NIELIT CHANDIGARH
May 29, 2025

Git & GitHub

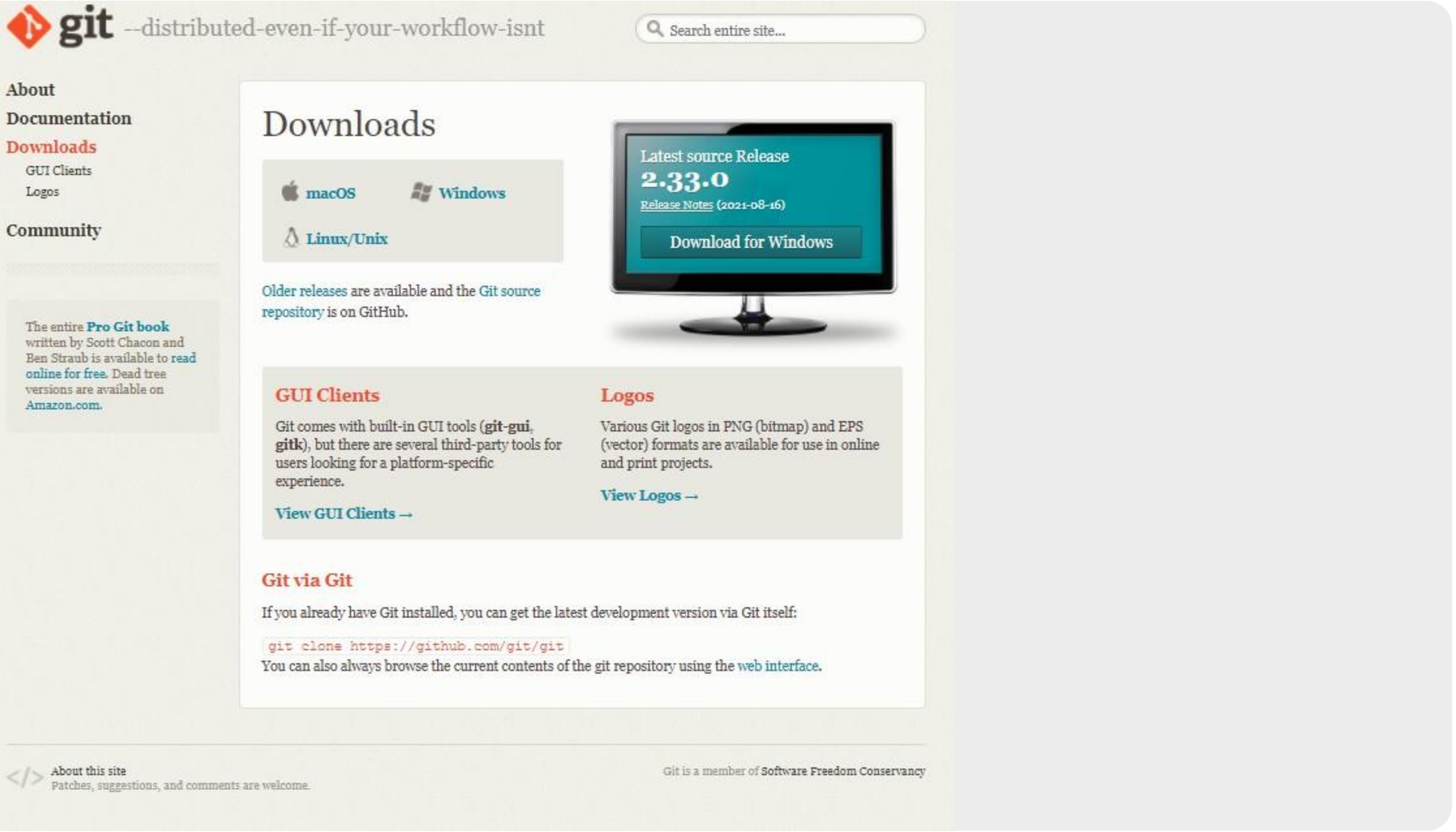
Today's Agenda

1. What is Git & GitHub?
2. Understanding GitHub Workflow
3. Practical session
4. Bonus ✨ + Wrap up

Prerequisite

https://git-scm.com/downloads

Git

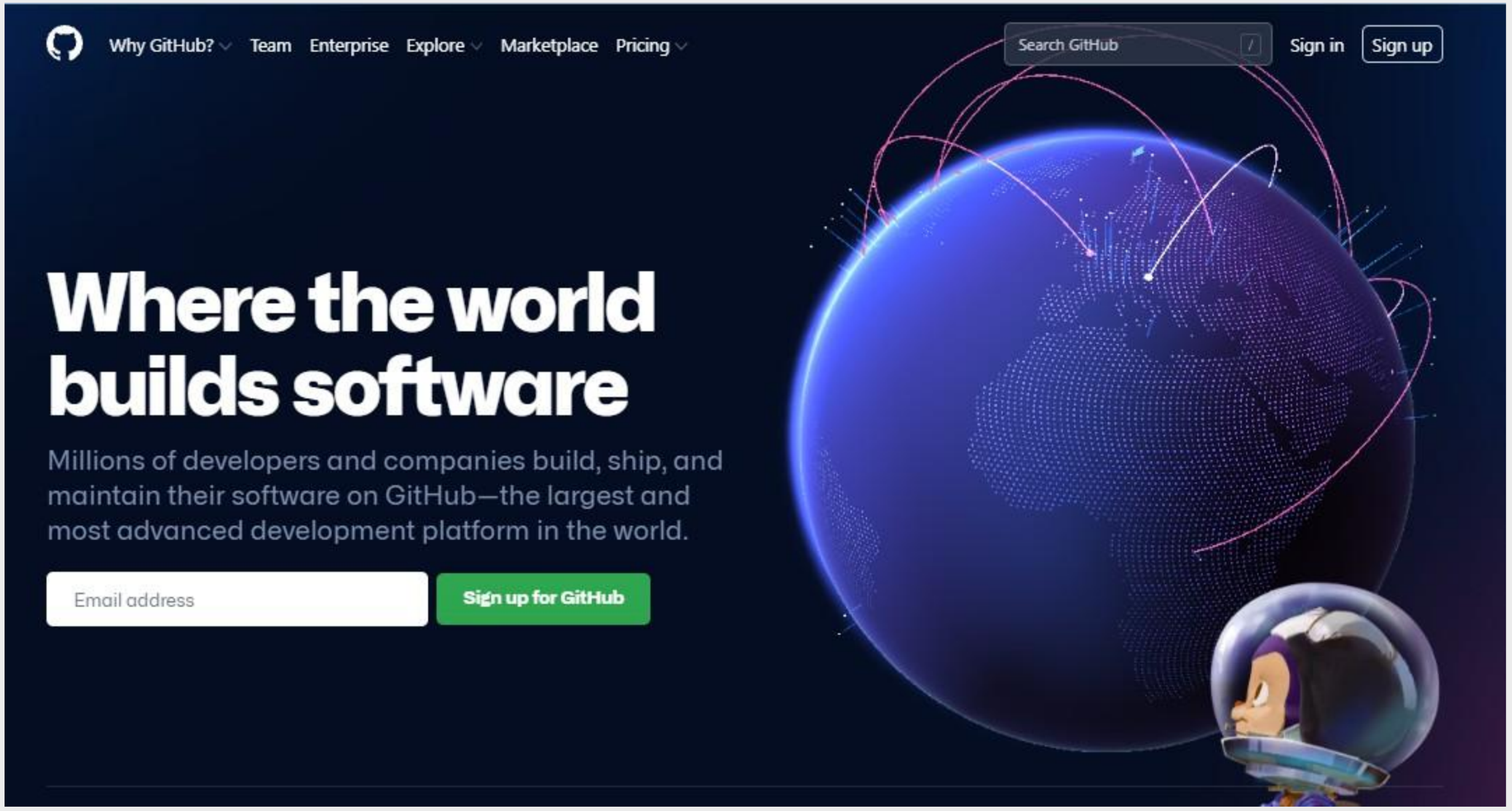


Take: 5 minutes

Prerequisite

https://github.com/

GitHub Account



Take: 5 minutes

Git

- Version Control System
- Save versions and
  Easily revert changes
- The project is called a
  "Repository"

GitHub

- Host for Git Repositories
- Store Repositories in the
  Cloud
- Easily collaborate over the
  Internet

WHAT IS VERSION CONTROL AND

WHY USE IT?

VERSION CONTROL

- A system that manages and keeps records of changes made to files
- Allows for collaborative development
- Allows you to know who made what changes and when
- **Allows you to revert any changes and go back to a previous state**

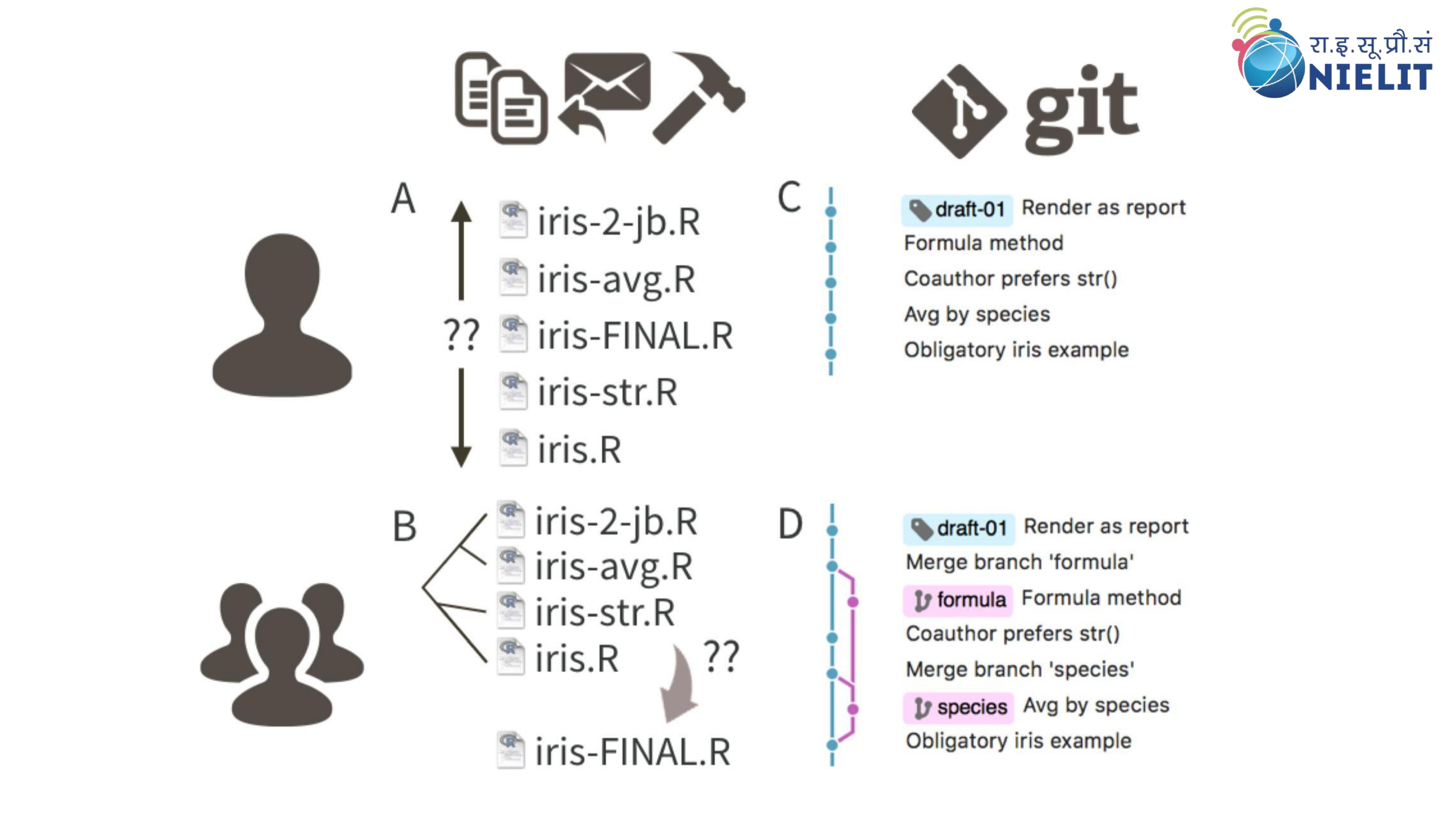
Article

Excuse Me, Do You Have a Moment to Talk About Version Control?

Jennifer Bryan

Pages 20-27 | Received 01 Jul 2017, Accepted author version posted online: 14 Nov 2017, Published online: 24 Apr 2018

Download citation https://doi.org/10.1080/00031305.2017.1399928 Check for updates


git
A
iris-2-jb.R
iris-avg.R
??
iris-FINAL.R
iris-str.R
iris.R
C
draft-01 Render as report
Formula method
Coauthor prefers str()
Avg by species
Obligatory iris example
B
iris-2-jb.R
iris-avg.R
iris-str.R
iris.R
??
iris-FINAL.R
D
draft-01 Render as report
Merge branch 'formula'
formula Formula method
Coauthor prefers str()
Merge branch 'species'
species Avg by species
Obligatory iris example

WHAT ARE GIT AND GITHUB?

- Git and GitHub are two different things
- Git is a particular implementation of version control originally designed by Linus Torvalds in 2005 as a way of managing the Linux kernel. Git manages the evolution of a set of files – called a repository or repo
- Essentially, the language of version control

- GitHub is an online hub for hosting Git repositories and provides GUI software for using Git
- GitHub complements Git by providing a slick user interface and distribution mechanism for repositories. Git is the software you will use locally to record changes to a set of files. GitHub is a hosting service that provides a Git-aware home for such projects on the internet
- GitHub is like DropBox or Google Drive, but more structured, powerful, and programmatic

What is Git?

- ✓ Git is a distributed version control system
- ✓ Tracks changes in source code during software development
- ✓ Helps collaborate without overwriting each other’s work

What is GitHub?

- GitHub is a cloud-based hosting service for Git repositories
- Allows collaboration, code sharing, pull requests, and issue tracking
- Supports open-source and private projects

Setting Up Git (Quick Setup)

1. Install Git from git-scm.com
2. Configure user info:

```
git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"
```

Starting a Git Repository

- Initialize a Git repo locally:

```
git init
```

- Add files to staging area:

```
git add filename.txt
```

- Commit files:

```
git commit -m "Initial commit"
```

Creating and Cloning a GitHub Repository

- Create a repo on GitHub at github.com/new
- Copy repo URL (HTTPS or SSH)
- Clone to local machine:

git clone https://github.com/username/repo.git

Git Clone

What is git clone?

•git clone is a **command to create a local copy** of an existing remote Git repository.

•It downloads the entire repository — including all files, commits, branches, and history.

•The cloned repo will automatically have the remote origin set to the URL you cloned from.

When to use git clone?

•When you want to start working on an existing project hosted on GitHub (or any Git server).

•Instead of creating a new repo, you copy an existing one to your local machine.

```
# Go to the folder where you want to clone the repo
cd projectfolder

# Clone the repo
git clone https://github.com/username/reponame.git

# Move into the cloned directory
cd reponame

# Check remote
git remote -v
```

Common next steps after cloning:

•Pull latest changes (if needed):

```
git pull origin main
```

•Make changes, then:

```
git add . git commit -m "Your commit message"
git push origin main
```

Pushing Changes to GitHub

- `cd /path/to/your/project`
- `git init`
- `git add .`
- `git commit -m "Initial commit"`
- `git remote add origin https://github.com/username/repo.git`
- `git push -u origin master`

Push Guide:
https://github.com/lovnishverma/lovnishverma/blob/main/push.md

Pulling Changes from GitHub

1.Initialize repo (if new):
git init

2.Connect to remote:
git remote add origin https://github.com/username/repo.git

3.Pull from remote (to get the latest code if any):
git pull origin main

4.Make your changes

5.Add, commit, and push your changes

Important

•If you cloned the repository directly from GitHub, e.g.:

git clone https://github.com/username/repo.git
then the remote is **already connected** and you don't need to add remote manually.

Pull Guide: https://github.com/lovnishverma/lovnishverma/blob/main/pull.md

Branching Basics

- Branch = separate line of development
- Create a branch:
- git checkout -b feature-branch

Switch branches:
git checkout master

Command	Description
git branch <branch-name>	Create a new branch
git checkout <branch-name>	Switch to an existing branch
git checkout -b <branch-name>	Create **and** switch to a new branch
git branch	List all branches and show current branch

Example Workflow:

```
# Create and switch to new branch
git checkout -b feature-login

# Work, add, commit your changes here

# Switch back to main branch
git checkout main

# Merge feature-login changes into main (optional)
git merge feature-login
```

Making Changes in Branch

1. Switch to the branch you want to work on

```
git checkout <branch-name>
```

Example:

```
git checkout feature-login
```

2. Make your changes in the files

•Open files in your editor

•Add or modify code, text, or resources

3. Check the status to see changed files

```
git status
```

Shows which files have been modified, added, or deleted.

4. Stage the changes for commit

•Stage all changes:

```
git add .
```

•Or stage specific files:

```
git add filename1 filename2
```

5. **Commit your changes with a descriptive message**

```
git commit -m "Add login feature
with form validation"
```

git push -u origin <branch-name>

Example:

```
git push -u origin feature-login
```

For subsequent pushes, simply:

```
git push
```

7. Collaborate

Others can now pull your branch:

```
git pull origin <branch-name>
```

Or review your changes via Pull Requests on GitHub.

Step	Command	Description
Switch to branch	git checkout <branch-name>	Work on desired branch
Check status	git status	See modified files
Stage changes	git add . or git add <files>	Prepare files for commit
Commit changes	git commit -m "message"	Save changes locally
Push branch (first time)	git push -u origin <branch-name>	Upload branch and commits
Push branch (after)	git push	Upload new commits

Merging Branches

What is merging?

Merging takes the changes from one branch (e.g., a feature branch) and integrates them into another branch (usually main or master).This lets you combine completed work into the main codebase.

Typical Workflow for Merging

1. Switch to branch to merge into (e.g., master):

```
git checkout master
```

2. Update your current branch

Make sure it's up to date with the remote repository:

```
git pull origin master
```

3. Merge the other branch into this branch

Example: merge feature-login branch into master:

```
git merge feature-login
```

4. Resolve any merge conflicts (if they occur)

- Git will tell you which files have conflicts.
- Open those files, look for conflict markers (<<<<<<, ======, >>>>>>).
- Edit to keep desired code.
- Stage the resolved files:

```
git add <file>
```

- Then complete the merge commit:

```
git commit
```

5. Push the merged changes to remote

```
git push origin main
```

Resolving Merge Conflicts (Overview)

- Sometimes changes conflict
- Git will mark conflicting files
- Open conflicted files and edit
- After resolving:

git add conflicted-file.txt
git commit

Guide:
https://github.com/lovnishverma/lovnishverma/blob/main/pull.md#step-5-resolve-merge-conflicts-if-any

Deleting a Branch

After merging, delete branch locally:
git branch -d feature-branch # Delete local branch

Delete branch on remote:

git push origin --delete feature-branch # Delete remote branch

Useful Git Commands Cheat Sheet

Command	Description
git status	Show current repo status
git log	View commit history
git branch	List branches
git checkout <branch>	Switch branches
git pull	Pull changes from remote
git push	Push commits to remote
git merge <branch>	Merge branch into current
git clone <url>	Clone remote repo

What is Git LFS?

•Git LFS is an extension for Git to manage **large files** (e.g., images, videos, datasets) efficiently.

•Instead of storing large files directly in the Git repository, it stores **pointers** to the files.

•The actual large files are stored on a separate server or cloud storage.

•Helps keep the Git repo lightweight and fast.

Why Use Git LFS?

•Avoid bloated repository size.

•Faster cloning and fetching.

•Better handling of binary files that don't diff well.

Basic Git LFS Workflow

```
# Install Git LFS (one-time)
git lfs install

# Track large file types
git lfs track "*.psd"

# Add files normally
git add largefile.psd

# Commit and push as usual
git commit -m "Add large file"
git push origin main
```

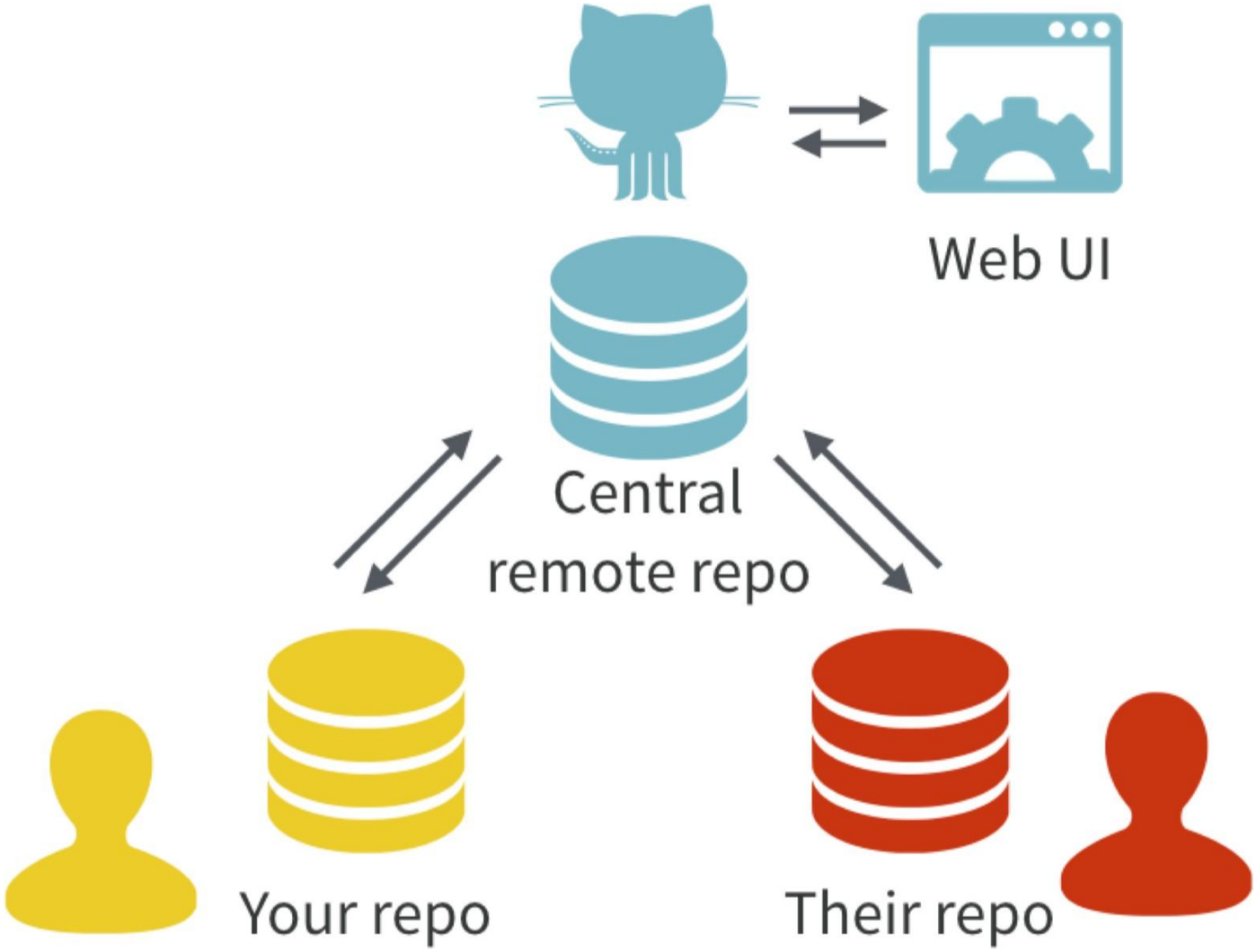
GIT LFS GUIDE:
https://github.com/lovnishverma/lovnishverma/blob/main/Git%20LFS%20(Large%20File%20Storage).md

Summary and Best Practices

- Commit often with clear messages
- Use branches for features and bug fixes
- Pull frequently to avoid conflicts
- Write meaningful commit messages
- Always review changes before pushing
- GitHub imposes a 100MB file size limit for files in a repository and recommends using Git LFS (Large File Storage) for files larger than 50MB.

Demo Time

```
# Initialize repo
git init

# Create file and commit
echo "Hello GitHub" > README.md
git add README.md
git commit -m "Add README"

# Connect remote repo
git remote add origin https://github.com/username/demo.git

# Push to GitHub
git push -u origin master

# Create and switch branch
git checkout -b feature-update

# Make changes and commit
echo "New feature" >> README.md
git add README.md
git commit -m "Add new feature"
```

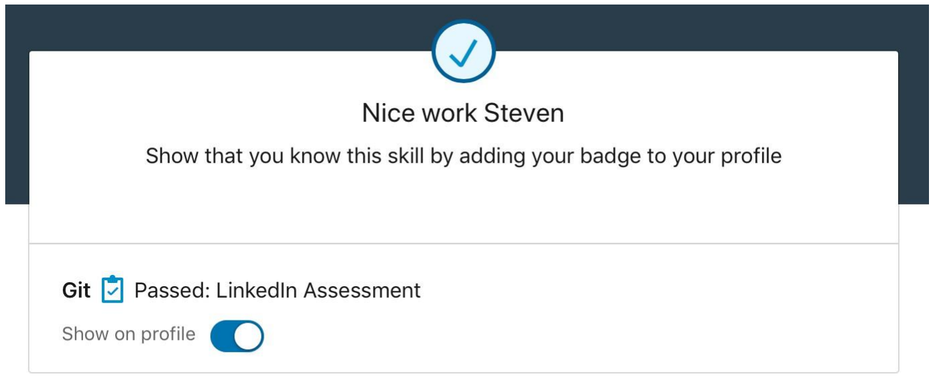


With Git, all contributors have a copy of the repo, with all files and the full history. It is typical to stay in sync through the use of a central remote repo, such as GitHub. Hosted remotes like GitHub also provide access to the repo through a web browser.

GIT CAN BE COMPLICATED AT FIRST BUT IS EASY TO LEARN

Here are some great places to go to learn Git:

- Udemy - Git complete step-by-step guide
- TeamTreehouse - Git Basics
- Codecademy - Learn Git
- Lynda.com - Git Essential Training
- atlassian.com - Getting Git Right



GLOSSARY:

- `cd` - change directory
- directory - the same thing as a folder
- `ls` - list the files and folders in a folder
- `touch` - create an empty file
- repository - the saved history of a folder and files, used by git.
- `init` - start or initialize a git repository
- `add` - put a file into the staging area, so that git starts tracking it
- staging/index area - where files are stored before going into the history
- `commit` - send files in the staging/index area into the history (the git repository)
- `status` - check the status of the folder and the git repository
- `diff` - compare a file to the a file in the history
- `log` - view the commit history in the git repository

MANY GIT COMMANDS CAN BE EXECTUED WITH GITHUB

```
cd ~/Desktop ## Move to your desktop
mkdir playing ## Create a folder (aka directory)
cd playing
git init ## Create the repository (init = initialize)
```

```
touch bio.txt ## Command to create a file called bio.txt
ls ## Check that you created the file, ls = list files
git add bio.txt ## Track the file
## Save the file to the history with a message (-m)
git commit -m "Initial commit"
```

```
git status ## Check the activity
git diff bio.txt ## Compare to the one in the history
git add bio.txt ## This sends it to the staging area
git commit -m "Added my bio" ## This sends it to the history
```

```
git remote add origin https://github.com/yourusername/playing.git
git push origin master
git pull
```

HOW DOES GIT WORK: KEY CONCEPTS

- Git keeps track of your files using “snapshots” that record what the files in your repository look like at any given point in time
- You decide when to take a snapshot and of what files, this is known as a **commit**
    - Can be used as noun or verb
    - *“I commited code”*
      “I just made a new commit”
- Have the ability to go back and visit any commit
- A project is made up of a bunch of commits

KEY CONCEPTS: COMMIT

Commits contain 3 pieces of information:

1. Information about how the files have changed from the previous commit
2. A reference to the commit that came before it, known as the **parent commit**
3. An SHA-1 hash code (e.g. fb2d2ec5069fc6776c80b3ad6b7cbde3cade4e)

WHAT IS AN SHA-1 HASH?

SHA-1 is an algorithm and what it does is:
It takes some data as input and generates a unique 40 character string from it.

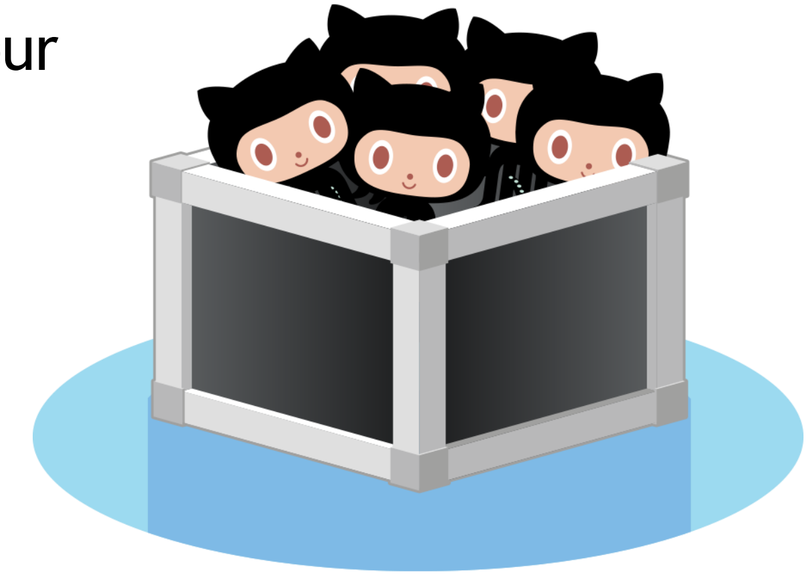
What does *unique* mean in this context? Unique means that no other input data should ever produce the same hash. The same input data however should always produce exactly the same hash.

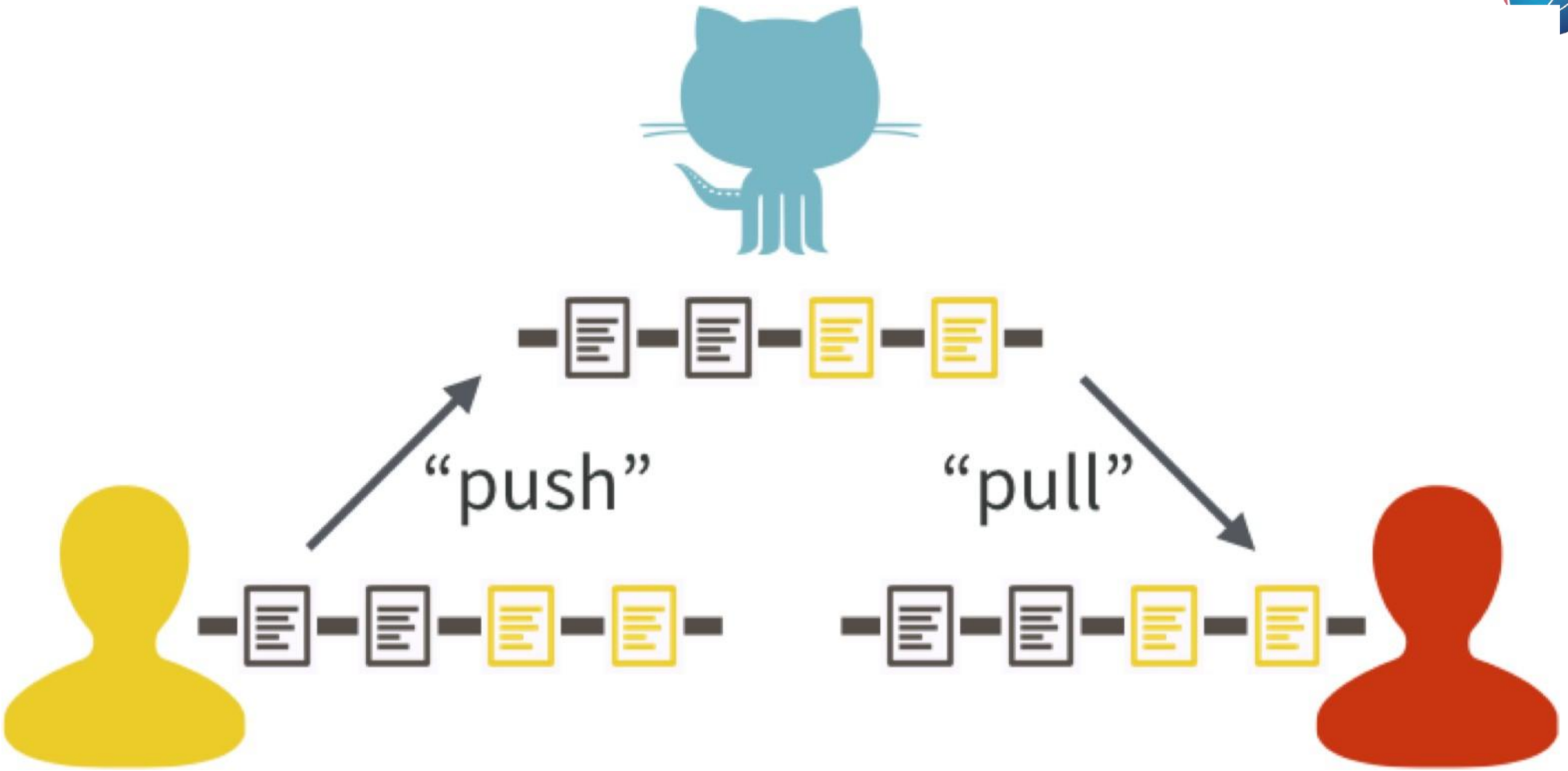
If you and I both look at `revision f4f78b319c308600eab015a5d6529add21660dc1` on our machines and Git tells us that we have a clean working directory, we can be 100% certain that we are looking at exactly the same files.

The hash ensures there is no way someone could manipulate a single bit without Git knowing about it.

KEY CONCEPTS: REPOSITORIES

- Often referred to as a **repo**, this a collection of all your files and the history of those files (i.e. commits)
- Can live on a local machine or on a remote server (e.g. GitHub)
- The act of copying a repo from a remote server is called cloning
- Cloning from a remote server allows teams to work together
- The process of downloading commits that don't exist on your machine from a remote server is called **pulling** changes
- The process of adding your local changes to a remote repo is called **pushing** changes

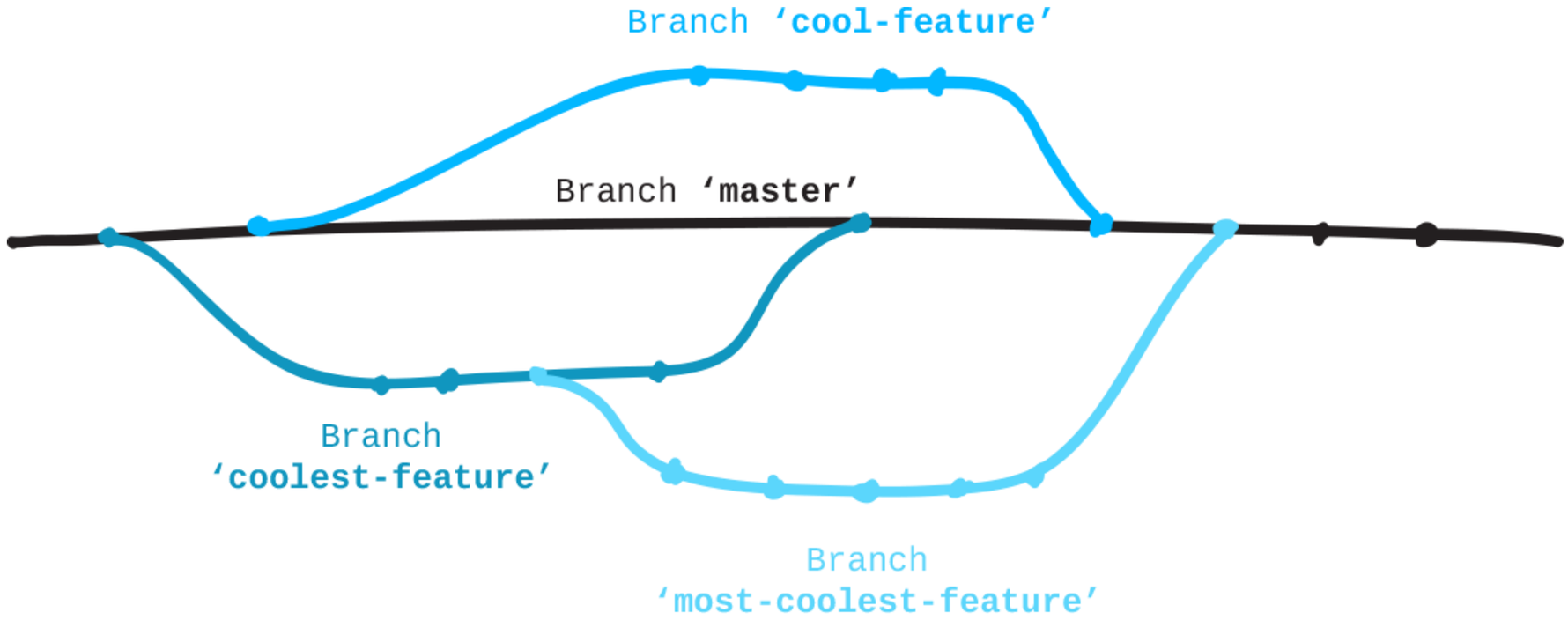


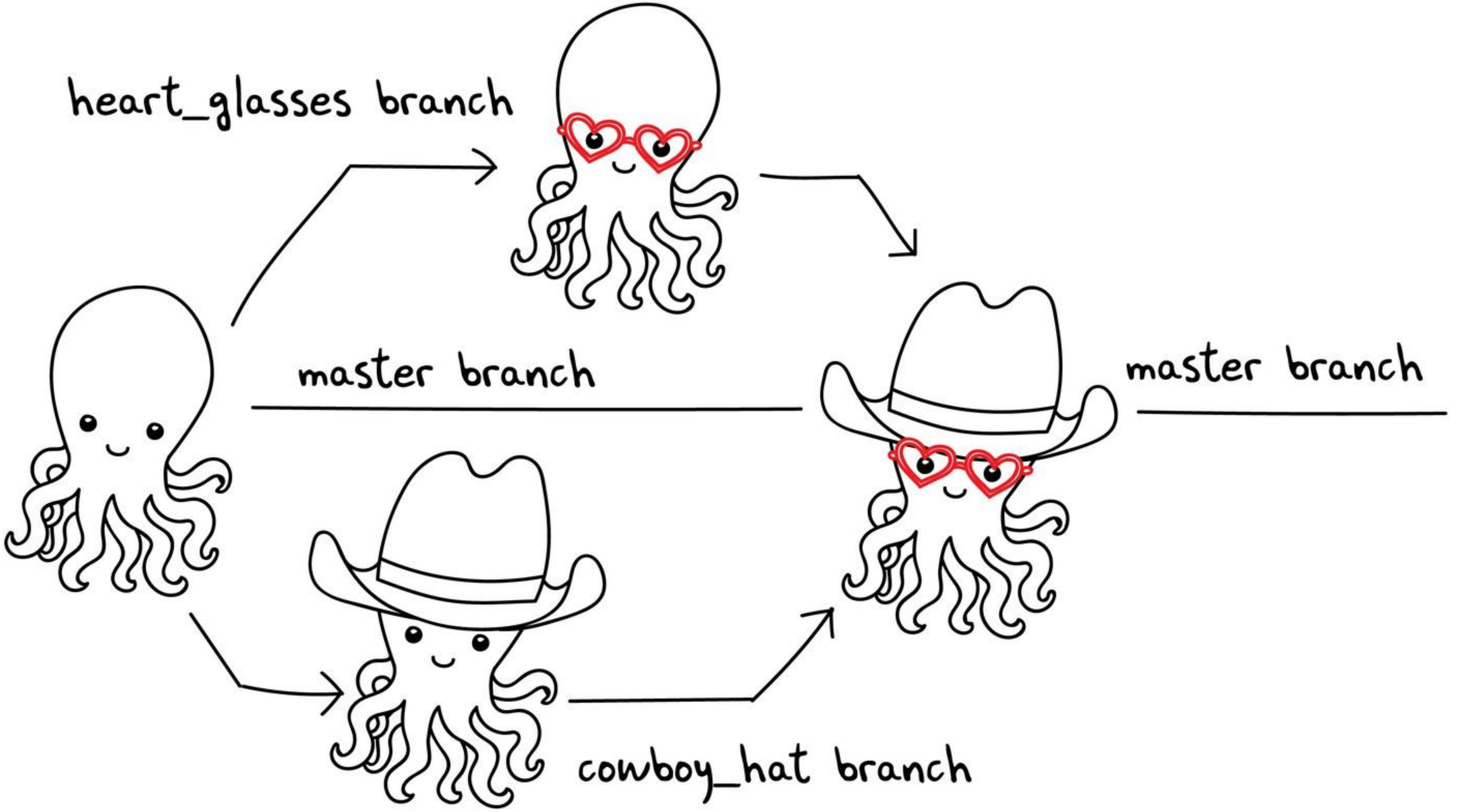


One contributor has made two new commits and updates the master copy on GitHub with a push.
Another contributor stays up-to-date with a pull from GitHub.

KEY CONCEPTS: BRANCHES

- All commits live on a branch
- There can be many branches
- The project's main branch is called the **master**



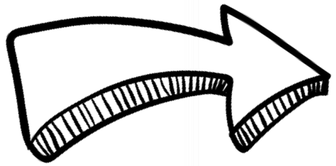

heart_glasses branch
master branch
master branch
cowboy_hat branch

KEY CONCEPTS: README.MD

- Collaborate responsibly! Every repository should be initialized with a `README.MD` file
- This is where you detail all the information about the project so that the work can be understood and recreated by another student or collaborator
- Arguably the most important file in your repo
- Use markdown syntax to make your file clear and easy to read

MARKDOWN

```
319
320  ## Analysis 2: HIV Phylogenetics
321
322  1. Run `/Code/CombineFASTAs.py` with arg1 = directory of .fasta files arg2 = outp
323  `python3 ~/github/Dissertation-Aim1/Code/CombineFASTAs.py ~/github/Dissertation-A
324
325  | HIV | Total | AM | DH | KA | IM | AZ |
326  | ---- | ---- | ---- | ---- | ---- | ---- | ---- |
327  | env | 669 | 131 | 287 | 54 | 20 | 177 |
328
329  *See `/Sequences/sample_list_env.txt` for listing of all env samples*
330  *See `Sequence_Availablility.xlsx` for full listing of samples with sequence by r
331
332  2. Remove AZ samples from combined fasta file using grep in TextWrangler.
333
334    - Find: `>AZ(?s).*?\n>` Replace: `>`
335    - Save as: `env_combined_sansAZ.fasta`
336
337  3. Determine subtype distribution using: http://dbpartners.stanford.edu:8080/Rega
338  Env Results: http://dbpartners.stanford.edu:8080/RegaSubtyping/stanford-hiv/typin
339  Pol Results: http://dbpartners.stanford.edu:8080/RegaSubtyping/stanford-hiv/typin
340
341  | HIV Subtype | Number of sequences | Percentage |
342  | ---- | ---- | ---- |
343  | C | 467 | 94.92% |
344  | A | 2 | 0.41% |
345  | Recombinant C,A1 | 8 | 1.63% |
346  | Recombinant C,B | 10 | 2.03% |
347  | Unassigned | 5 | 1.02% |
348  | Total | 492 | 100% |
349
350  4. Align sequences in combined fasta file using MUSCLE in MEGA7 with default para
351    - Gap Open: -400
352    - Gap Extend: 0
353    - Max Iterations: 8
354    - Clustering Method (Iteration 1, 2 and others): UPGMB
355    - Min Diag Length (lambda): 24
```



Analysis 2: HIV Phylogenetics

1. Run `/Code/CombineFASTAs.py` with arg1 = directory of .fasta files arg2 = output filename `python3 ~/github/Dissertation-Aim1/Code/CombineFASTAs.py ~/github/Dissertation-Aim1/Sequences/hiv_env env_combined.fasta`

HIV	Total	AM	DH	KA	IM	AZ
env	669	131	287	54	20	177

See `/Sequences/sample_list_env.txt` for listing of all env samples See `Sequence_Availablility.xlsx` for full listing of samples with sequence by region

2. Remove AZ samples from combined fasta file using grep in TextWrangler.

- Find: `>AZ(?s).*?\n>` Replace: `>`
- Save as: `env_combined_sansAZ.fasta`

3. Determine subtype distribution using: http://dbpartners.stanford.edu:8080/RegaSubtyping/stanford-hiv/typingtool/ Env Results: http://dbpartners.stanford.edu:8080/RegaSubtyping/stanford-hiv/typingtool/job/343559231/ Pol Results: http://dbpartners.stanford.edu:8080/RegaSubtyping/stanford-hiv/typingtool/job/1773838861/

HIV Subtype	Number of sequences	Percentage
C	467	94.92%
A	2	0.41%
Recombinant C,A1	8	1.63%
Recombinant C,B	10	2.03%
Unassigned	5	1.02%
Total	492	100%

4. Align sequences in combined fasta file using MUSCLE in MEGA7 with default parameters and trim sequences as necessary.

- Gap Open: -400
- Gap Extend: 0
- Max Iterations: 8
- Clustering Method (Iteration 1, 2 and others): UPGMB
- Min Diag Length (lambda): 24

MARKDOWN

Further reading:
github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet/

Headers

```
# H1
## H2
### H3
#### H4
##### H5
###### H6

Alternatively, for H1 and H2, an underline-ish style:

Alt-H1
======

Alt-H2
------
```

H1

H2

H3

H4

H5

H6

Blockquotes

```
> Blockquotes are very handy in email to emulate reply text.
> This line is part of the same quote.

Quote break.

> This is a very long line that will still be quoted properly when it wraps. Oh boy let
```

> Blockquotes are very handy in email to emulate reply text. This line is part of the same quote.

Quote break.

> This is a very long line that will still be quoted properly when it wraps. Oh boy let's keep writing to make sure this is long enough to actually wrap for everyone. Oh, you can *put* **Markdown** into a blockquote.

Tables

Tables aren't part of the core Markdown spec, but they are part of GFM and *Markdown Here* supports them. They are an easy way of adding tables to your email -- a task that would otherwise require copy-pasting from another application.

```
Colons can be used to align columns.

| Tables        | Are           | Cool  |
| ------------- |:-------------:| -----:|
| col 3 is      | right-aligned | $1600 |
| col 2 is      | centered      |   $12 |
| zebra stripes | are neat      |    $1 |

There must be at least 3 dashes separating each header cell.
The outer pipes (|) are optional, and you don't need to make the
raw Markdown line up prettily. You can also use inline Markdown.

Markdown | Less | Pretty
--- | --- | ---
*Still* | `renders` | **nicely**
1 | 2 | 3
```

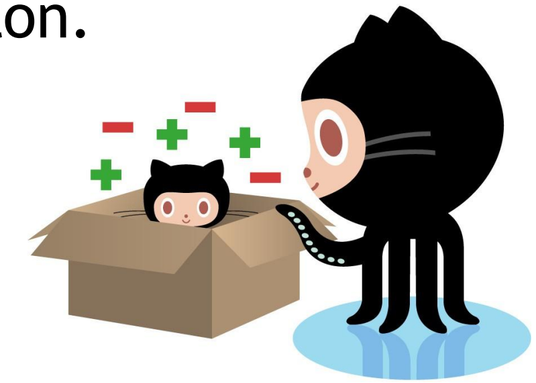
Colons can be used to align columns.

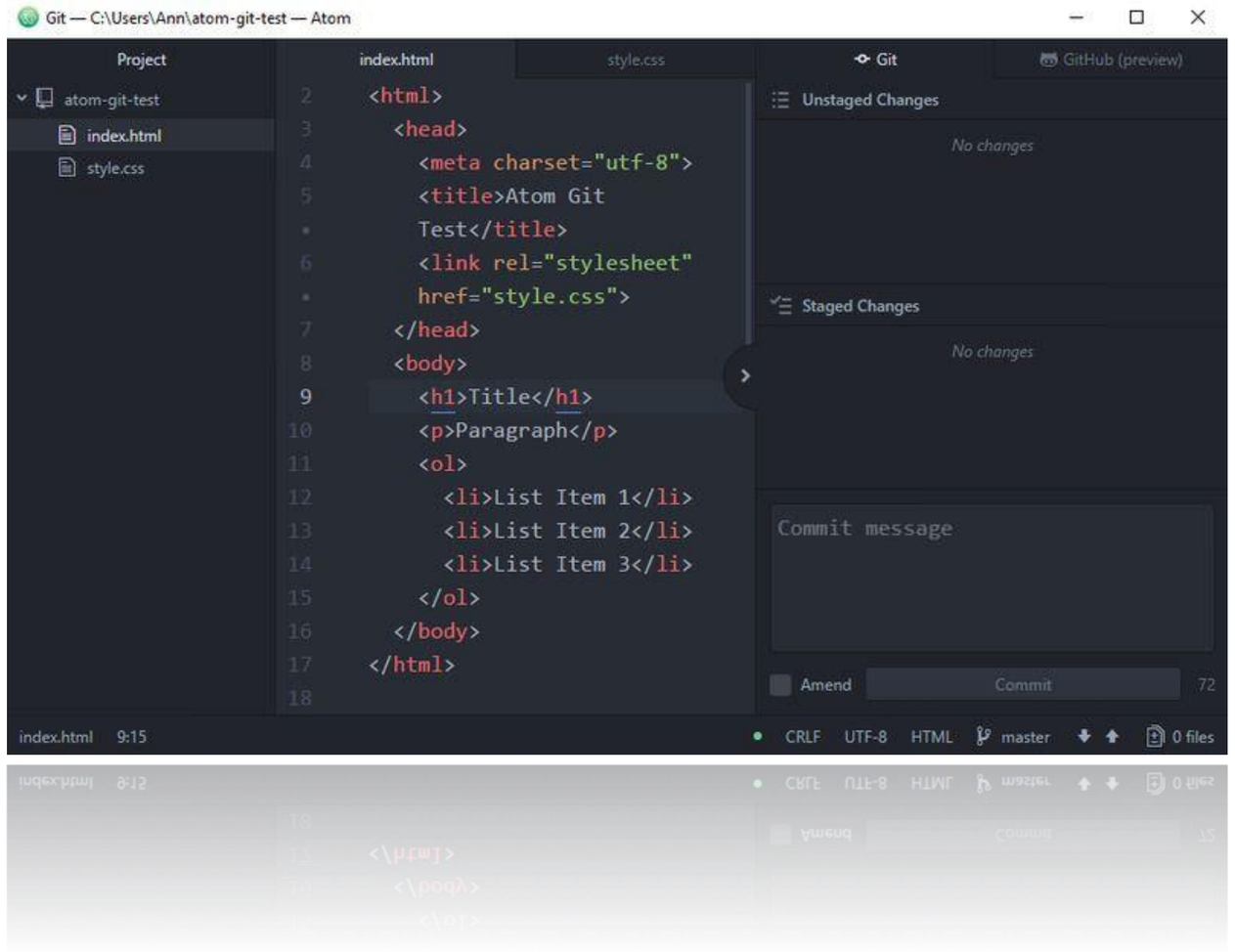
Tables	Are	Cool
col 3 is	right-aligned	$1600
col 2 is	centered	$12
zebra stripes	are neat	$1

KEY CONCEPS: MAKING ACOMMIT

The process:

- Make some changes to a file
- Use the `git add` command or Atom to add files to the **staging area**
- Type a commit message (that shortly describes the changes you made since the last commit) into the commit message box, and click the Commit button.





WRITING A GOOD COMMIT MESSAGE

The 7 rules of a great commit message:

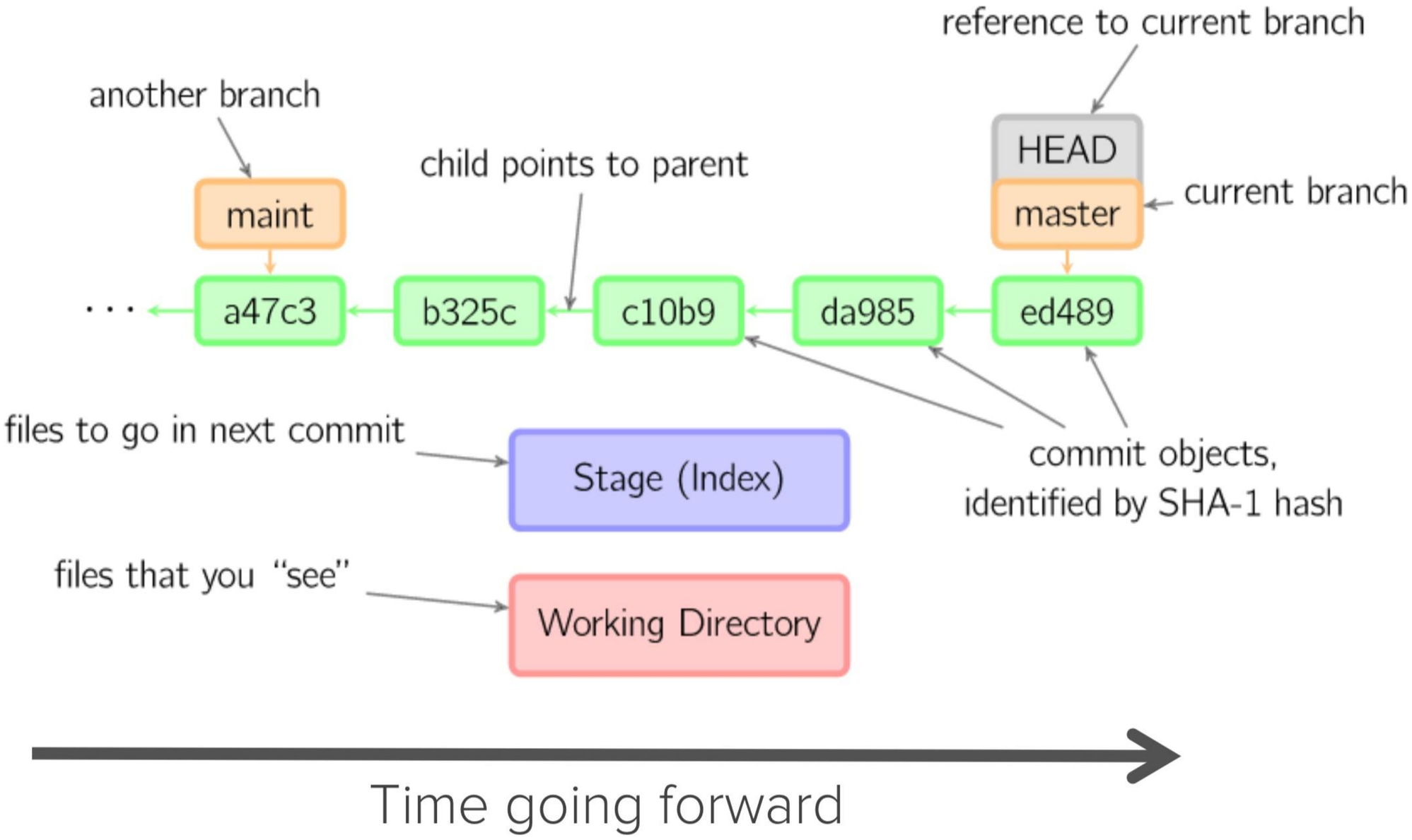
1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters
7. Use the body to explain what and *why* vs. *how*

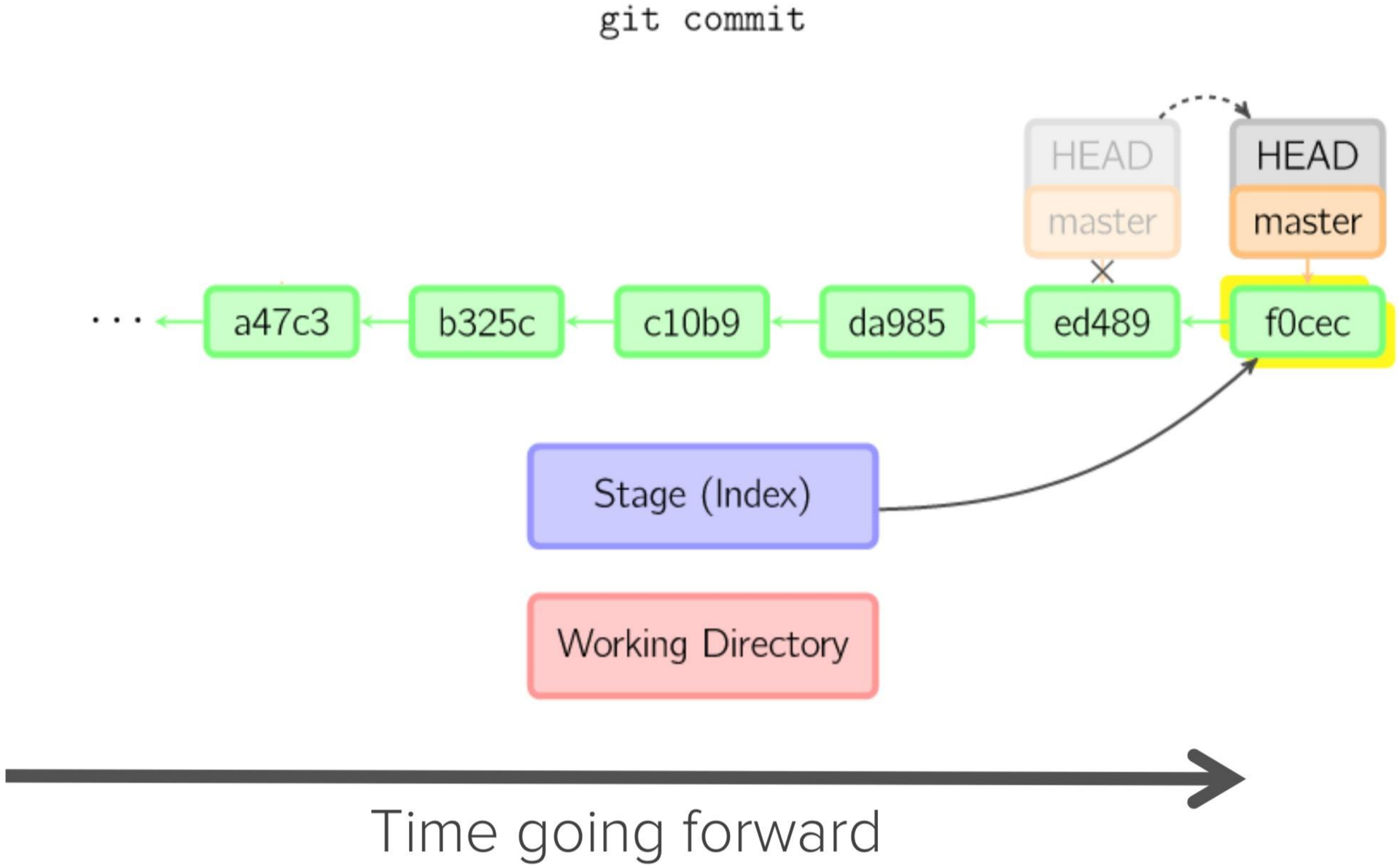
WRITING A GOOD COMMIT MESSAGE

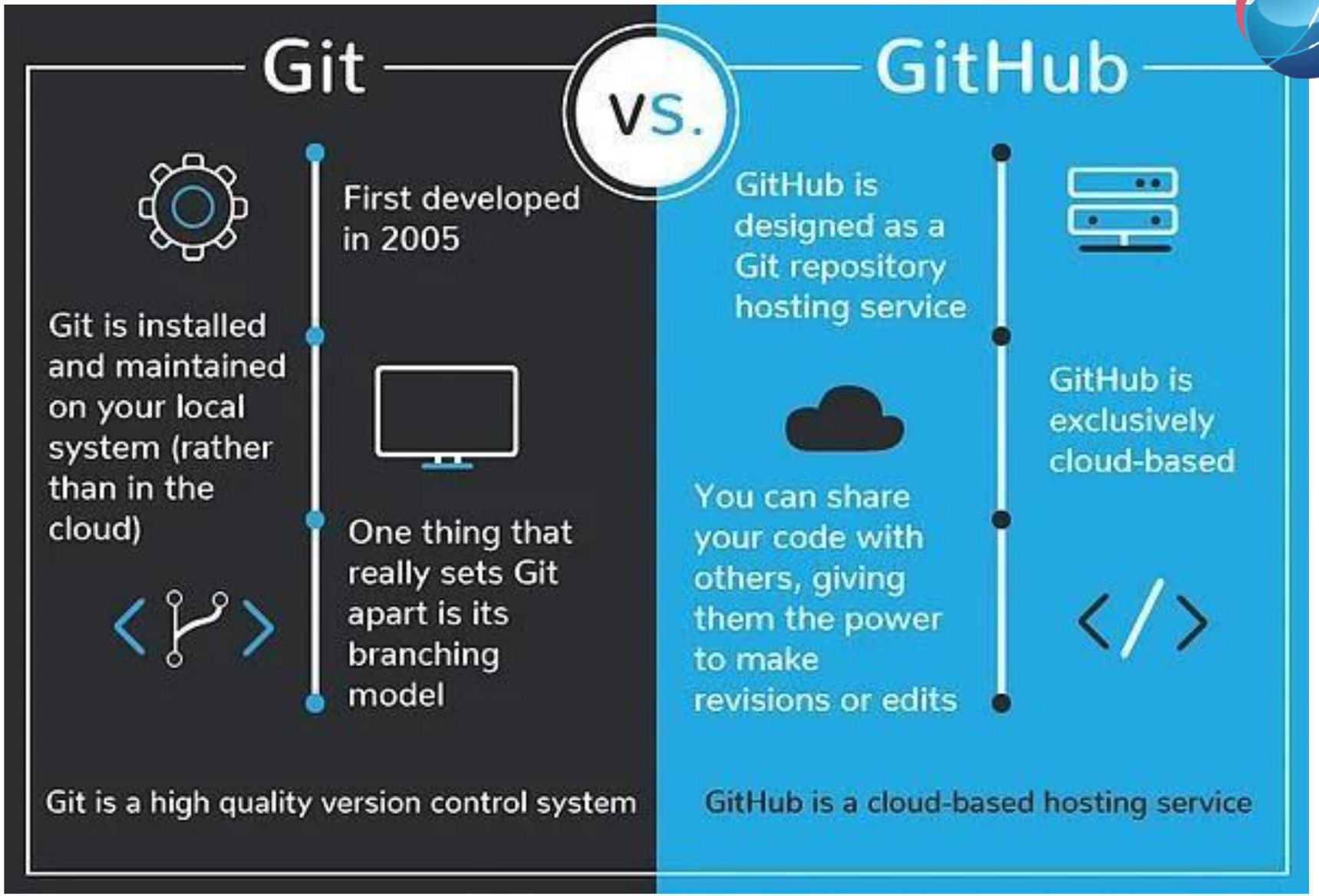
```
Summarize changes in around 50 characters or less

More detailed explanatory text, if necessary. Wrap it to about 72
characters or so. In some contexts, the first line is treated as the
subject of the commit and the rest of the text as the body. The
blank line separating the summary from the body is critical (unless
you omit the body entirely); various tools like `log`, `shortlog`
and `rebase` can get confused if you run the two together.

Explain the problem that this commit is solving. Focus on why you
are making this change as opposed to how (the code explains that).
Are there side effects or other unintuitive consequences of this
change? Here's the place to explain them.

Further paragraphs come after blank lines.

 - Bullet points are okay, too

 - Typically a hyphen or asterisk is used for the bullet, preceded
   by a single space, with blank lines in between, but conventions
   vary here

If you use an issue tracker, put references to them at the bottom,
like this:

Resolves: #123
See also: #456, #789
```

Further reading:

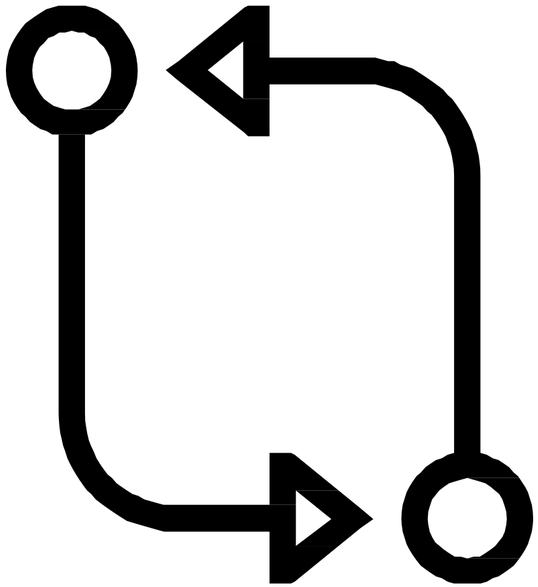
chris.beams.io/posts/git-commit/

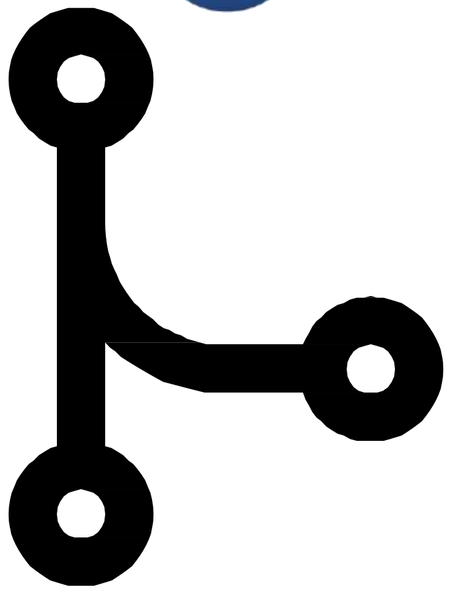

reference to current branch
another branch
HEAD
child points to parent
maint
master
current branch
. . .
a47c3
b325c
c10b9
da985
ed489
files to go in next commit
Stage (Index)
commit objects,
identified by SHA-1 hash
files that you "see"
Working Directory
Time going forward

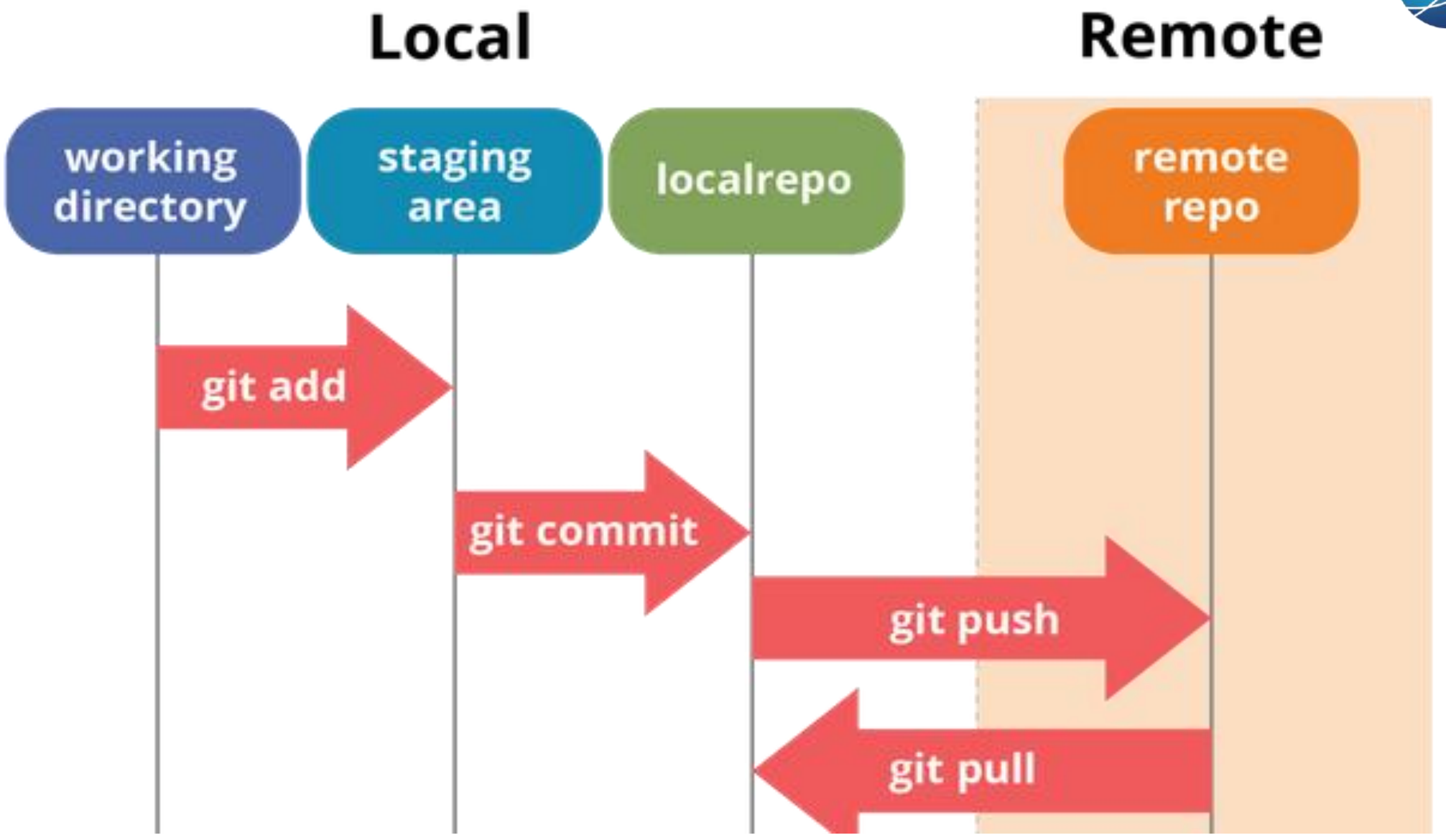




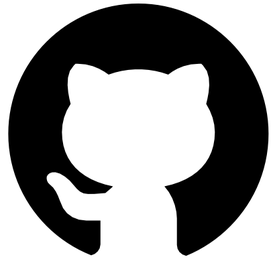
Understanding GitHub Workflow






Local
Remote
working directory
staging area
localrepo
remote repo
git add
git commit
git push
git pull

Let's Git it!



Git Commands

- git init
- git add - A
- git commit - m "message"
- git remote add origin [URL]
- git push origin main
- git status
- git log
- git branch feature
- git checkout feature

ADDITIONAL RESOURCES

Official git site and tutorial:

https://git-scm.com/

GitHub guides:

https://guides.github.com/

Command cheatsheet:

https://training.github.com/kit/_downloads/github-git-cheat-sheet.pdf

Mardown Cheatsheet:

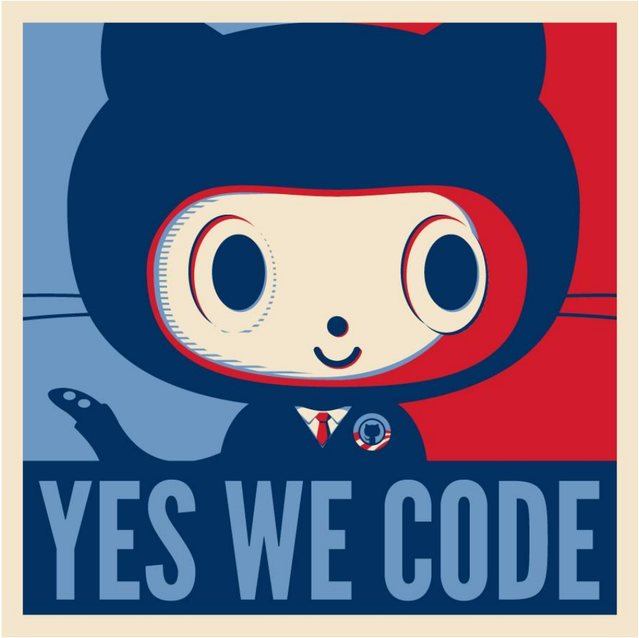
https://github.com/adam-p/markdown-here/wiki/Markdown-Here-Cheatsheet

Interactive git tutorial:

https://try.github.io/levels/1/challenges/1

Visual/interactive cheatsheet:

http://ndpsoftware.com/git-cheatsheet.html



We're done!

If you found this helpful, don't forget to follow me on GitHub.
https://github.com/lovnishverma

Repurposing version control software for science represents a transparent, replicable, and streamlined process that fosters responsible collaboration

This repo provide a clear and comprehensive demonstration of how branches and issues are utilized in the lovnishverma/mailjet repository, illustrating core GitHub functionalities.
https://github.com/lovnishverma/mailjet