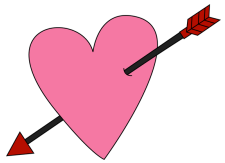
INTRODUCTION TO DATA SCIENCE

JOHN P DICKERSON



Lecture #6 – 2/14/2017

CMSC320
Tuesdays & Thursdays
3:30pm – 4:45pm

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

ANNOUNCEMENTS

Register on Piazza: piazza.com/umd/spring2017/cmsc320

- 90 have registered already. ♡
- I think we're good ☺.

We will release the first mini-project in less than one week.

- Please make sure Anaconda installed correctly!
- (See any of us if it didn't.)
- ELMS will be used for submission.

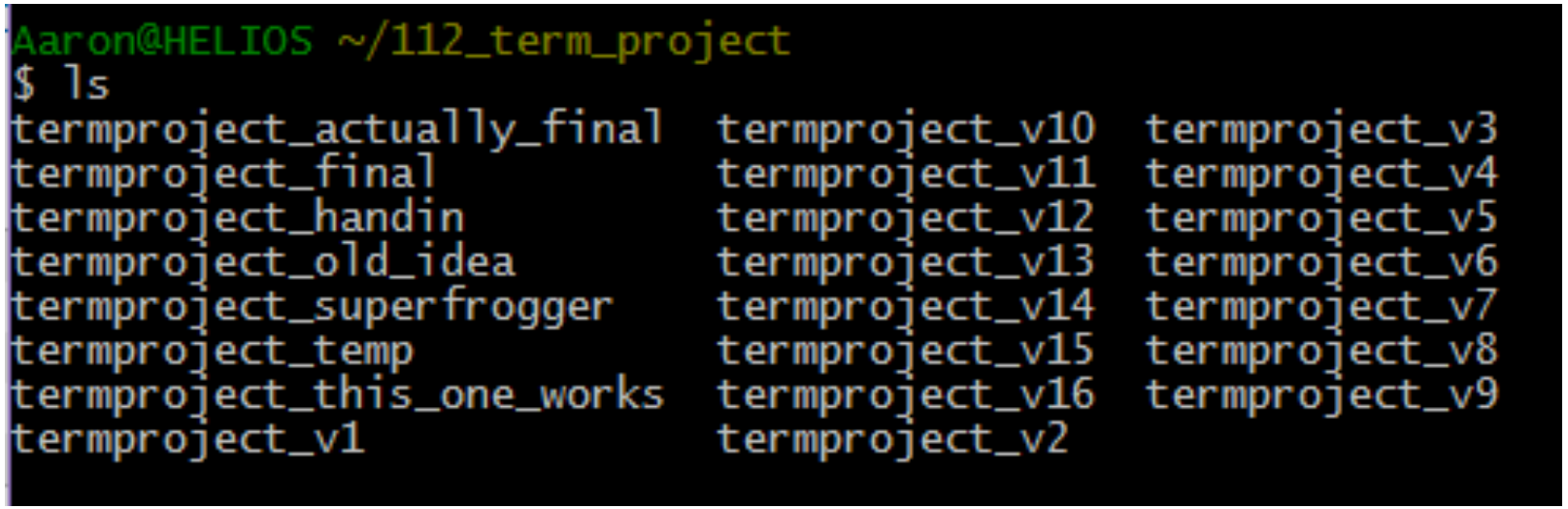
TODAY'S LECTURE

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?



DEVELOPMENT TOOL

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

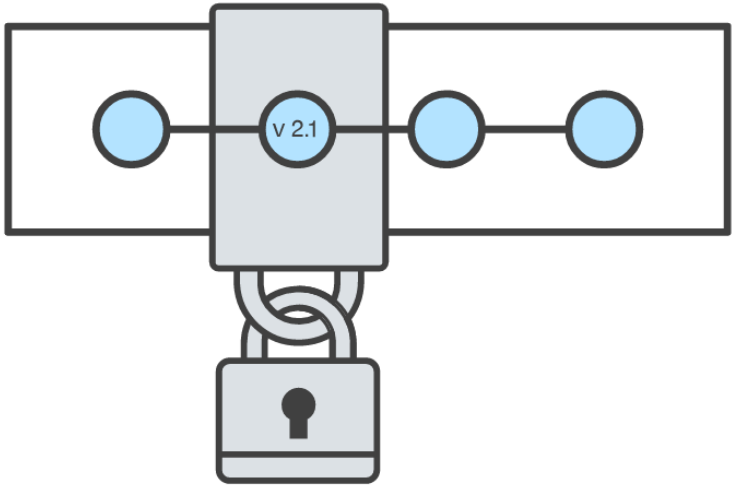
- Need a system to allow versioning, and a way to acquire the latest edition of the code
- A system to track and manage bugs was also needed

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

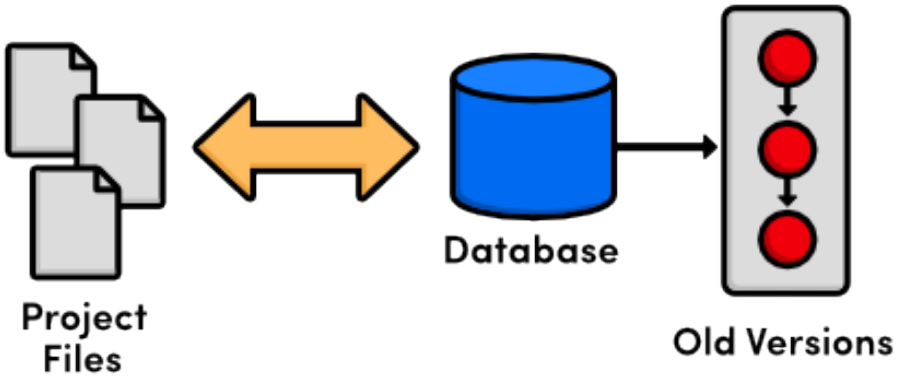
Can be slow

Hard to share

No record of authorship

```
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
```

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

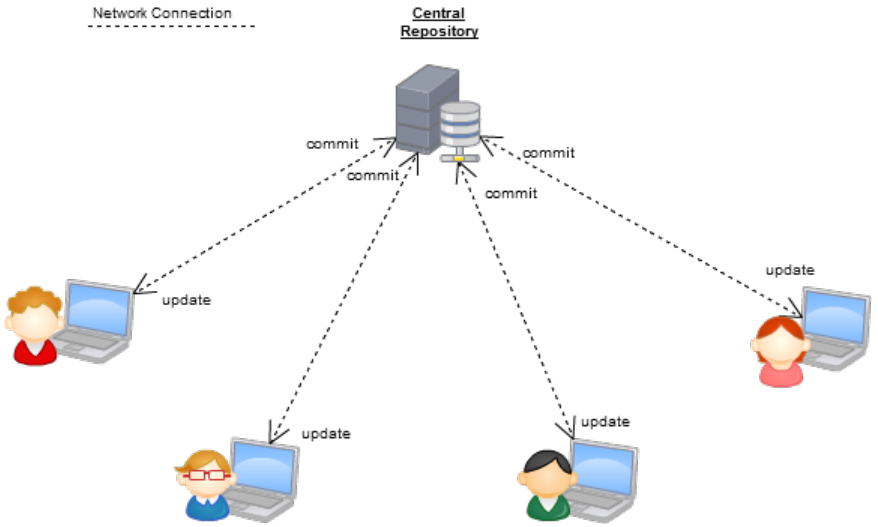
Can’t share with collaborators

CENTRALIZED VERSION CONTROL SYSTEMS

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

Collaborators “push” changes (commits) to this repository.

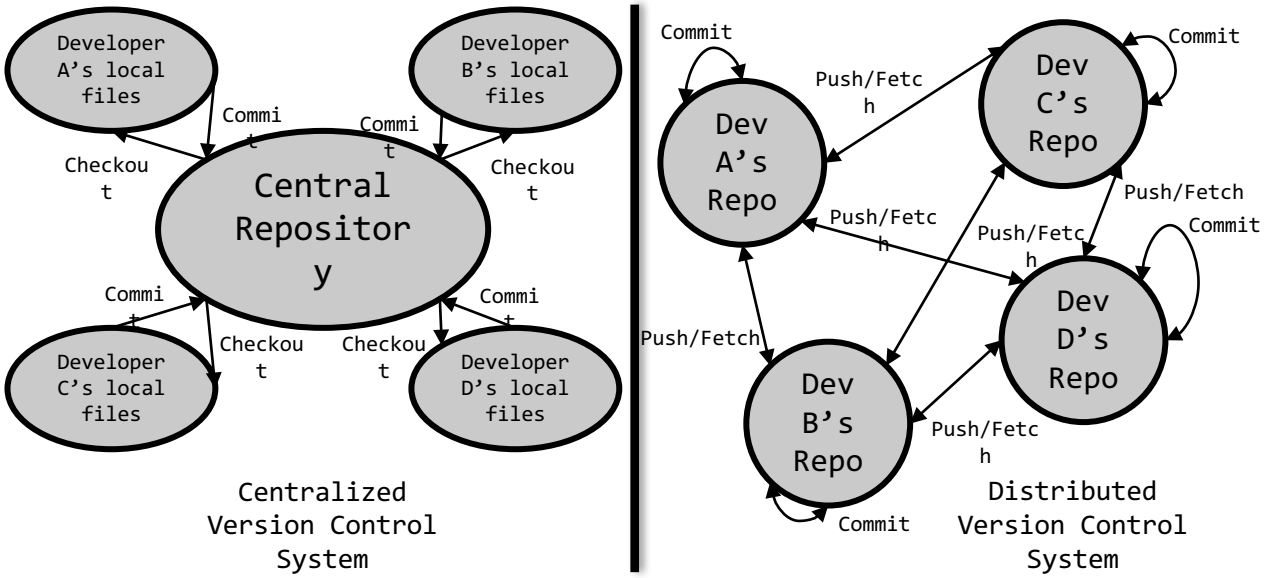
Any new commits must be compatible with the most recent commit. If it isn’t, somebody must “merge” it in.



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

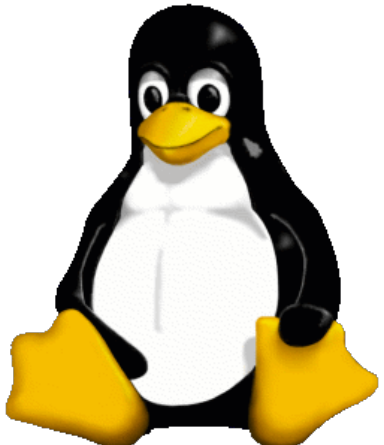
- Changes passed around as archived file

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)

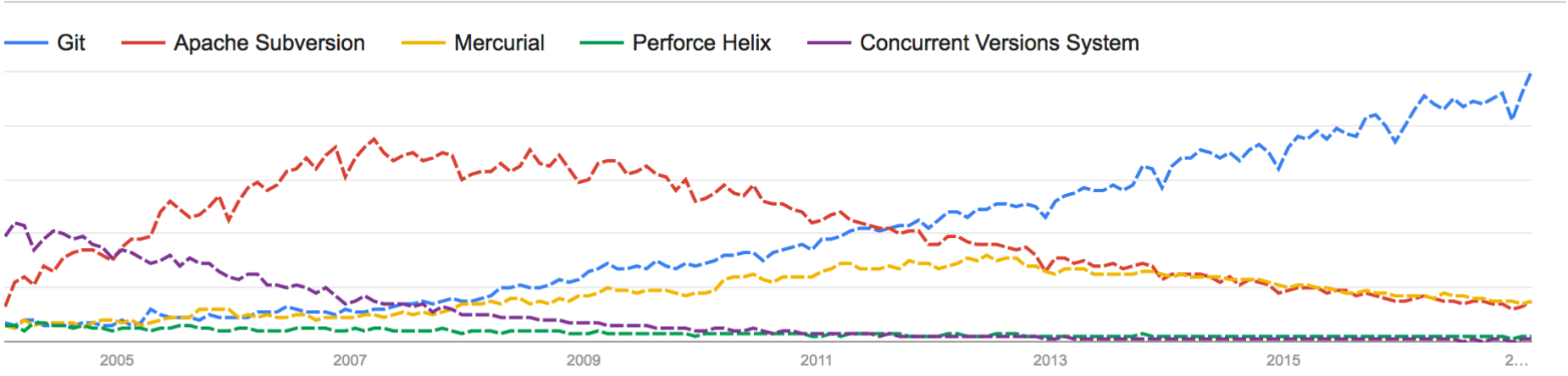
A SHORT HISTORY OF GIT

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]

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



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

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:

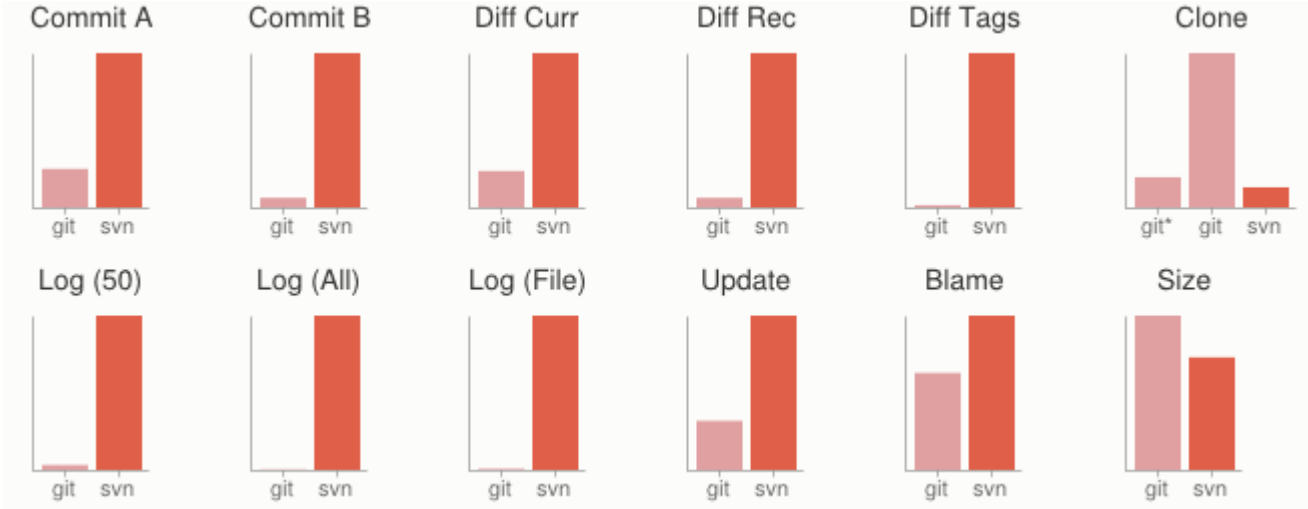
- Many places use legacy systems that will cause problems in the future – be the change you believe in!

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

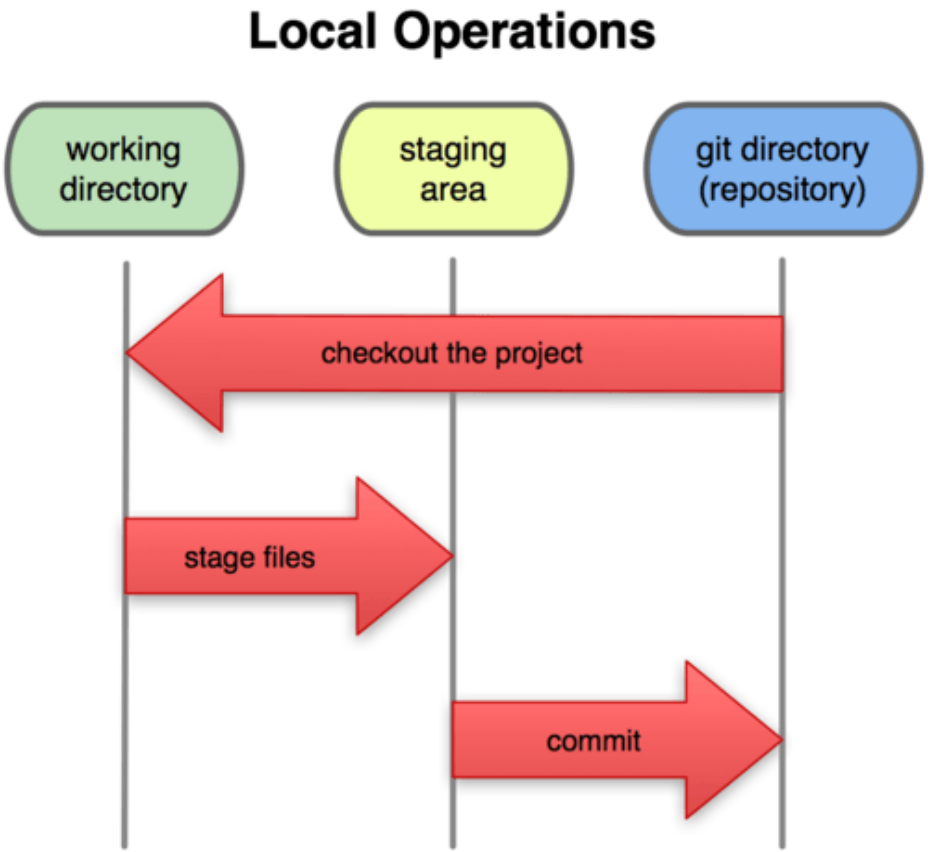
GIT BASICS II

Three main areas of a git project:

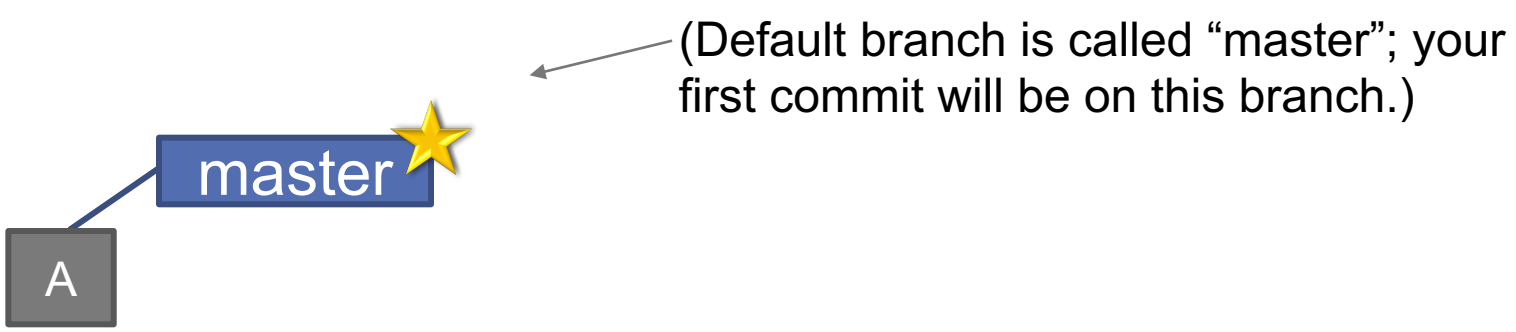
- **Working directory**
  - Single checkout of one version of the project.
- **Staging area**
  - Simple file storing information about what will go into your next commit
- **Git directory**
  - What is copied when cloning a repository

GIT BASICS III

Three main areas of a git project:

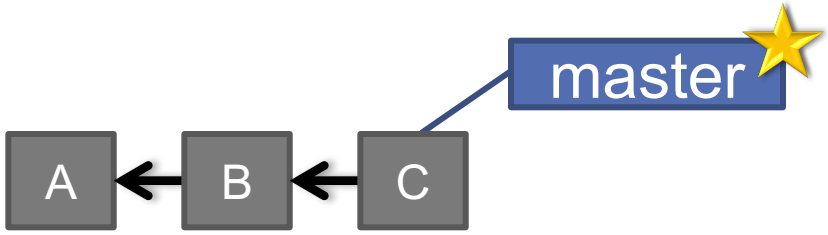


BRANCHES ILLUSTRATED



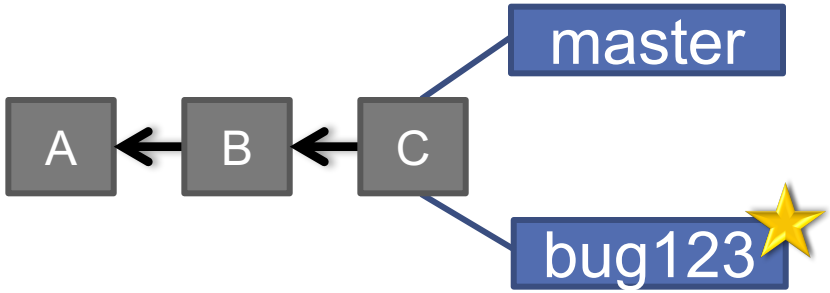

```
> git commit –m ‘my first commit’
```

BRANCHES ILLUSTRATED



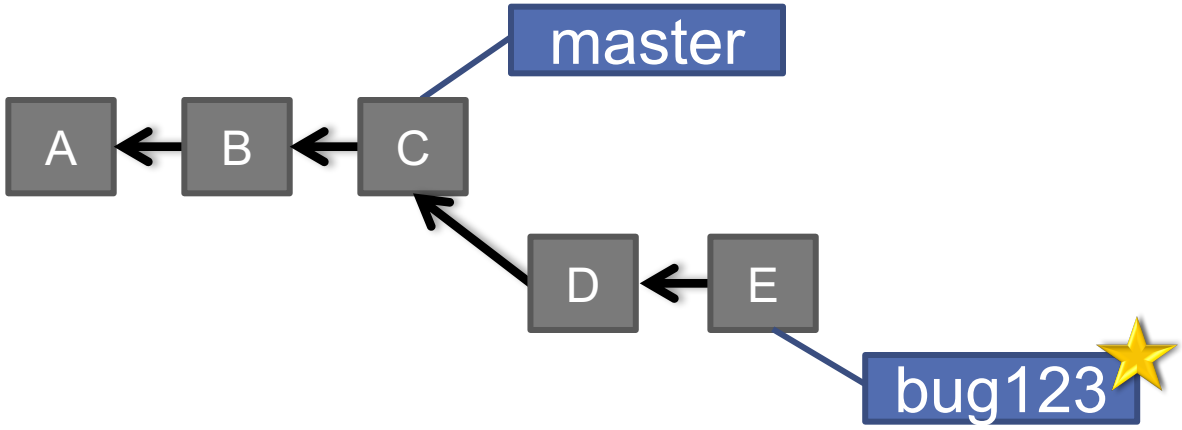

```
> git commit (x2)
```

BRANCHES ILLUSTRATED



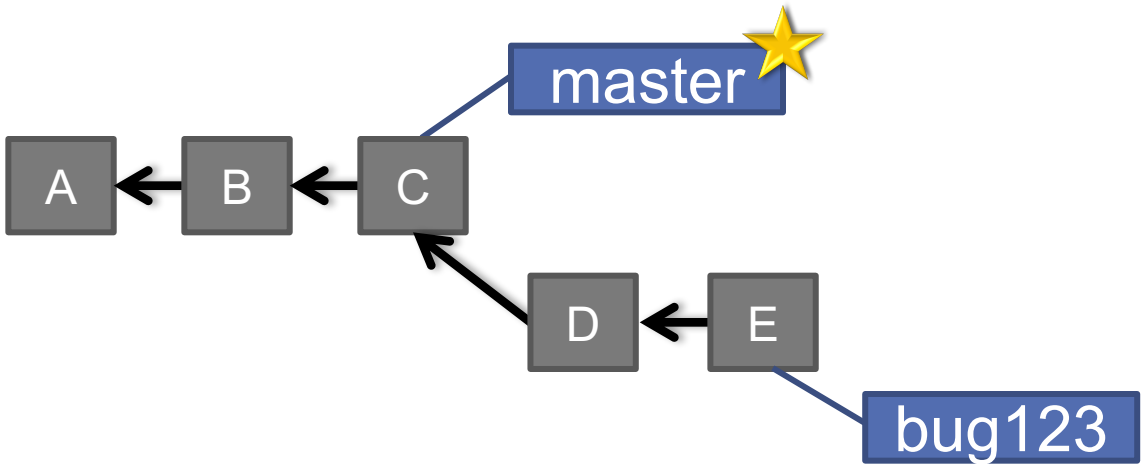

```
> git checkout -b bug123
```

BRANCHES ILLUSTRATED



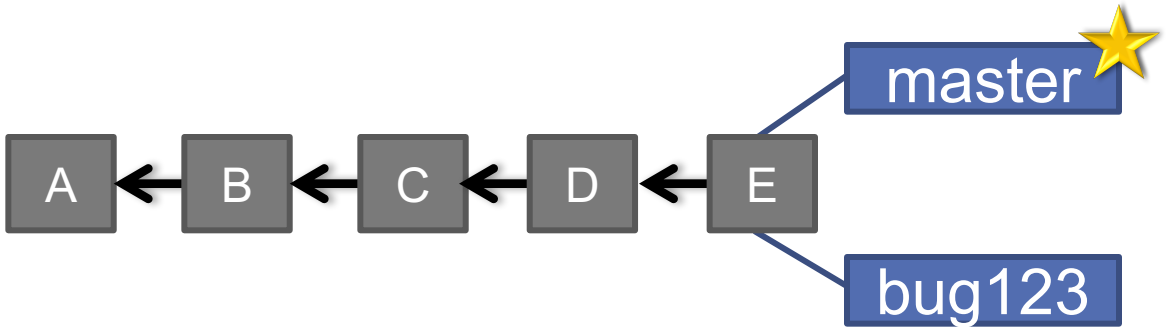

```
> git commit (x2)
```

BRANCHES ILLUSTRATED



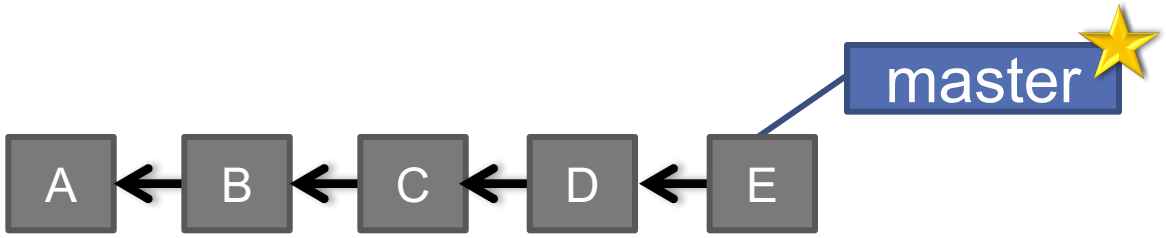

```
> git checkout master
```

BRANCHES ILLUSTRATED



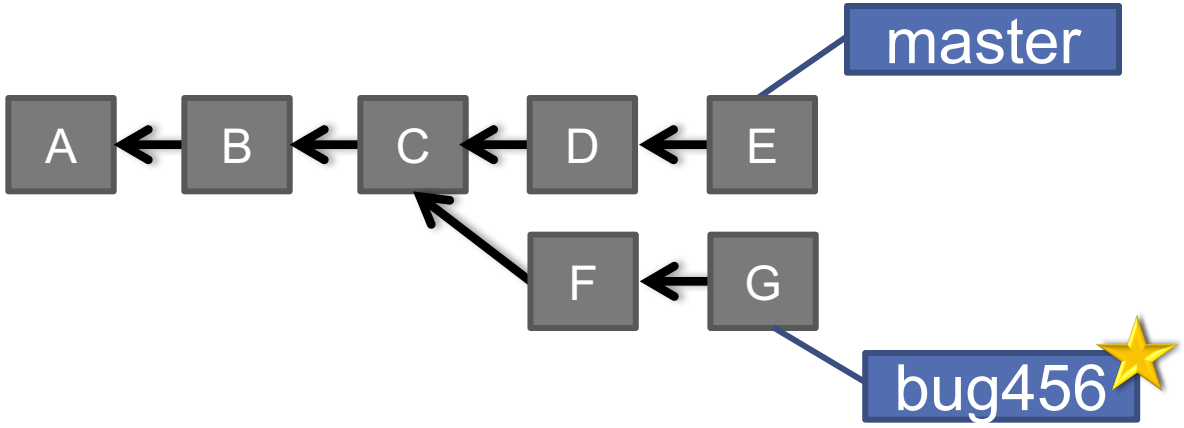

```
> git merge bug123
```

BRANCHES ILLUSTRATED

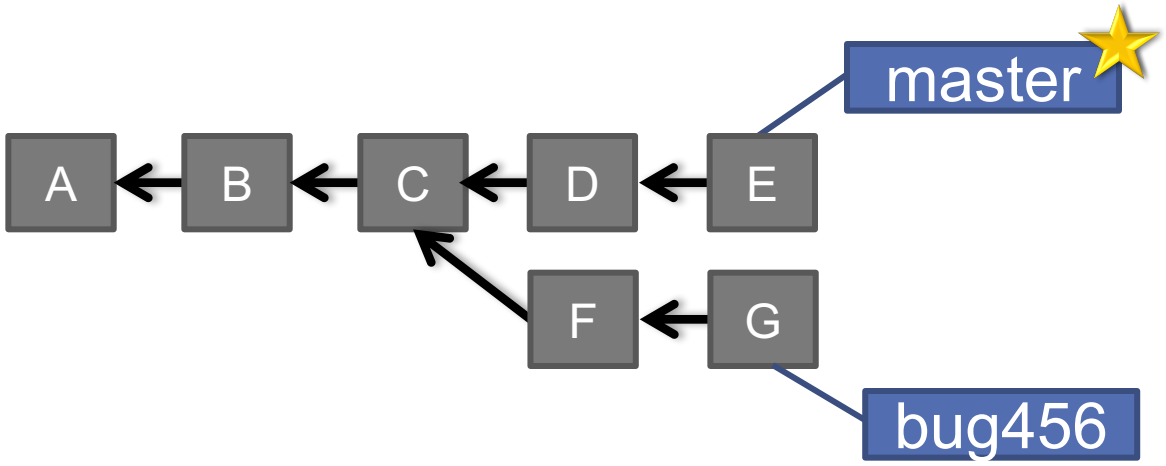



```
> git branch -d bug123
```

BRANCHES ILLUSTRATED

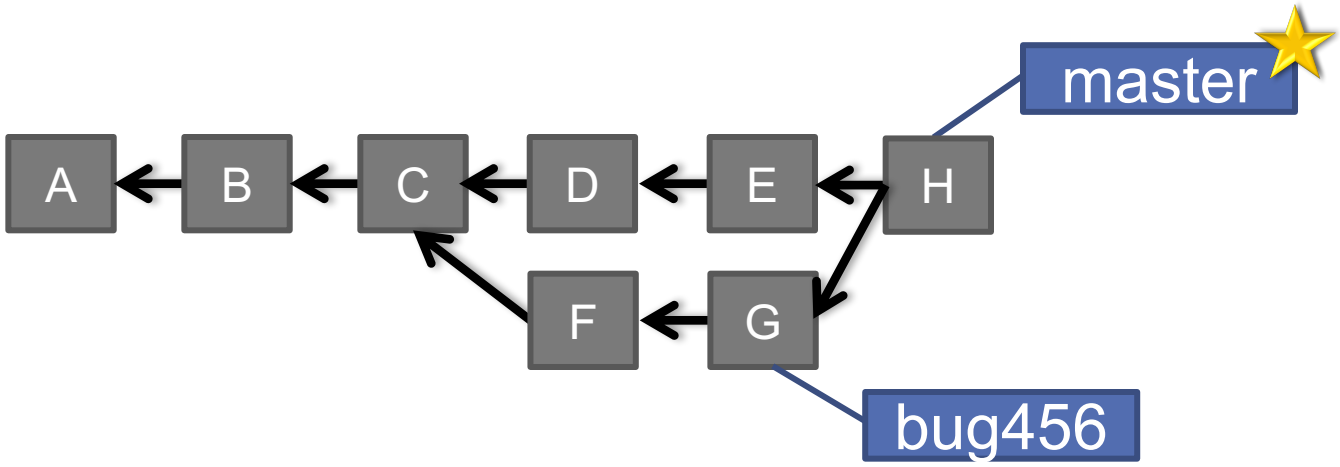


BRANCHES ILLUSTRATED



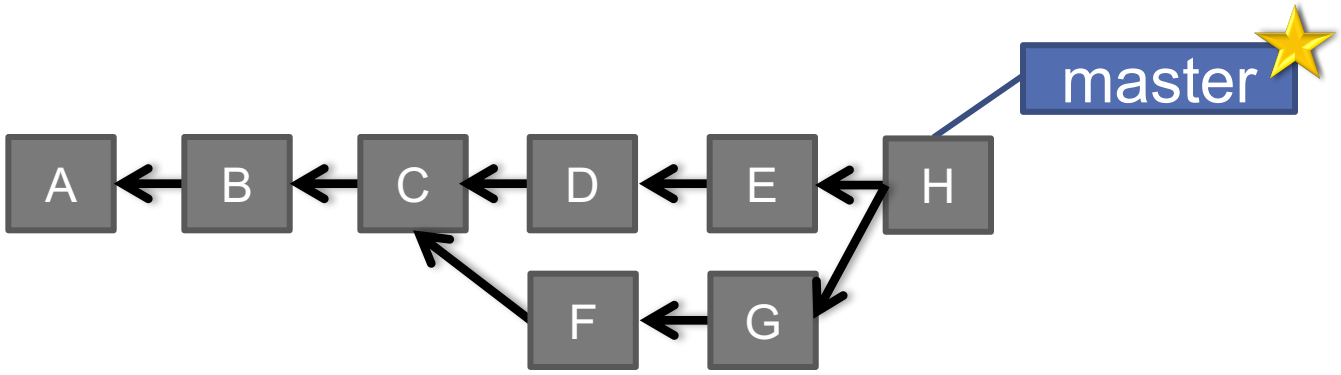

```
> git checkout master
```

BRANCHES ILLUSTRATED



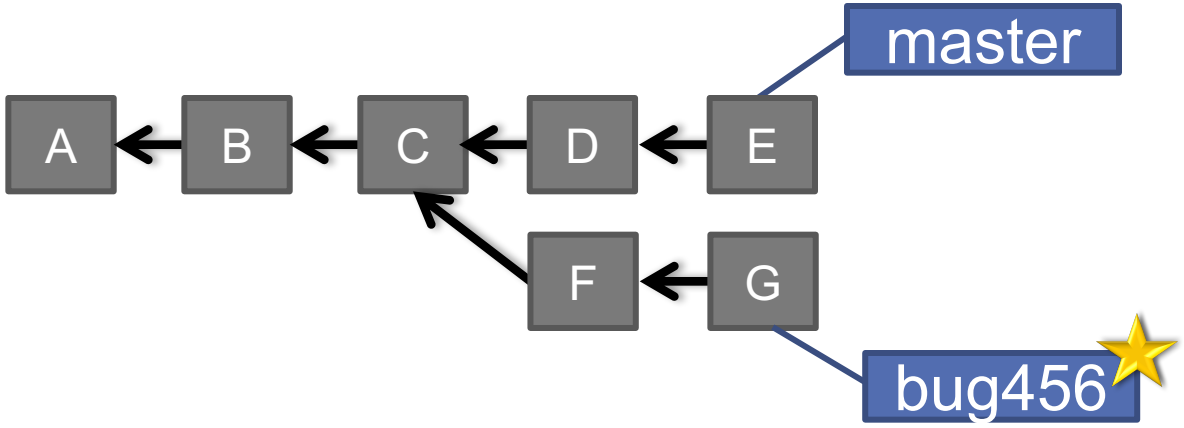

```
> git merge bug456
```

BRANCHES ILLUSTRATED

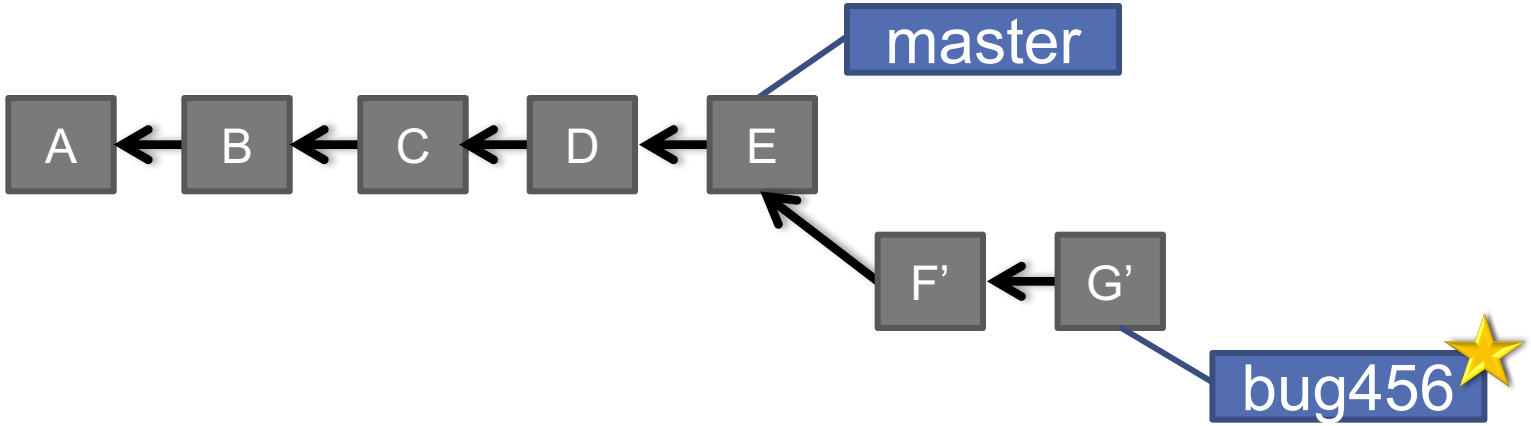



```
> git branch -d bug456
```

BRANCHES ILLUSTRATED

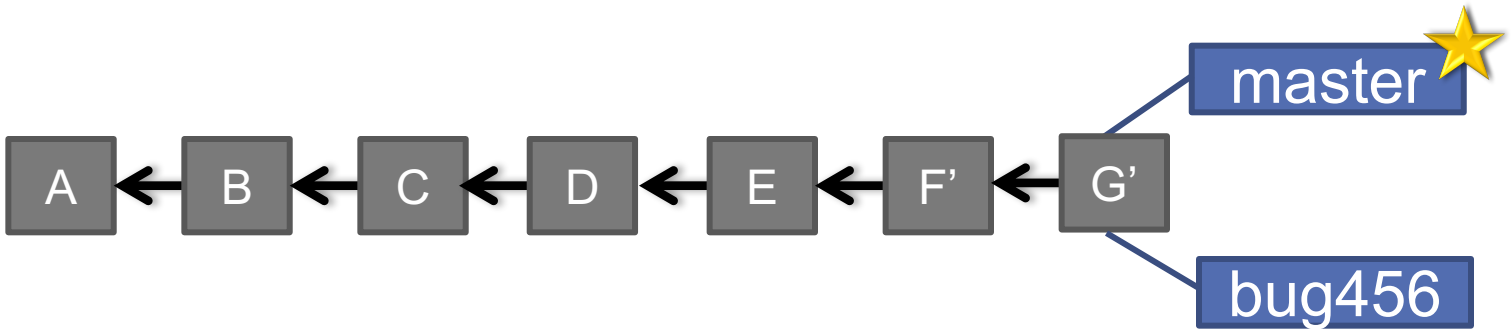


BRANCHES ILLUSTRATED




```
> 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.**

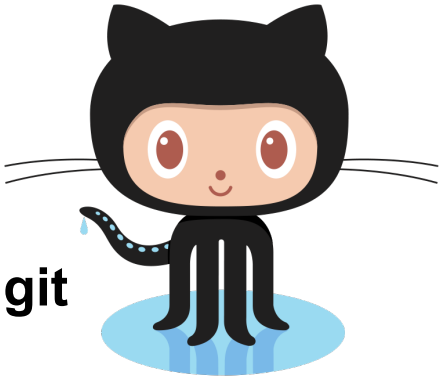
Local branching is very lightweight!

- 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 ☺.**

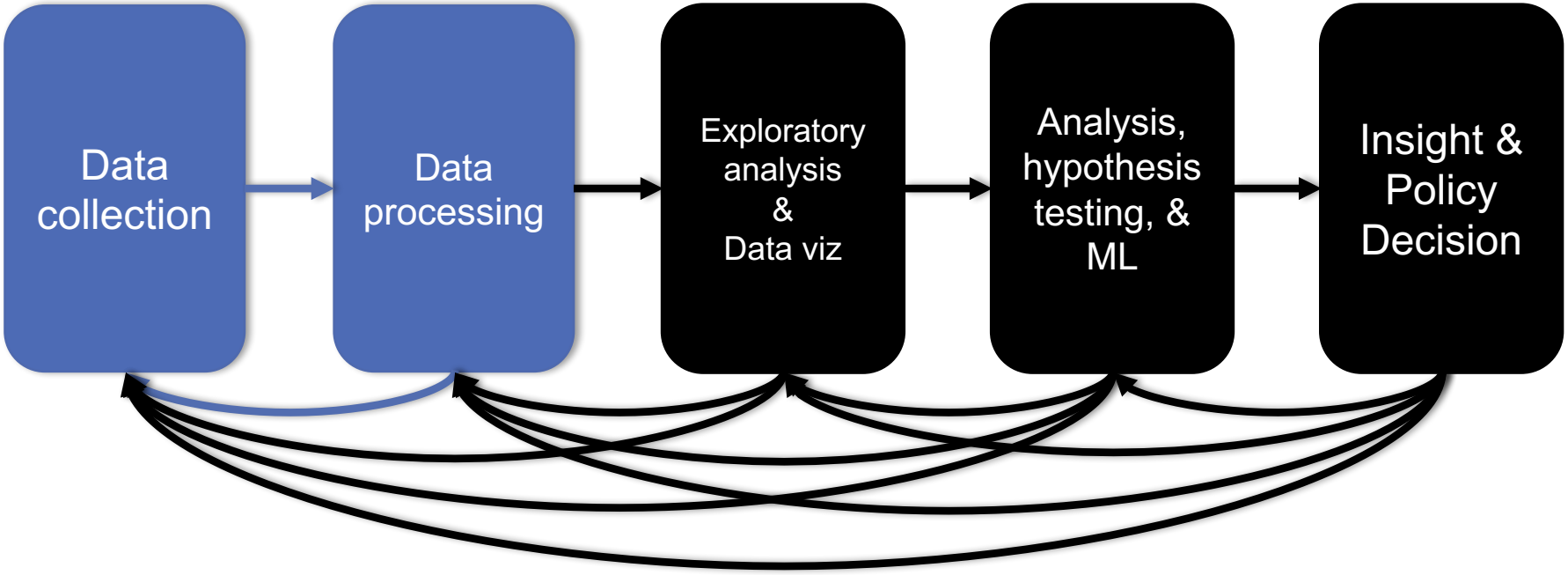
GitLab

- http://gitlab.org/

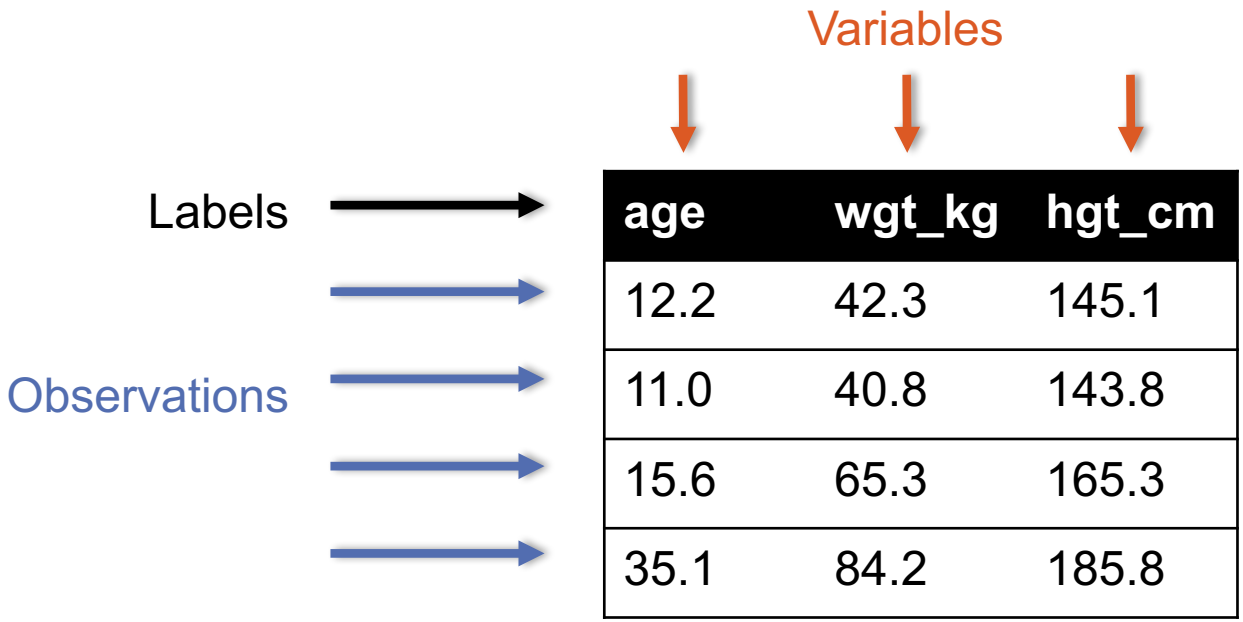
Git and SVN Comparison

- https://git.wiki.kernel.org/index.php/GitSvnComparison

THE REST OF TODAY'S LECTURE



TIDY DATA



But also:

- **Names of files/DataFrames = description of one dataset**
- **Enforce one data type per dataset (ish)**

EXAMPLE

Variable: measure or attribute:

- age, weight, height, sex

Value: measurement of attribute:

- 12.2, 42.3kg, 145.1cm, M/F

Observation: all measurements for an object

- A specific person is [12.2, 42.3, 145.1, F]

TIDYING DATA I

Name	Treatment A	Treatment B
John Smith	-	2
Jane Doe	16	11
Mary Johnson	3	1

??????????????

Name	Treatment A	Treatment B	Treatment C	Treatment D
John Smith	-	2	-	-
Jane Doe	16	11	4	1
Mary Johnson	3	1	-	2

??????????????

Thanks to http://jeannicholashould.com/tidy-data-in-python.html

TIDYING DATA II

2/21

Name	Treatment	Result
John Smith	A	-
John Smith	B	2
John Smith	C	-
John Smith	D	-
Jane Doe	A	16
Jane Doe	B	11
Jane Doe	C	4
Jane Doe	D	1
Mary Johnson	A	3
Mary Johnson	B	1
Mary Johnson	C	-
Mary Johnson	D	2

MELTING DATA I

religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k
Agnostic	27	34	60	81	76	137
Atheist	12	27	37	52	35	70
Buddhist	27	21	30	34	33	58
Catholic	418	617	732	670	638	1116
Dont know/refused	15	14	15	11	10	35
Evangelical Prot	575	869	1064	982	881	1486
Hindu	1	9	7	9	11	34
Historically Black Prot	228	244	236	238	197	223
Jehovahs Witness	20	27	24	24	21	30
Jewish	19	19	25	25	30	95

??????????????

MELTING DATA II

```
f_df = pd.melt(df,
               ["religion"],
               var_name="income",
               value_name="freq")
f_df = f_df.sort_values(by=["religion"])
f_df.head(10)
```

religion	income	freq
Agnostic	<$10k	27
Agnostic	$30-40k	81
Agnostic	$40-50k	76
Agnostic	$50-75k	137
Agnostic	$10-20k	34
Agnostic	$20-30k	60
Atheist	$40-50k	35
Atheist	$20-30k	37
Atheist	$10-20k	27
Atheist	$30-40k	52

MORE COMPLICATED EXAMPLE

Billboard Top 100 data for songs, covering their position on the Top 100 for 75 weeks, with two "messy" bits:

- Column headers for each of the 75 weeks
- If a song didn't last 75 weeks, those columns have are null

year	artist.inverted	track	time	genre	date.entered	date.peaked	x1st.week	x2nd.week	...
2000	Destiny's Child	Independent Women Part I	3:38	Rock	2000-09-23	2000-11-18	78	63.0	...
2000	Santana	Maria, Maria	4:18	Rock	2000-02-12	2000-04-08	15	8.0	...
2000	Savage Garden	I Knew I Loved You	4:07	Rock	1999-10-23	2000-01-29	71	48.0	...
2000	Madonna	Music	3:45	Rock	2000-08-12	2000-09-16	41	23.0	...
2000	Aguilera, Christina	Come On Over Baby	3:38	Rock	2000-08-05	2000-10-14	57	47.0	...
2000	Janet	Doesn't Really Matter	4:17	Rock	2000-06-17	2000-08-26	59	52.0	...

Messy columns!

Thanks to http://jeannicholashould.com/tidy-data-in-python.html

MORE COMPLICATED EXAMPLE

```
# Keep identifier variables
id_vars = ["year",
           "artist.inverted",
           "track",
           "time",
           "genre",
           "date.entered",
           "date.peaked"]

# Melt the rest into week and rank columns
df = pd.melt(frame=df,
             id_vars=id_vars,
             var_name="week",
             value_name="rank")
```

Creates one row per week, per record, with its rank

MORE COMPLICATED EXAMPLE

```
# Formatting
df["week"] = df['week'].str.extract('(\d+)',
                                    expand=False).astype(int)
df["rank"] = df["rank"].astype(int)
```

[…, "x2nd.week", 63.0] → […, 2, 63]

```
# Cleaning out unnecessary rows
df = df.dropna()

# Create "date" columns
df['date'] = pd.to_datetime(
                df['date.entered']) +
                pd.to_timedelta(df['week'], unit='w') –
                pd.DateOffset(weeks=1)
```

MORE COMPLICATED EXAMPLE

```
# Ignore now-redundant, messy columns
df = df[["year",
         "artist.inverted",
         "track",
         "time",
         "genre",
         "week",
         "rank",
         "date"]]

df = df.sort_values(ascending=True,
 by=["year","artist.inverted","track","week","rank"])

# Keep tidy dataset for future usage
billboard = df

df.head(10)
```

MORE COMPLICATED EXAMPLE

year	artist.inverted	track	time	genre	week	rank	date
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	1	87	2000-02-26
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	2	82	2000-03-04
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	3	72	2000-03-11
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	4	77	2000-03-18
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	5	87	2000-03-25
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	6	94	2000-04-01
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	7	99	2000-04-08
2000	2Ge+her	The Hardest Part Of Breaking Up (Is Getting Ba...	3:15	R&B	1	91	2000-09-02
2000	2Ge+her	The Hardest Part Of Breaking Up (Is Getting Ba...	3:15	R&B	2	87	2000-09-09
2000	2Ge+her	The Hardest Part Of Breaking Up (Is Getting Ba...	3:15	R&B	3	92	2000-09-16

??????????????

MORE TO DO?

Column headers are values, not variable names?

- Good to go!

Multiple variables are stored in one column?

- Maybe (depends on if genre text in raw data was multiple)

Variables are stored in both rows and columns?

- Good to go!

Multiple types of observational units in the same table?

- Good to go! One row per song's week on the Top 100.

A single observational unit is stored in multiple tables?

- Don't do this!

Repetition of data?

- Lots! Artist and song title's text names. Which leads us to …

NEXT CLASS:

RELATIONAL DATABASE STUFF