Building software: Version control with Git

Data Sciences Institute

University of Toronto

Simeon Wong

Asking questions

- Zoom chat during class
  - Feel free to post and answer questions at any time
  - I will pause for questions occasionally, and review questions from the chat
- Pre- / Post-class office hours with Tong
- Email
  - simeonm.wong@mail.utoronto.ca
  - tong.su@mail.utoronto.ca

Course objective

How to write robust software in a team that we, our colleagues, and the public can trust and use with confidence.

What is version control?

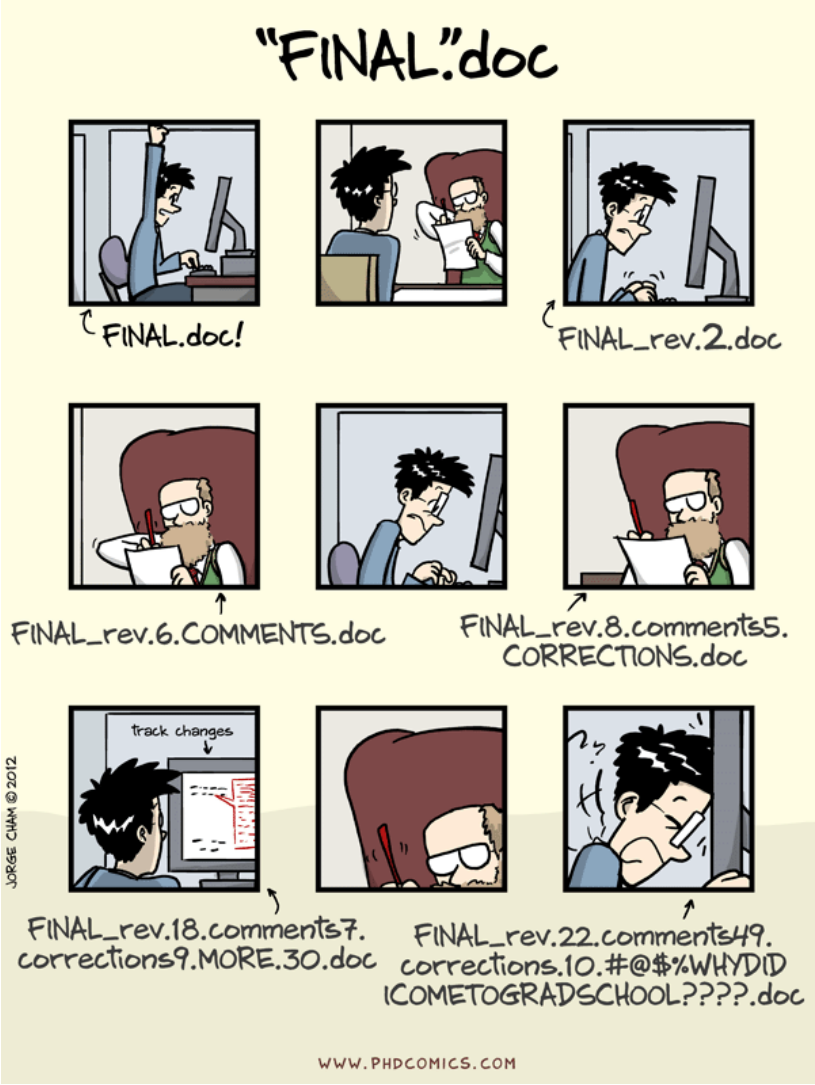
- A system that records changes to a file or a set of files over time
- Enables us to recall a specific version later
- **e.g.** Copying files to another directory to save past versions is a form of version control.
  - While it is simple, it lacks flexibility and ability to handle complexity

What is version control?

Version Control Systems (VCS) can do a number of things and can be applied on nearly any type of file on our computers:

- revert files to a previous state
- revert entire project to a previous state
- compare changes over time
- see who modified something last
- who introduced an issue and when
- recover lost files

Why version control for software?



Why version control for software?

- Robust software is documented as it is written
  - Log changes and reasoning for why changes are made
- Working in teams requires code-specific version control
  - Changing one part of a code project can affect behaviour in seemingly unrelated features
  - In-progress state of one component can render the entire program temporarily unusable (e.g. syntax error)
- Backup of your work

Git: Installation

System check!

1. Open your terminal
2. Type

```
git --version
```

You should see something like this:

```
simeo@chronos2 MINGW64 ~
$ git --version
git version 2.39.2.windows.1
```

Installing Git

Please see the environment setup instructions in the Onboarding repository:

https://github.com/UofT-DSI/Onboarding/tree/main/environment_setup

$> Interactive live coding

Setup identity information on the Git command line:

```
1. git config --list
2. git config --global user.name
3. git config --global user.email
4. git config --global core.editor “code --wait”
```

Launching from the command line

You can also run VS Code from the terminal by typing 'code' after adding it to the path:

- Launch VS Code.
- Open the **Command Palette** (`Cmd+Shift+P`) and type 'shell command' to find the **Shell Command: Install 'code' command in PATH** command.

- Restart the terminal for the new `$PATH` value to take effect. You'll be able to type 'code .' in any folder to start editing files in that folder.

Getting help

- `git help <verb>`
- `git <verb> --help`

Git reference manual: https://git-scm.com/docs

What questions do we have?

Meet our analyst, Alex

- Alex is a data engineer
- Alex works on a team at a mid-sized company
- Alex is starting a new project:
  - develop a data processing pipeline that aggregates sales data from multiple sources into a centralized data warehouse
  - develop a new module for the sales business intelligence dashboard with this analysis
- Follow along as Alex uses Git to simplify her work

$> Interactive live coding

Alex sets up a code repository before writing any code.

1. `mkdir myproject ;  cd myproject`
2. `git init`
   - Initialize a new repository
3. Create README.md
4. `git status`
   - Query the current state

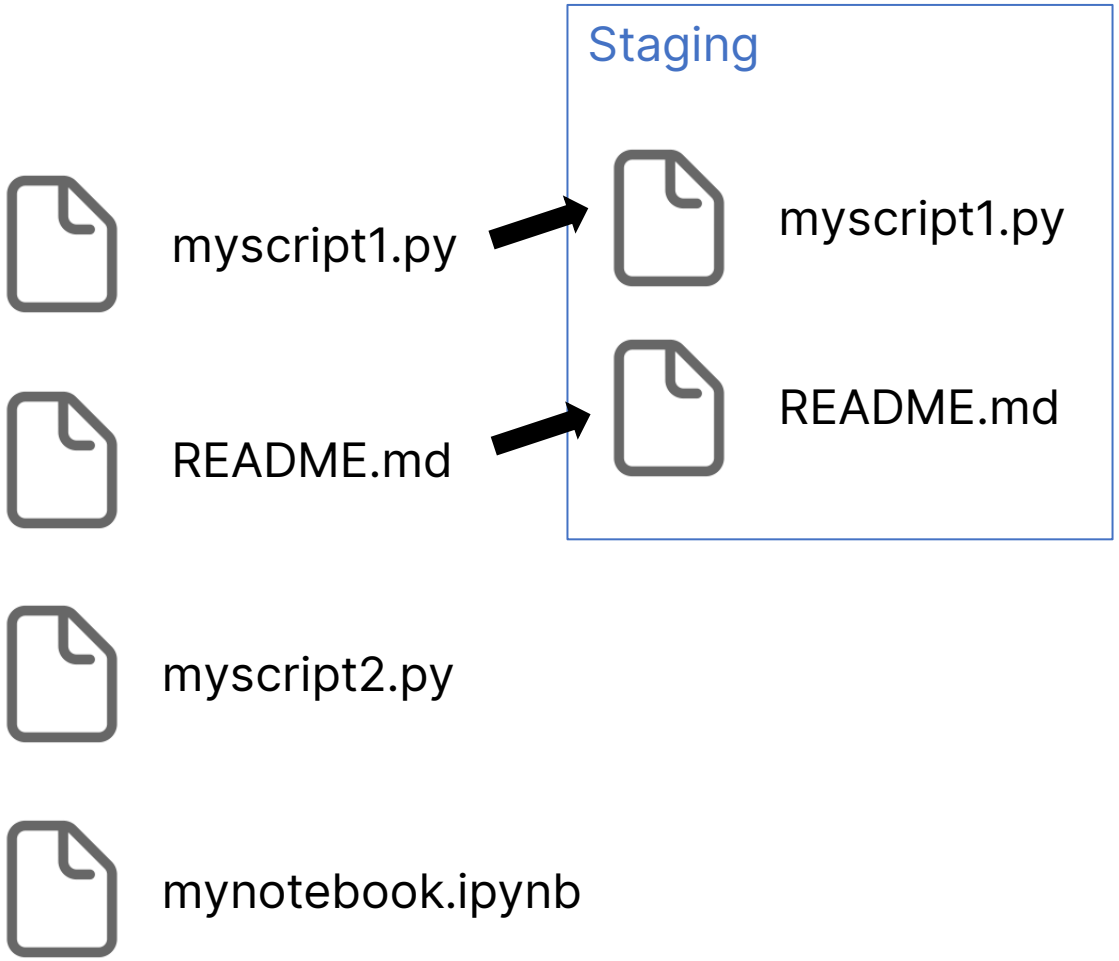
Git: Basic commands

Add to cart

- We need to tell Git which files it should **track**
- Indicate that the current state of a file should be tracked by **adding** it to the git **staging** area

```
git add myscript1.py README.md
```

- <u>Note:</u> If a file is modified after staging, this doesn't change the copy in the staging area



Add to cart

- Remove from staging: `git reset <file>`
- Update staging area with new changes: `git add <file>`
- Add all files in the repository folder: `git add -A`

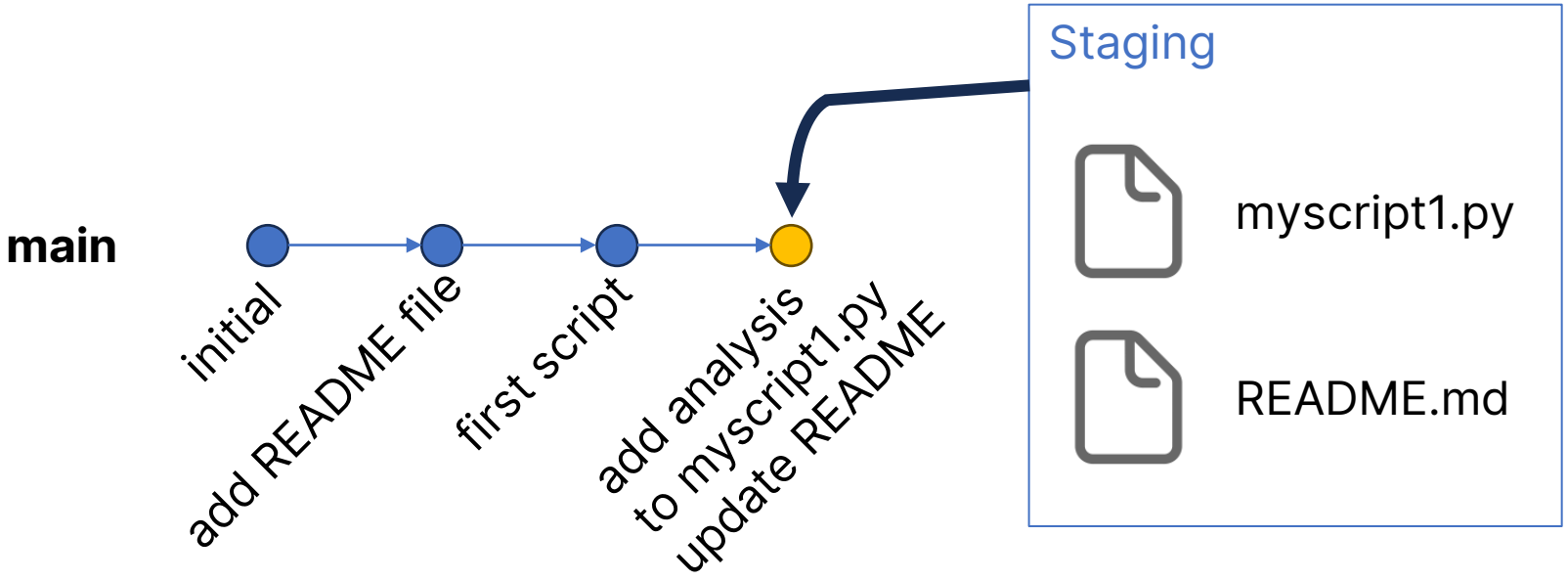
$> Interactive live coding

Alex adds her README file into staging.

1. `git add README.md`
2. `git status`
   - Query the current state
3. `git reset README.md`
4. `git status`

Commit staged changes

- Applies to all changes in the **staging** area, but not changes made since adding to staging
- Appends to the git **tree** (repository history) with a **commit message**
  - The tree can have multiple **branches** (more on this tomorrow)



$> Interactive live coding

Alex makes a commit with her README file.

1. `git add README.md`
2. `git status`
   - Query the current state
3. `git commit`

Also try:

```
git commit -m “commit message here”
git commit -a
```

Best practices for git commits

- **Commit messages** are extremely important:
  - for our own records as a reminder for we did
  - when collaborating with others
- Commit often
  - mistakes are easier to locate and fix
- Committed code can always be fixed with another commit

Exercise: Commit some files

- Try staging and committing some more files:
  - your Python files that you wrote earlier in this course
  - random files you create now

- Try different parameters:
  - `git add -A` add all files to staging
  - `git reset` unstage files
  - `git commit -am “msg”` commit all modified files with message

Tracking changes with Alex

- Follow along as Alex uses Git to simplify her work
  - Created a new repository: `git init`
  - Added a new file to staging: `git add README.md`
  - Committed that file to the version history: `git commit`
  - Checked status of repository: `git status`

What questions do we have?

Tracking changes with Alex

- Oh no, Alex's current code has an error!
- She wants to find out what changed in her code since the last commit

Viewing commit history

- To see a history of our commits:

```
git log
```

- For more details:

```
git log --stat
```

- For less details:

```
git log --oneline
```

Looking for differences

- Compare the current directory with a previous commit

```
git diff <commit id>
```

- Compare two different commits using

```
git diff <commit 1 id> <commit 2 id>
```

$> Interactive live coding

Alex looks through the commit history and compares differences.

```
1. git log --stat

2. git diff <commit id>
```

Tracking changes with Alex

- Oh no, Alex's current code has an error!
- She compared her current file with one from before and wants to revert to a previous version

Reverting to a previous commit

- Revert the entire repository to a previous commit:

```
git switch -c <newname> <commit id>
```

create new branch

- Revert a single file to a previous commit:

```
git restore -s <commit id> <file name>
```

source to revert to

- **Be careful:** this could overwrite uncommitted changes

$> Interactive live coding

Alex looks through the commit history and reverts her code to a previous known good state.

```
1. git log --stat

2. git switch -c prev1 <commit id>

3. git restore -s <commit id> <file>
```

Tracking changes with Alex

- Follow along as Alex uses Git to simplify her work
  - Looked through the history: `git log`
  - Compared differences: `git diff`
  - Reverted to a previous version: `git restore`

What questions do we have?

GitHub

- GitHub is an online service for hosting and collaborating on code
- Based on Git version control software
- Graphical display of code history, commit messages
- Code review, project management, and many other useful features!

Login to GitHub in the command-line

- **Git Credential Manager** stores your GitHub account details safely:
  - enables the Git command to authenticate with GitHub
  - without needing to type your password every time
- Check if you're logged in:

```
git-credential-manager github list
```

- If not, login on Git Credential Manager:

```
git-credential-manager github login
```

Tracking remote repositories

- Manage remote repositories you are **tracking** using `git remote`
- View list of tracked repositories

```
git remote -v
```

- Add / remove tracked repositories

```
git remote add <nickname> <url>

git remote rm <nickname>
```

- Default name is `origin`

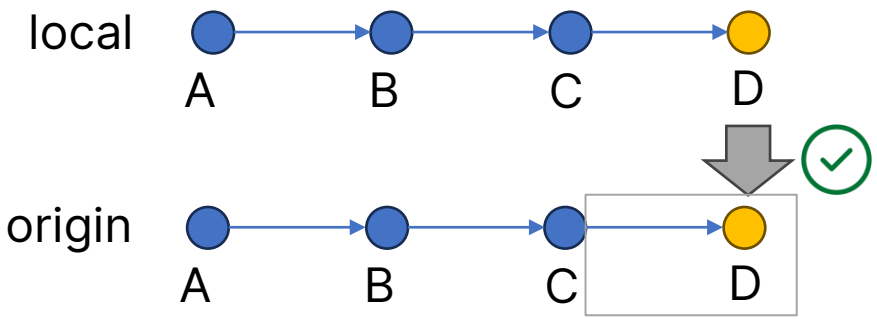
Git push

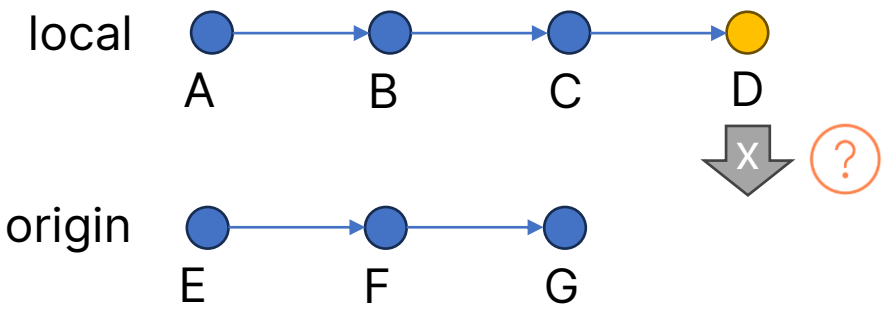
- Upload new commits using

```
git push <remote nickname> <branch name>

git push origin main
```

- Can only push commits with matching histories





$> Interactive live coding

Alex uploads her code to GitHub.

1. Create a repository on GitHub
2. `git remote add origin <url>`
3. `git remote -v`
4. `git push origin main`

What questions do we have?

Tracking changes with Alex

- Alex is asked by a teammate to help with a part of their code.
- Alex needs to:
  - download their code
  - make edits
  - track her changes
  - upload her changes

Cloning a remote repository

- Cloning downloads an entire code repository and all its history
  - Enables quick browsing and navigation through history
  - Allows you to add your commits to that history!

```
git clone <url>
```

$> Interactive live coding

Alex downloads her colleagues' repository.

```
1. cd ~
2. git clone https://github.com/dtxe/DSI_assignmentpkg
3. git status
4. git log
```

Tracking changes with Alex

- Alex is asked by a teammate to help with a part of their code.
- Alex needs to:
  - download their code
  - make edits
  - track her changes
  - upload her changes

$> Interactive live coding

Alex makes edits to the code.

1. `git commit -am “commit message”`
2. `git status`
3. `git log`
4. `git push`
   - Why doesn’t this work?

Forking a repo on GitHub

- We don’t usually have permission to edit/write to other people’s repositories
- To make changes, we can **fork** (create a working copy of) a public repository that we can write to
- Then, we can ask the original repository owner to incorporate our changes
  - This is called a **Pull Request** (we will discuss this later!)

$> Interactive live coding

Alex makes a copy of the repo and uploads her changes.

1. Fork https://github.com/dtxe/DSI_assignmentpkg
2. `git remote rename origin upstream`
3. `git remote add origin <your repo>`
4. `git push origin main`

Tracking changes with Alex

- Alex is asked by a teammate to help with a part of their code.
- Alex needs to:
  - download their code `git clone`
  - make edits `git add`
  - track her changes `git commit`
  - upload her changes `git push`
  - update tracked repos `git remote`

What questions do we have?

Course objective

How to write robust software in a team that we, our colleagues, and the public can trust and use with confidence.

Homework #1

- Due tomorrow before class
- Create a README.md file, commit, then upload to GitHub

- Detailed instructions on the GitHub repo