INTRODUCTION TO DATA SCIENCE

JOHN P DICKERSON

Lecture #8 – 09/23/2021

CMSC320
Tuesdays & Thursdays
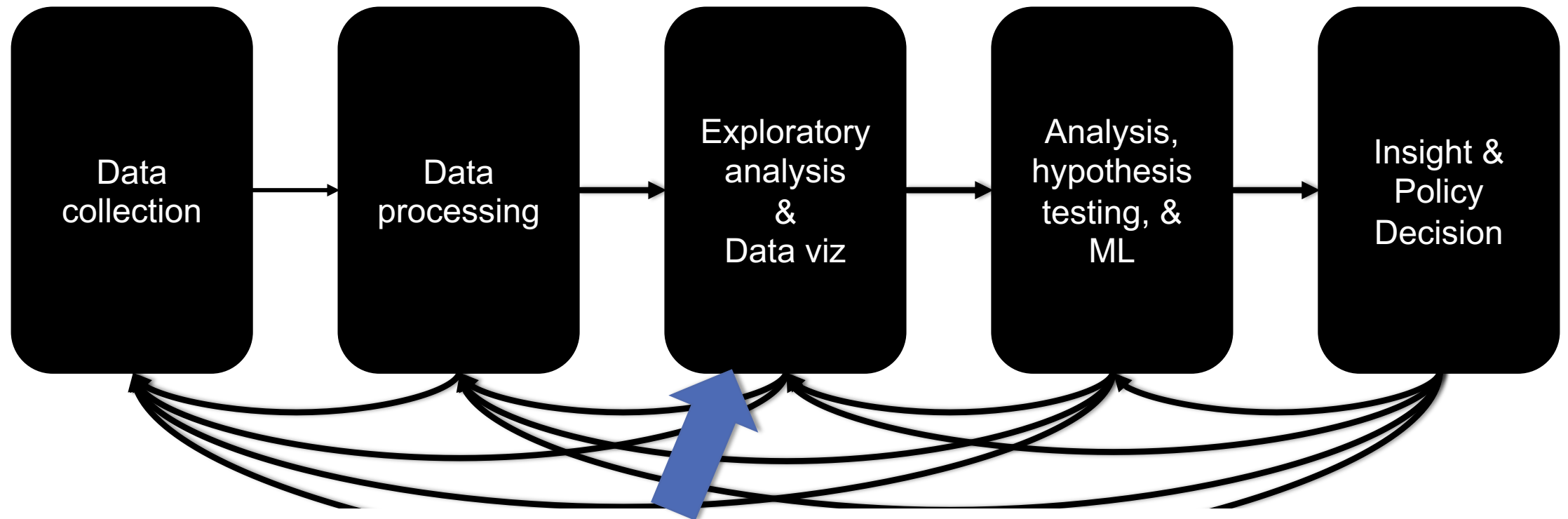**5:00pm – 6:15pm**

https://cmsc320.github.io/

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND
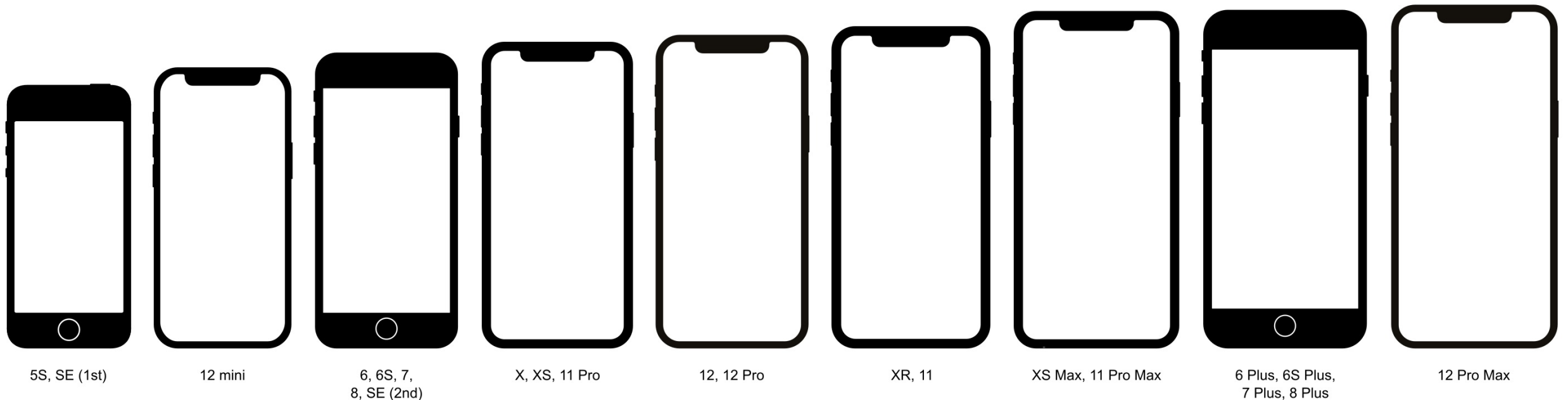
THE DATA LIFECYCLE

Version control for tracking code/data and for managing collaboration.

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!**

| 5S, SE (1st) | 12 mini | 6, 6S, 7, 8, SE (2nd) | X, XS, 11 Pro | 12, 12 Pro | XR, 11 | XS Max, 11 Pro Max | 6 Plus, 6S Plus, 7 Plus, 8 Plus | 12 Pro Max |

WHAT IS VERSION CONTROL?

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

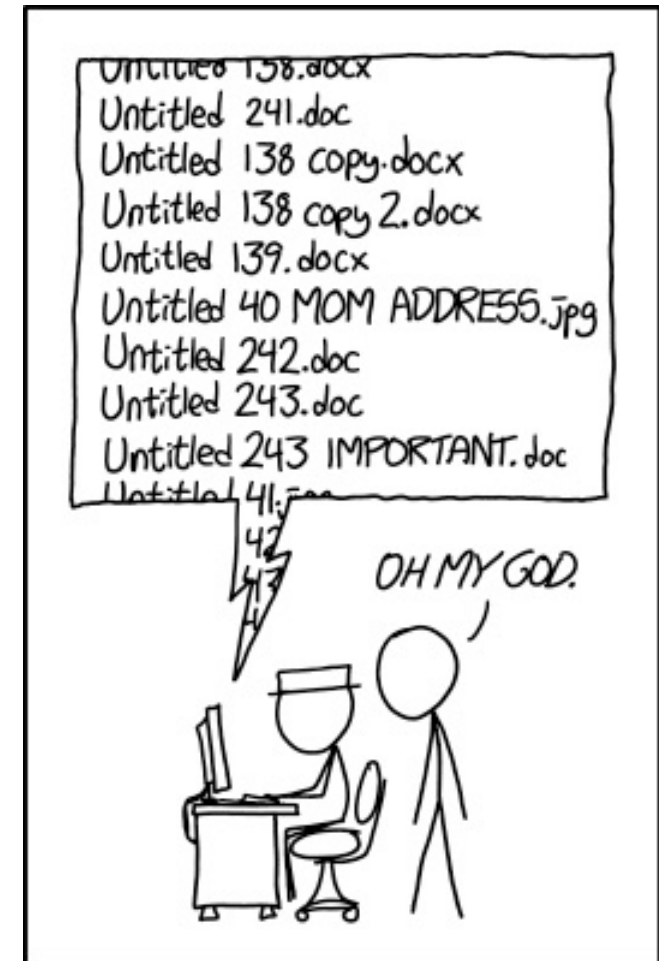**Be able to confidently make large changes to existing files**

atlassian.com/git/tutorials/what-is-version-control
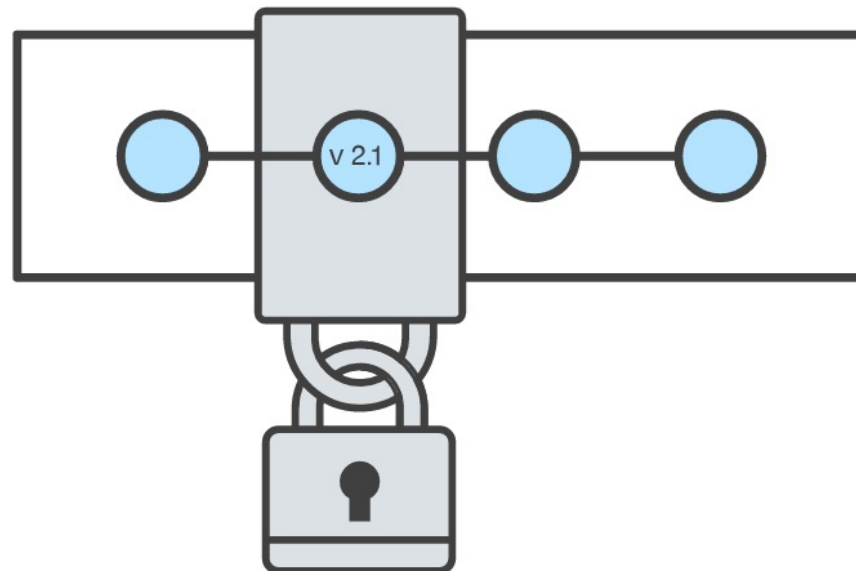
NAMED FOLDERS APPROACH

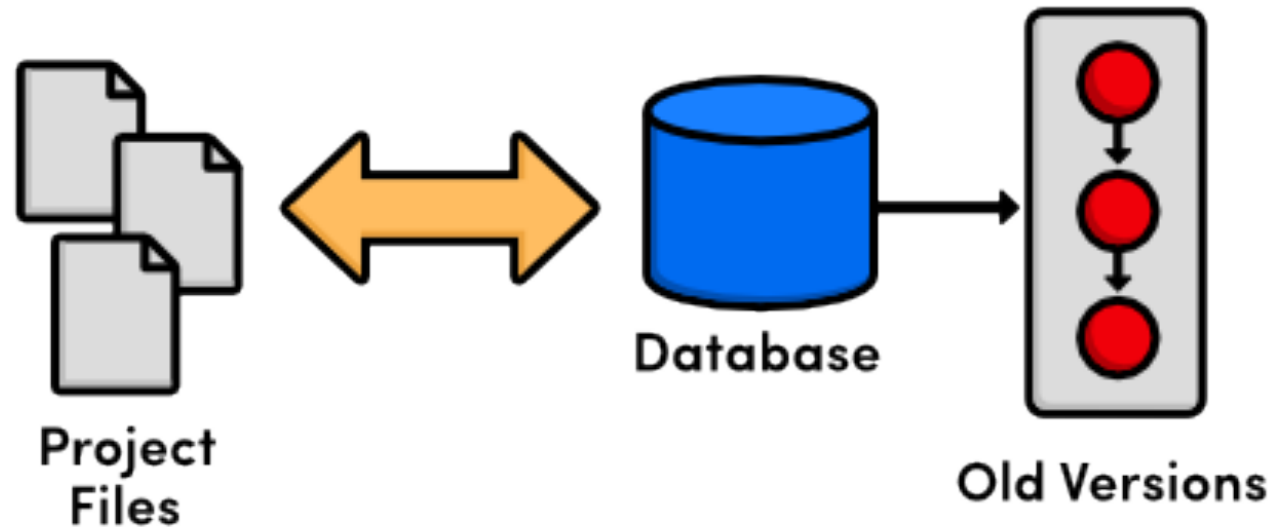**Can be hard to track**

Memory-intensive

Can be slow

Hard to share

No record of authorship

PROTIP: NEVER LOOK IN SOMEONE ELSE'S DOCUMENTS FOLDER.

LOCAL DATABASE OF VERSIONS APPROACH

Project Files

Database

Old Versions

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
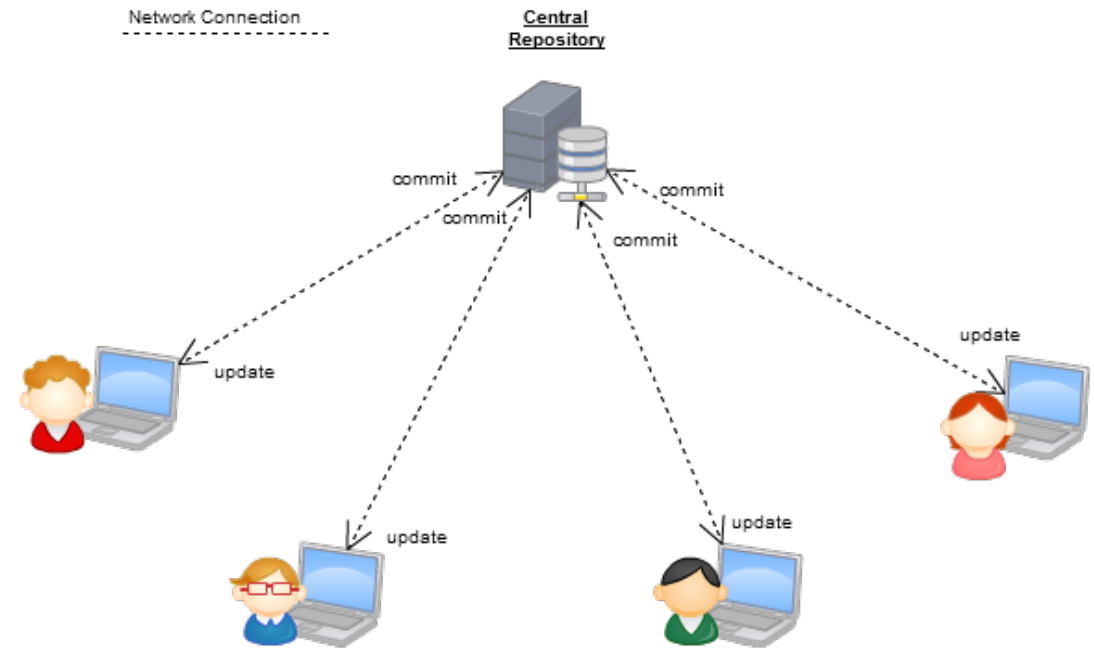
Can't share with collaborators

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.
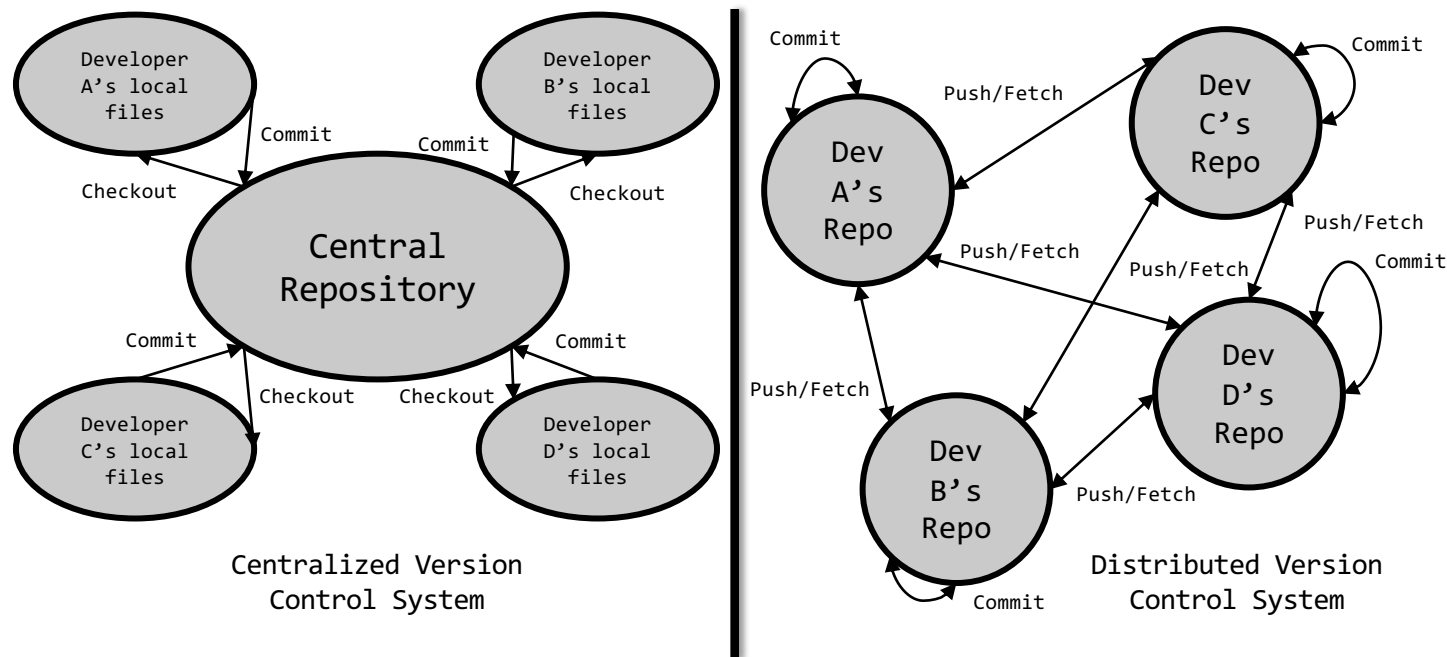
Examples: SVN, CVS, Perforce

DISTRIBUTED VERSION CONTROL SYSTEMS (DVCS)

No central repository

Every repository has every commit

Examples: Git, Mercurial

Centralized Version Control System

Distributed Version Control System

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
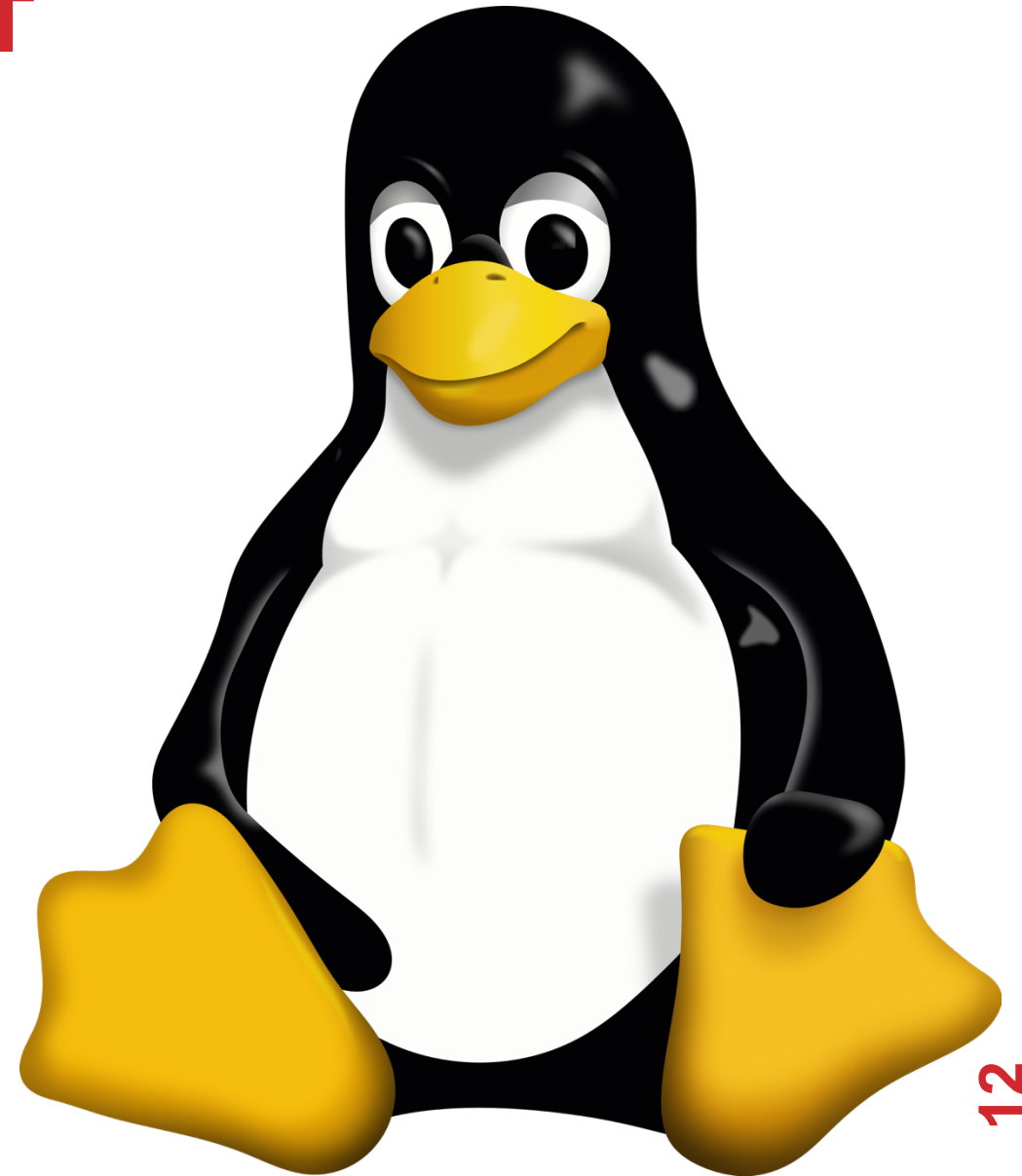
A SHORT HISTORY OF GIT

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

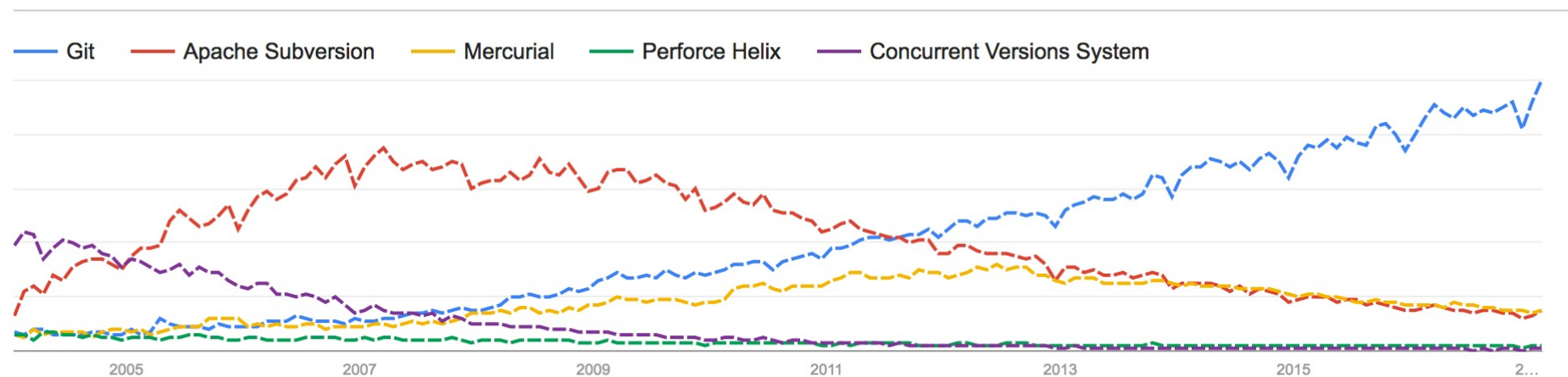- Able to handle large projects like the Linux kernel efficiently (speed and data size)

A SHORT HISTORY OF GIT

Popularity:

- Git is now the most widely used source code management tool

- 50% of professional software developers use Git (often through GitHub) as their primary source control system

[citation needed, IMO much more ☺]

Interest over time. Web Search. Worldwide, 2004 - present.

Git — Apache Subversion — Mercurial — Perforce Helix — Concurrent Versions System

2005   2007   2009   2011   2013   2015   2…

Google   View full report in Google Trends

GIT IN INDUSTRY

Companies and projects currently using Git

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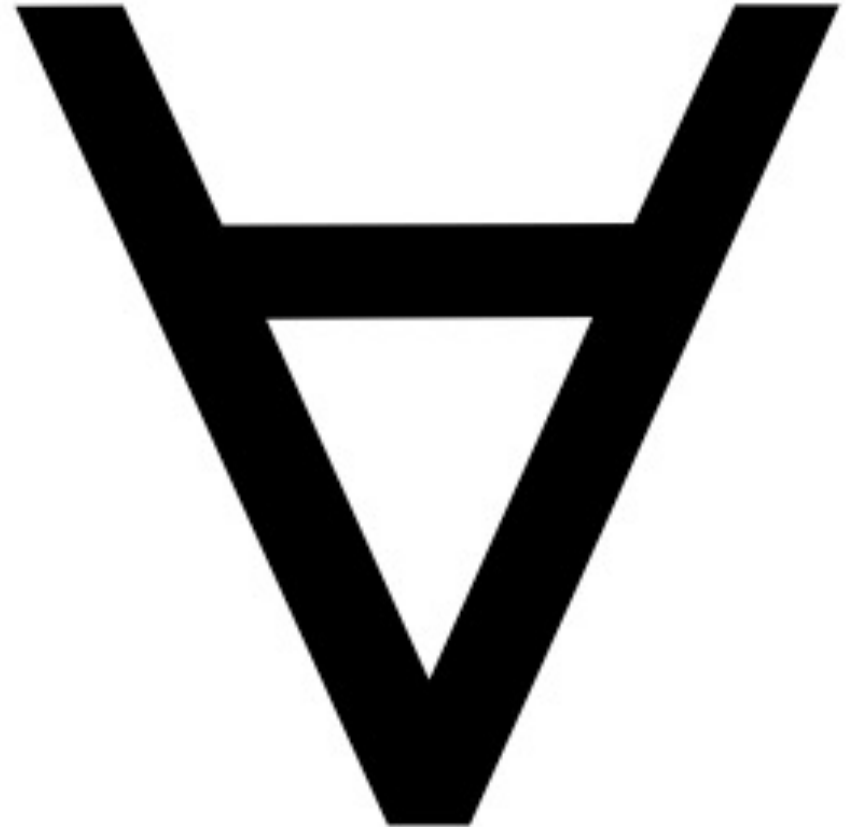
GIT BASICS

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

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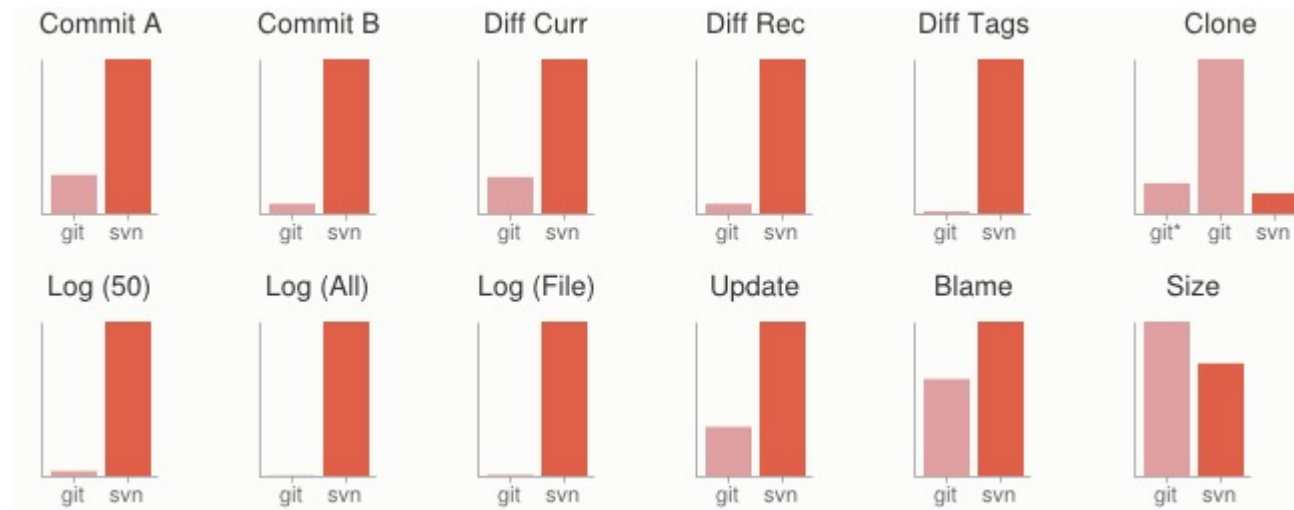
Benchmarks performed by http://git-scm.com/about/small-and-fast

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
```

Windows

UNIX®

A Standard of The Open Group®

-based systems such as MacOS (more related to FreeBSD) and Linux (e.g., the Ubuntu distros many of you are running)

```
john@Johns-MBP ~ % mkdir cmsc320 && cd cmsc320
john@Johns-MBP cmsc320 % git init
Initialized empty Git repository in /Users/john/cmsc320/.git/
john@Johns-MBP cmsc320 % touch README.md
john@Johns-MBP cmsc320 % git add .
john@Johns-MBP cmsc320 % git commit -m "First commit!"
```

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

master

A
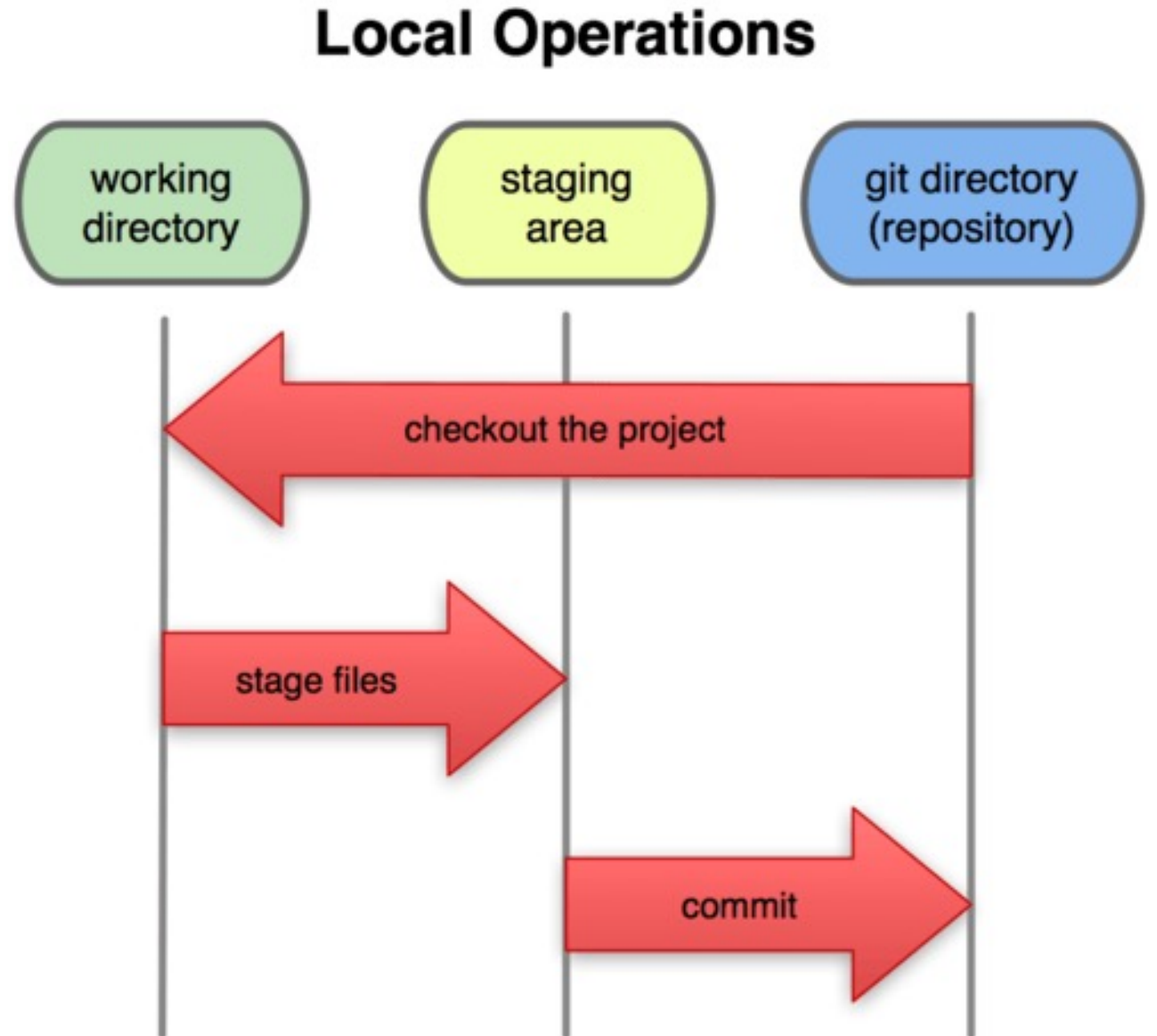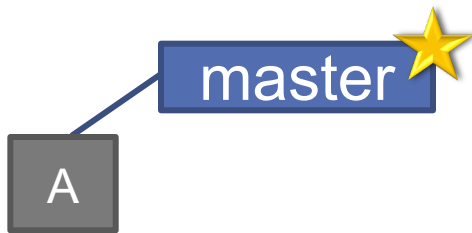
(Default branch is called "master"; your first commit will be on this branch. Starting October 1, 2020, this will be called "main" on GitHub.)

```
> git commit -m 'my first commit'
```

BRANCHES ILLUSTRATED


```
> git commit (x2)
```
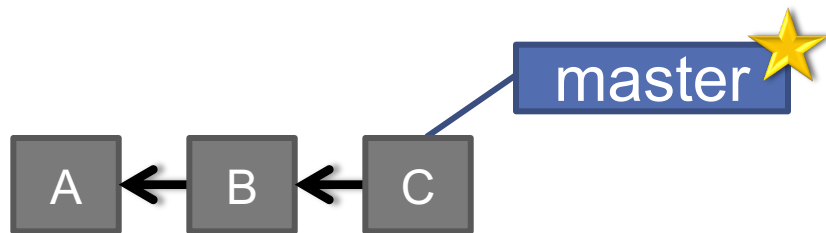
BRANCHES ILLUSTRATED
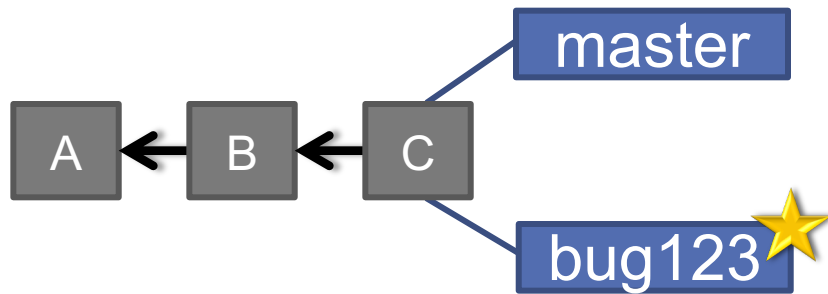

```
> git checkout –b bug123
```

BRANCHES ILLUSTRATED
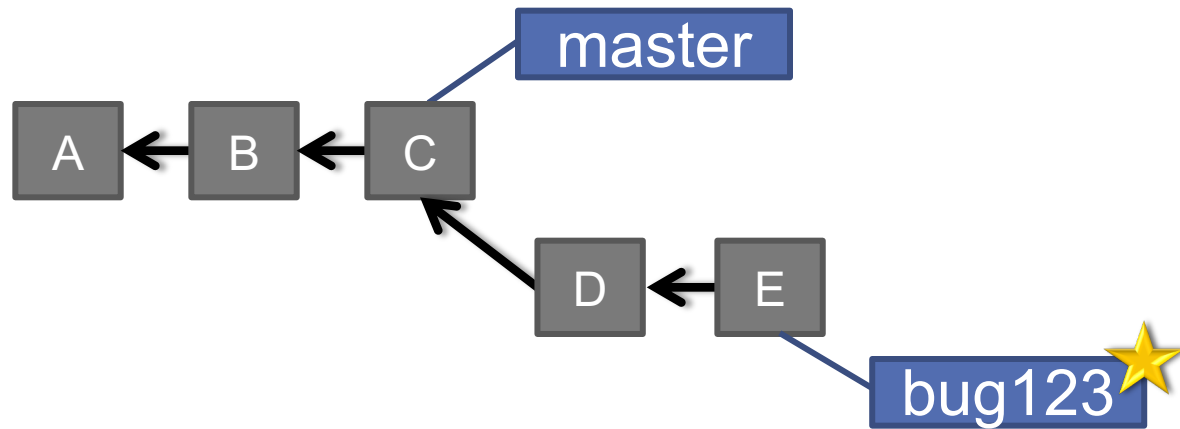

```
> git commit (x2)
```

BRANCHES ILLUSTRATED


```
> git checkout master
```
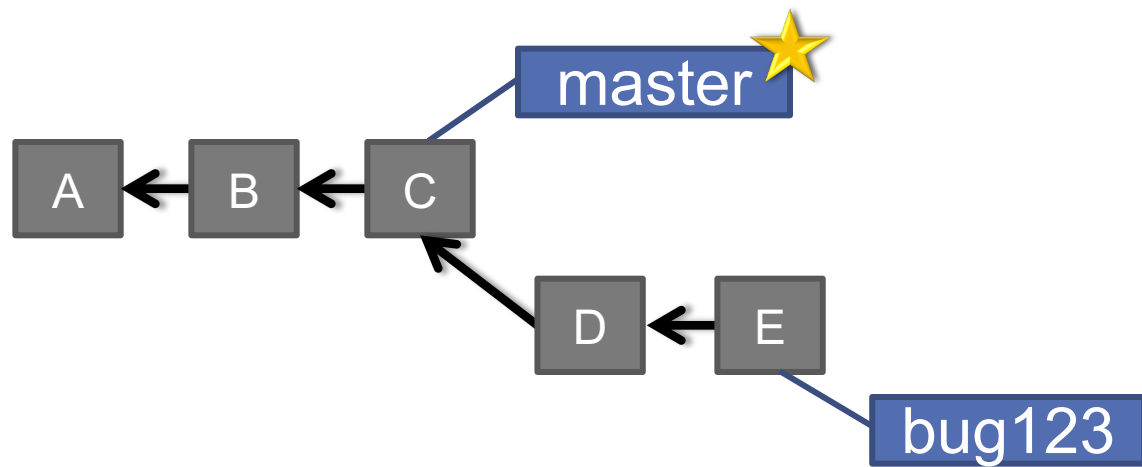
BRANCHES ILLUSTRATED
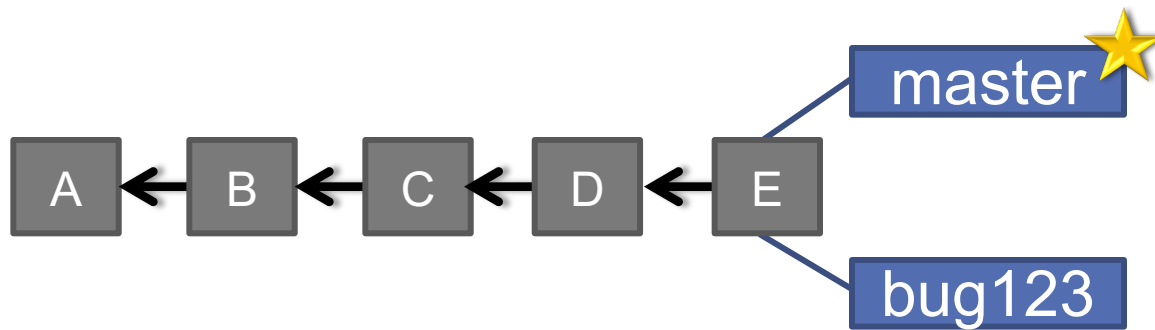
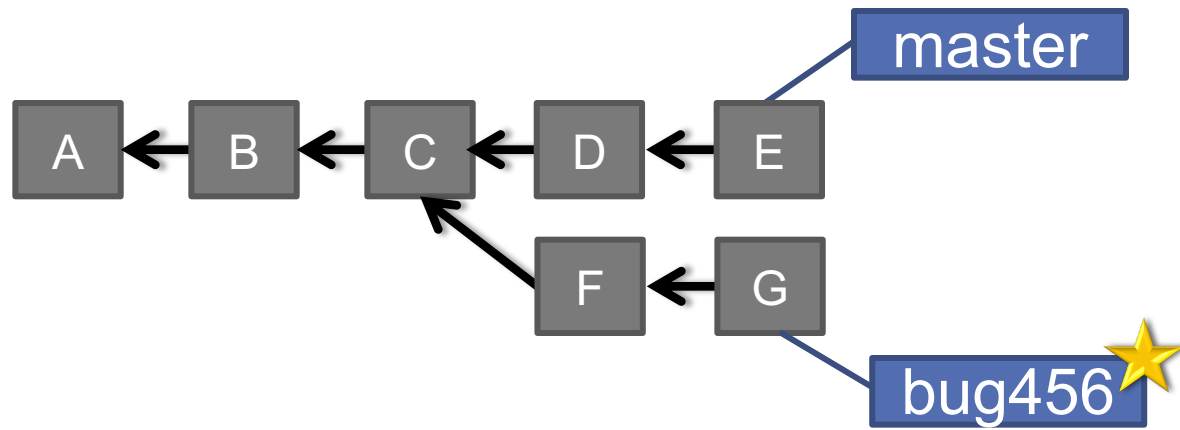```
> git merge bug123
```

# BRANCHES ILLUSTRATED
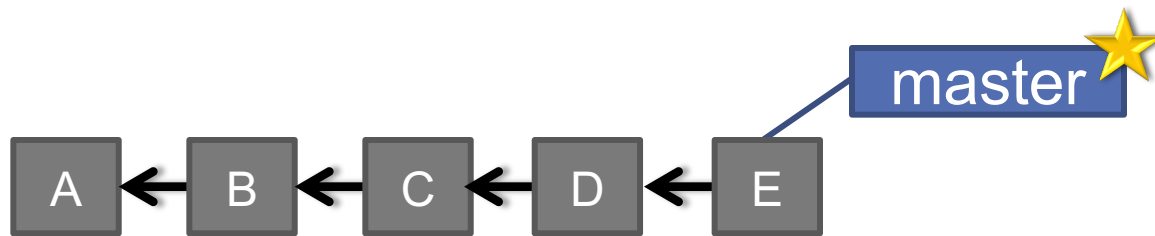


```
> git branch -d bug123
```
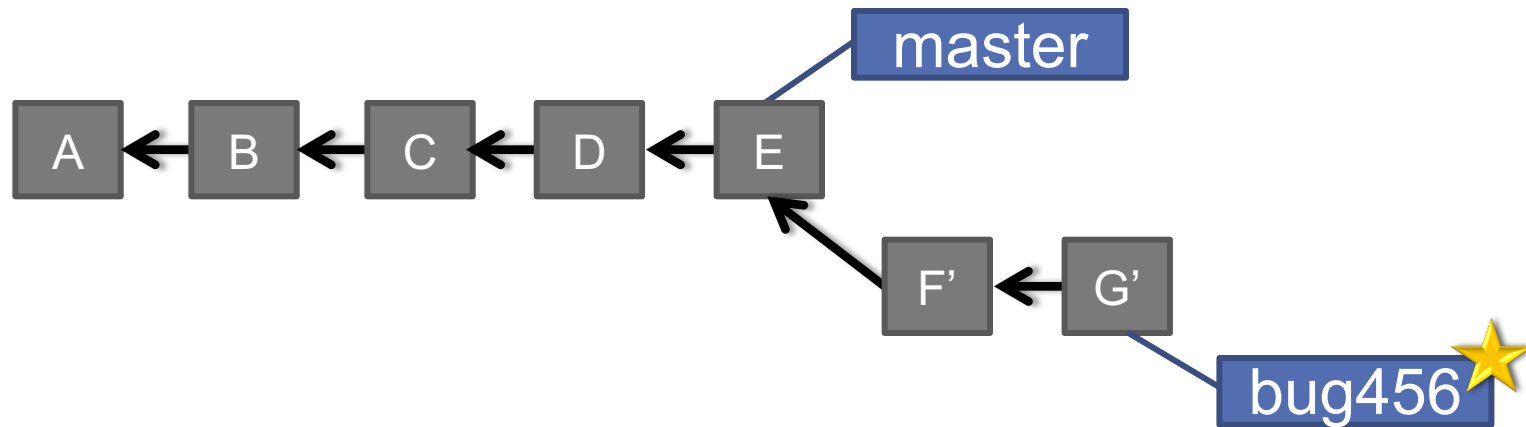
# BRANCHES ILLUSTRATED
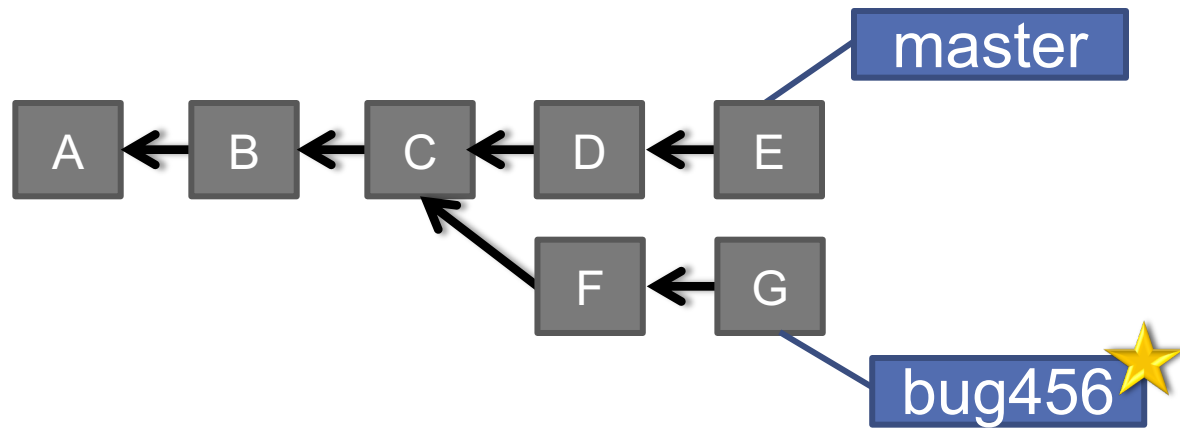
# BRANCHES ILLUSTRATED
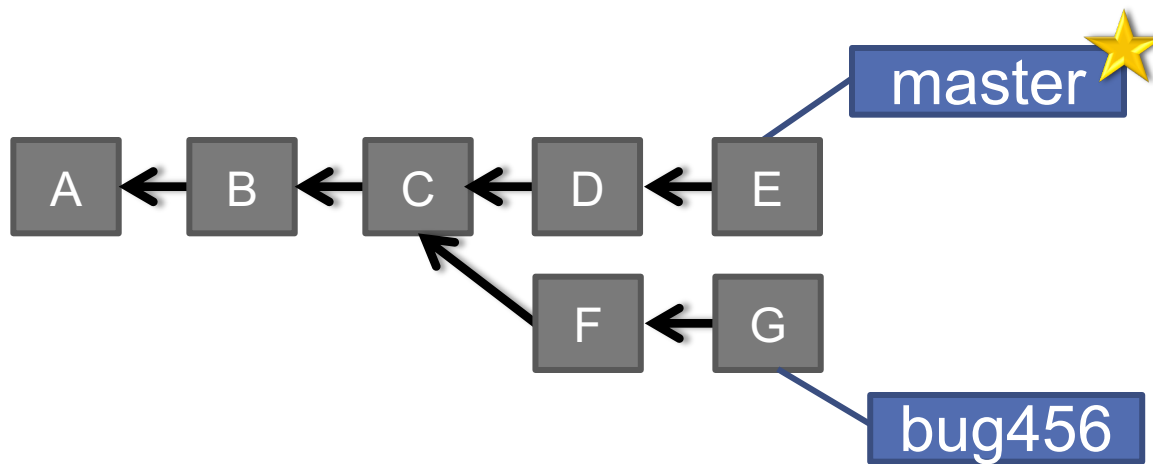


> git checkout master

# BRANCHES ILLUSTRATED
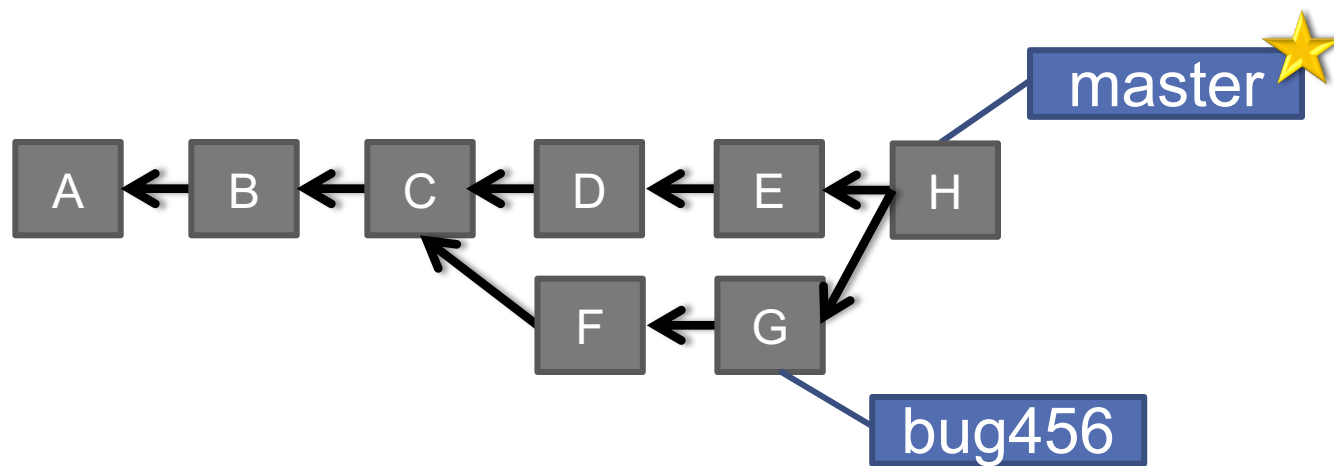


```
> git merge bug456
```

BRANCHES ILLUSTRATED
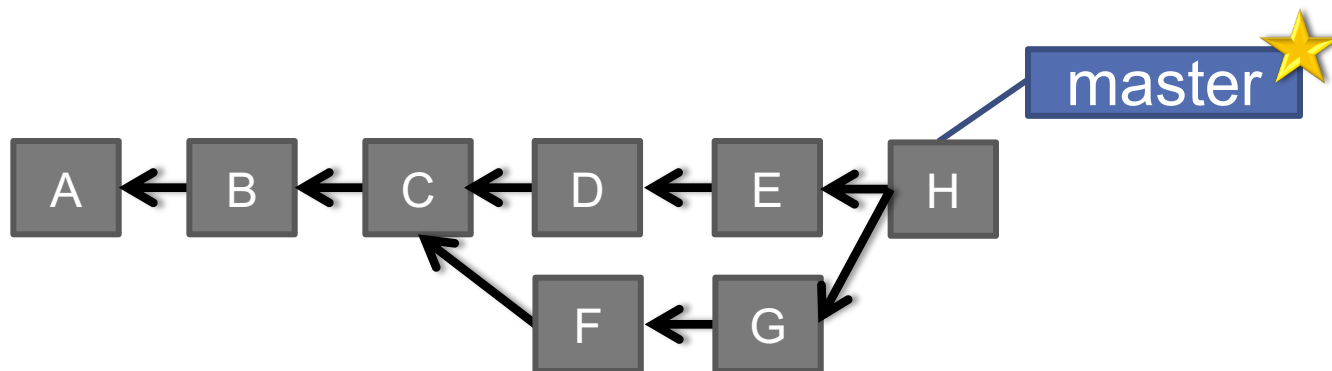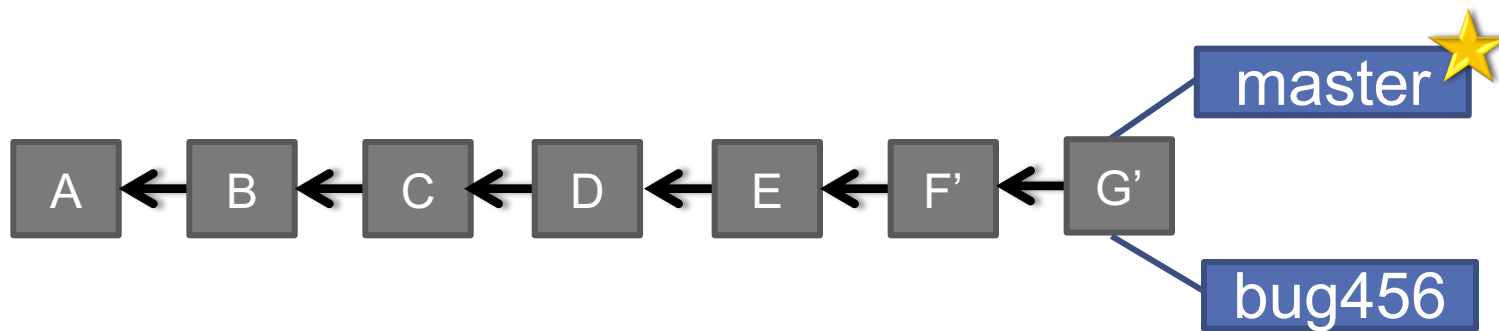


```
> git branch -d bug456
```

BRANCHES ILLUSTRATED

BRANCHES ILLUSTRATED


```
> git rebase master
```

BRANCHES ILLUSTRATED


```
> git checkout master
> git merge bug456
```
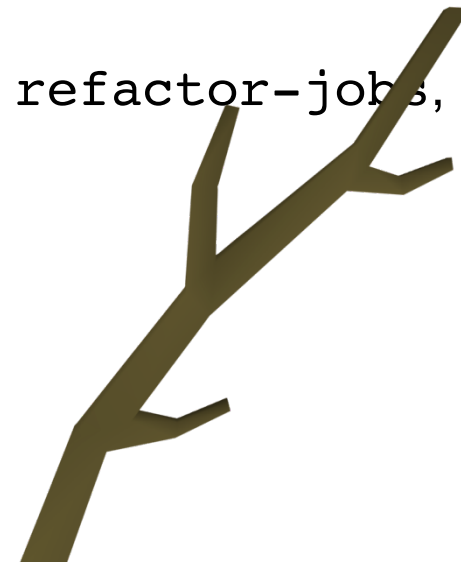
WHEN TO BRANCH?

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/"awesome" projects (e.g., **https://github.com/maxbbraun/trump2cash**)

There are reasons to use the competitors (e.g., private repositories, access control)

"SOCIAL CODING" – PROS AND CONS, PUBLIC VS PRIVATE

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