Introduction to Git

Anna Yannakopoulos
February 25, 2018

What is version control?

- *“The task of keeping a software system consisting of **many versions and configurations** well **organized**.”* - Google
- Many people have invented their own version control:
  - paper.docx
  - paper-FINAL.docx
  - paper-FINAL-corrected.docx
  - paper-FINAL-corrected-UPDATE.docx
- Problems with this approach:
  - Duplicates take up a lot of space
  - Usually inconsistent, uninformative names
  - Difficult to track specific changes
  - Prone to user error, especially if collaborating

What is Git(Hub)?

- **Git:** version control software developed to maintain the Linux kernel in 2005
  - *"I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git."* - Linus Torvalds
- **GitHub:** cloud storage for projects that use Git
- Why use Git and GitHub?
  - **Easy collaboration:** think Google Docs but for code
  - **Tracks who made changes:** so you know who to blame when your code breaks
  - **Allows you to revert changes:** so you can go back to before it broke
  - **Does this automatically:** less time spent managing version control = more time spent coding

Vocabulary

- Your project folder is a **Git repository** (often called a **repo**)
- Copy a repo from GitHub to your machine by **cloning** it
- Add files to the **staging area** and **commit** to save a snapshot of staged files
  - **Push** commits from **local** (your computer) to **remote** (GitHub)
  - **Pull** commits from remote to local
  - **Checkout** to replace your local files with a different version
- Your repo can have both a main **trunk** or **master branch** (stable version) and many other **branches** (development versions, often for specific features)
  - Branches can be **merged** into each other or the master branch

Commits are snapshots of your repository

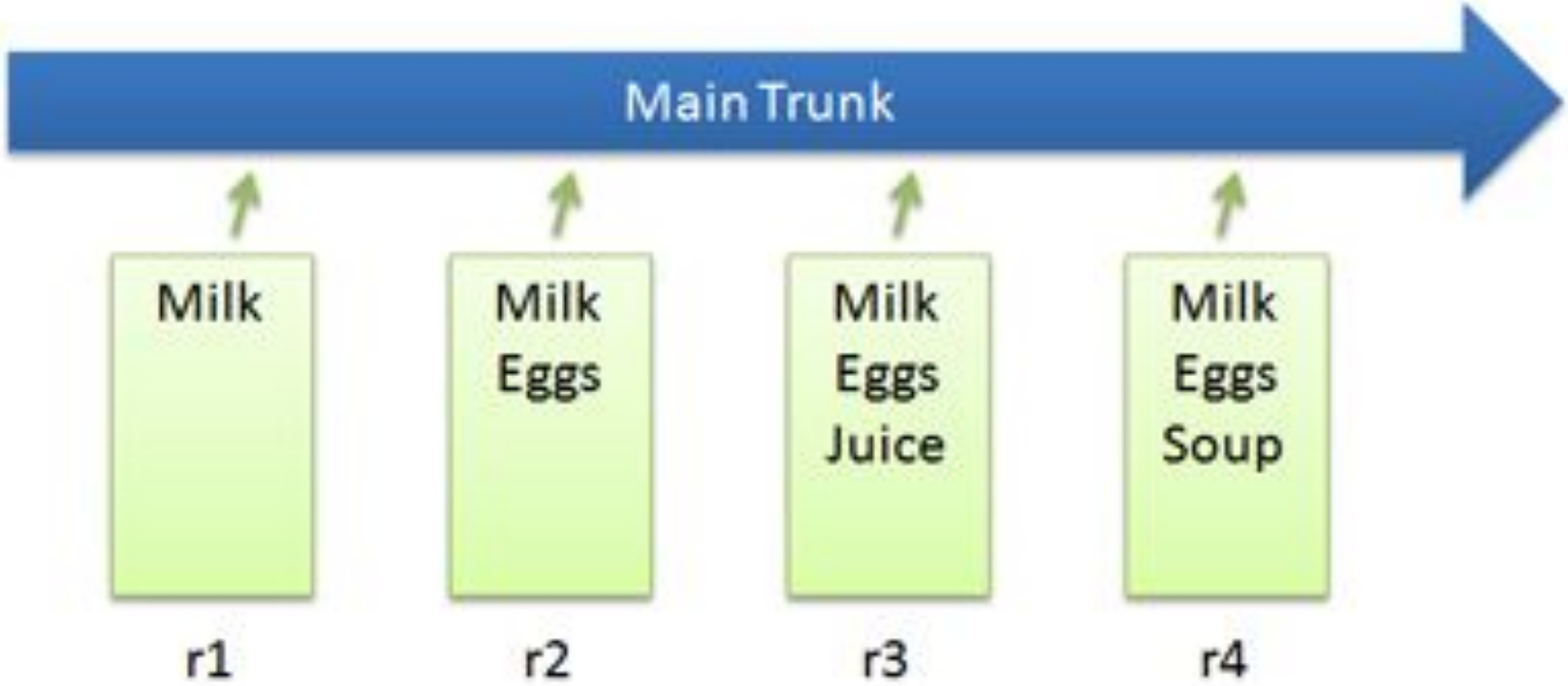

Image source: betterexplained.com/articles/a-visual-guide-to-version-control/

Commits only record changes

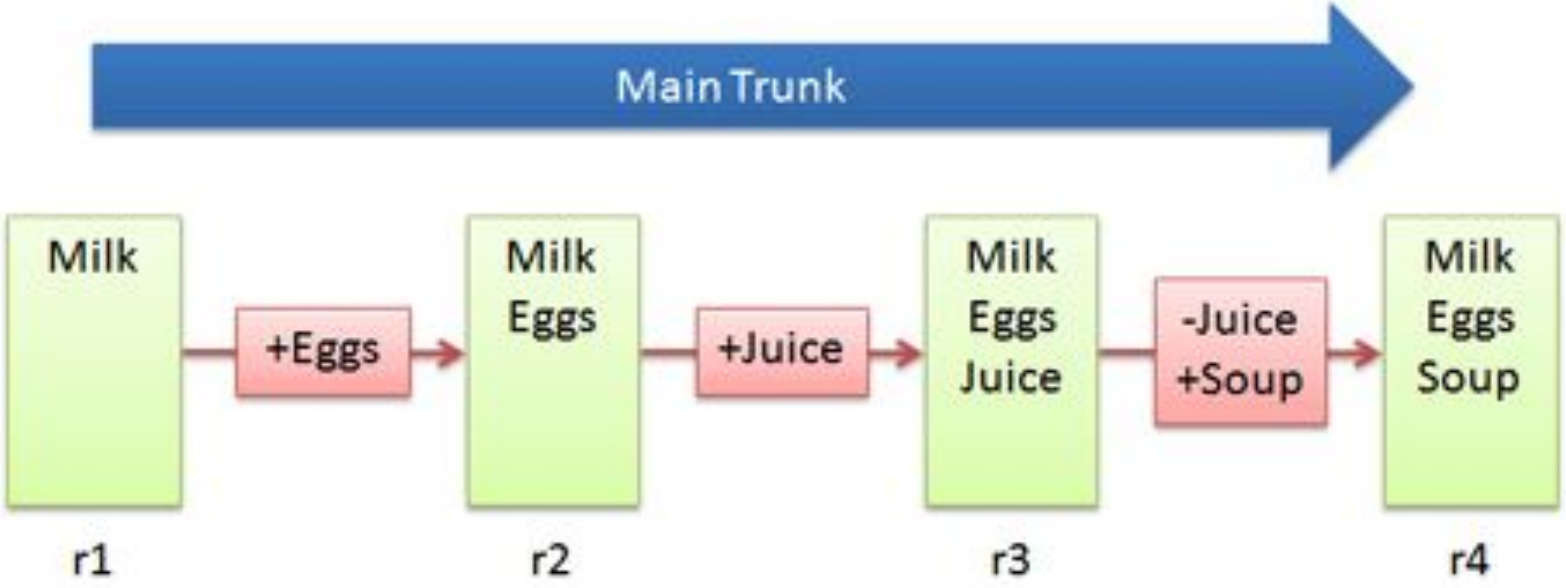

Image source: betterexplained.com/articles/a-visual-guide-to-version-control/

Notes on commits

- Each commit has a unique hash identifier
  - Use this to revert commits, etc.
  - The most recent commit is the **`HEAD`**
- Each commit should have a **commit message** explaining what the commit changes
- You can inspect a particular commit to see specific files and lines changed as well as who was responsible - useful for debugging!

Advanced Material

Feel free to skip ahead to the tutorial!

Branching

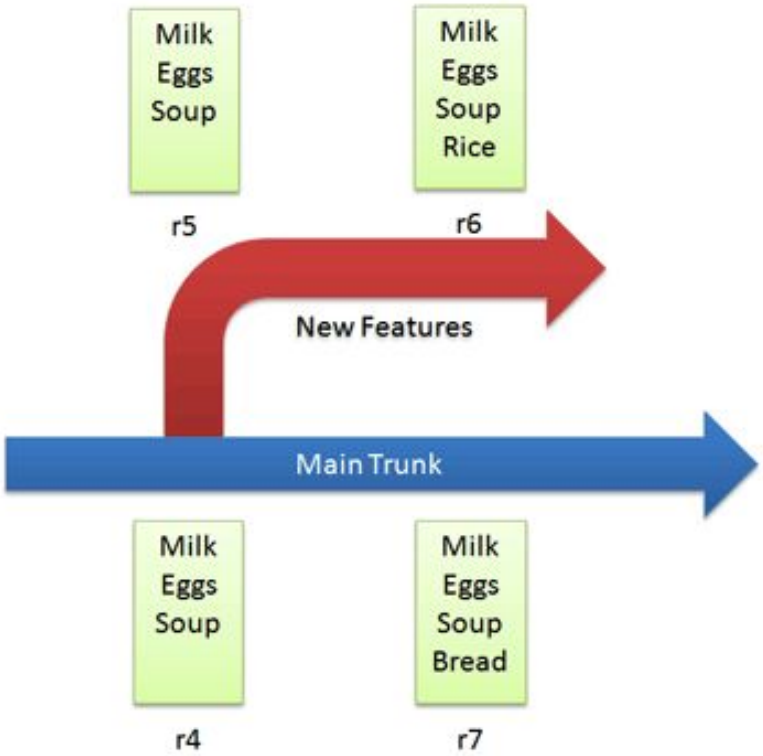

Image source: betterexplained.com/articles/a-visual-guide-to-version-control/

Merging

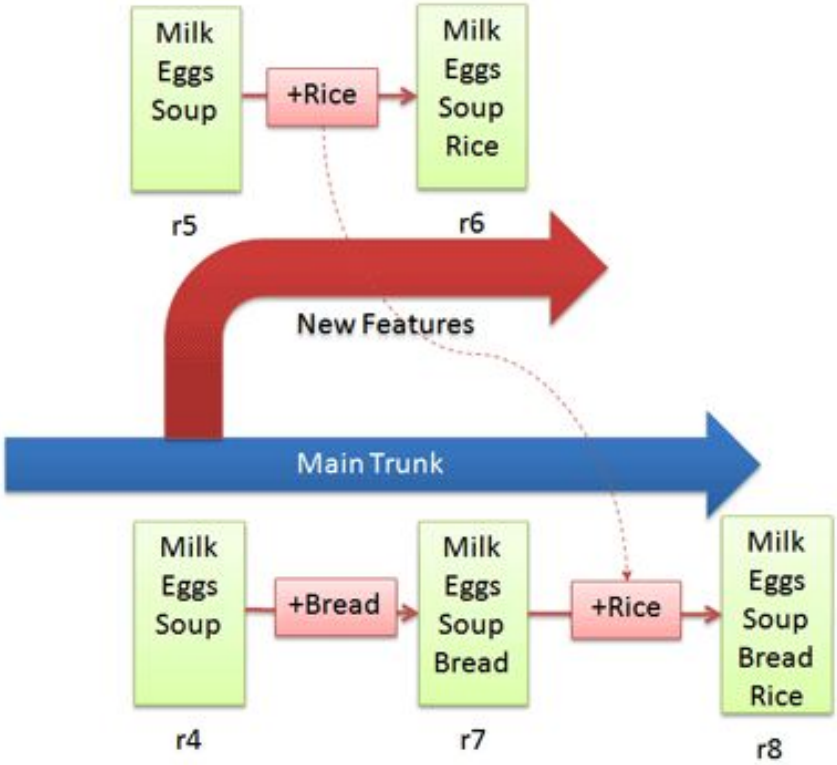

Image source: betterexplained.com/articles/a-visual-guide-to-version-control/

Notes on branching and merging

- It’s good practice to do most development work on branches
  - **One branch per major feature**
  - Keeps master branch stable until you’re ready to merge completed feature branch
  - Stops contributors working on different features from breaking each other’s work
- When you’re ready to merge, you submit a **pull request** asking the repo maintainers to pull your changes to the master branch
  - If you’re the maintainer, this is mostly a formality
  - However, on shared projects, pull requests invite code review
- Git may detect a **merge conflict** and ask you to manually resolve it

Merge conflicts

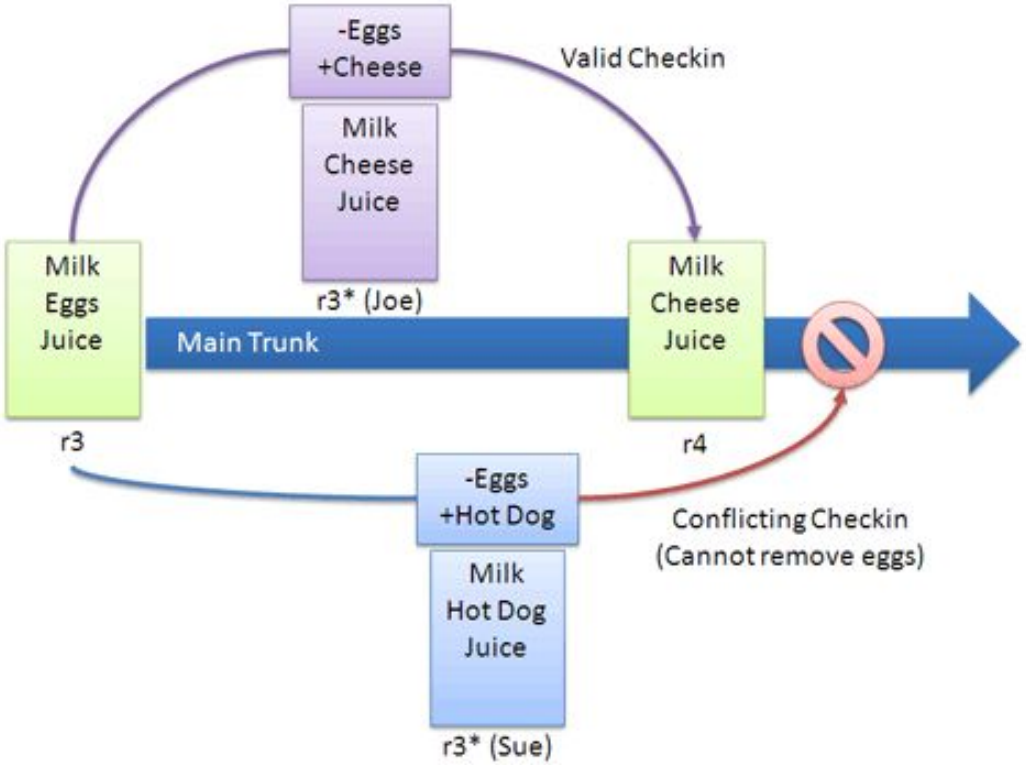

Image source: betterexplained.com/articles/a-visual-guide-to-version-control/

Git Tutorial

Initial Setup

1. Install Git:
   - Windows: git-scm.com/downloads
   - Mac: probably already installed, but if not, git-scm.com/downloads
   - Linux: use your package manager
2. Open a command prompt and make sure everything works:

   ```
   git --version
   ```

3. Tell Git who you are:

   ```
   git config --global user.name "Your Name"
   git config --global user.email "your.email@domain.com"
   ```

4. If you're not familiar with vi:

   ```
   git config --global core.editor "nano"
   ```

Create a GitHub account

- github.com
- Free for public repositories
- Free for private repositories with ≤ 3 collaborators
- Use your MSU email to get the Student Developer Pack
  - Cloud computing credit (AWS, DigitalOcean)
  - Other software and services

Push and pull without a password

- We can generate an **SSH key** to enable pushing and pulling to and from GitHub without typing your account password every time
- Run `ssh-keygen` to generate a public/private key pair
  - Accept default filename
  - Provide a password or hit enter to leave it blank
- Open your public key at `.ssh/id_rsa.pub` and copy the contents
- Paste into new key at github.com/settings/keys

Create a GitHub repo

- Go to github.com/new
  - Enter a repository name and description
  - Choose whether to make the repo public or private
  - Initialize the repo with a README
- Click “Clone or download” and copy the repo URL
- `git clone <URL>`

Staging and committing a file

- **`nano test.txt`** to create a file
  - Opens a terminal-based text editor
  - Type whatever you like and Ctrl-O to save
  - Ctrl-X to exit
- **`git status`** to see which files have changed
- **`git add test.txt`** to stage your changes
- **`git status`** to show what's in the staging area
- **`git commit -m "Some commit message"`** to commit
- **`git push`** to send changes to GitHub
- Check your changes on your repo page!

Advanced Material

Feel free to skip this!

Undoing changes and reverting commits

- **`git log`** to see commit IDs
  - You only need the first 5 characters of a commit ID
  - **`HEAD`** is a shortcut for the most recent commit ID
  - Leaving the commit ID empty generally assumes **`HEAD`**
- **`git show <commit>`** to inspect a commit
- Let's edit **`test.txt`** again. What if we break something?
- **`git diff test.txt`** to see specific changes
- **`git reset test.txt`** to unstage a staged file
- **`git checkout <commit> test.txt`** to replace file with some commit
- **`git revert <commit>`** to make a new commit that undoes an old one

Making and merging a branch

- **`git checkout -b <branch name>`** to create a new branch
- Commit some change to **`test.txt`**
- **`git push --set-upstream origin <branch name>`** to push to a new branch on GitHub
- Go to the repo on GitHub and you should be able to create a pull request for your branch, but don’t do it yet!

Dealing with merge conflicts

- **`git checkout master`** to return to the master branch
- Commit and push some incompatible change to **`test.txt`**
- Refresh the pull request page on GitHub - it should now complain of a merge conflict between your branch and the master
- To resolve, make a new commit that merges the conflicting commits

Ignoring files

- Sometimes your project will include files that you don't want to version control e.g. log files, compiled files, etc.
- You can specify rules for filenames that Git should ignore in **`.gitignore`**
- To ignore a specific file, add its path to **`.gitignore`**
- To ignore the **`data`** directory, add **`data/`** to **`.gitignore`**
- **`.gitignore`** uses globbing patterns to match filenames:
  - To ignore all tab-separated data files, add **`*.tsv`**
  - To ignore any directory named **`logs`** no matter where it is, add **`**/logs/`**
  - More here: linux.die.net/man/7/glob

Other commands

- **`git init`** creates a new repo from existing files
  - To upload this repo to GitHub, you'll need to make a new repo on GitHub and follow instructions for importing an existing local repo
- **`git pull`** fetches and merges changes from GitHub
  - If you're working with collaborators, you'll need to do this to sync your work with theirs
- **`git mv <old file> <new file>`** to rename files
  - Git can usually figure out if you moved a file without changing its contents, but you can use this command to make sure
- **`git rm <file>`** to remove files
  - Try to remember to do this instead of deleting files via file manager or **`rm`**

Additional Resources

Resources to Learn Git: try.github.io

Learn Enough Git to be Dangerous:
www.learnenough.com/git-tutorial/getting_started

Git Tutorials and Training by Atlassian (BitBucket instead of GitHub):
www.atlassian.com/git/tutorials