Version Control Systems (VCS) [Git/GitHub]

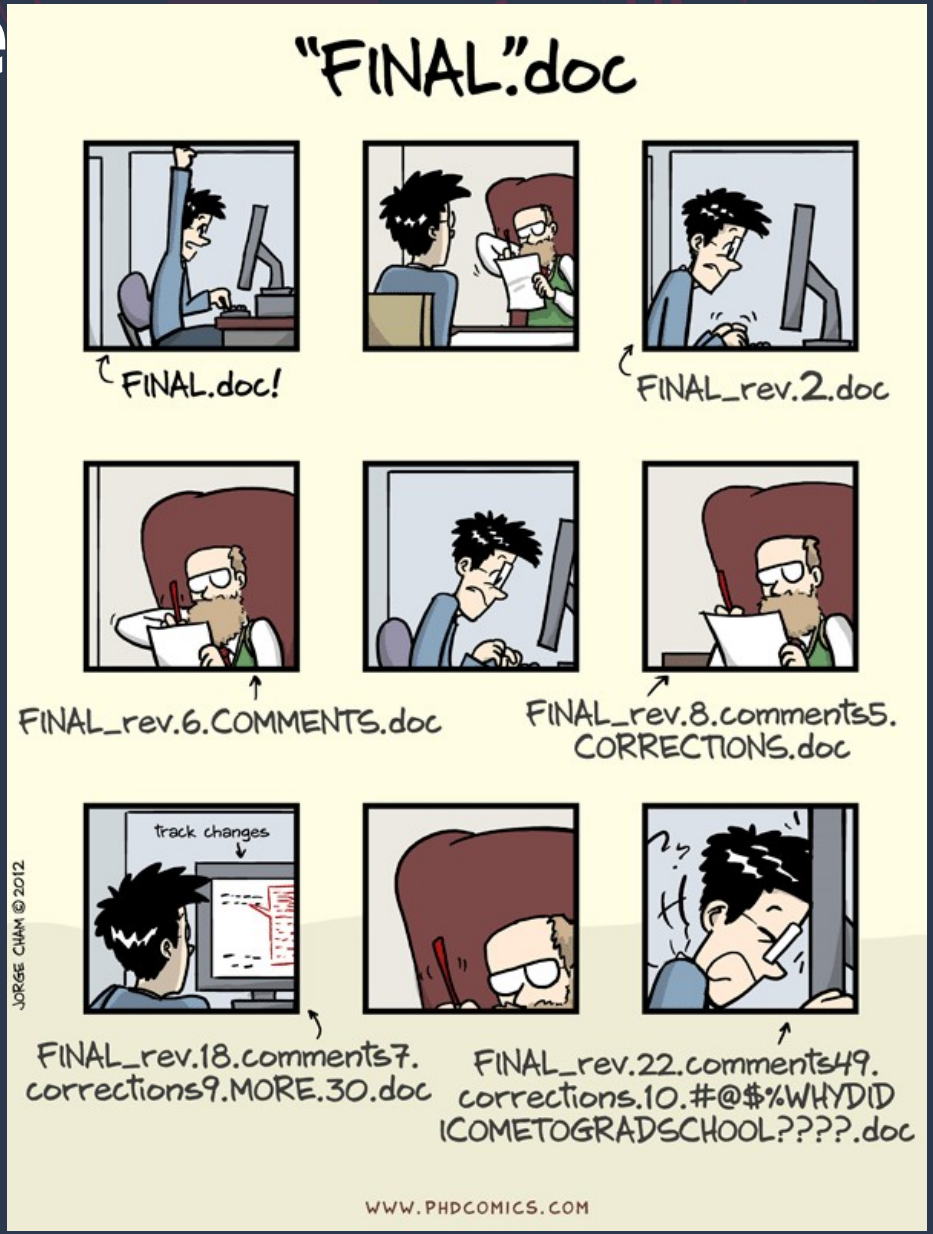
BEE2041

Key references & Material:
https://missing.csail.mit.edu/2020/version-control
https://git-scm.com/book/en/v2
https://ourcodingclub.github.io/tutorials/git
https://guides.github.com/introduction/git-handbook

Version Control Syste

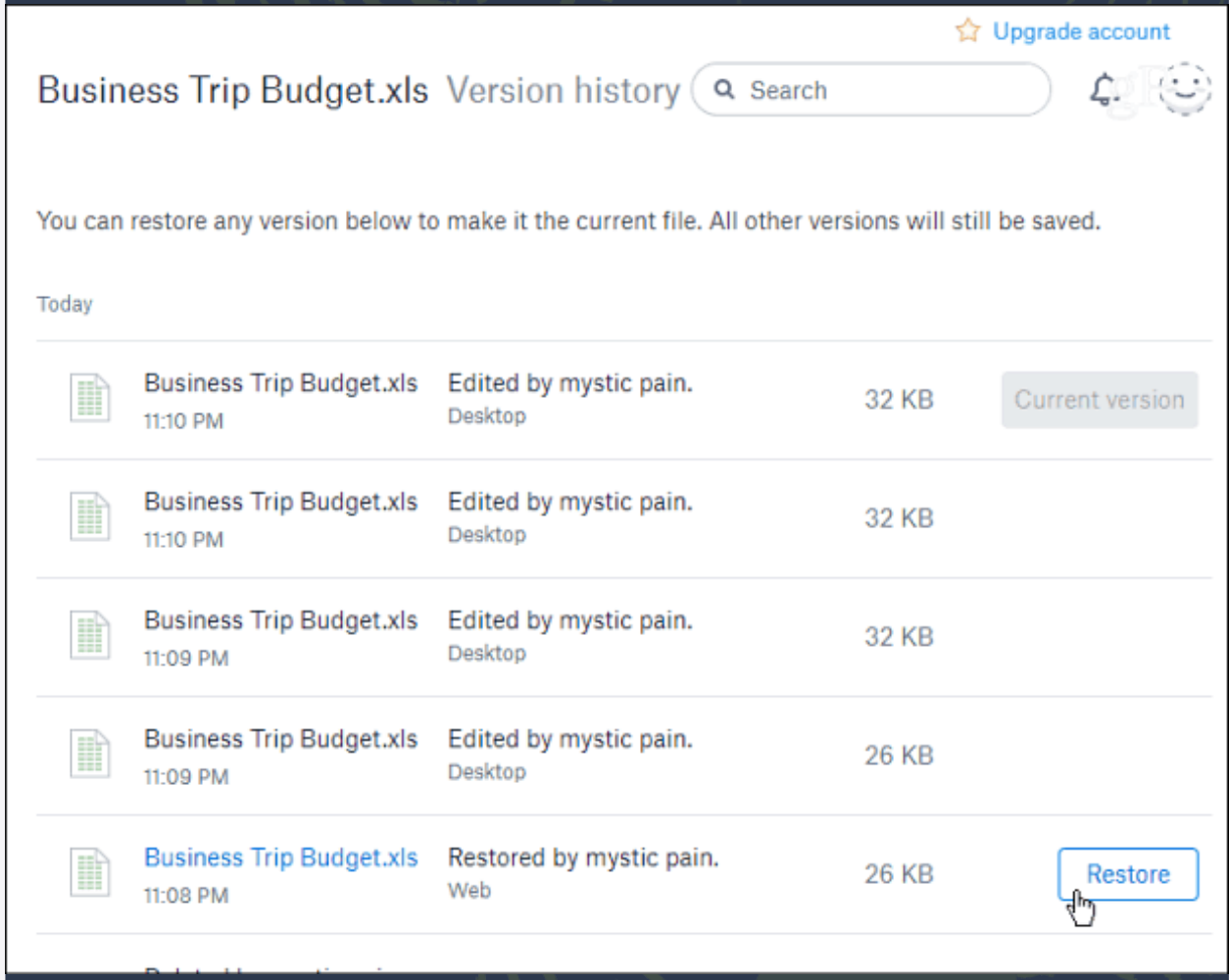
- You probably do some sort of version control

Name	Type
project draft.doc	Microsoft Word 9...
project draft1.doc	Microsoft Word 9...
project final.doc	Microsoft Word 9...
project final1.doc	Microsoft Word 9...
project final2.doc	Microsoft Word 9...
project FINAL final.doc	Microsoft Word 9...
project FINAL final1.doc	Microsoft Word 9...
project FINAL final2.doc	Microsoft Word 9...
project final THIS IS THE ONE TO SUBMIT.doc	Microsoft Word 9...
project final THIS IS THE ONE TO SUBMIT v2.doc	Microsoft Word 9...



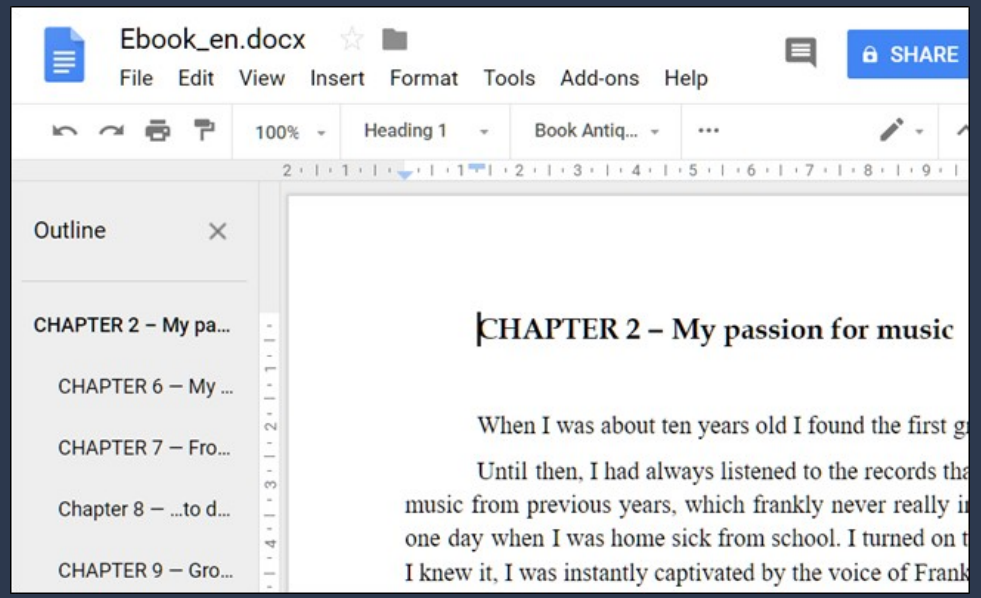
Version Control Systems

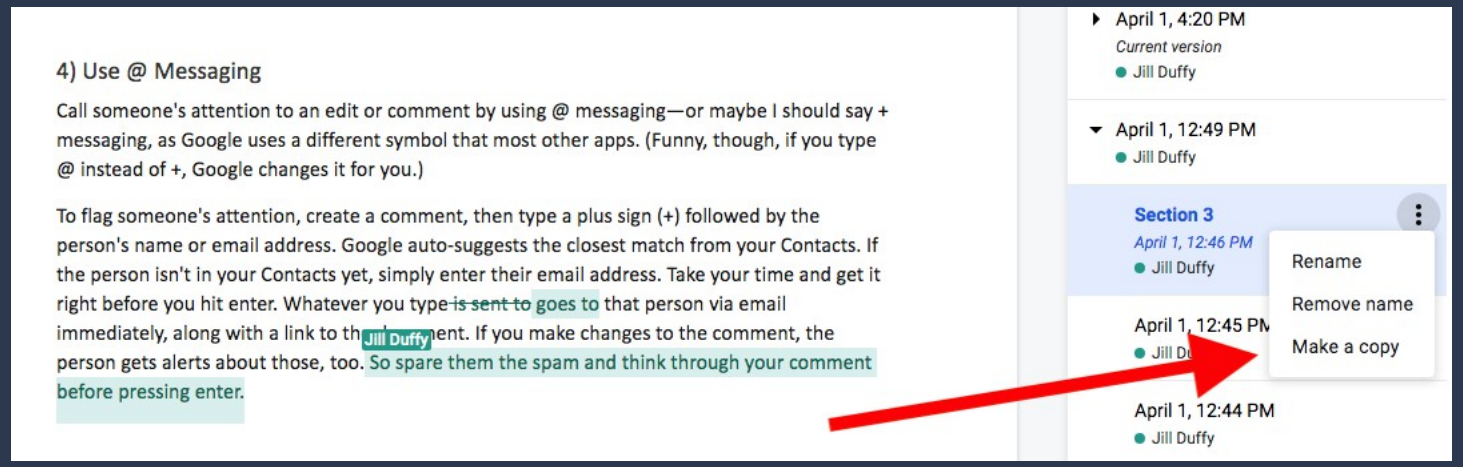
- If you use Dropbox, it has its implementation of version control



Version Control Systems

- Google Docs also has its own version control





Version Control Systems (VCS)

- A system that records changes to a file or set of files over time so that you can recall specific versions later.

Advantages of VCS

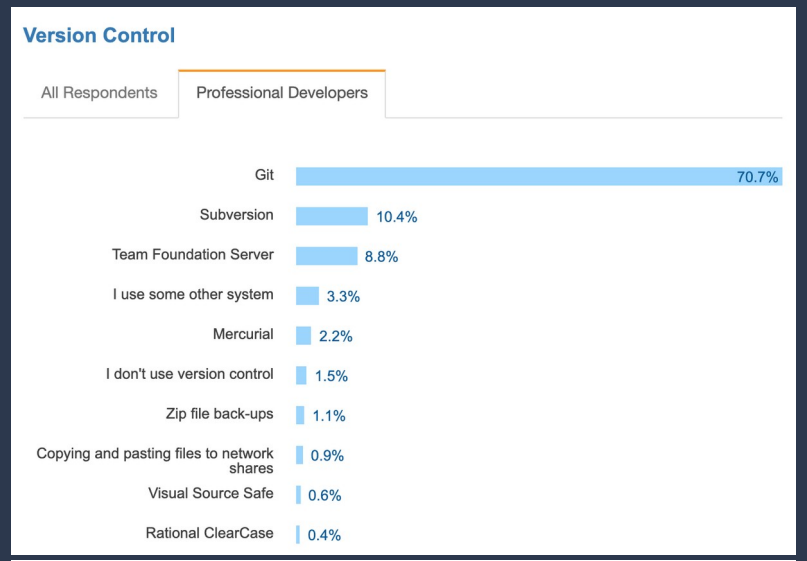
- Archiving: keep track of all different versions of a project
- Access to earlier versions: easy to navigate
- Debugging: identify where problems started to arise
- Fixing: restore last working version
- Collaboration:
  - Share work effectively
  - Keep track of who changed what (and when and why)
  - Simultaneous distributed development

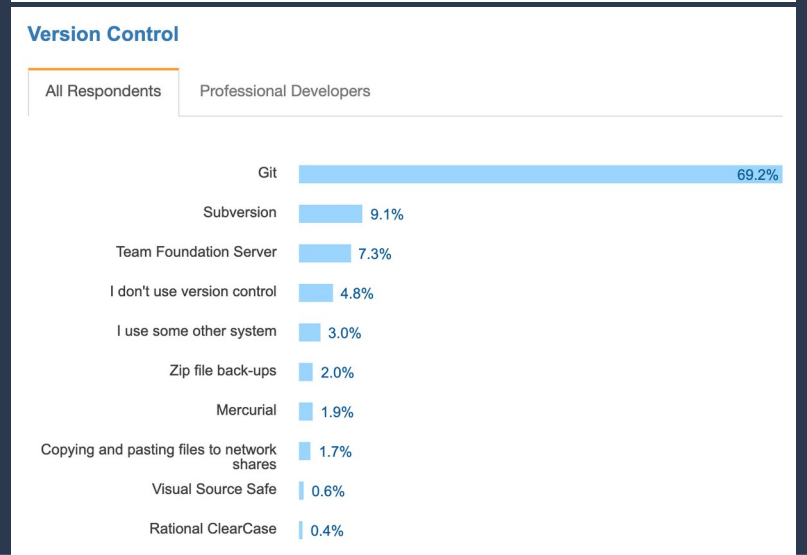
Functionality of VCS

- Which changes were made?
- Who made the changes?
- When were the changes made?
- Why were changes needed?
- When was this particular line of this particular file edited? By whom? Why was it edited?
- Over the last 1000 revisions, when/why did a particular unit test stop working?

Git – Distributed VCS, Snapshot based

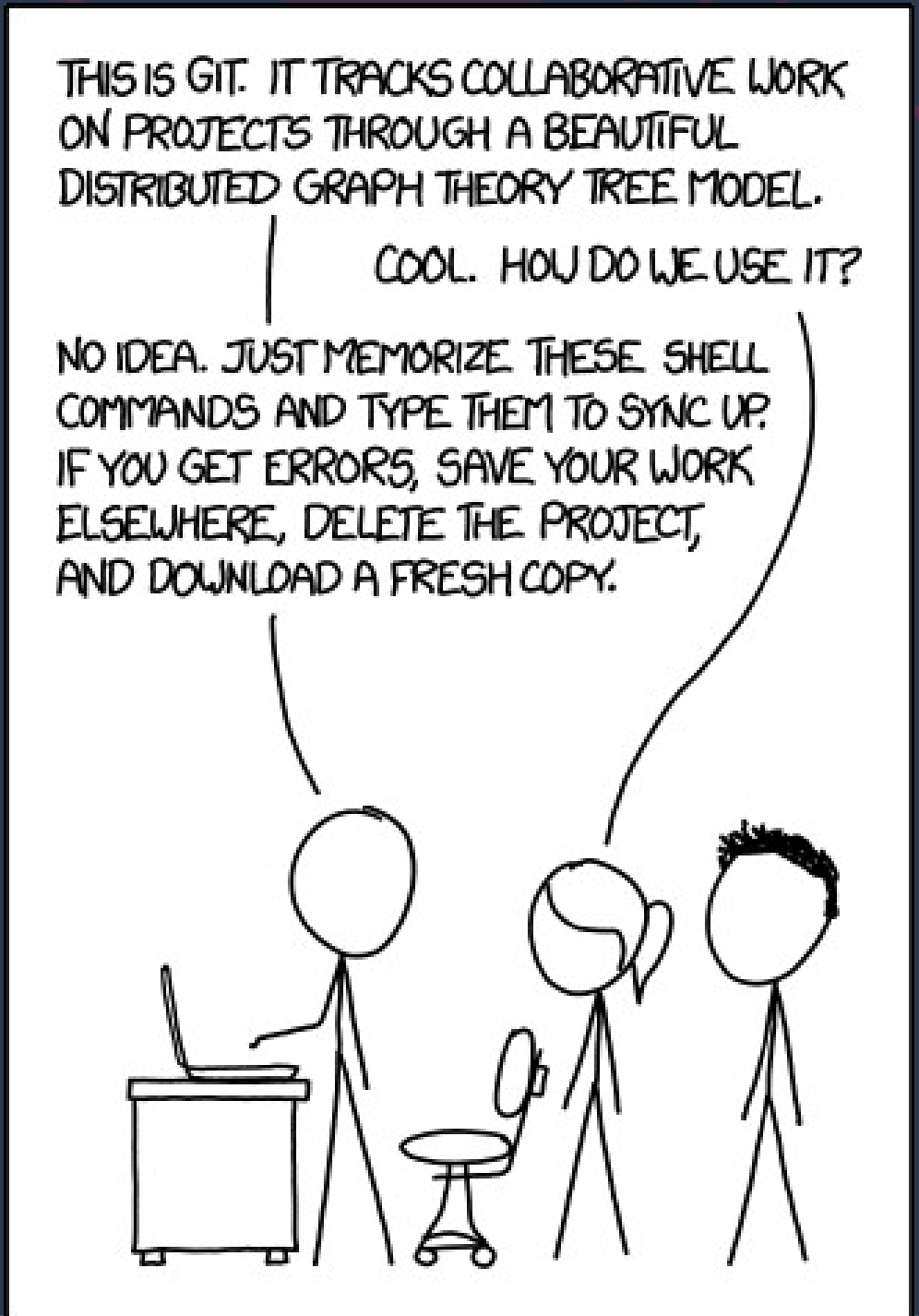
- Speed
- Simple design
- Strong support for non-linear development
- Fully distributed
- Able to handle large projects





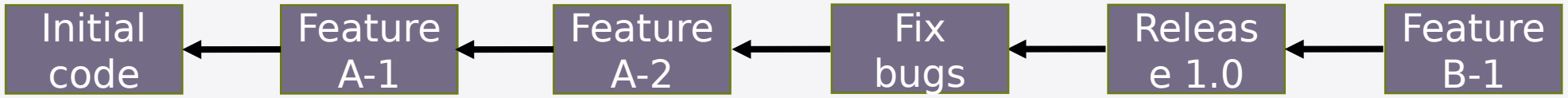
Git

- On the other hand...

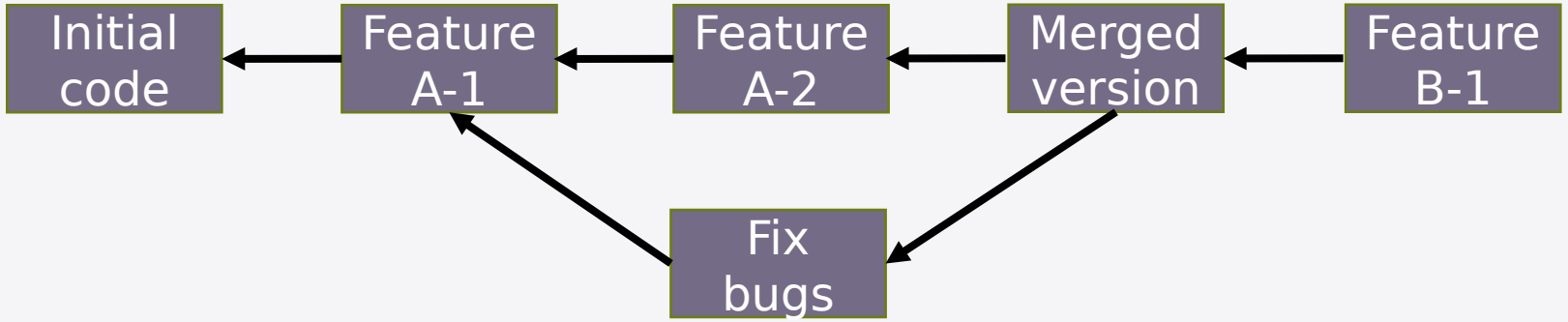


Git Model: nonlinear (vs. linear)

Linear

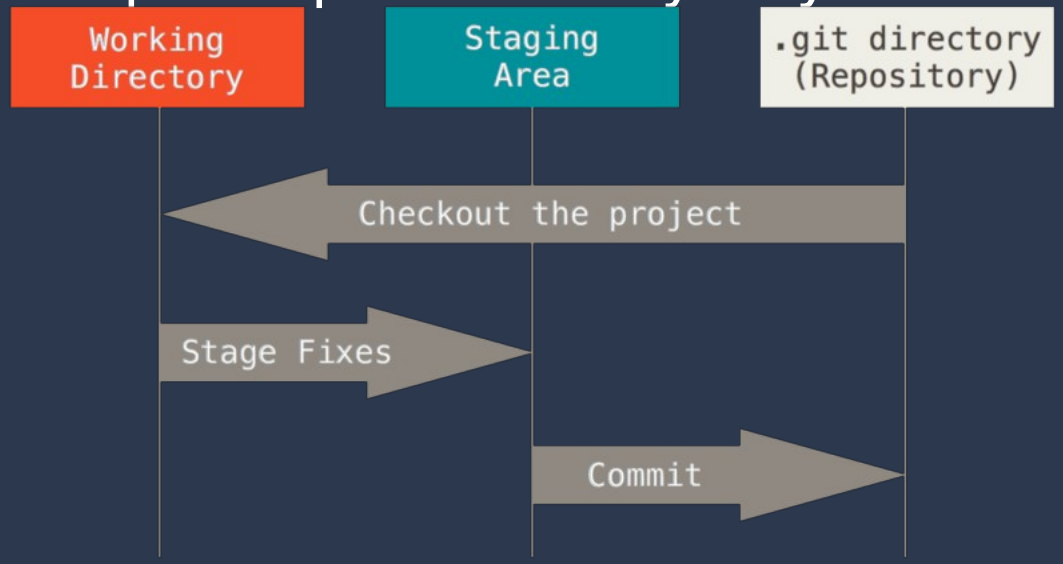


Nonlinear



Git Workflow

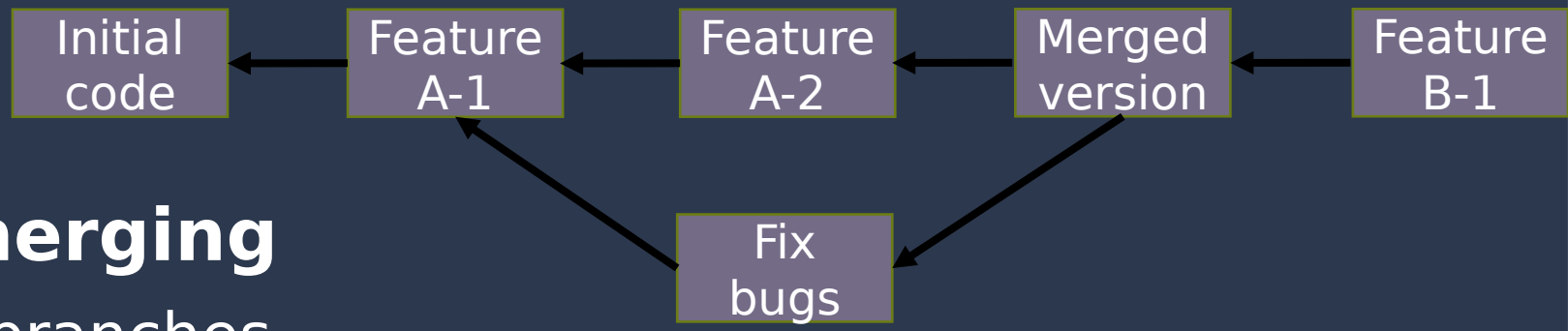
1. You modify files in your working tree.
2. You selectively stage just those changes you want to be part of your next commit, which adds **only** those changes to the staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.



Git Commands

- **Basics**
  - git help <command>: get help for a git command
  - git init: creates a new git repo, with data stored in the .git directory
  - git status: tells you what’s going on
  - git add <filename>: adds files to staging area
  - git commit: creates a new commit
  - git log: shows a flattened log of history
  - git log --all --graph --decorate: visualizes history as a DAG

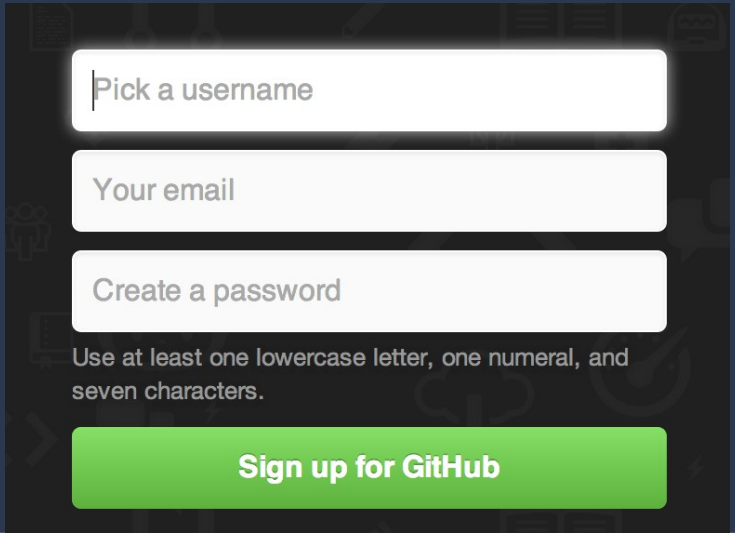
Branch/Merge



- **Branching and merging**
  - git branch: shows branches
  - git branch <name>: creates a branch
  - git checkout -b <name>: creates a branch and switches to it
    - same as git branch <name>; git checkout <name>
  - git merge <revision>: merges into current branch
  - git mergetool: use a fancy tool to help resolve merge conflicts

GitHub

- A hosting repository for Git
- Not a direct part of the Git open source project
- The single largest host for Git repositories
- A large percentage of all Git repositories are hosted on GitHub



GitHub Commands

- **Case1: Start a new repository and publish it to GitHub**

```
# create a new directory, and initialize it with git-specific functions
git init my-repo

# change into the `my-repo` directory
cd my-repo

# create the first file in the project
touch README.md

# git isn't aware of the file, stage it
git add README.md

# take a snapshot of the staging area
git commit -m "add README to initial commit"

# provide the path for the repository you created on github
git remote add origin https://github.com/YOUR-USERNAME/YOUR-REPOSITORY.git

# push changes to github
git push --set-upstream origin main
```

GitHub Commands

- **Case2: contribute to an existing branch on GitHub**

```
# assumption: a project called `repo` already exists on the machine, and a new bra

# change into the `repo` directory
cd repo

# update all remote tracking branches, and the currently checked out branch
git pull

# change into the existing branch called `feature-a`
git checkout feature-a

# make changes, for example, edit `file1.md` using the text editor

# stage the changed file
git add file1.md

# take a snapshot of the staging area
git commit -m "edit file1"

# push changes to github
git push
```

GitHub Commands

- **Case3: contribute to an existing project on GitHub, to which you don’t have push access**

1. Fork the project.
2. Create a topic branch from master.
3. Make some commits to improve the project.
4. Push this branch to your GitHub project.
5. Open a Pull Request on GitHub.
6. Discuss, and optionally continue committing.
7. The project owner merges or closes the Pull Request.
8. Sync the updated master back to your fork.

Let’s try this out …