶ To create and start editing a file simply type `vi` followed by the name of the new file:

```
vi newfile.txt
```

- ▶ Press the **I** key to start editing content
- ▶ When you're done, press the **ESC** key
- ▶ Then type `:wq` to save and quit
- ▶ You can reopen it again with: `vi newfile.txt`

Google **vi cheat sheet** to find more information

File Management

Managing Files

Common actions

- ▶ creating a directory
- ▶ creating a file
- ▶ copying a file
- ▶ moving a file
- ▶ deleting a file
- ▶ searching a file

Managing Files

Common actions

- ▶ creating a directory: `mkdir`
- ▶ creating a file: usually through a text editor
- ▶ copying a file: `cp`
- ▶ moving a file: `mv`
- ▶ deleting a file: `rm`
- ▶ searching a file: `?`

Creating Directories and Files

Create a directory "summer2015" in my Documents

```
cd ~/Documents  
mkdir summer2015
```

Create an empty file "README.md" in summer2015

```
cd summer2015  
touch README.md
```

Copying Files

- ▶ `cp` is the command to copy files
- ▶ `cp` can be used in two ways:
- ▶ `cp file1 file2` copies `file1` into `file2`
- ▶ `cp file1 directory` copies `file1` into a directory (directory must already exist)

Copying Files

Copying functions.R from Documents to HW6

```
cp ~/Documents/functions.R ~/Desktop/HW6/
```

Copying starwars.csv to current directory

```
cp ~/Documents/starwars.csv .
```

Deleting files

Deleting README.md and starwars2.csv

```
cd ~/Documents/summer2015  
rm README.md  
rm starwars2.csv
```

Wildcards

- ▶ the shell provides special characters to specify filenames
- ▶ these special characters are called **wildcards**
- ▶ using wildcards allow you to select filenames based on patterns of characters
- ▶ these wildcards are similar to some regular expression characters

Wildcards

wildcard	description
*	matches any characters
?	matches any single character
[characters]	matches any character that is a member of the set <i>characters</i>
[!characters]	matches any character that is not a member of the set <i>characters</i>
[[:class:]]	matches any character that is a member of the specified <i>class</i>

Example

Create a directory dummy, cd to it, and then create empty files:

```
$ mkdir dummy  
$ cd dummy  
$ touch AGing.txt Bing.xt Gagging.text Going.nxt ing.ext  
$ ls
```

* Wildcard

Use ***** to refer to multiple files at once; it stands for *anything*

```
$ ls
AGing.txt    Bing.xt
Gagging.text Going.nxt  ing.ext
```

```
$ ls G*
Gagging.txt  Going.nxt
```

```
$ ls *.xt
Bing.xt
```

? Wildcard

The question mark **?** represents a *single* character

```
$ ls
AGing.txt  Bing.txt
Gagging.text  Going.nxt  ing.ext

$ ls ?ing.txt
Bing.txt
```

[] Wildcard

Brackets [] can be replaced by whatever characters are within those characters:

```
$ ls
AGing.txt    Bing.xt
Gagging.text Going.nxt  ing.ext
```

```
$ ls [B]ing.*
Bing.xt
```

```
$ ls [A-G]ing.*
Bing.xt
```

Combining Wildcards

Wildcards can be combined:

```
$ ls
AGing.txt    Bing.xt
Gagging.text Going.nxt ing.ext

$ ls *G*
AGing.txt Gagging.txt Going.nxt

$ ls *i*.*e*
Gagging.text ing.ext
```

Test Yourself

```
AGing.txt    Bing.xt  
Gagging.text Going.nxt ing.ext
```

What command produces the output above:

- A) `ls *ing.*xt`
- B) `ls ?ing.*xt`
- C) `ls ?ing.?xt`
- D) `ls ?ing.xt`
- E) `ls *ing.?xt`

Test Yourself

```
AGing.txt  Going.nxt  ing.ext
```

What command produces the output above:

- A) `ls *ing.*xt`
- B) `ls ?ing.*xt`
- C) `ls ?ing.?xt`
- D) `ls ?ing.xt`
- E) `ls *ing.?xt`

Wildcard Examples

Pattern	Matches
<code>*</code>	all files
<code>a*</code>	any file beginning with "a"
<code>*.txt</code>	any file ending with ".txt"
<code>b*.txt</code>	any file beginning with "b" followed by any characters and ending with ".txt"
<code>[gst]*</code>	any file beginning with either a "g", and "s", or a "t"
<code>[[digit:]]*</code>	any file beginning with a number
<code>[[upper:]]*</code>	any file beginning with an uppercase letter