Tulane University

Best Practices & Git

Nicholas Mattei, *Tulane University*

CMPS3660 – Introduction to Data Science – Fall 2019

https://rebrand.ly/TUDataScience

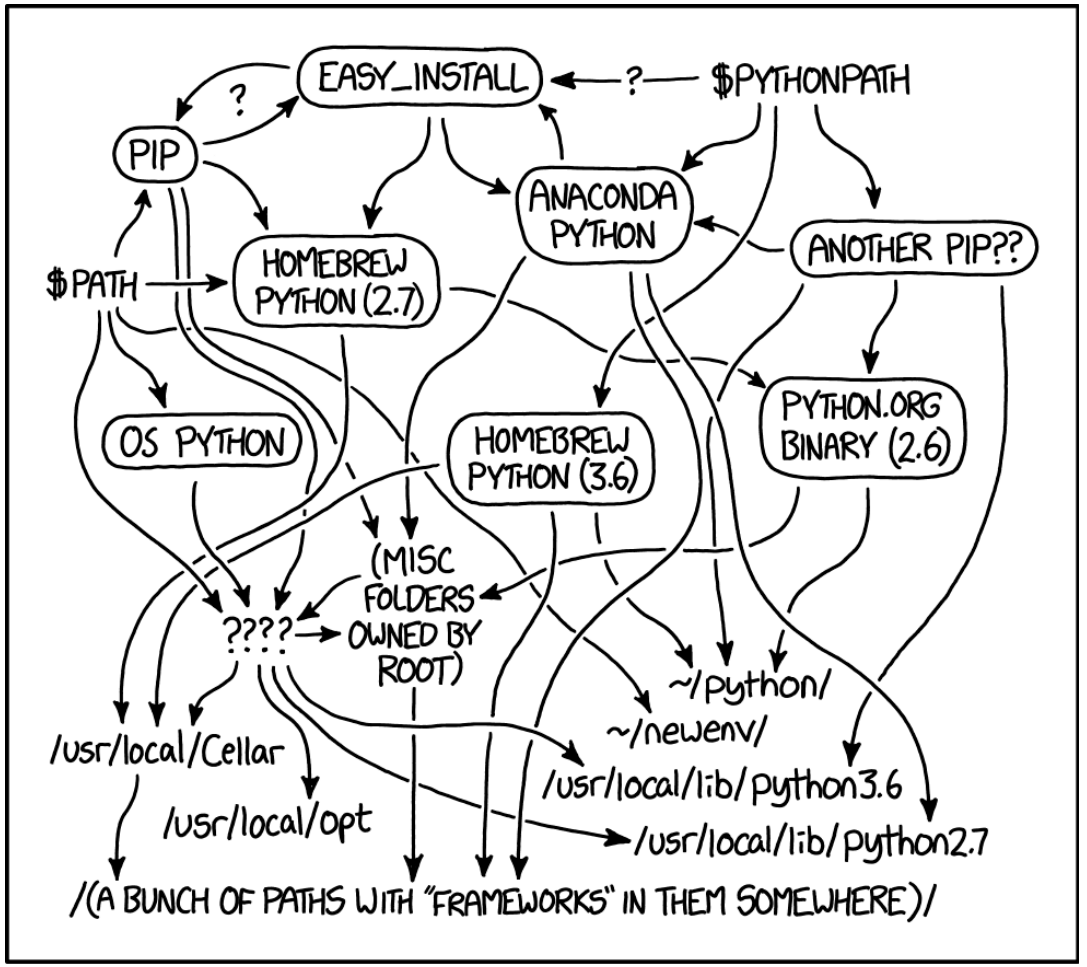
Many Thanks
Slides based off Introduction to Data Science from John P. Dickerson - https://cmsc320.github.io/

Announcements

- Lab day moved to Tuesday 9/10
  - Make sure you can run Docker or Anaconda on your laptop.
  - Note that you can develop on either Docker, Anaconda, System Python… but it must run on Docker for grading.
  - Make sure you can run the Notebook from Tuesday somehow.

- Going over Quiz 1

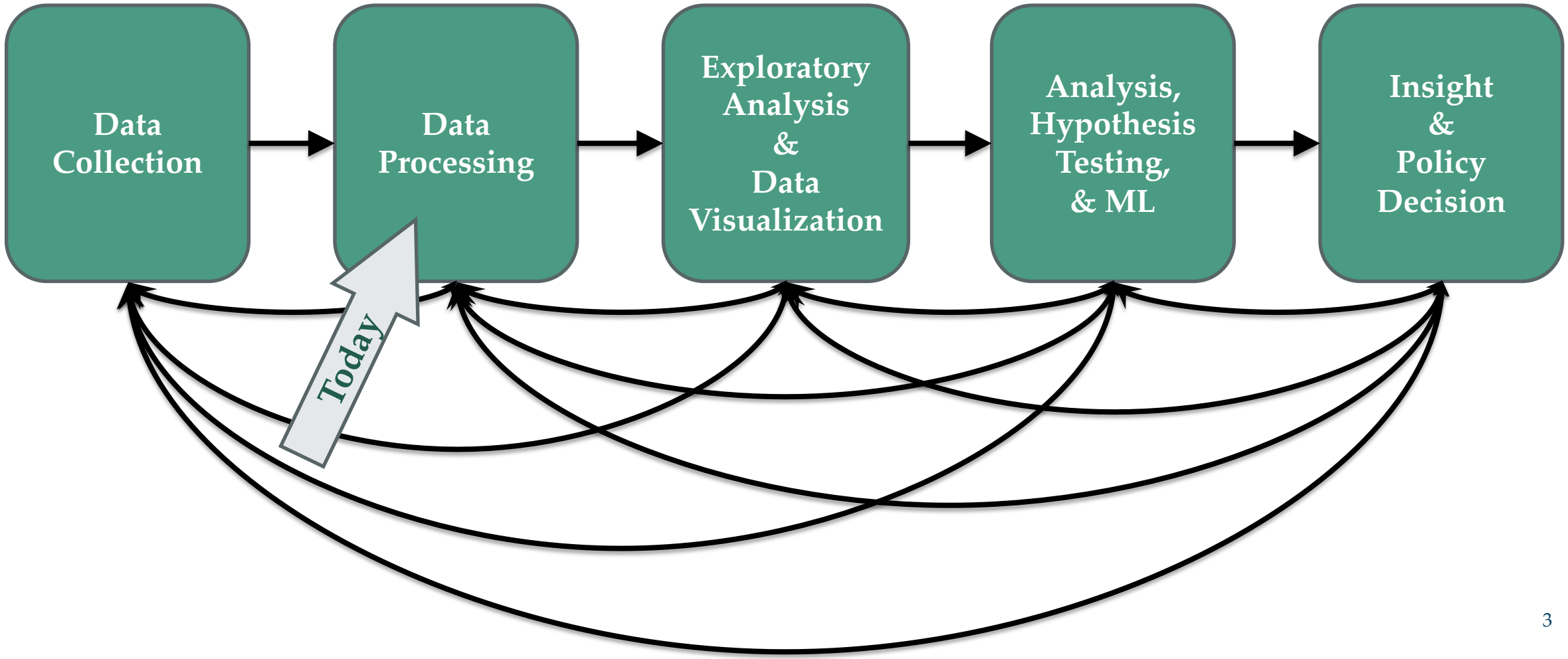
- Finish up Notebook from Lecture 3



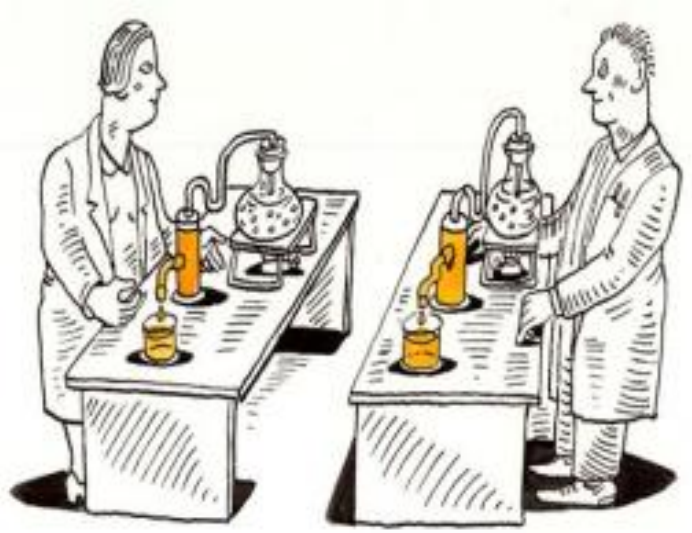
MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

www.xkcd.com

The Data LifeCycle



Reproducibility

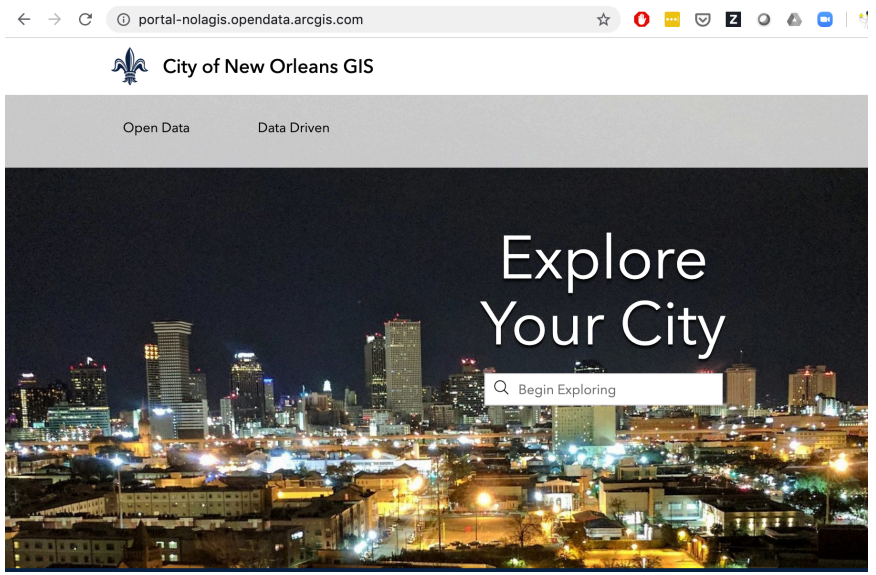


- Extremely important aspect of data analysis
  - "Starting from the same raw data, can we reproduce your analysis and obtain the same results?"
  - https://ropensci.github.io/reproducibility-guide/sections/introduction/
- Using libraries helps:
  - Since you don't reimplement everything, reduce programmer error
  - Large user bases serve as "watchdog" for quality and correctness
- Standard practices help:
  - Version control: git, git, git, …, git, svn, cvs, hg, Dropbox
  - Unit testing: unittest (Python), RUnit (R), testthat
  - Share and publish: github, gitlab

Many slides in this lecture adapted from Hector Corrado Bravo

Reproducibility

- "**Open data** is the idea that some data should be freely available to everyone to use and republish as they wish, without restrictions from copyright, patents or other mechanisms of control"
- Open Data Websites.
  - http://www.opendatafoundation.org/
  - https://portal-nolagis.opendata.arcgis.com/



Practical Tips

- Many tasks can be organized in modular manner:
- Data acquisition:
  - Get data, put it in usable format (many 'join' operations), clean it up, checkpoint it!
- Algorithm/tool development:
  - If new analysis tools are required.
- Computational analysis:
  - Use tools to analyze data.
- Communication of results:
  - Prepare summaries of experimental results, plots, publication, upload processed data to repositories.

Usually a single language or tool does not handle all of these equally well – **choose the best tool for the job!**

Practical Tips

- Modularity requires organization and careful thought
- In Data Science, we wear two hats:
  - Algorithm/tool developer
  - **Experimentalist:** we don't get trained to think this way enough!
- **It helps two consciously separate these two jobs**

- **Think like an experimentalist!**
  - **Plan your experiment**
  - **Gather your raw data**
  - **Gather your tools**
  - **Execute experiment**
  - **Analyze**
  - **Communicate**

Think Like An Experimentalist

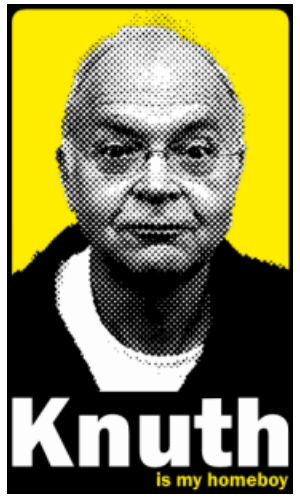
- Let this guide your organization. One potential structure for organizing a project:

```
project/
| data/
| | processing_scripts
| | raw/
| | proc/
| tools/
| | src/
| | bin/
| exps
| | pipeline_scripts
| | results/
| | analysis_scripts
| | figures/
```

Think Like An Experimentalist

- Keep a lab notebook!

- Literate programming tools are making this easier for computational projects:
  - http://en.wikipedia.org/wiki/Literate_programming (Lec #2!)
  - https://ipython.org/
  - http://rmarkdown.rstudio.com/
  - http://jupyter.org/

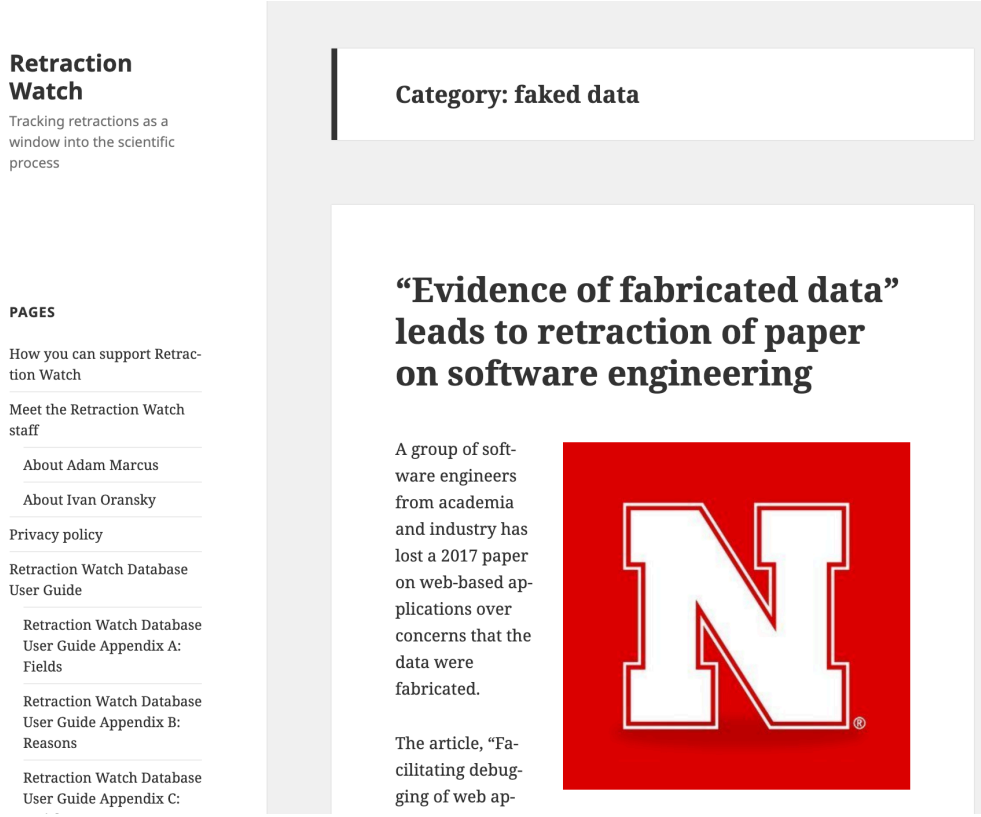


Think Like An Experimentalist

- Separate experiment from analysis from communication
  - Store results of computations
  - Write separate scripts to analyze results and make plots/tables
- **Aim for reproducibility!**

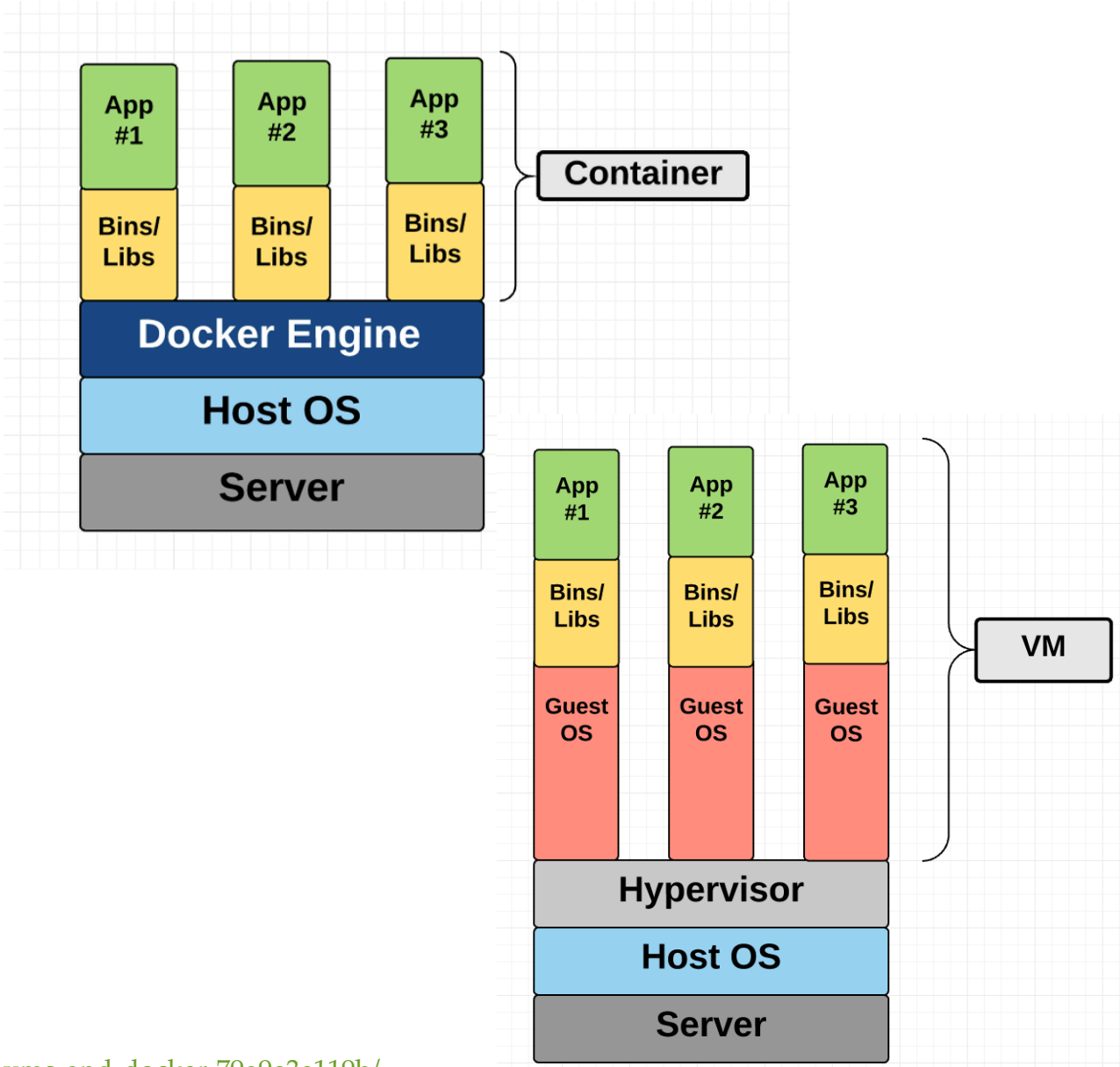
- There are serious consequences for not being careful
  - Publication retraction
  - Worse: http://videolectures.net/cancerbioinformatics2010_baggerly_irrh/

- Lots of tools available to help, use them! Be proactive: learn about them on your own!



Docker

- Docker is a tool for creating *containers* which allow you to easily distribute and scale code.
  - Like a VM it abstracts away the actual machine in order to increase portability.
  - A VM abstracts the hardware, kernel, and user space for every machine.
  - A container is more lightweight, only the binaries and libraries are are unique for each container.
- Both containers and VMs have private space for processing, can execute commands as root, have a private network interface and IP address, allow custom routes and iptable rules, can mount file systems, and etc…

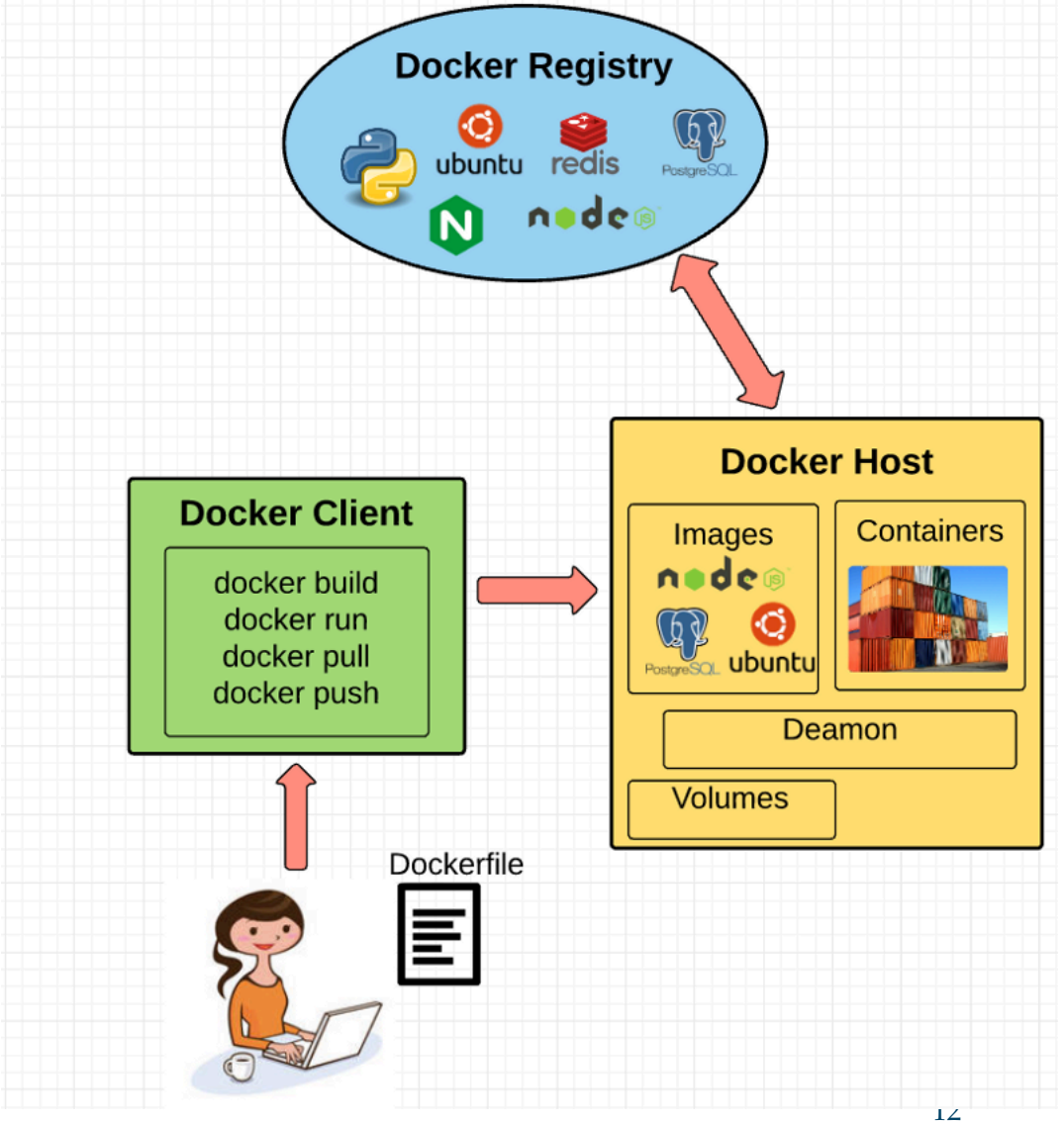


https://www.freecodecamp.org/news/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b/

Building Docker Files

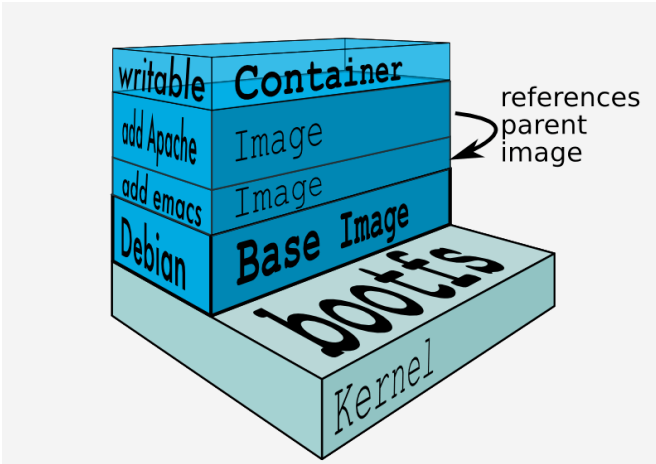
- A Docker container is built out of a Dockerfile
  - For Project0 we are using the jupyter/datascience-notebook docker image: https://hub.docker.com/r/jupyter/datascience-notebook/dockerfile
  - List of images: https://hub.docker.com/search/?type=image
  - A dockerfile contains a line by line what packages should be included in the *image* to build the container.
  - A dockerfile can stack on top of other images, for instance, the datascience-notebook is built on top of the jupyter-base notebook.

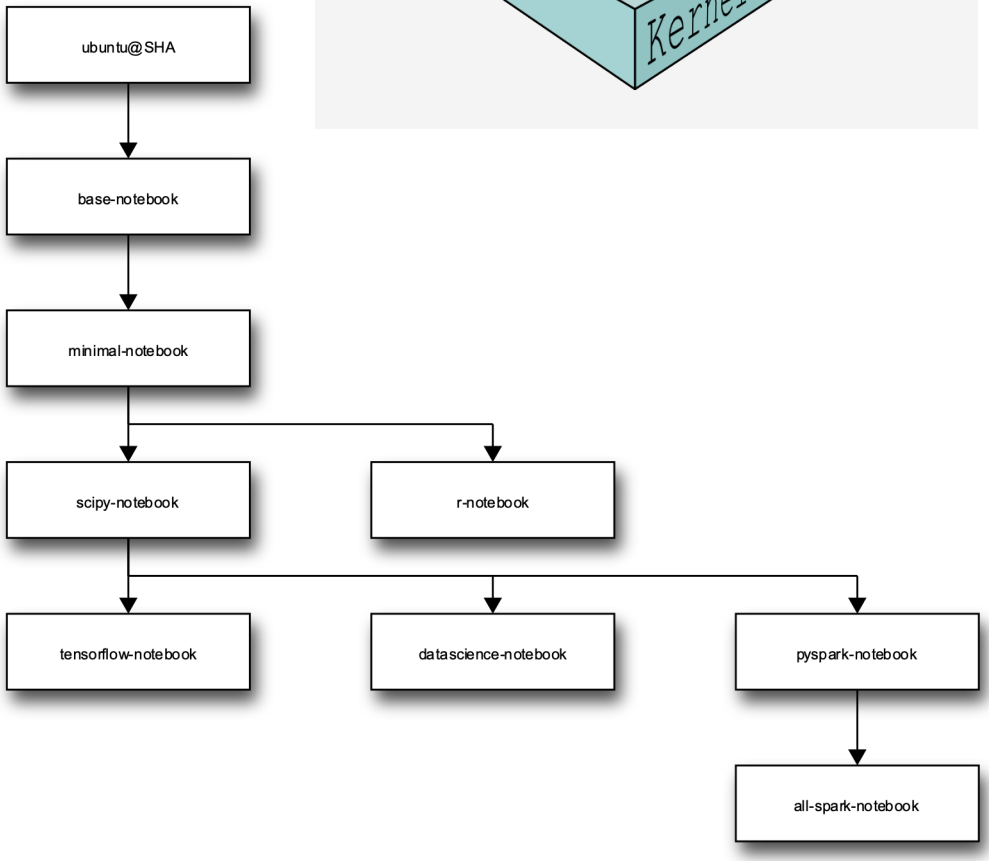
- Once the container is made it volumes are connected back to the host operating system to allow you to read and edit files.



Deep Dive – Jupyter Docker Files

- Project Jupyter maintains a set of Docker images for easy use of the notebook and related software.
  - More about Jupyter Images: https://jupyter-docker-stacks.readthedocs.io/en/latest/using/selecting.html
  - Looking at the base notebook stack we see it's built on Ubuntu https://github.com/jupyter/docker-stacks/blob/master/base-notebook/Dockerfile





134 lines (113 sloc) | 4.74 KB | Raw | Blame | History

```
# Copyright (c) Jupyter Development Team.
# Distributed under the terms of the Modified BSD License.

# Ubuntu 18.04 (bionic) from 2019-06-12
# https://github.com/tianon/docker-brew-ubuntu-core/commit/3c462555392cb188830b7c91e29311b5fad90cfe
ARG BASE_CONTAINER=ubuntu:bionic-20190612@sha256:9b1702dcfe32c873a770a32cfd306dd7fc1c4fd134adfb783db68defc8894b3c
FROM $BASE_CONTAINER
```

Unpacking a Docker Command

docker run -it -v /Users/nsmattei/project0:/home/jovyan/notebooks --rm -p 8888:8888 jupyter/datascience-notebook

- *docker run* the command to tell Docker to run a container.
- *-it* Since we are using a program that needs a shell, this tells Docker to give us an interactive terminal
- *-v /Users/nsmattei/project0:/home/jovyan/notebooks* mounts the current project0 directory on the guest OS, so that everything in project0 directory will be available in notebooks directory on the guest.
- *-rm* this tells Docker to clean up after we close the notebook terminal.
- *-p 8888:8888* maps the 8888 port on the host OS to the 8888 port on the guest container. So if you were to go to http://localhost:8888, it will redirect to the 8888 port on the container - Jupyter Notebook starts a web server on that port on the guest.
- *jupyter/datascience-notebook* tells Docker which container to load.

- *More details:* https://docs.docker.com/engine/reference/run/

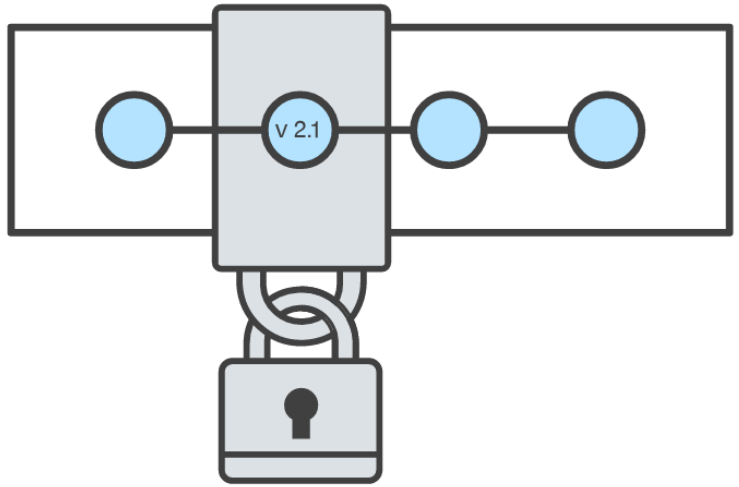
What is Version Control?

```
Aaron@HELIOS ~/112_term_project
$ ls
termproject_actually_final  termproject_v10  termproject_v3
termproject_final           termproject_v11  termproject_v4
termproject_handin          termproject_v12  termproject_v5
termproject_old_idea        termproject_v13  termproject_v6
termproject_superfrogger    termproject_v14  termproject_v7
termproject_temp            termproject_v15  termproject_v8
termproject_this_one_works  termproject_v16  termproject_v9
termproject_v1              termproject_v2
```

Goals of Version Control

- When working with a team, the need for a central repository is essential
  - Need a system to allow versioning, and a way to acquire the latest edition of the code
  - A system to track and manage bugs was also needed

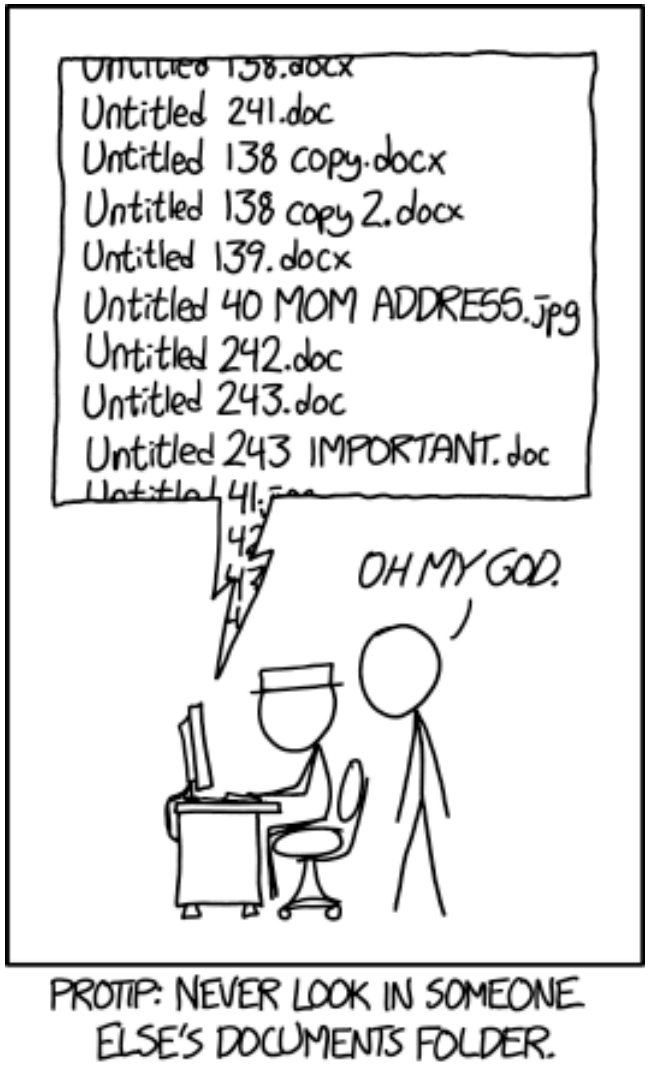
- Be able to search through revision history and retrieve previous versions of any file in a project
- Be able to share changes with collaborators on a project
- Be able to confidently make large changes to existing files



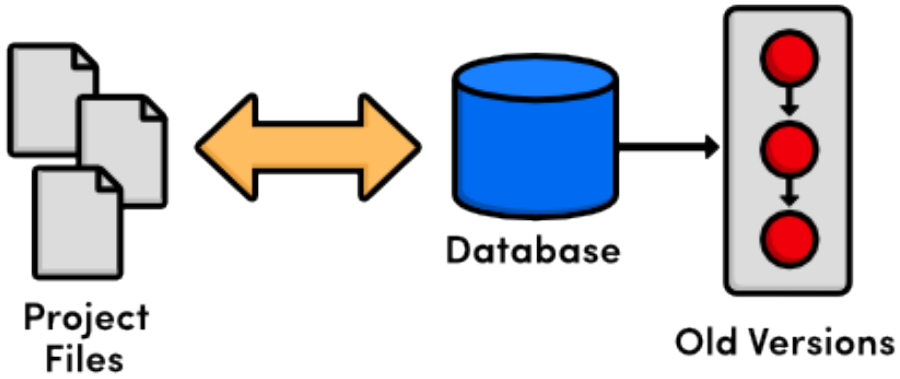
atlassian.com/git/tutorials/what-is-version-control

Named Folders Approach

- Can be hard to track
- Memory-intensive
- Can be slow
- Hard to share
- No record of authorship



Local Database of Versions Approach

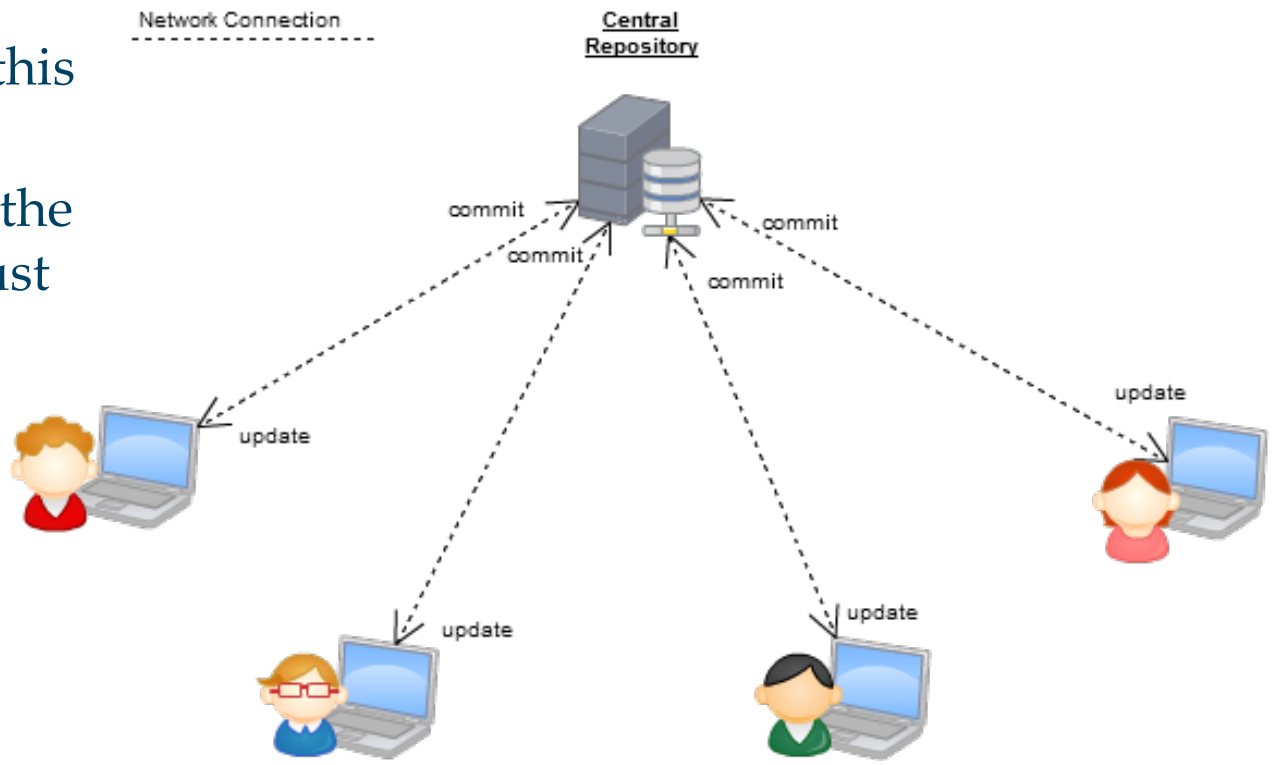


- Provides an abstraction over finding the right versions of files and replacing them in the project
- Records who changes what, but hard to parse that
- **Can't share with collaborators!!**

Centralized Version Control Systems

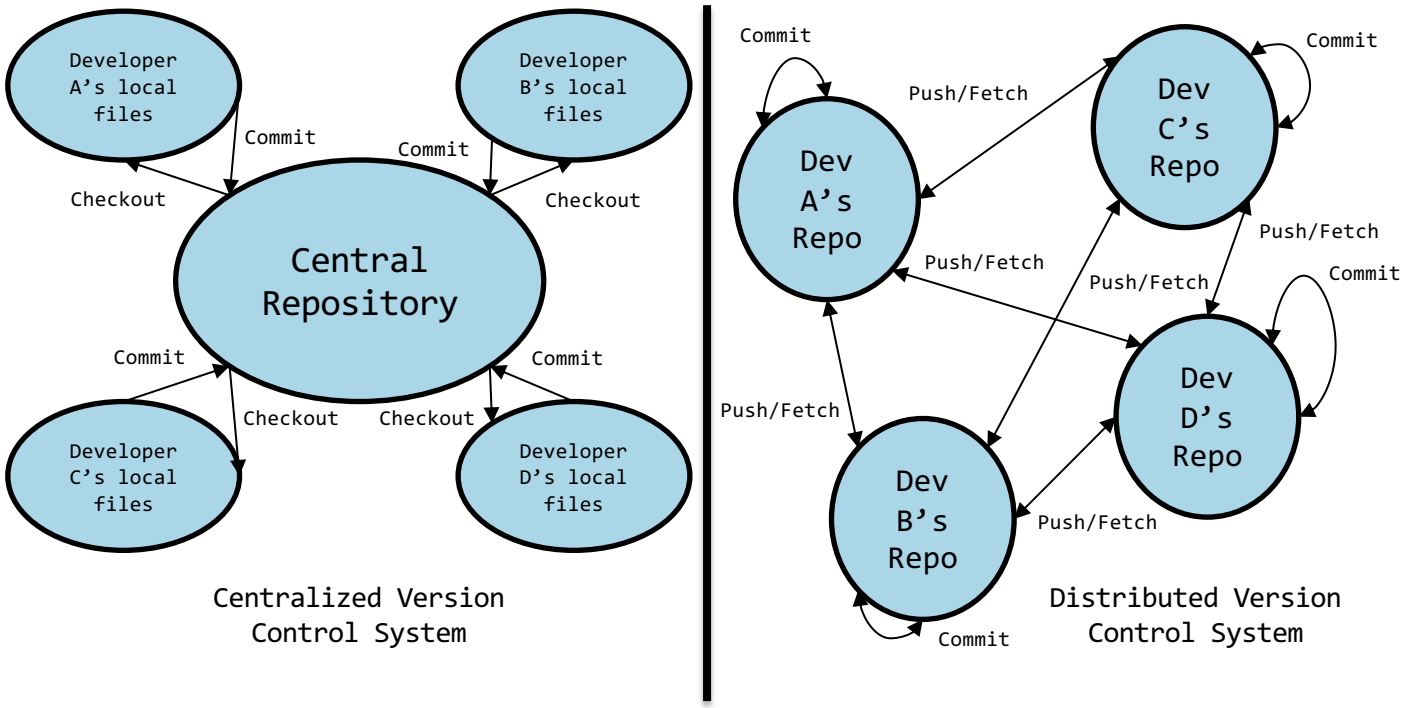
- A central, trusted repository determines the order of commits ("versions" of the project)
- Collaborators "push" changes (commits) to this repository.
- Any new commits must be compatible with the most recent commit. If it isn't, somebody must "merge" it in.

- Examples: SVN, CVS, Perforce



Distributed Version Control Systems (DVCS)

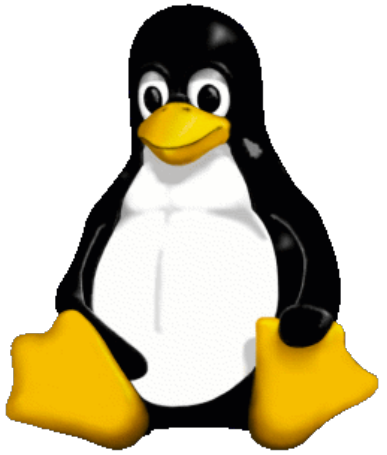
- No central repository
- Every repository has every commit
- Examples: Git, Mercurial



What is Git

- Git is a version control system
- Developed as a repository system for both local and remote changes
- Allows teammates to work simultaneously on a project
- Tracks each commit, allowing for a detailed documentation of the project along every step
- Allows for advanced merging and branching operations

A Short History of Git



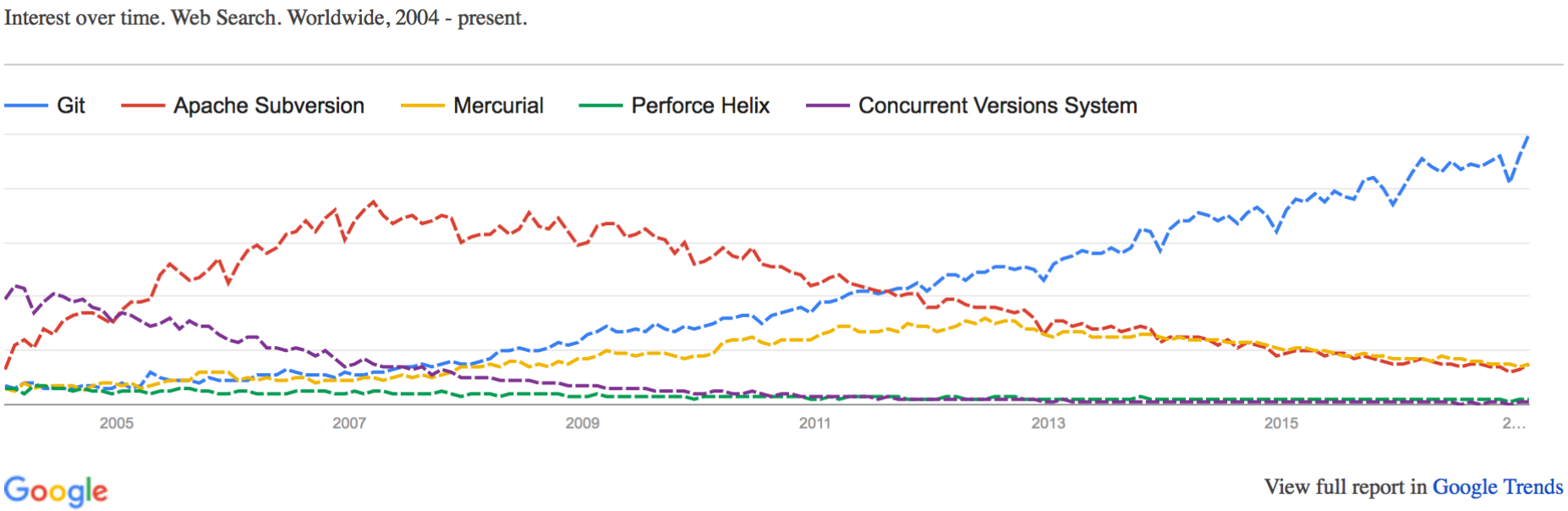
- Linux kernel development
- 1991-2002.
  - Changes passed around as archived file – PATCH files.
- 2002-2005.
  - Using a DVCS called BitKeeper
- 2005
  - Relationship broke down between two communities (BitKeeper licensing issues)

- **Goals of Git**
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - **Fully distributed** – not a requirement, can be centralized
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)

A short history of Git

- Popularity:
- Git is now the most widely used source code management tool
- 33.3% of professional software developers use Git (often through GitHub) as their primary source control system

[citation needed]



Git in Industry

- Companies and projects currently using Git
- Google
- Android
- Facebook
- Microsoft
- Netflix
- Linux
- Ruby on Rails
- Gnome
- KDE
- Eclipse
- X.org
- IBM

Git Basics

- **Snapshots**, not changes
  - A picture of what all your files look like at that moment
  - If a file has not changed, store a reference
- Nearly every operation is local
  - Browsing the history of project
  - See changes between two versions (diff)

Why Git is Better

- Git tracks the content rather than the files
- Branches are lightweight, and merging is a simple process
- Allows for a more streamlined offline development process
- Repositories are smaller in size and are stored in a single .git directory
- Allows for advanced staging operations, and the use of stashing when working through troublesome sections

What about SVN?

> Subversion has been the most pointless project ever started … Subversion used to say CVS done right: with that slogan there is nowhere you can go. There is no way to do CVS right … If you like using CVS, you should be in some kind of mental institution or somewhere else.

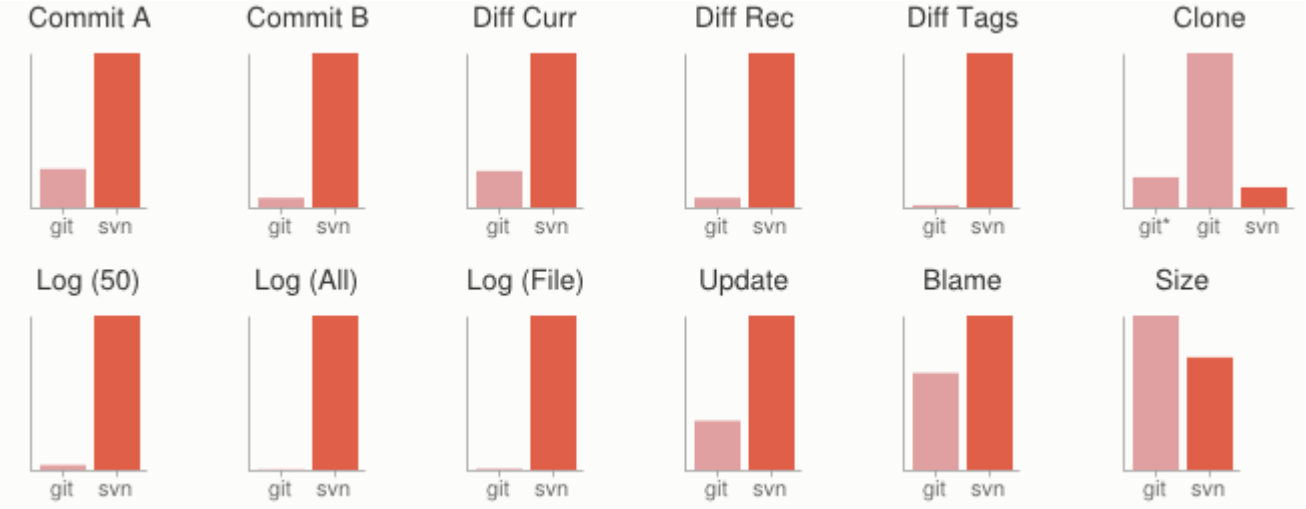
Linus Torvalds

Git vs {CVS, SVN, …}

- Why you should care:
  - Many places use legacy systems that will cause problems in the future – be the change you believe in!
- Git is **much** faster than SVN:
  - Coded in C, which allows for a great amount of optimization
  - Accomplishes much of the logic client side, thereby reducing time needed for communication
  - Developed to work on the Linux kernel, so that large project manipulation is at the forefront of the benchmarks

Git vs {CVS, SVN, …}

- Speed benchmarks:



Benchmarks performed by http://git-scm.com/about/small-and-fast

Git vs {CVS, SVN, …}

- Git is significantly smaller than SVN
  - All files are contained in a small decentralized .git file
  - In the case of Mozilla's projects, a Git repository was 30 times smaller than an identical SVN repository
  - Entire Linux kernel with 5 years of versioning contained in a single 1 GB .git file
  - SVN carries two complete copies of each file, while Git maintains a simple and separate 100 bytes of data per file, noting changes and supporting operations

- Nice because you can (and do!) store the whole thing locally

Git vs {CVS, SVN, …}

- Git is **more secure** than SVN
  - All commits are uniquely hashed for both security and indexing purposes
  - Commits can be authenticated through numerous means
  - In the case of SSH commits, a key may be provided by both the client and server to guarantee authenticity and prevent against unauthorized access

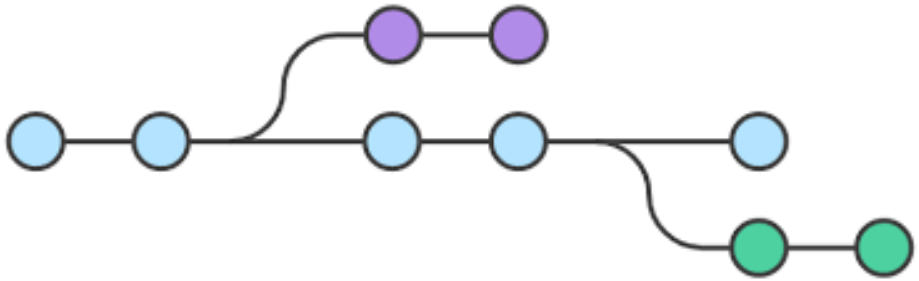
- Git is **decentralized**:
  - Each user contains an individual repository and can check commits against itself, allowing for detailed local revisioning
  - Being decentralized allows for easy replication and deployment
  - In this case, SVN relies on a single centralized repository and is unusable without a connection to this repository!

Git vs {CVS, SVN, …}

- Git is **flexible**:
  - Due to it's decentralized nature, git commits can be stored locally, or committed through HTTP, SSH, FTP, or even by Email
  - No need for a centralized repository
  - Developed as a command line utility, which allows a large amount of features to be built and customized on top of it
- **Data assurance:** a checksum is performed on both upload and download to ensure sure that the file hasn't been corrupted.
  - Commit IDs are generated upon each commit:
  - Linked list style of commits
  - Each commit is linked to the next, so that if something in the history was changed, each following commit will be rebranded to indicate the modification

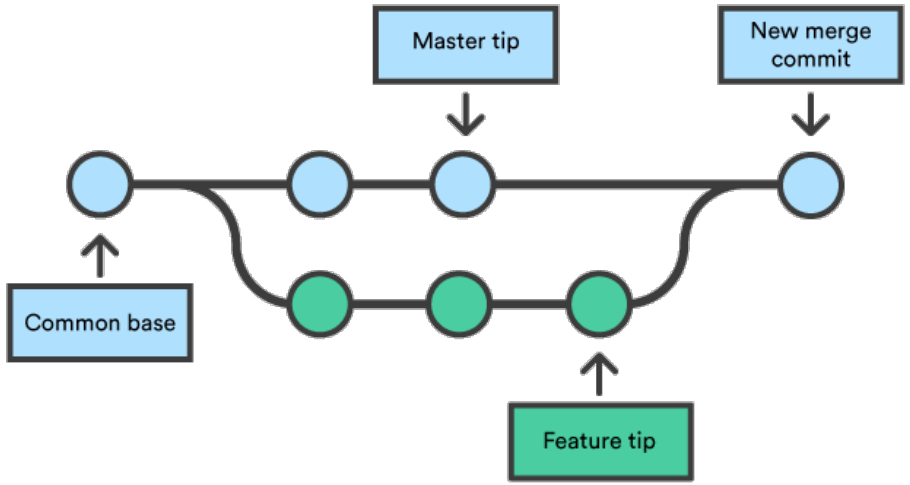
Git vs {CVS, SVN, …}

- **Branching**
  - Git allows the usage of advanced **branching** mechanisms and procedures
  - Individual divisions of the code can be separated and developed separately within separate branches of the code
  - Branches can allow for the separation of work between developers, or even for disposable experimentation
  - Branching is a precursor and a component of the merging process
- **Merging**
  - The process of merging is directly related to the process of branching
  - Individual branches may be merged together, solving code conflicts, back into the default or master branch of the project
  - Merges are usually done automatically, unless a conflict is presented, in which case the user is presented with several options with which to handle the conflict



Git vs {CVS, SVN, …}

- **Merging** – The big differences…
  - Content of the files is tracked rather than the file itself
  - This allows for a greater element of tracking and a smarter and more automated process of merging
  - SVN is unable to accomplish this, and will throw a conflict if, e.g., a file name is changed and differs from the name in the central repository
  - Git is able to solve this problem with its use of managing a local repository and tracking individual changes to the code



Initialization of a Git repository

```
C:\> mkdir CoolProject
C:\> cd CoolProject
C:\CoolProject > git init
Initialized empty Git repository in
C:/CoolProject/.git
C:\CoolProject > notepad README.txt
C:\CoolProject > git add .
C:\CoolProject > git commit -m 'my first
commit'
[master (root-commit) 7106a52] my first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.txt
```

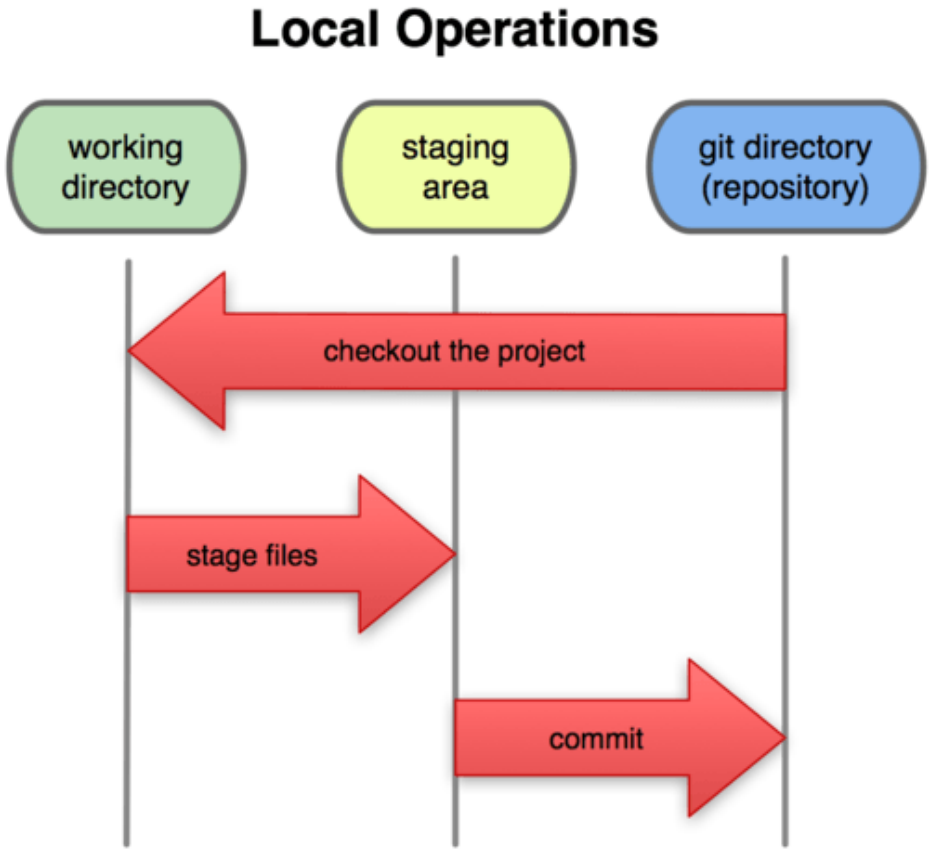
Git Basics I

- The three (or four) states of a **file**:
- **Modified:**
  - File has changed but not committed
- **Staged:**
  - Marked to go to next commit snapshot
- **Committed:**
  - Safely stored in local database
- **Untracked!**
  - Newly added or removed files

```
nsmattei@MatteiMac:~/repo/github/review_market$ git status
On branch master
Your branch is behind 'origin/master' by 29 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   allocation/cap_discrete_alloc.py
        modified:   allocation/utility_functions.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        allocation/tmp/

no changes added to commit (use "git add" and/or "git commit -a")
```

Git Basics II

- Three main areas of a git **project**:
- **Working directory**
  - Single checkout of one version of the project.
- **Staging area**
  - Simple file storing information about what will go into your next commit
- **Git directory**
  - What is copied when cloning a repository

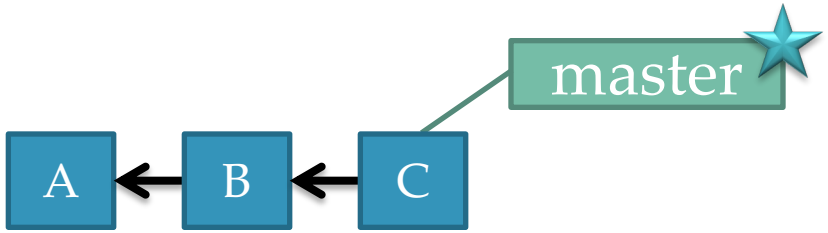


Branches Illustrated

(Default branch is called “master”; your first commit will be on this branch.)

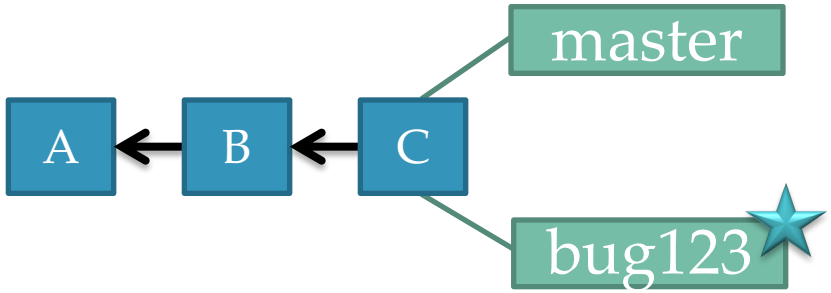
```
> git commit –m ‘my first commit’
```

Branches Illustrated



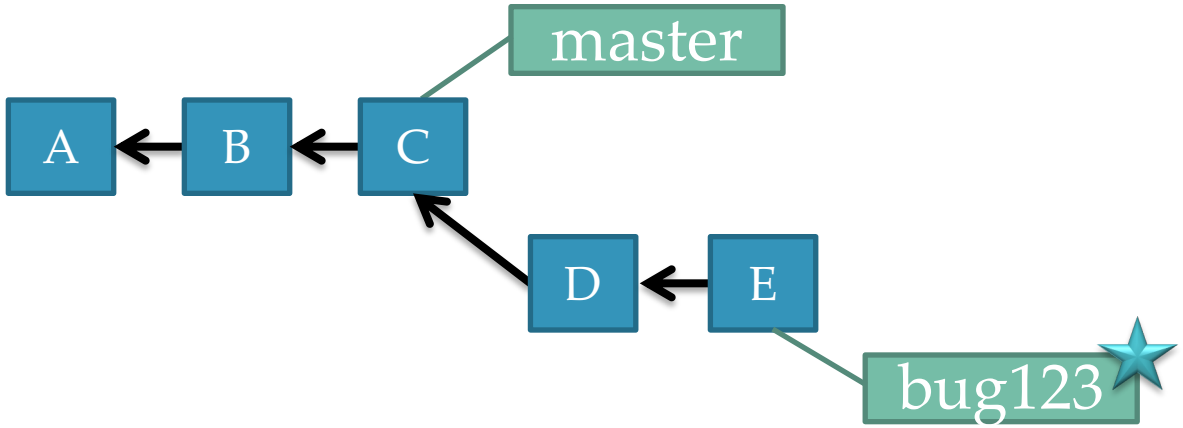

```
> git commit (x2)
```

Branches Illustrated



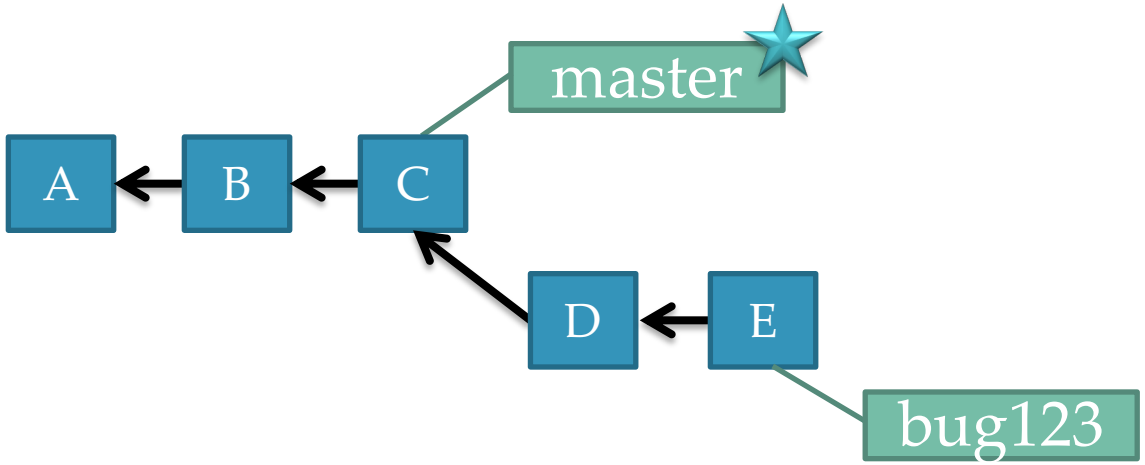

```
> git checkout -b bug123
```

Branches Illustrated



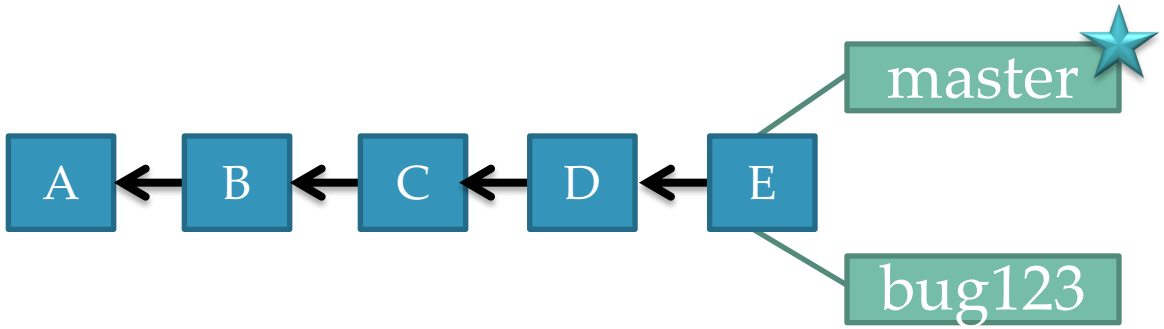

```
> git commit (x2)
```

Branches Illustrated



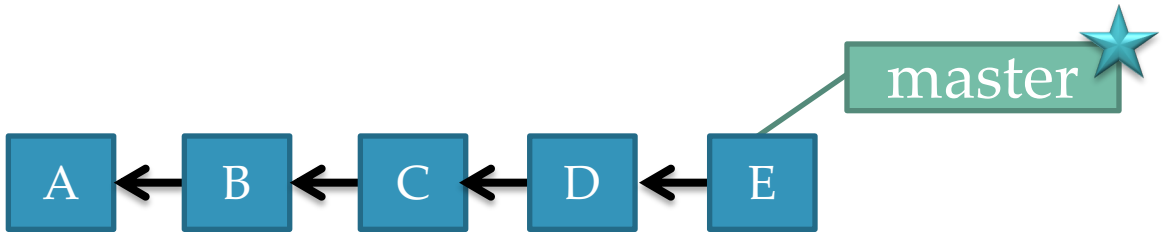

```
> git checkout master
```

Branches Illustrated



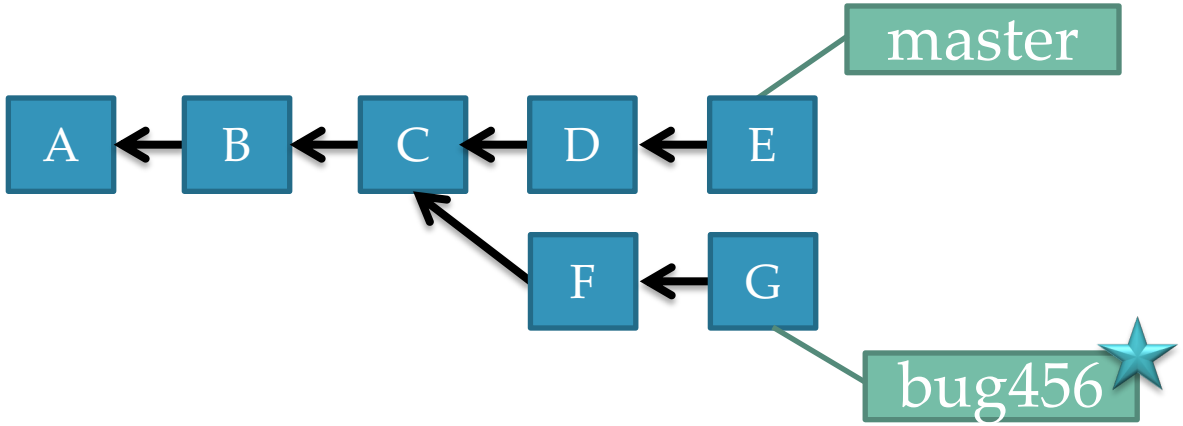

```
> git merge bug123
```

Branches Illustrated

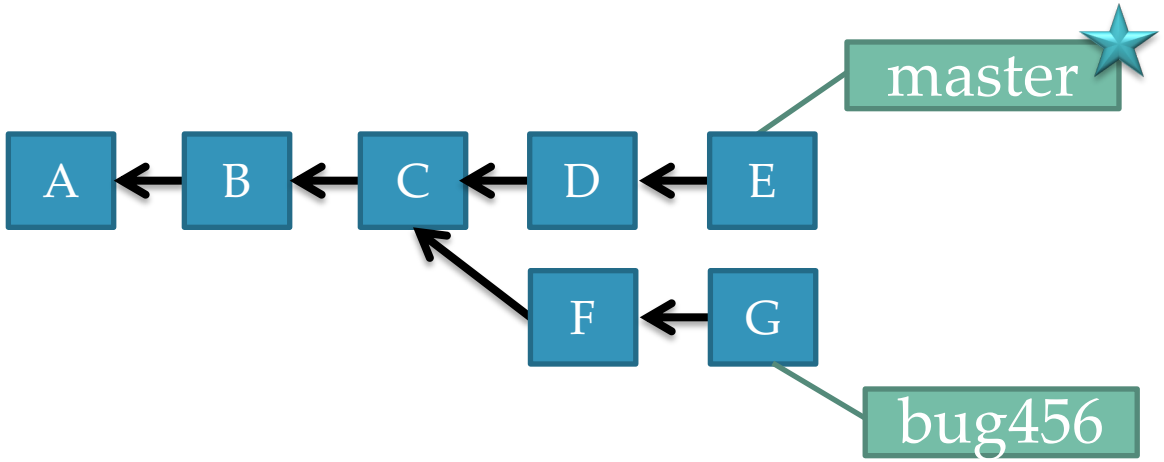



```
> git branch -d bug123
```

Branches Illustrated

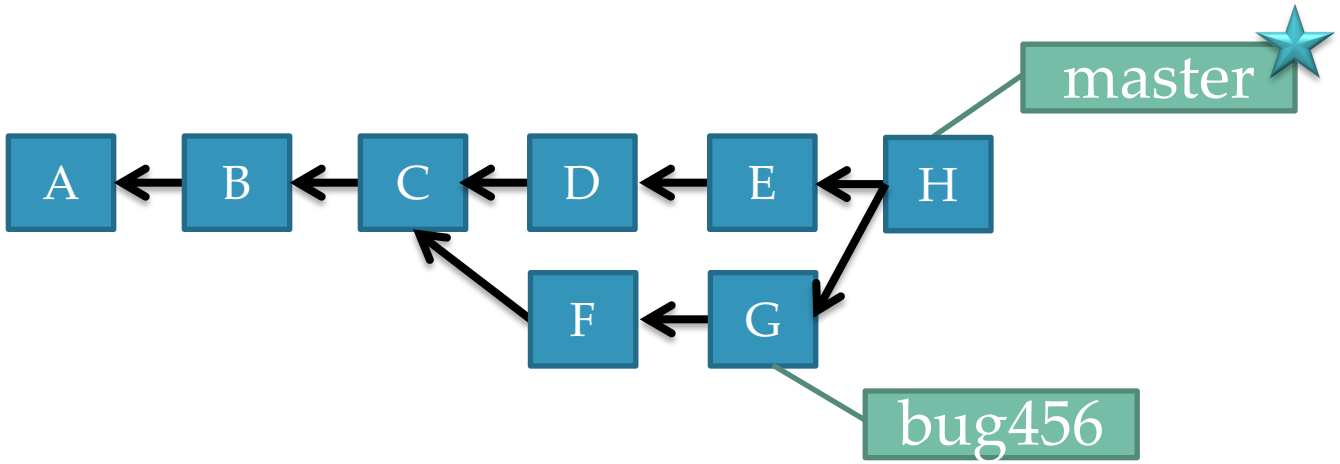


Branches Illustrated



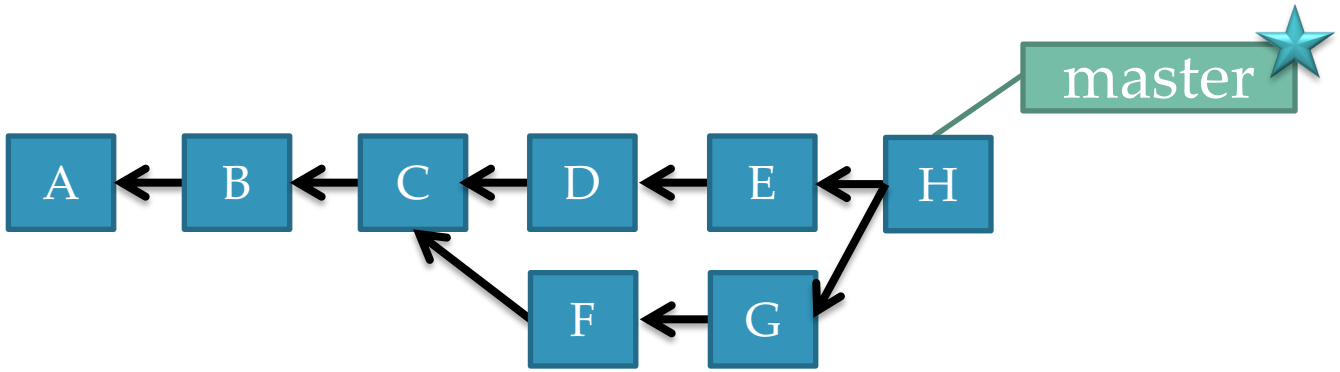

```
> git checkout master
```

Branches Illustrated



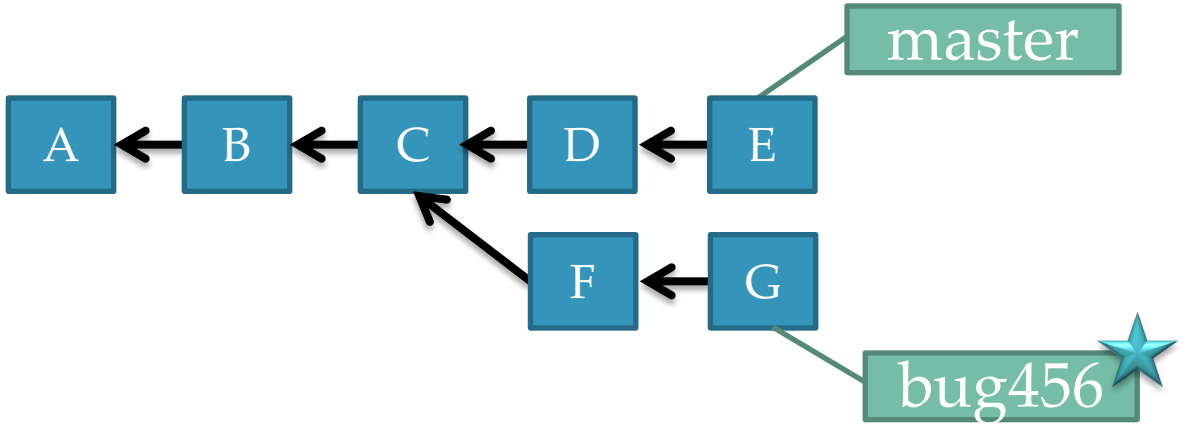

```
> git merge bug456
```

Branches Illustrated

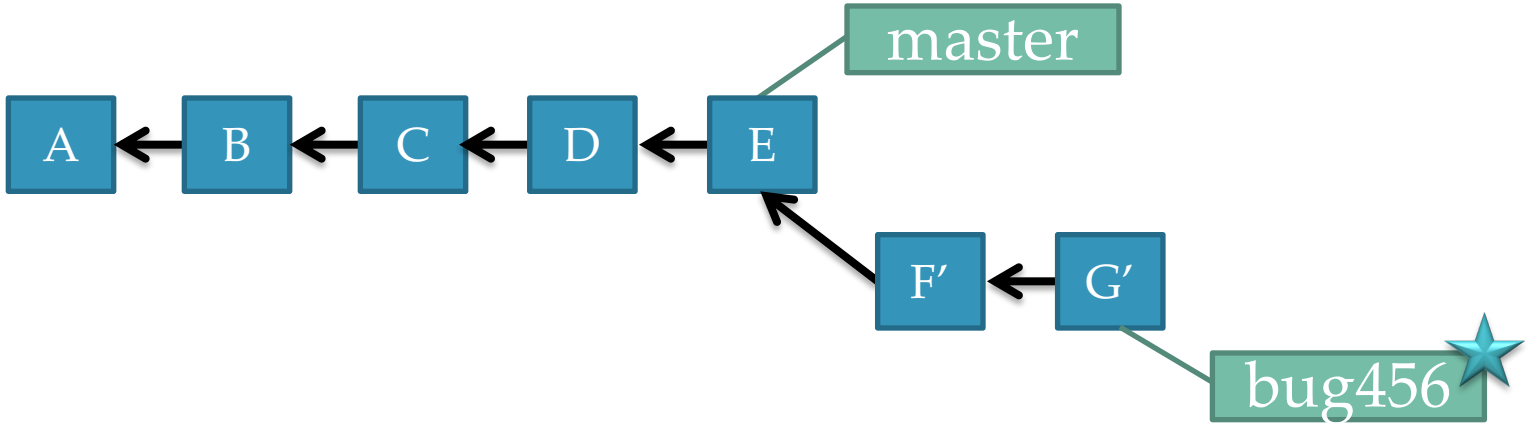



```
> git branch -d bug456
```

Branches Illustrated

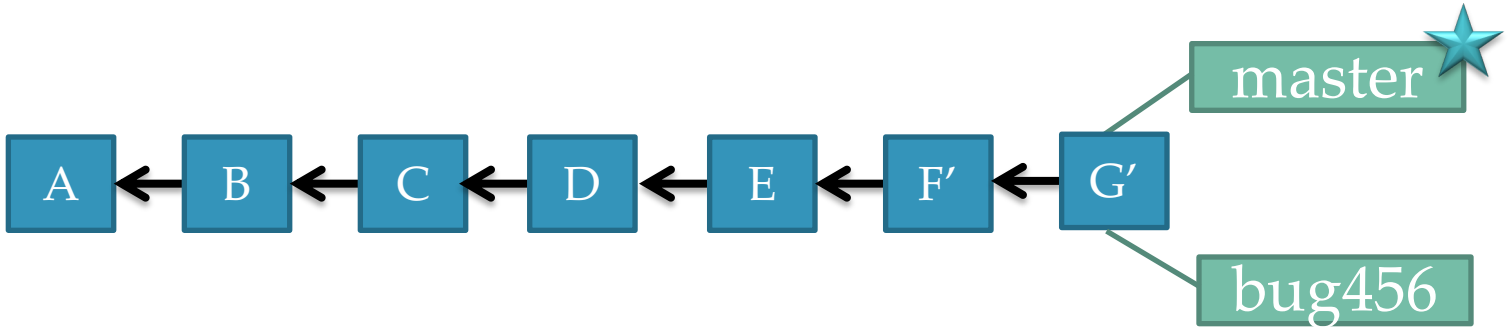


Branches Illustrated



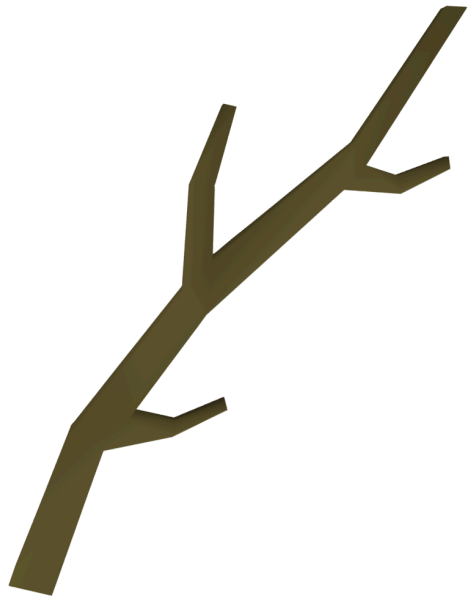

```
> git rebase master
```

Branches Illustrated




```
> git checkout master
> git merge bug456
```

When to Branch?



- General rule of thumb:
  - **Anything in the master branch is always deployable.**
- Local branching is very lightweight!
  - New feature? Branch!
  - Experiment that you won't ever deploy? Branch!
- Good habits:
  - Name your branch something descriptive (`add-like-button`, `refactor-jobs`, `create-ai-singularity`)
  - Make your commit messages descriptive, too!

So you want somebody else to host this for you …

- Git: general distributed version control system
- GitHub / BitBucket / GitLab / …: **hosting** services for git repositories
- In general, GitHub is the most popular:
- Lots of big projects (e.g., Python, Bootstrap, Angular, D3, node, Django, Visual Studio)
- Lots of ridiculously awesome projects (e.g., https://github.com/maxbbraun/trump2cash)
- There are reasons to use the competitors (e.g., private repositories, access control)

Bitbucket

Social Coding

Set status

Nicholas Mattei
nmattei

PRO

Edit profile

Tulane University
nsmattei@gmail.com
http://www.nickmattei.net

Organizations

Overview | Repositories 16 | Projects 0 | Stars 11 | Followers 9 | Following 8

Pinned — Customize your pins

PrefLib-Tools
A small and lightweight set of Python tools for working with and generating data from www.PrefLib.org.
Python ★ 13 5

GenCPnet
Code to generate CP-nets uniformly at random. Can provide bounds on in-degree and other useful properties.
C++

peerselection
Implementations of Stragegyproof Peer Selection Mechanisms
Python

PrefLib-www
All data, code, and pages for the PrefLib website.
PHP 1

ShapleyTSG
Data and Code from our paper, "A Study of Proxies for Shapley Allocations of Transport Costs."

InterdependentSchedulingGames
Code from our IJCAI 2016 paper on Interdependent Scheduling Games
Jupyter Notebook

98 contributions in the last year — Contribution settings

Sep Oct Nov Dec Jan Feb Mar Apr May Jun Jul Aug

Learn how we count contributions. Less More

Review: How to Use

- Git commands for everyday usage are relatively simple
- git pull
  - Get the latest changes to the code
- git add .
  - Add any newly created files to the repository for tracking
- git add –u
  - Remove any deleted files from tracking and the repository
- git commit –m ‘Changes’
  - Make a version of changes you have made
- git push
  - Deploy the latest changes to the central repository
- Make a repo on GitHub and clone it to your machine:
  - https://guides.github.com/activities/hello-world/

Stuff to click on

- Git
  - http://git-scm.com/
- GitHub
  - https://github.com/
  - https://guides.github.com/activities/hello-world/
  - ^-- Just do this one. You'll need it for your tutorial ☺.
- GitLab
  - http://gitlab.org/
- Git and SVN Comparison
  - https://git.wiki.kernel.org/index.php/GitSvnComparison
- BitBucket and Sourcetree
  - https://www.sourcetreeapp.com/