INTRODUCTION TO DATA SCIENCE

JOHN P DICKERSON

PREM SAGGAR

Lecture #4 – 9/10/2018

CMSC320
Mondays & Wednesdays
2:00pm – 3:15pm

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

ANNOUNCEMENTS

Register on Piazza: piazza.com/umd/fall2018/cmsc320

- 228 have registered already?! ♡
- -3 have not registered yet?! 💔

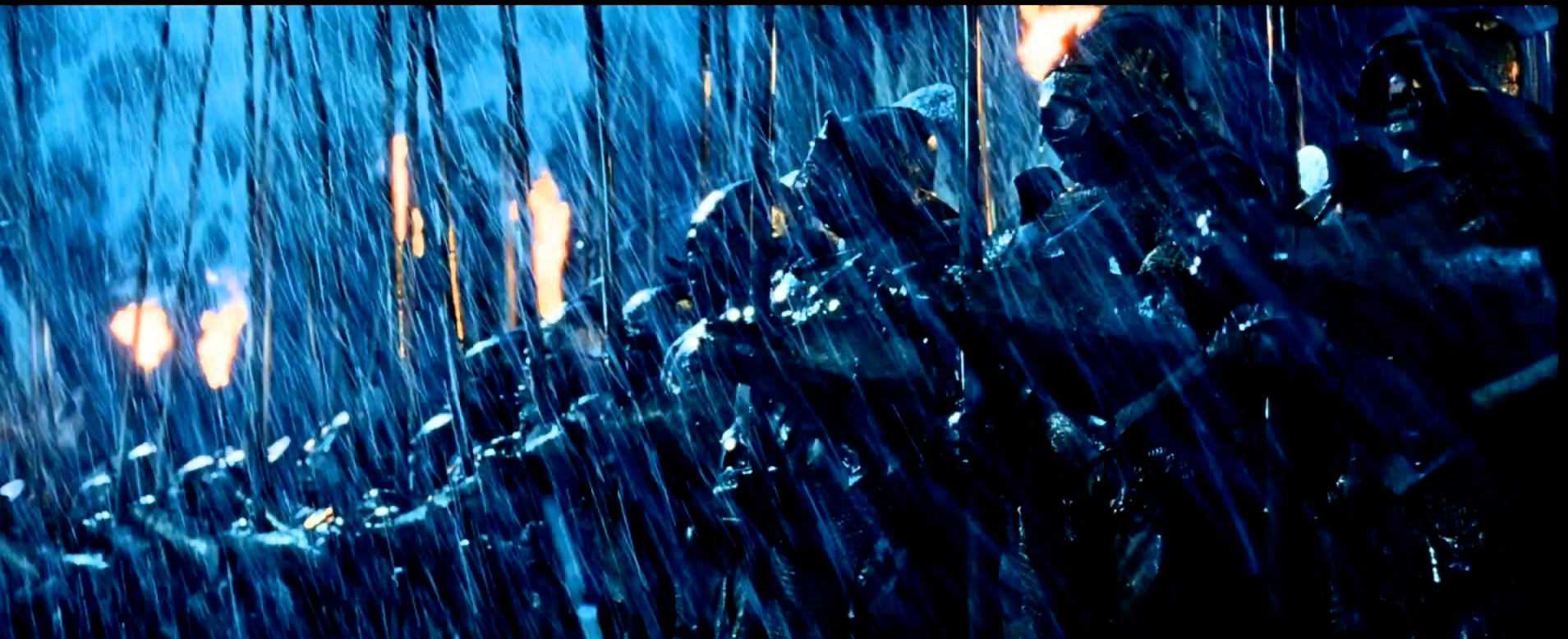
We will release the first mini-project this week.

- Please make sure Jupyter installed correctly!
- (See any of us if it didn't.)

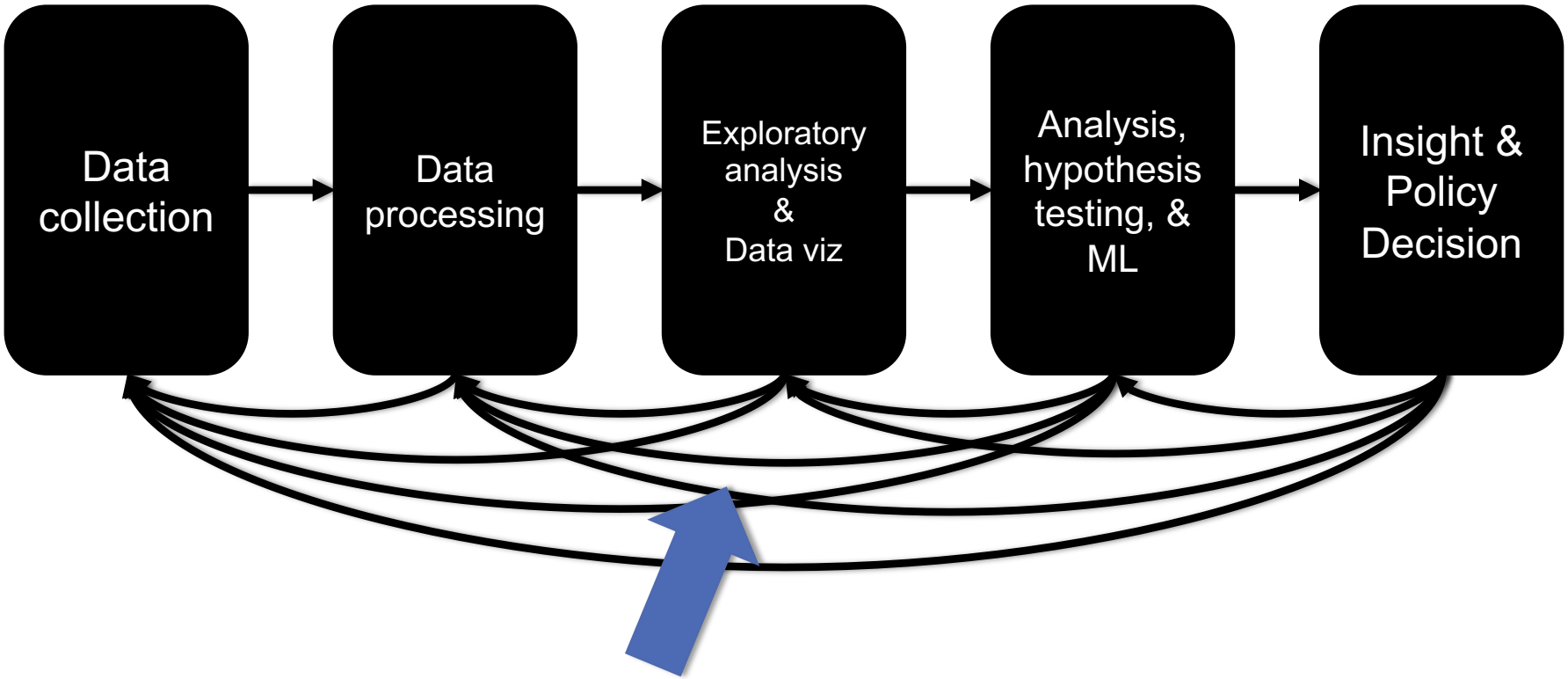
WAITLIST UPDATE

“Sure, we can expand it. Will take care of it. --Mike [Hicks]”

Waitlist should clear soon – if not, talk to me.



TODAY'S LECTURE



Best practices
for managing this monstrosity.

REPRODUCIBILITY

Extremely important aspect of data analysis

- "Starting from the same raw data, can we reproduce your analysis and obtain the same results?"

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:
"**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 movement website

- http://www.opendatafoundation.org/

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
- 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!

BIAS, ETHICS, & RESPONSIBILITY

DATA SCIENCE LIFECYCLE: AN ALTERNATE VIEW

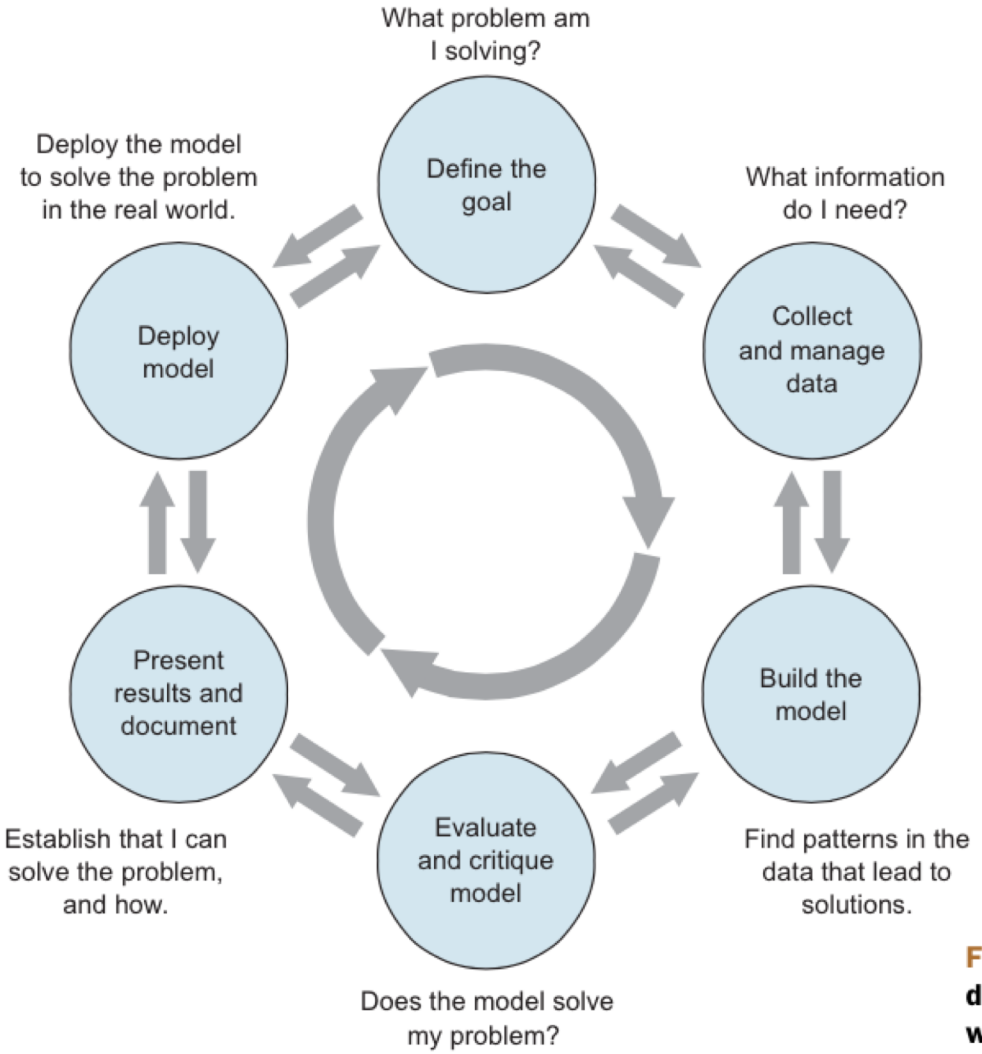


Figure 1.1 The lifecycle of a data science project: loops within loops

EXAMPLES OF BIAS

Genetic testing

- Genetic tests for heart disorder and race-biased risk (NYTimes)
- Race-bias in ancestry reports

Search results / feed optimization

- Google
- Facebook

COMBATING BIAS

Fairness through blindness:

- Don’t let an algorithm look at protected attributes

Examples currently in use ??????????

- Race
- Gender
- Sexuality
- Disability
- Religion

Problems with this approach ?????????

COMBATING BIAS

Demographic parity:

- A decision must be independent of the protected attribute
- E.g., a loan application's acceptance rate is independent of an applicant's race (but can be depenedent on non-protected features like salary)

Formally: binary decision variable C, protected attribute A

- P{ C = 1 | A = 0 } = P{ C = 1 | A = 1 }

Membership in a protected class should have no correlation with the final decision.

- Problems ????????

Example from Moritz Hardt's blog

COMBATING BIAS

What if the decision isn't the thing that matters?

"Consider, for example, a luxury hotel chain that renders a promotion to a subset of wealthy whites (who are likely to visit the hotel) and a subset of less affluent blacks (who are unlikely to visit the hotel). The situation is obviously quite icky, but demographic parity is completely fine with it so long as the same fraction of people in each group see the promotion."

Demographic parity allows classifiers that select qualified candidates in the "majority" demographic and unqualified candidate in the "minority" demographic, within a protected attribute, so long as the expected percentages work out.

More: http://blog.mrtz.org/2016/09/06/approaching-fairness.html

Example from Moritz Hardt's blog

FATML

This stuff is really tricky (and really important).

- It's also not solved, even remotely, yet!
- CMSC498/499 ♡

New community: Fairness, Accountability, and Transparency in Machine Learning (aka **FATML**)

"… policymakers, regulators, and advocates have expressed fears about the potentially discriminatory impact of machine learning, with many calling for further technical research into the dangers of inadvertently encoding bias into automated decisions."

F IS FOR FAIRNESS

In large data sets, there is always proportionally less data available about minorities.

Statistical patterns that hold for the majority may be invalid for a given minority group.

Fairness can be viewed as a measure of diversity in the combinatorial space of sensitive attributes, as opposed to the geometric space of features.

Thanks to: Faez Ahmed

A IS FOR ACCOUNTABILITY

Accountability of a mechanism implies an obligation to report, explain, or justify algorithmic decision-making as well as mitigate any negative social impacts or potential harms.

- Current accountability tools were developed to oversee human decision makers
- They often fail when applied to algorithms and mechanisms instead

Example, no established methods exist to judge the intent of a piece of software. Because automated decision systems can return potentially incorrect, unjustified or unfair results, additional approaches are needed to make such systems accountable and governable.

Thanks to: Faez Ahmed

T IS FOR TRANSPARENCY

Automated ML-based algorithms make many important decisions in life.

- Decision-making process is opaque, hard to audit

A transparent mechanism should be:

- understandable;
- more meaningful;
- more accessible; and
- more measurable.

Thanks to: Faez Ahmed

DATA COLLECTION

What data should (not) be collected

Who owns the data

Whose data can (not) be shared

What technology for collecting, storing, managing data

Whose data can (not) be traded

What data can (not) be merged

What to do with prejudicial data

Thanks to: Kaiser Fung

DATA MODELING

Data is biased (known/unknown)

- Invalid assumptions
- Confirmation bias

Publication bias

- WSDM 2017: https://arxiv.org/abs/1702.00502

Badly handling missing values

Thanks to: Kaiser Fung

DEPLOYMENT

Spurious correlation / over-generalization

Using “black-box” methods that cannot be explained

Using heuristics that are not well understood

Releasing untested code

Extrapolating

Not measuring lifecycle performance (concept drift in ML)

We will go over ways to counter this in the ML/stats/hypothesis testing portion of the course

Thanks to: Kaiser Fung

GUIDING PRINCIPLES

Start with clear user need and public benefit

Use data and tools which have minimum intrusion necessary

Create robust data science models

Be alert to public perceptions

Be as open and accountable as possible

Keep data secure

Thanks to: UK cabinet office

SOME REFERENCES

Presentation on ethics and data analysis, Kaiser Fung @ Columbia Univ. http://andrewgelman.com/wp-content/uploads/2016/04/fung_ethics_v3.pdf

O'Neil, Weapons of math destruction. https://www.amazon.com/Weapons-Math-Destruction-Increases-Inequality/dp/0553418815

UK Cabinet Office, Data Science Ethical Framework. https://www.gov.uk/government/publications/data-science-ethical-framework

Derman, Modelers' Hippocratic Oath. http://www.iijournals.com/doi/pdfplus/10.3905/jod.2012.20.1.035

Nick D's MIT Tech Review Article. https://www.technologyreview.com/s/602933/how-to-hold-algorithms-accountable/

REST OF TODAY'S LECTURE

By popular request …

- **Version control primer!**
- **Specifically, git via GitHub and GitLab**
- **Thanks: Mark Groves (Microsoft), Ilan Biala & Aaron Perley (CMU), Sharif U., & the HJCB Senior Design Team!**

And then a bit on keeping your data … tidy data.

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
```

DEVELOPMENT TOOL

When working with a team, the need for a central repository is essential

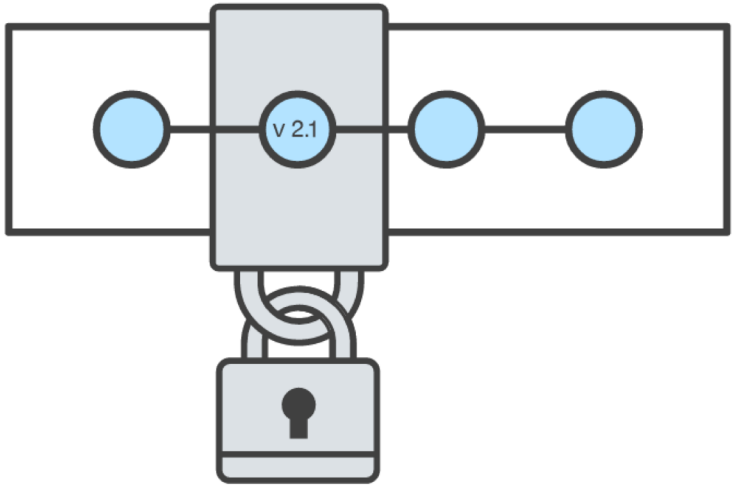
- Need a system to allow versioning, and a way to acquire the latest edition of the code
- A system to track and manage bugs was also needed

GOALS OF VERSION CONTROL

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

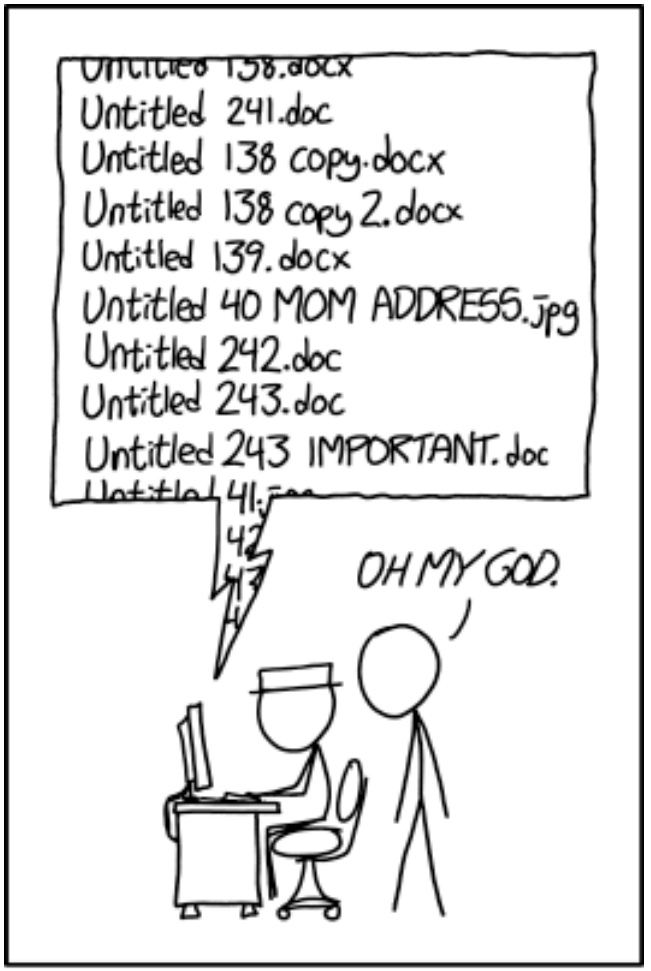
Can be hard to track

Memory-intensive

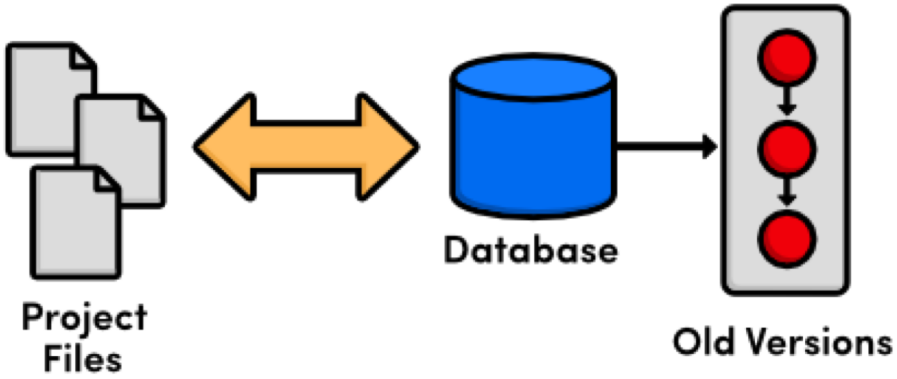
Can be slow

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

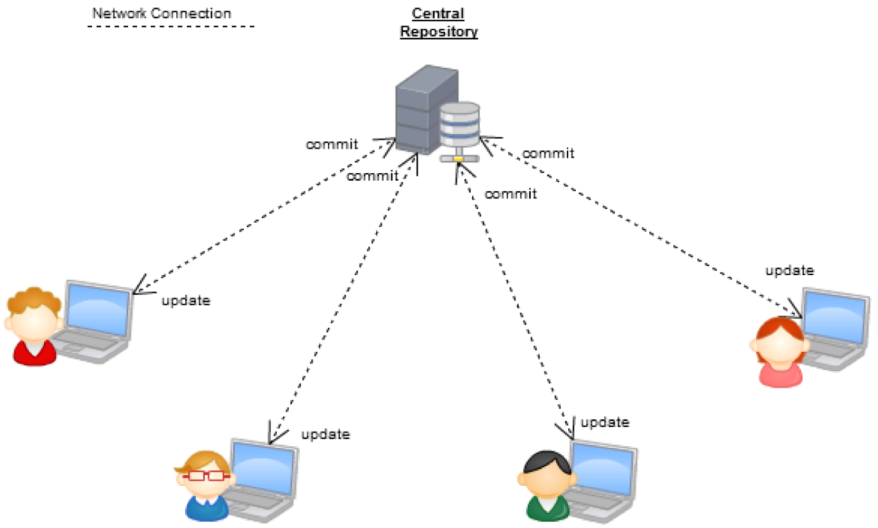
Can’t share with collaborators

CENTRALIZED VERSION CONTROL SYSTEMS

A central, trusted repository determines the order of commits ("versions" of the project)

Collaborators "push" changes (commits) to this repository.

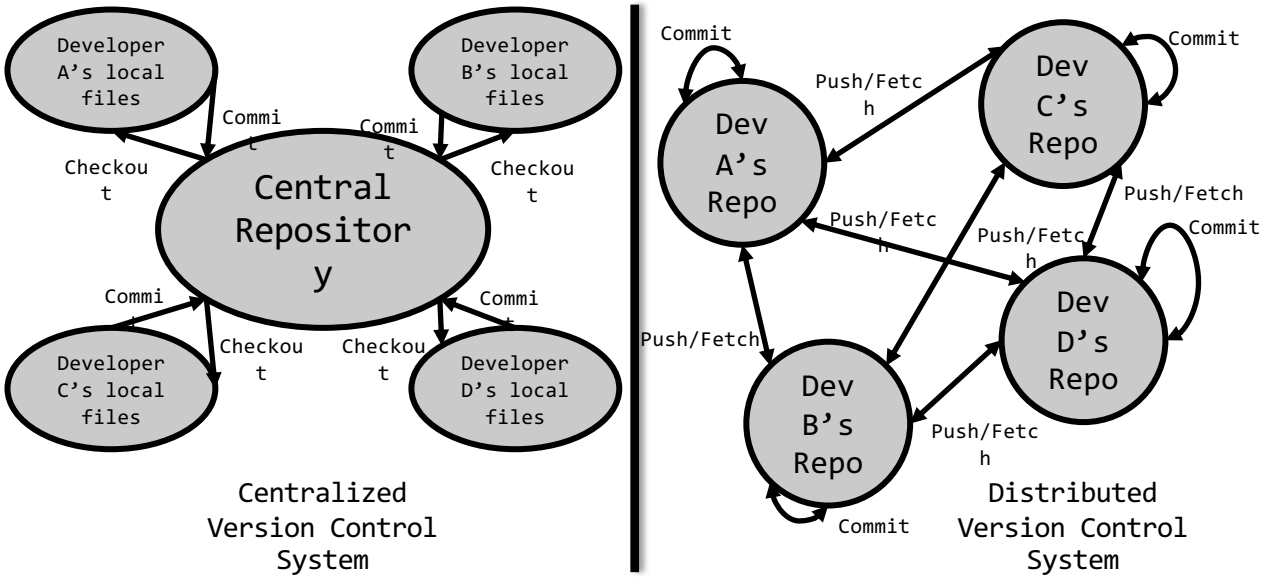
Any new commits must be compatible with the most recent commit. If it isn't, somebody must "merge" it in.



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

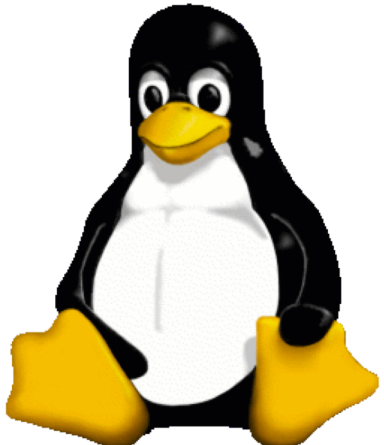
- Changes passed around as archived file

2002-2005

- Using a DVCS called BitKeeper

2005

- Relationship broke down between two communities (BitKeeper licensing issues)



A SHORT HISTORY OF GIT

Goals:

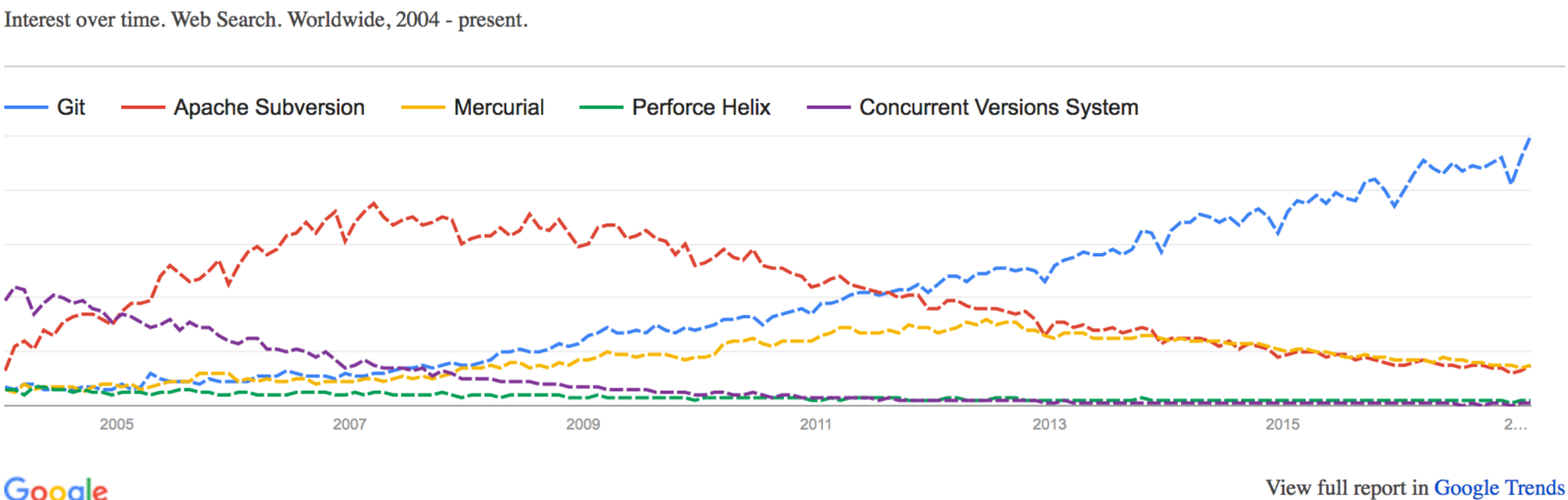
- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed – not a requirement, can be centralized
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

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

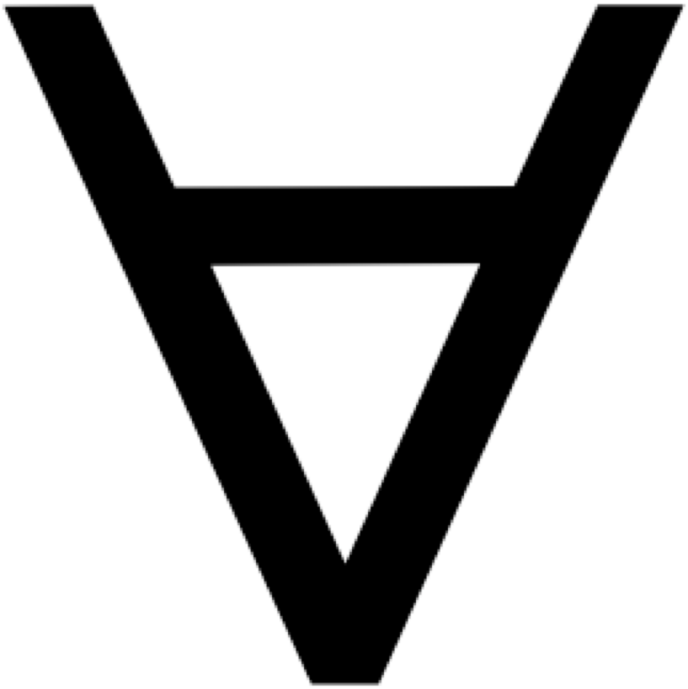
[citation needed]



GIT IN INDUSTRY

Companies and projects currently using Git

- Google
- Android
- Facebook
- Microsoft
- Netflix
- Linux
- Ruby on Rails
- Gnome
- KDE
- Eclipse
- X.org



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

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:

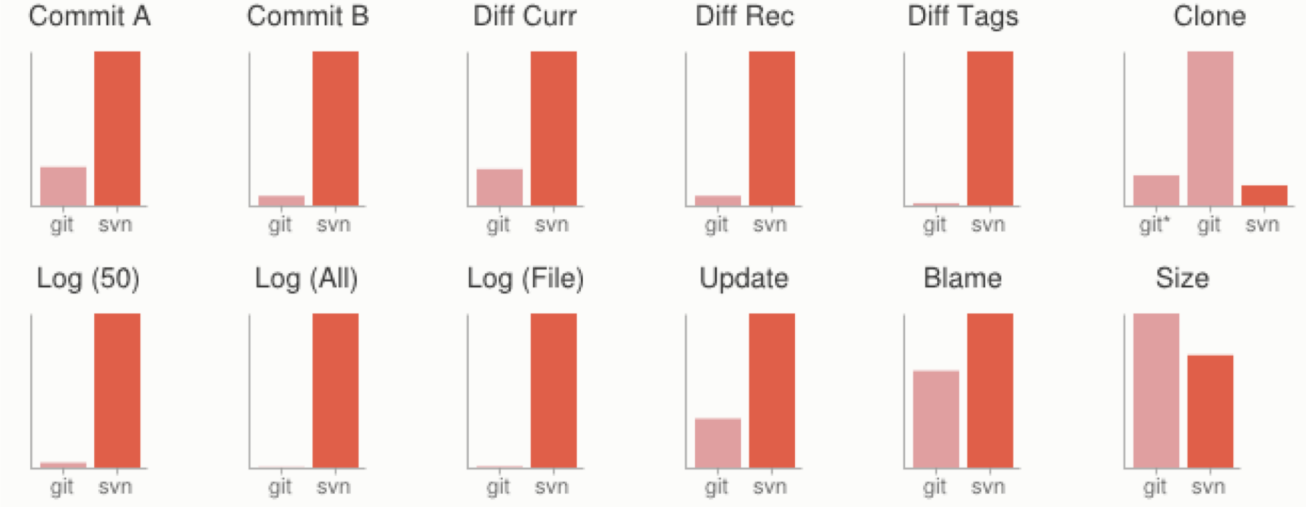
- Many places use legacy systems that will cause problems in the future – be the change you believe in!

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 VS {CVS, SVN, ...}

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

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

GIT VS {CVS, SVN, …}

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

Will give an example shortly.

GIT VS {CVS, SVN, ...}

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

Will give an example shortly.

GIT VS {CVS, SVN, ...}

Merging: 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

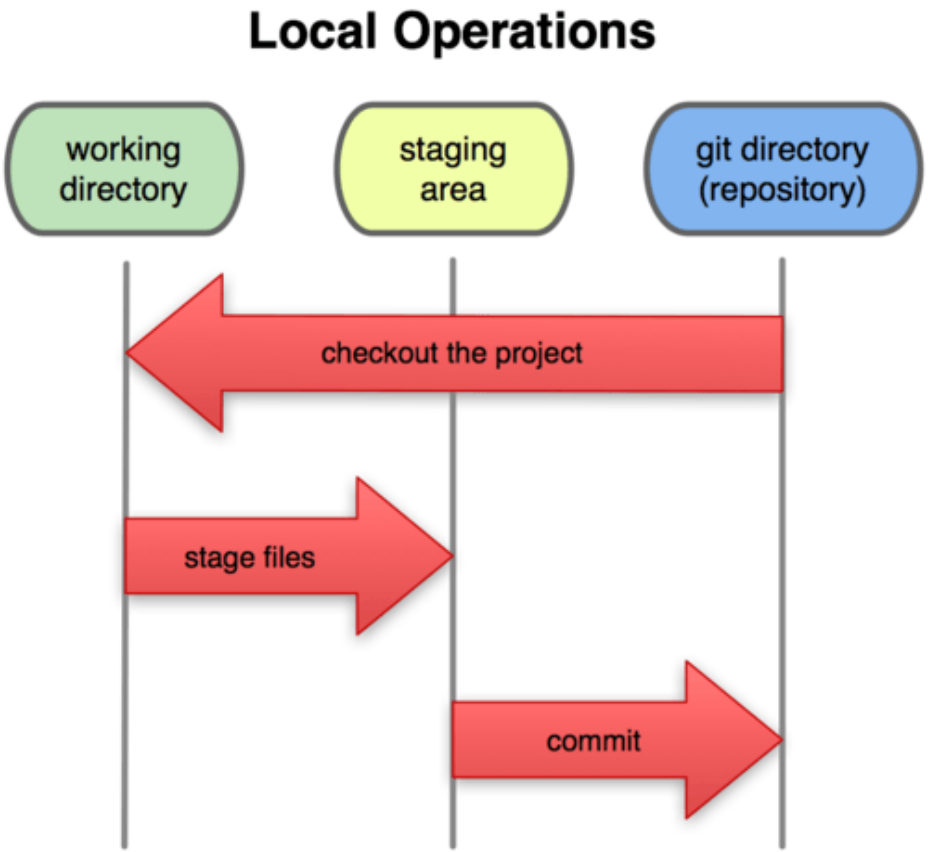
GIT BASICS II

Three main areas of a git project:

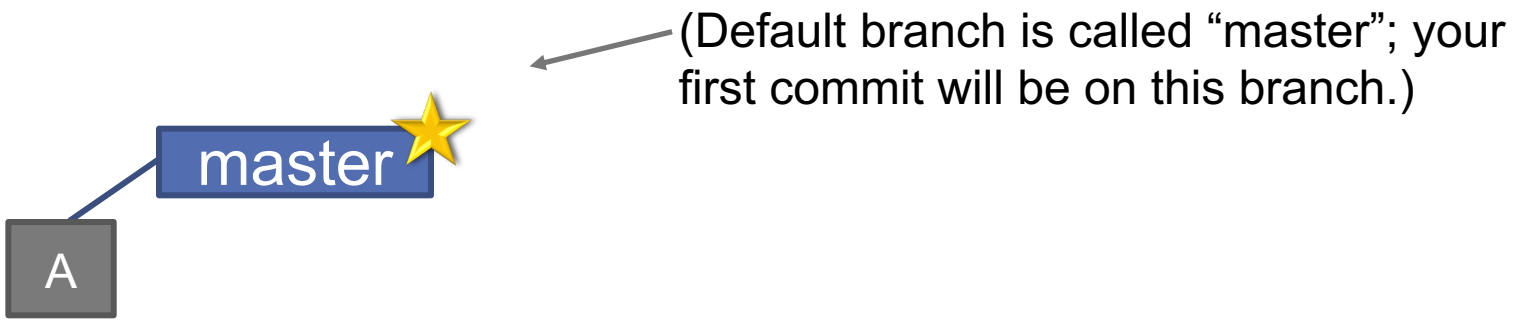
- **Working directory**
  - Single checkout of one version of the project.
- **Staging area**
  - Simple file storing information about what will go into your next commit
- **Git directory**
  - What is copied when cloning a repository

GIT BASICS III

Three main areas of a git project:

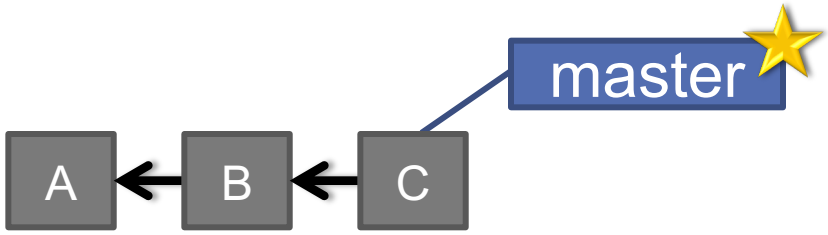


BRANCHES ILLUSTRATED



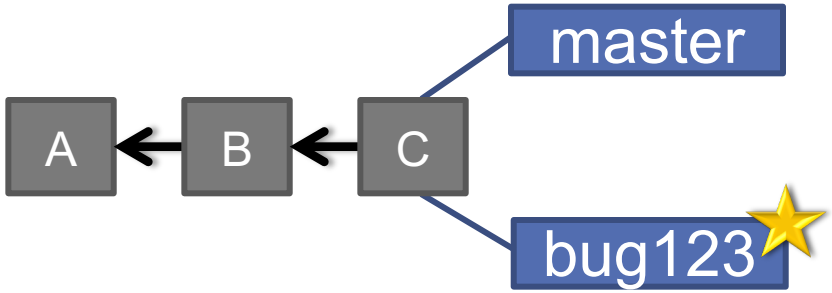

```
> git commit –m ‘my first commit’
```

BRANCHES ILLUSTRATED



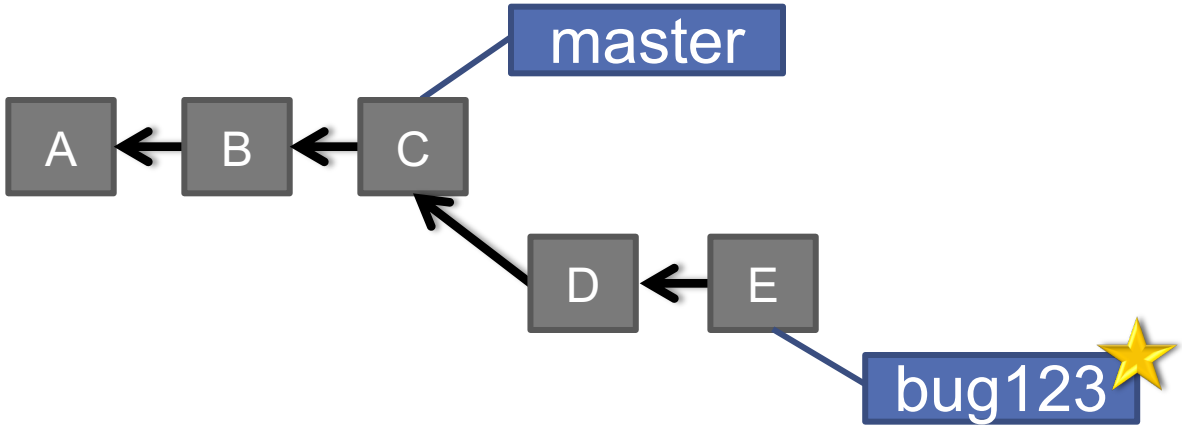

```
> git commit (x2)
```

BRANCHES ILLUSTRATED



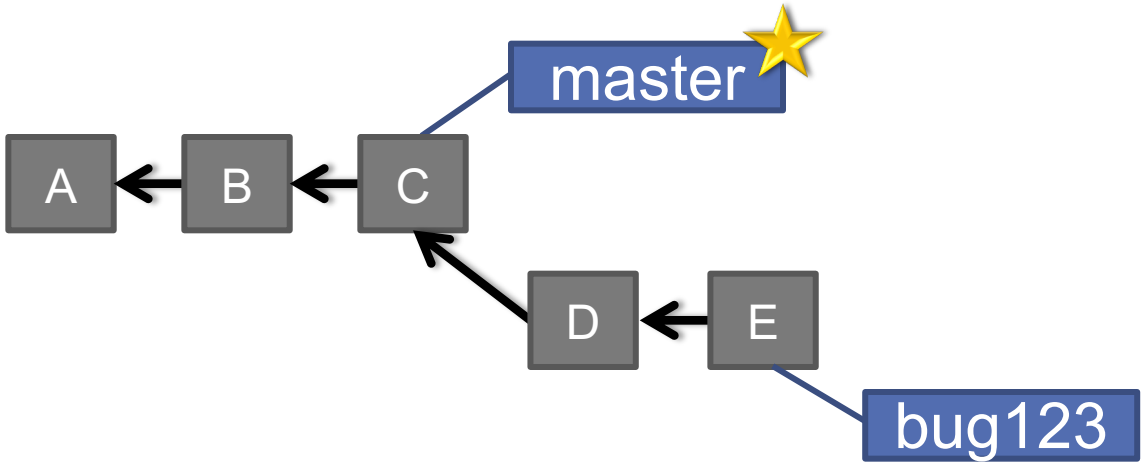

```
> git checkout -b bug123
```

BRANCHES ILLUSTRATED



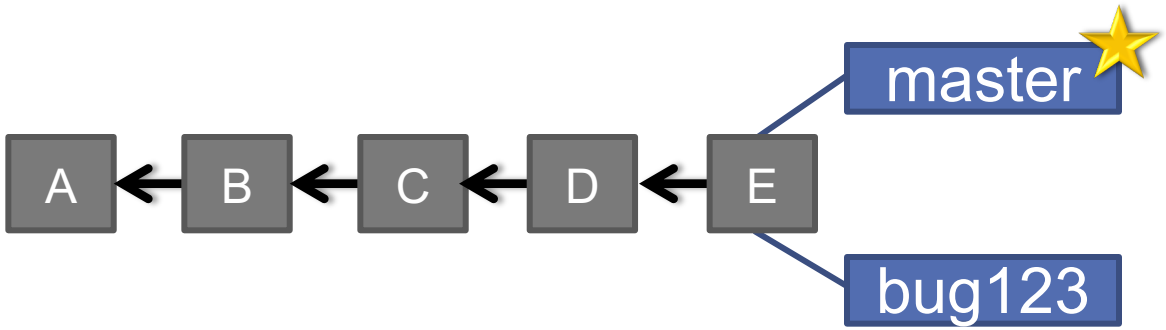

```
> git commit (x2)
```

BRANCHES ILLUSTRATED



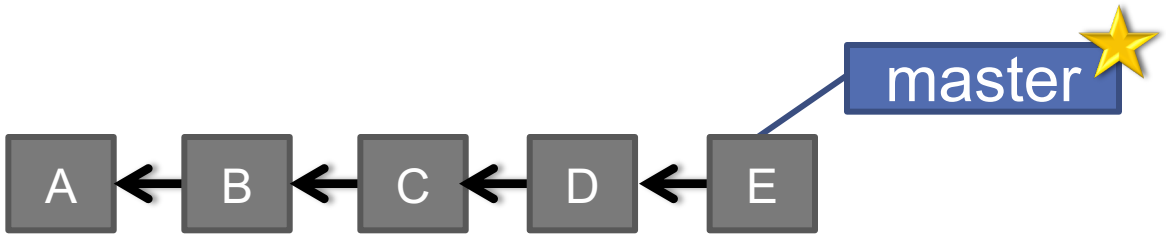

```
> git checkout master
```

BRANCHES ILLUSTRATED



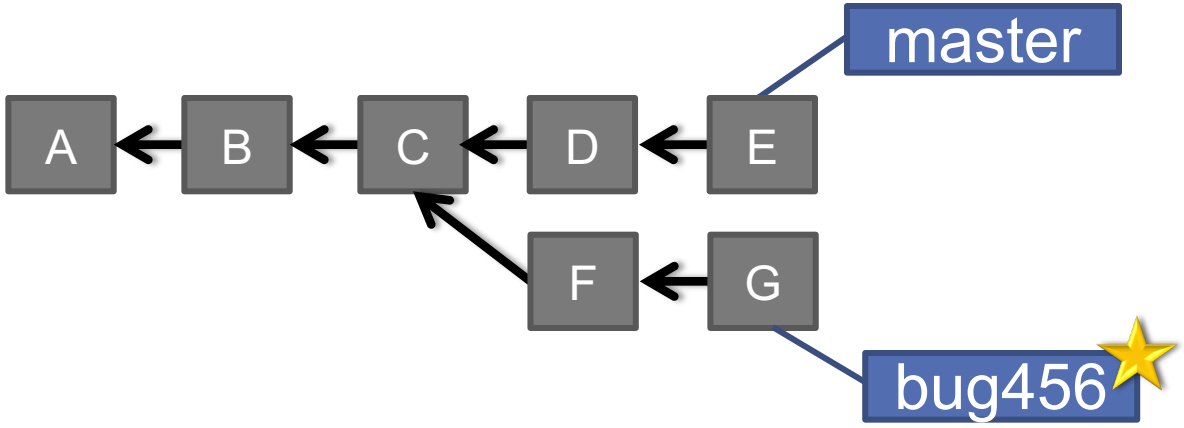

```
> git merge bug123
```

BRANCHES ILLUSTRATED

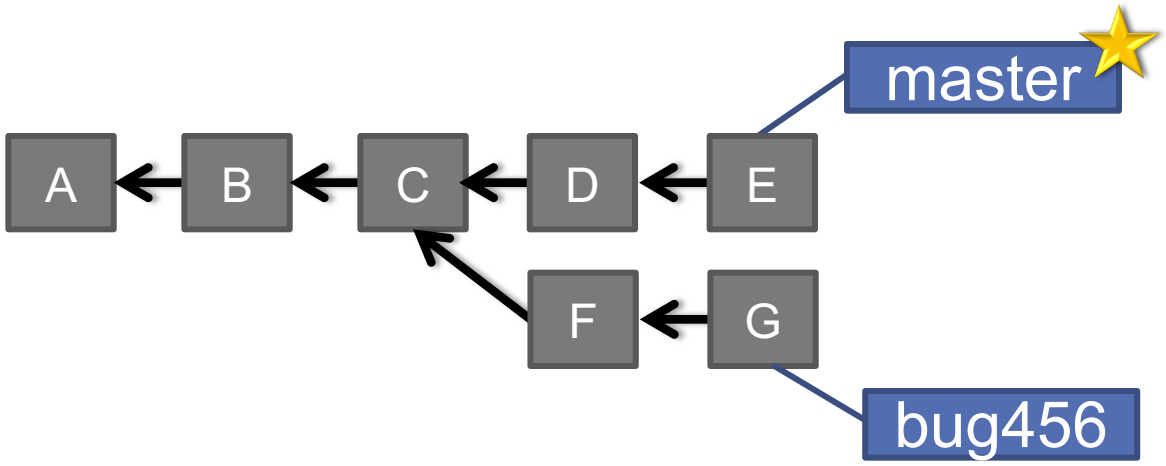



```
> git branch -d bug123
```

BRANCHES ILLUSTRATED

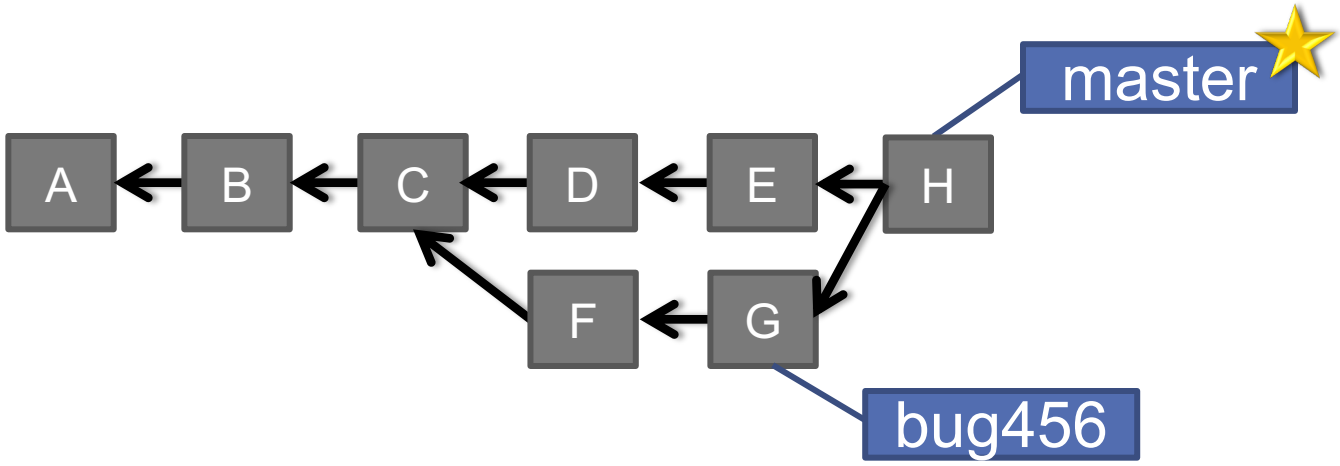


BRANCHES ILLUSTRATED



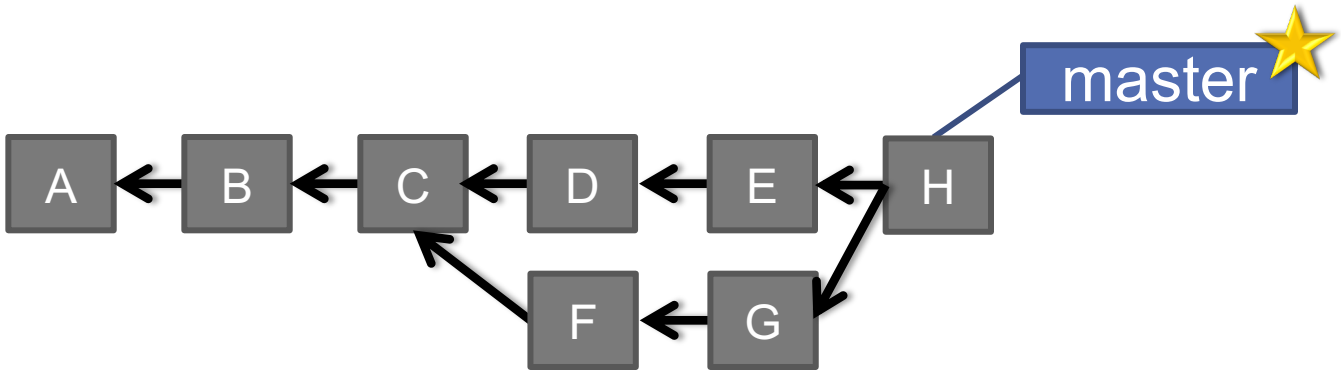

```
> git checkout master
```

BRANCHES ILLUSTRATED



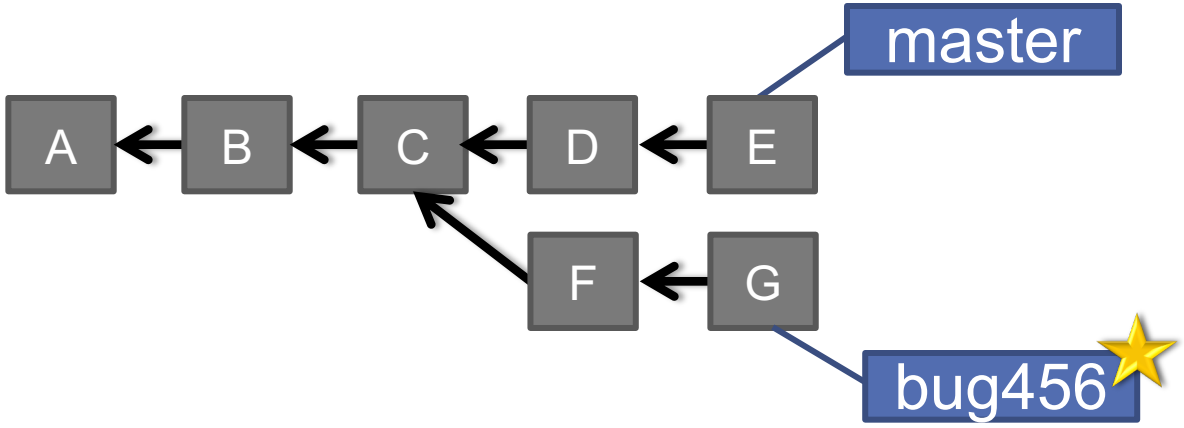

```
> git merge bug456
```

BRANCHES ILLUSTRATED

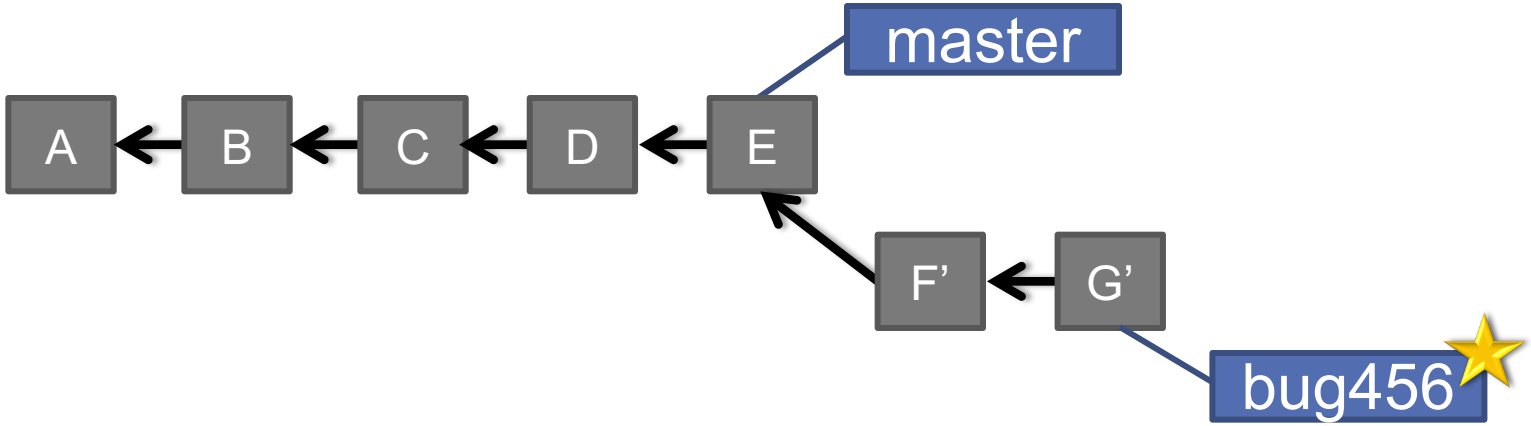



```
> git branch -d bug456
```

BRANCHES ILLUSTRATED

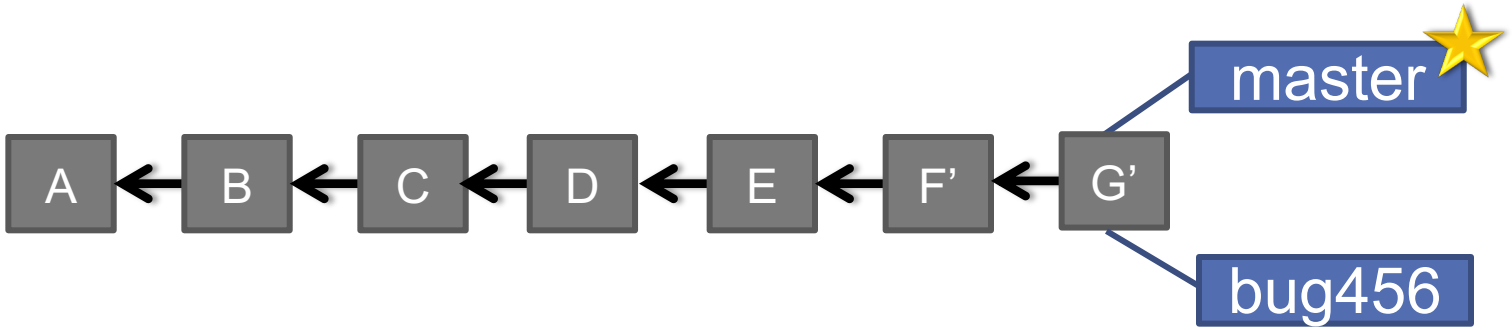


BRANCHES ILLUSTRATED




```
> 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.**

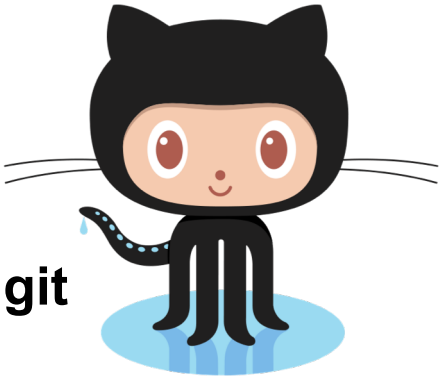
Local branching is very lightweight!

- 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)

“SOCIAL CODING”

John P. Dickerson
JohnDickerson

Assistant Professor of Computer Science, University of Maryland; Ph.D. in Computer Science, Carnegie Mellon University

University of Maryland
Washington, DC
http://jpdickerson.com

Organizations

Overview | Repositories 14 | Stars 71 | Followers 34 | Following 47

Popular repositories — Customize your pinned repositories

KidneyExchange
Kidney paired donation optimization code
Java ★ 8 7

TrackIt
Modular suite that measures a child's sustained selective attention.
Java ★ 3 2

EnvyFree
Computes envy-free allocations of items to agents.
Python ★ 2

VotingRules
Compute winners in elections based on various voting rules.
Java ★ 1

muffins
Fairly feeding hungry students
Python ★ 1

website
My academic website.
TeX

75 contributions in the last year — Contribution settings

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

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

NEXT CLASS:

TIDY DATA & MAYBE SOME RELATIONAL DATABASE STUFF