Version Control Using `git`

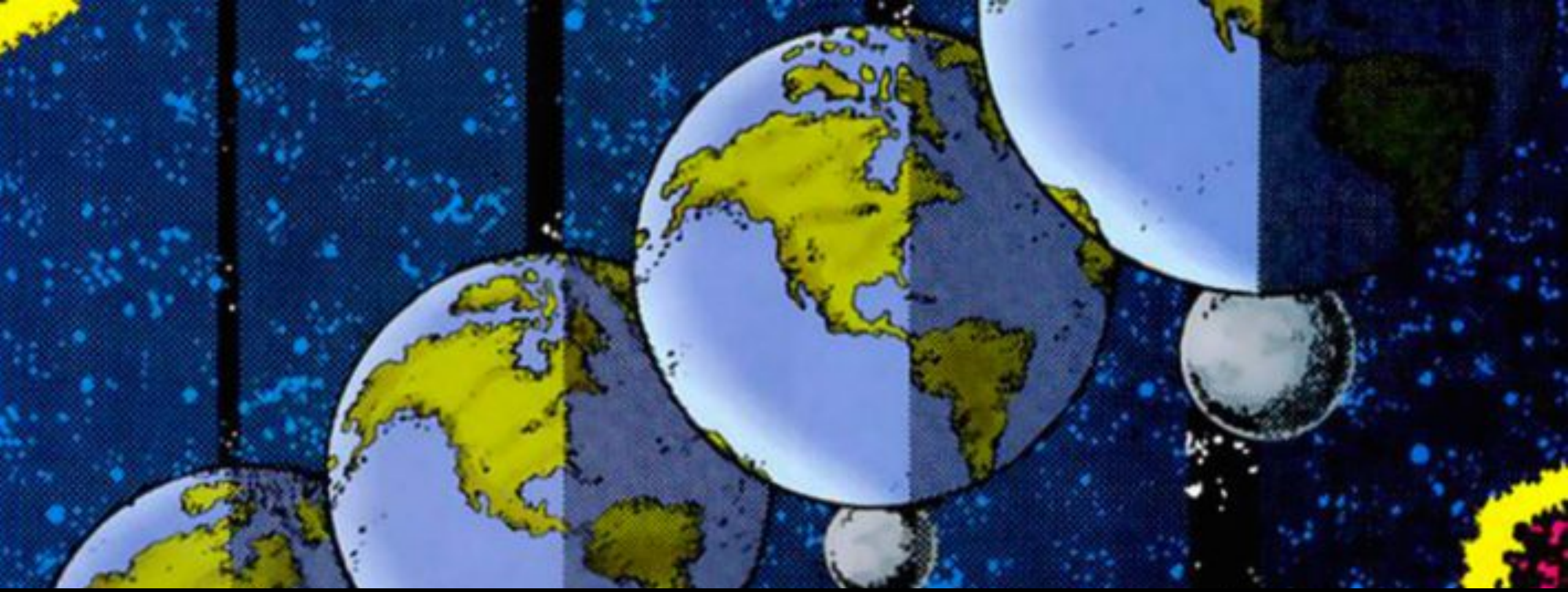
Adam Sweeney

Introduction

- For our development projects, it can be very handy to keep track of the changes we make to our code
- If something breaks, we can roll back
- We can try out new features without affecting the “main” program
- Version control is an extremely important tool used in industry

Agenda

- Version control
- Using git locally
  - Repository creation
  - Staging & Committing
  - Basic branching



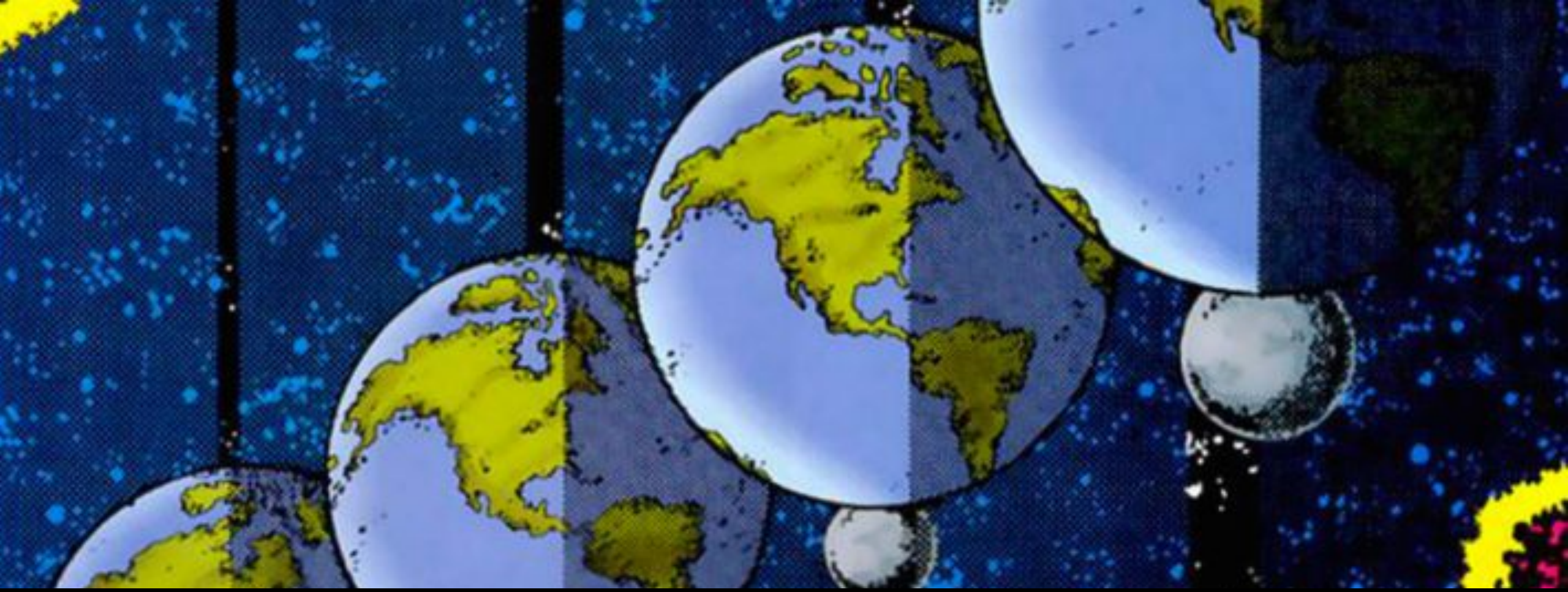
Version Control

What Even is Version Control

- The idea of version control is simple
- Create a controlled way to track changes we make to our project and provide access to all the different versions
- Version control has existed long before programming
  - Look at any textbook, it’s usually on its nth edition

We Always Have a Choice

- There are many “schemes” for software version control
  - svn
  - git
  - mercurial
  - cvs
  - …etc.
- The focus of this lecture is using the git scheme
  - Open source, developed by Linus Torvalds
  - Rapidly gaining in popularity across industry



Using git Locally

Installing git

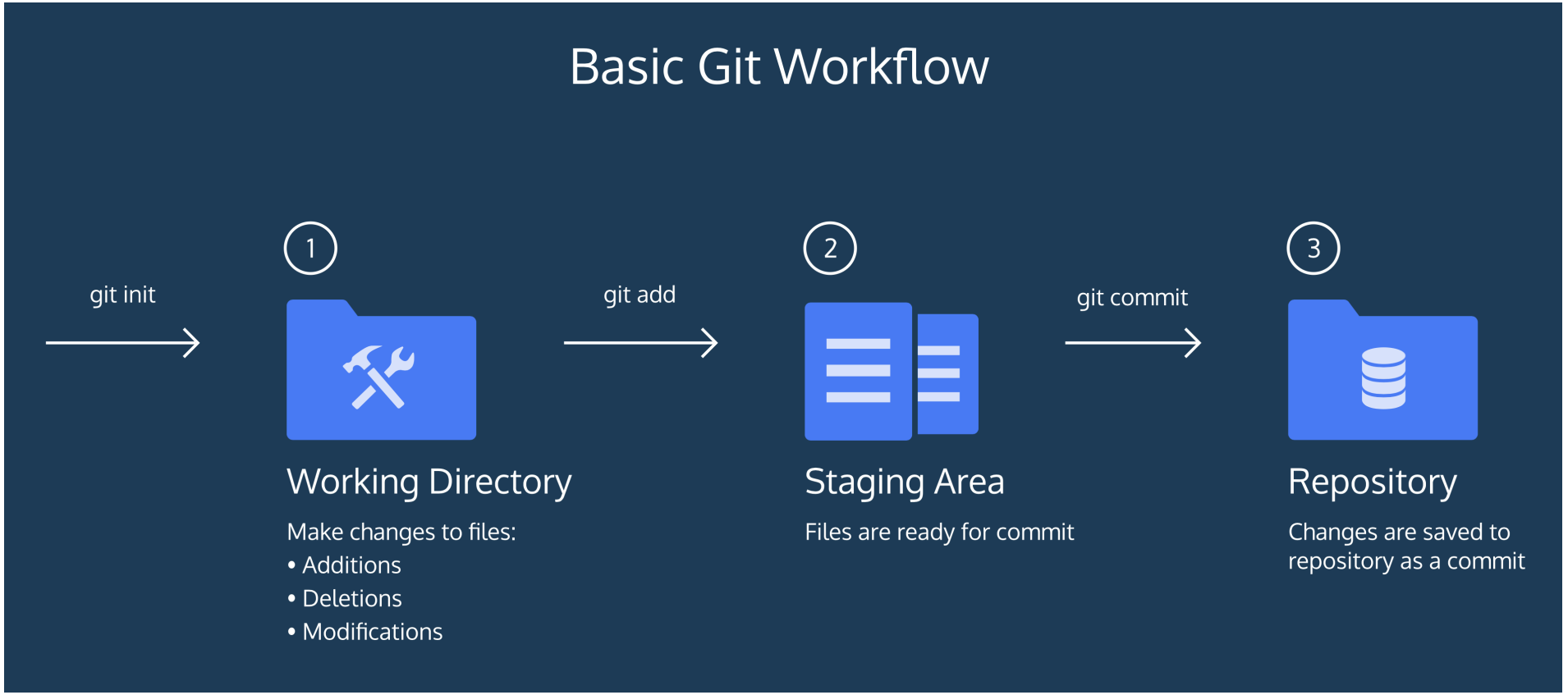
- We can’t use git if it’s not on our system
- Linux - install git using your distro’s package manager
- Windows - install from https://git-scm.com
  - Optionally use a Windows package manager like chocolatey or scoop
- Mac - install from https://git-scm.com
  - Optionally use homebrew

Setting up git

- We just need to enter two commands
- Open a terminal:
  - git config --global user.name [your name]
  - git config --global user.email [your email]
- Optional setup
  - Set your editor (for commit messages, vim by default)
    - git config --global core.editor “path/to/editor”

Getting Started

- We do not have to wait until we have a new project
  - It’s never too late to put our work under version control
- We do need to organize our work, however
  - At a minimum, each project should be contained in its own folder
- From a terminal, navigate to your project folder
- Type `git init`
- An empty repository is created



The git Process*

Our Workflow

- We edit our code, we save our code
  - Same as we've ever done
- To save it to our repository, we must first stage the files to be committed
  - Staging is a way of saying that we're done altering that file for now
- Stage by using the command `git add [FILENAMES]`
- We can always check the status of our repository by using the command `git status`

Committing Files

- After we stage all the files that we have edited/want to add, we are ready to commit them
- `git commit -m “MESSAGE”`
  - Message should be written in the present-tense, and kept short (< 150 characters)
  - `git commit` is technically all we need
    - Opens a text editor for us to enter a message
  - ALWAYS include a message; it is best practice

git Tracks EVERYTHING

- In a project folder, git keeps track of everything
- There will likely be files we do not want it to track and bother us about
- We can tell git to ignore these files
- In the working directory, create a file called .gitignore
  - It is a plaintext file where we list the files we don’t want git to track
  - We can name specific files and use wildcards
- We can get gitignore lists from GitHub
  - https://github.com/github/gitignore

Viewing Our Commits

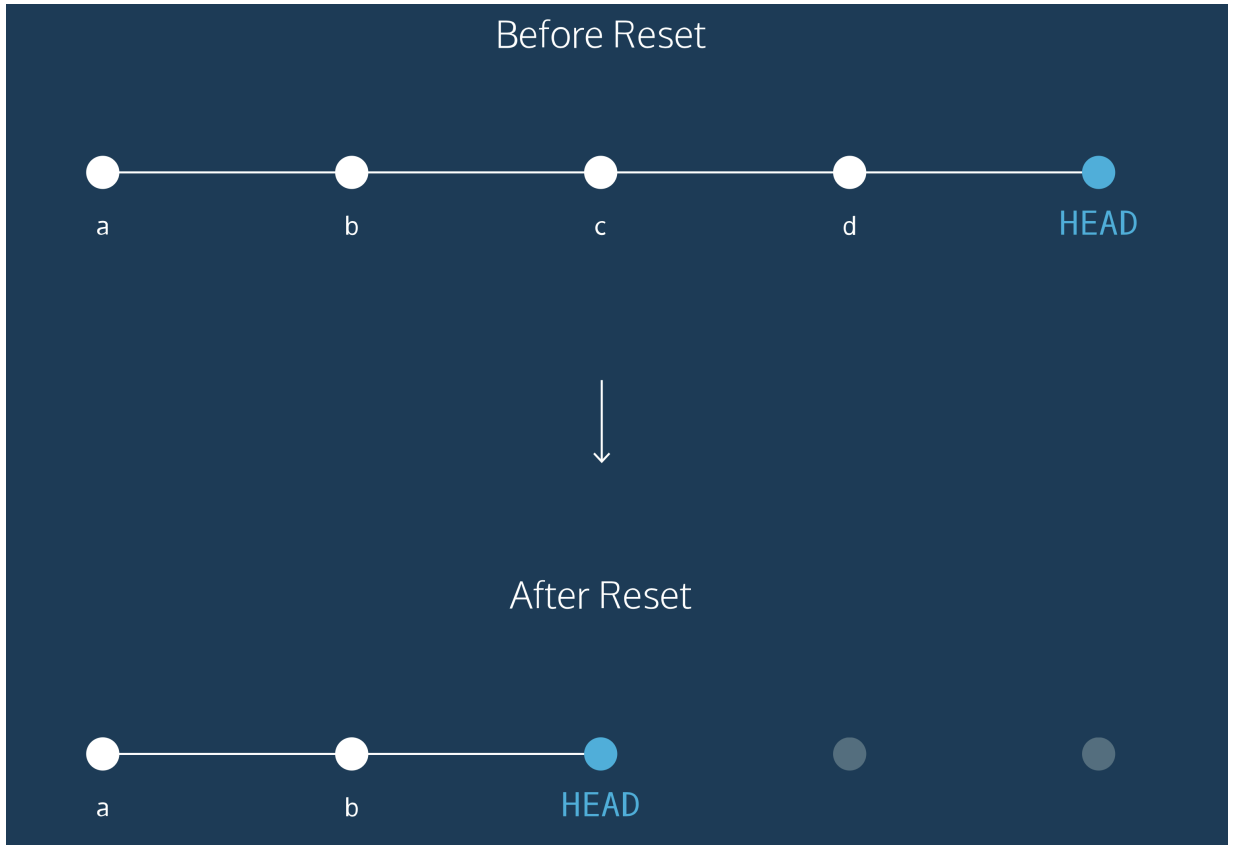
- There will come a time when we want to see our commit history
  - Finding out when a bug cropped up
  - Observing evolution of project
  - Finding who to blame when everything's broken
- This can be done using the command `git log`

What `git log` Shows You

- Commit number
  - Every commit is numbered with a hash value, generated using SHA
- What branch the commit was made to (more on that later)
- Commit author
- Commit timestamp
- Commit message

`git reset`

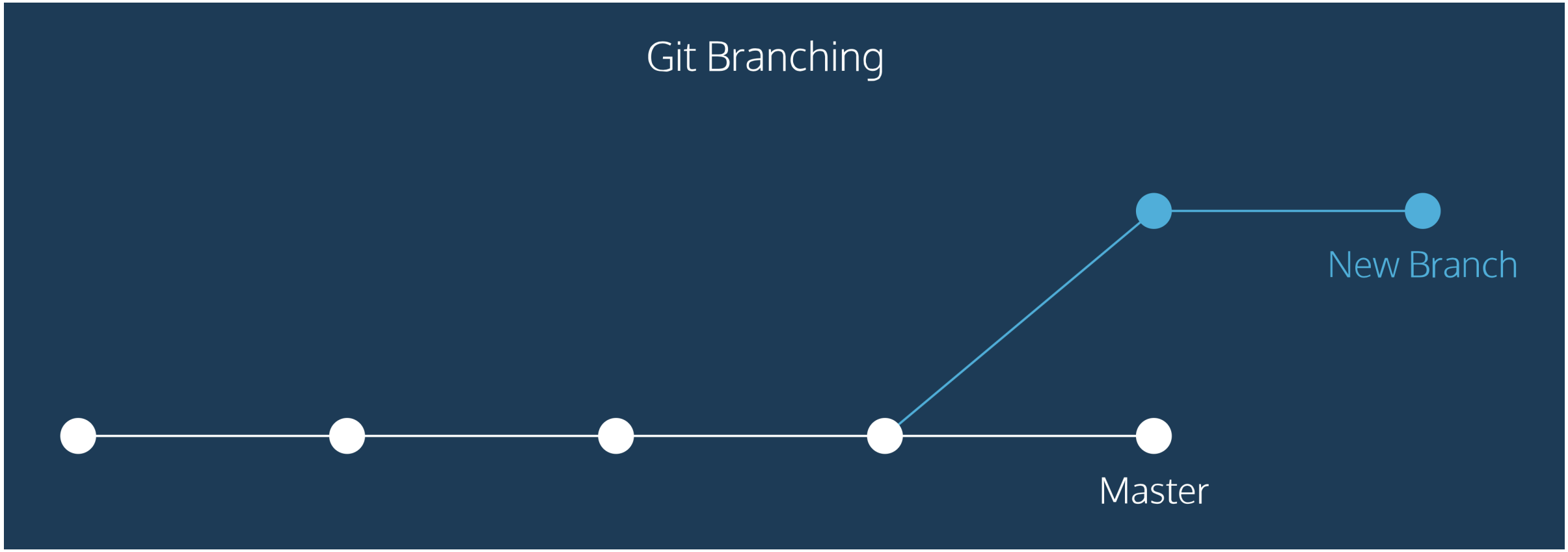
- If I have staged a file but wish to unstage it, I can use the following command
- `git reset FILENAME`
  - Does not change working directory, the file is merely unstaged
- Also possible to roll back entire repository if no filename is specified
- `git reset --hard COMMIT`
  - Changes working directory to match a specific commit



Visualization of git reset

Branching

- Oftentimes, we are afraid of adding new features or experimenting with new syntax because we could break what we have
- More importantly, no company wants their money maker in a broken state just to see if a new feature or refactor will pan out
- git allows us to perform these experiments without affecting our “master” code



A Visualization of git Branching

Creating a New Branch

- Simply use the command `git branch BRANCH_NAME`
- The branch name may not contain whitespace
- If we type `git branch` without specifying a branch name, git lists all the branches, with a * marking our current branch

Switching Branches

- `git checkout BRANCH_NAME`
- This will switch your branch; the branch must exist first
  - `git checkout -b BRANCH_NAME` will create a branch and switch to it
- Now, when staging and committing changes, they will be committed to the current branch, and not to the master
- This can be observed my making changes to a branch, committing those changes, and switching branches back to master

Bringing it Back

- The goal of a branch is to incorporate it back to master
- From the master branch: `git merge BRANCH_NAME`
- The branch specified in the merge command is the “giver” branch
- The branch you execute the command from is the “receiver” branch

Upon Successful Merging

- We usually want to delete the branch
  - Branches are a means to an end
  - Many branches have no use after being merged
- `git branch -d BRANCH_NAME`

git is a Powerful Tool

- We've only scratched the surface
- Most glaring omission right now is lack of discussion about online repositories for collaboration
- But that's coming
- In the meantime, here some resources for learning more about git

Resources

- https://www.codecademy.com
- https://www.atlassian.com/git
- https://www.git-tower.com/learn/git/ebook
- https://git-scm.com/doc
- `git help`
  - `git help TOPIC`

Review

- Version control
- Using git locally
  - Repository creation
  - Staging & Committing
  - Branching
  - Merge Conflicts