Shell, git, and GitHub

Programming for Statistical Science

Shawn Santo

Supplementary materials

Full video lecture available in Zoom Cloud Recordings

Companion videos

- Command line basics
- Using globs

Videos were created for STA 323 & 523 - Summer 2020

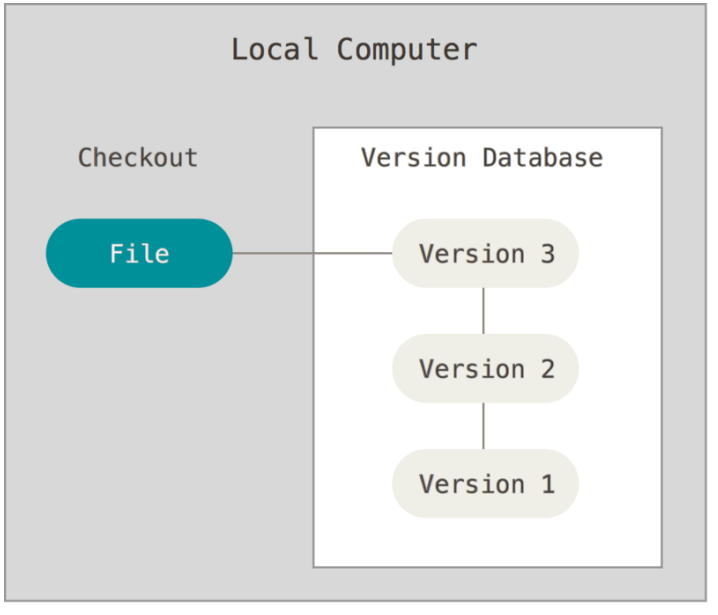
Additional resources

- Getting Started Pro Git
- Git Basics Pro Git
- Git cheatsheet
- Happy Git and GitHub for the useR

git and GitHub

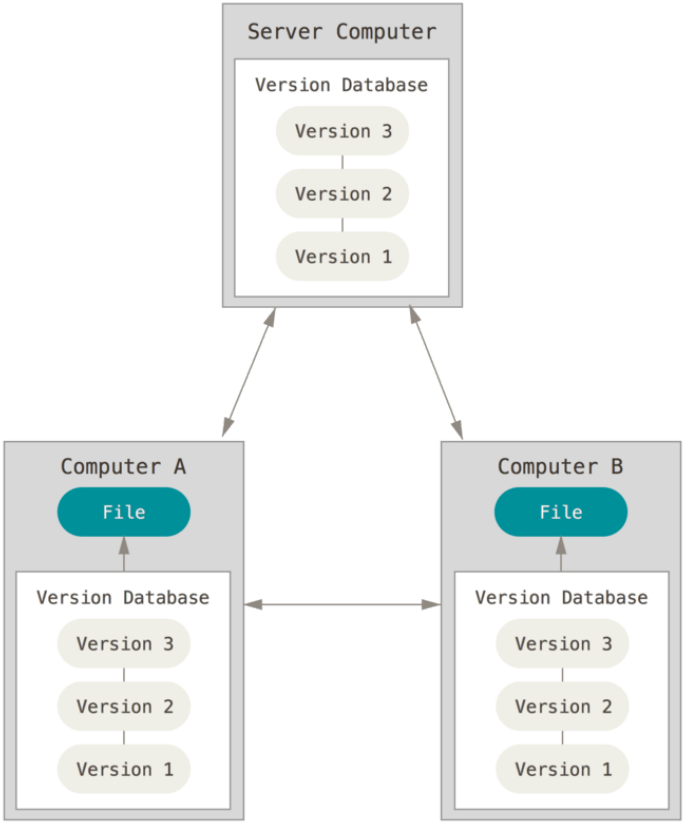
Why version control?

- Simple formal system for tracking all changes to a project
- Time machine for your projects
  - Track blame and/or praise
  - Remove the fear of breaking things
- Learning curve is steep, but when you need it you *REALLY* need it



Why git?

- Distributed
  - Work online or offline
  - Collaborate with large groups
- Popular and Successful
  - Active development
  - Shiny new tools and ecosystems
  - Fast
- Tracks any type of file
- Branching
  - Smarter merges



Verifying git exists

git is already set-up on the DSS servers. You can see this by

```
[sms185@numeric1 ~]$ git --version
git version 2.20.1
```

```
[sms185@numeric1 ~]$ which git
/usr/bin/git
```

To install git and connect git, GitHub and RStudio on your own computer follow the directions in Happy Git and GitHub for the useR.

git and GitHub live demo

Configure git

Enter the following in your terminal to tell git who you are, what editor you want to use, and to store your GitHub credentials.

```
git config --global user.name "name"
git config --global user.email "github email"
git config --global push.default simple
git config --global core.editor [editor-of-choice]
git config --global credential.helper 'cache --timeout=600000'
```

On DSS my editor of choice is vim; on my own computer my editor of choice is Sublime Text 3.

You will need to do this configuration on each machine in which you choose to use git.

For windows users, the last line should be

```
git config --global credential.helper wincred
```

Configure git verification

To verify you configured git correctly, run

```
[sms185@numeric1 ~]$ git config --global -l
user.name=Shawn Santo
user.email=shawn.santo@duke.edu
core.editor=vim
push.default=simple
credential.helper=cache --timeout=600000
```

You should see output similar to above.

Using command `git config --global -l` on Windows:

```
user.name=Shawn Santo
user.email=shawn.santo@duke.edu
core.editor='c:/program files/sublime text 3/sublime_text.exe' -w
credential.helper=wincred
push.default=simple
```

Using command `git config --global -l` on Mac:

```
credential.helper=osxkeychain
user.name=Shawn Santo
user.email=shawn.santo@duke.edu
core.editor=/Applications/Sublime\ Text.app/Contents/SharedSupport/bin/subl -n -w
push.default=simple
credential.helper=cache --timeout=600000
```

Version control best practices

- Commit early, often, and with complete code.
- Write clear and concise commit summary messages.
- Test code before you commit.
- Use branches.
- Communicate with your team.

Shell

Before we get started...

On GitHub:

1. Navigate to https://github.com/sta523-fa20/cl-demo.
2. Fork the repository.
3. Navigate to the forked repository and copy the repository URL - available by clicking on the green `Code` button.

In RStudio:

1. Go to `File > New Project...`
2. Select `Version Control`
3. Select `Git`
4. Paste the Repository URL into the dialog box
5. Click `Create Project`

We will learn to navigate and manipulate files in folder `cl-demo` using the terminal and shell commands.

Unix and Unix variants

- Unix was an operating system developed at Bell Labs in the early 1970s.
- Bell Labs distributed the operating system in its source language form, so anyone who obtained a copy could modify and customize it for his own purposes.
- Linux, Mac OS X, Android, iOS, Chrome OS, Orbis OS all use Unix-like operating systems.

Source: https://www.bell-labs.com/usr/dmr/www/picture.html

What is the shell?

- A shell is software that provides an interface for an operating system's users to provide access to the kernel's services.
- Shells that take commands from the keyboard and give them to the operating system were originally the only interface available on a Unix like system
- Today we have both graphical user interfaces (GUIs) and command line interfaces (CLIs).
- Many shell variants exist including sh, bash, csh, ksh, and more.

```
[sms185@numeric1 ~]$ which bash
/usr/bin/bash
[sms185@numeric1 ~]$ which zsh
/usr/bin/zsh
[sms185@numeric1 ~]$ which sh
/usr/bin/sh
```

What is the terminal?

- A program that accepts input from the keyboard and displays output on the screen.
- Also known as a terminal emulator.
- The terminal needs the shell. The shell interprets what you input into the terminal.
- **Windows users**: use Git Bash for BASH emulation
- **Mac OS X and Linux**: use built-in terminal emulator

Why use the terminal and shell?

- Automate tasks
- A lot of functionality with minimal keyboard entry
- Full functionality of programs
- System administration, computer maintenance, and application start-up scripts

Command line basics

Where am I?

```
[sms185@numeric1 ~/cl-demo]$
```

- `sms185`: user
- `numeric1`: computer (server) name
- `~`: home directory
- `/cl-demo`: folder `cl-demo`
- `$`: signifies I am a regular user as opposed to a root user with #

To print my working directory (where I am located) use

```
[sms185@numeric1 ~/cl-demo]$ pwd
/home/fac/sms185/cl-demo
```

Change directories

Change directories with command `cd`.

Let's move to the home directory, print that directory, then move to directory `cl-demo` and print the working directory once there.

```
[sms185@numeric1 ~/cl-demo]$ cd ~
[sms185@numeric1 ~]$ pwd
/home/fac/sms185
```

```
[sms185@numeric1 ~]$ cd cl-demo/
[sms185@numeric1 ~/cl-demo]$ pwd
/home/fac/sms185/cl-demo
```

Navigate to your home directory. Then go to `cl-demo/data/` and print the working directory once there.

Navigate through multiple directories by specifying the complete relative path.

Absolute vs. relative paths

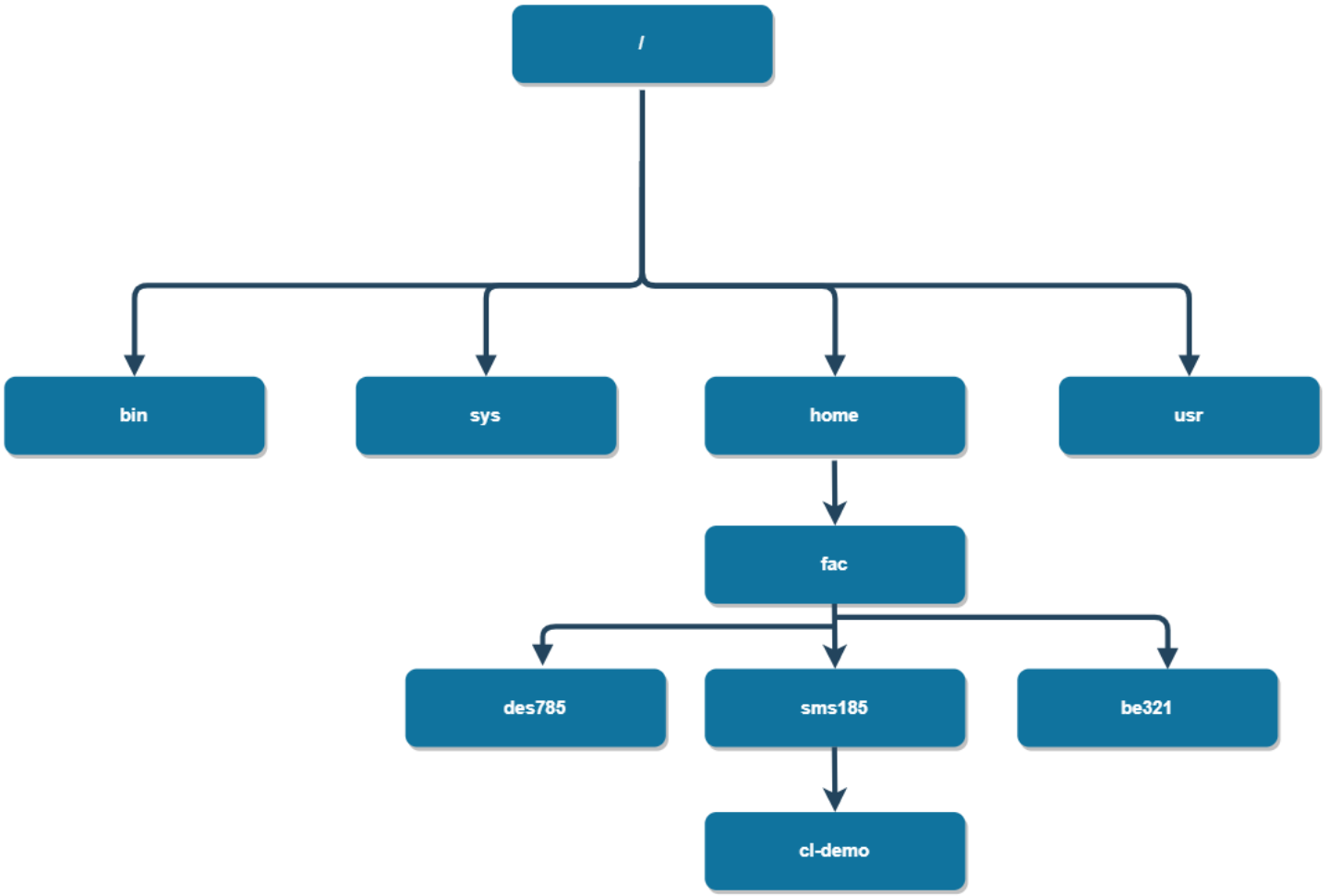
An absolute path is defined as specifying the location of a file or directory from the root directory.

```
[sms185@numeric1 ~]$ cd /home/fac/sms185/cl-demo/
[sms185@numeric1 cl-demo]$
```

A relative path is defined as the path related to your current working directory.

```
[sms185@numeric1 ~]$ cd cl-demo/
[sms185@numeric1 cl-demo]$
```

Paths diagram example



Anatomy of a Unix command

```
[sms185@numeric1 cl-demo]$ ls -l --all ./
total 88
drwxr-xr-x  6 sms185 prof   7 Aug 18 15:34 .
drwxr-xr-x 37 sms185 prof  63 Aug 18 15:34 ..
drwxr-xr-x  2 sms185 prof   5 Aug 18 15:34 data
drwxr-xr-x  8 sms185 prof  13 Aug 18 15:34 .git
-rw-r--r--  1 sms185 prof 156 Aug 18 15:34 readme.md
drwxr-xr-x  2 sms185 prof   5 Aug 18 15:34 scripts
drwxr-xr-x  2 sms185 prof 594 Aug 18 15:34 simulation
```

- `[sms185@numeric1 cl-demo]$` is the prompt
- `ls` is the command, list directory contents in this case
- `-l` is a short flag / option, `l` here means long listing format
- `--all` is a long flag / option, `--all` is equivalent to `-a`
- `./` is the argument, `./` here refers to the current directory

General structure: `command [options] [arguments]`. To get help, input `man [command]` in the terminal.

```
man ls
```

Parse the result

```
[sms185@numeric1 cl-demo]$ ls -l --all ./
total 88
drwxr-xr-x  6 sms185 prof   7 Aug 18 15:34 .
drwxr-xr-x 37 sms185 prof  63 Aug 18 15:34 ..
drwxr-xr-x  2 sms185 prof   5 Aug 18 15:34 data
drwxr-xr-x  8 sms185 prof  13 Aug 18 15:34 .git
-rw-r--r--  1 sms185 prof 156 Aug 18 15:34 readme.md
drwxr-xr-x  2 sms185 prof   5 Aug 18 15:34 scripts
drwxr-xr-x  2 sms185 prof 594 Aug 18 15:34 simulation
```

d	**rwx**	**r-x**	**r-x**	**2**	**sms185**	**prof**	**5**	**Aug 18 15:34**	**data**
directory	owner permissions	group permissions	everyone permissions	links or directories inside	owner	group	size (bytes)	date modified	name

Parse the result

```
[sms185@numeric1 cl-demo]$ ls -l --all ./
total 88
drwxr-xr-x  6 sms185 prof   7 Aug 18 15:34 .
drwxr-xr-x 37 sms185 prof  63 Aug 18 15:34 ..
drwxr-xr-x  2 sms185 prof   5 Aug 18 15:34 data
drwxr-xr-x  8 sms185 prof  13 Aug 18 15:34 .git
-rw-r--r--  1 sms185 prof 156 Aug 18 15:34 readme.md
drwxr-xr-x  2 sms185 prof   5 Aug 18 15:34 scripts
drwxr-xr-x  2 sms185 prof 594 Aug 18 15:34 simulation
```

-	rw-	r--	r--	1	sms185	prof	64	Aug 18 15:34	readme.md
file	owner permissions	group permissions	everyone permissions	links or directories inside	owner	group	size (bytes)	date modified	name

Copy (`cp`), move (`mv`), and delete (`rm`)

Copy `cars.txt` and call the new file `cars2.txt`.

```
[sms185@numeric1 cl-demo]$ cd data/
[sms185@numeric1 data]$ ls
cars.txt  diamonds.txt  mtcars.txt
[sms185@numeric1 data]$ cp cars.txt cars2.txt
[sms185@numeric1 data]$ ls
cars2.txt  cars.txt  diamonds.txt  mtcars.txt
```

Move `cars2.txt` up one directory.

```
[sms185@numeric1 data]$ mv cars2.txt ../
[sms185@numeric1 data]$ cd ../
[sms185@numeric1 cl-demo]$ ls
cars2.txt  data  readme.md  scripts  simulation
[sms185@numeric1 cl-demo]$
```

Delete `cars2.txt` and verify it is gone.

```
[sms185@numeric1 cl-demo]$ rm cars2.txt
[sms185@numeric1 cl-demo]$ ls
data  readme.md  scripts  simulation
```

Wildcards and the shell (globs)

These are characters that the shell will try to expand to match existing paths and files.

- `*` matches any number of characters in a file name, including none.
- `?` matches any single character.
- `[ ]` set of characters that may match a single character at that position.

These are similar but distinct from regular expressions which we will discuss later.

To find out how many simulation results we have for when p is 500 and n is 60 we can use the `*` wildcard.

```
[sms185@numeric1 cl-demo]$ cd simulation/
[sms185@numeric1 simulation]$ ls -l testresultp500n60* | wc -l
98
```

Other useful commands

Command	Use	Example
`cd ../`	move up one directory	`cd ../`
`cd ../../`	move up two directories	`cd ../ ../`
`man`	show the manual for a command	`man mkdir`
`mkdir`	create a directory	`mkdir new_folder`
`rmdir`	remove a directory	`rmdir new_folder`
`touch`	create a file	`touch get_data.R`
`locate`	find a file	`locate -n 5 "*.R"`
`clear`	clear terminal window	`clear`
`history`	print history of commands used	`history`
`htop`	process viewer (Linux)	`htop -u sms185`

Use commands `rm` and `rmdir` with caution. Never run `rm -rf /`.

Exercises

1. Navigate to directory `cl-demo/`. Explore and map out everything in `cl-demo/`. Draw a tree diagram as in the slide titled Path Diagram.

2. Create two new folders in `cl-demo/simulation/` with names `p500` and `p1000`. Move all simulation results with parameter p being 500 to folder `p500/`. Similarly, move all simulation results with parameter p being 1000 to folder `p1000/`.

git from the command line

git clone, status, add, commit, push

git Command	Description
`git clone <repo>`	Clone repo located at `<repo>` onto local machine
`git status`	List which files are staged, unstaged, and untracked
`git diff`	Show unstaged changes
`git add <directory/file>`	Stage changes of `<directory/file>`
`git commit -m "<message>"`	Commit staged snapshot with a summary `<message>`
`git push <remote> <branch>`	Push the `<branch>` to `<remote>`

These commands are enough to focus on for now as you work on individual assignments and projects. We'll soon add more that will facilitate team-based projects.

References

1. Git - Book. (2020). https://git-scm.com/book/en/v2.

2. Nokia Bell Labs: The invention of Unix. (2019). https://www.bell-labs.com/var/articles/invention-unix/.