git

for COMS 3157 Advanced Programming

What you need to know for AP

1. Understanding version control, repositories
2. Configuring Git
3. Cloning, adding, committing
4. Submitting your solutions

That’s it!

Why learn more about Git & GitHub?

- Shareable online code backups
  - GitHub == Google Docs for programming projects
  - People can contribute to your copy on the server
- An industry standard for a programming portfolio
  - Tech companies want to see your GitHub
  - You can upload school* or personal projects
- Octocat (GitHub logo) is adorable

*Do not make school projects public if your professor doesn't want you to! Jae doesn't want you to! It counts as cheating.

This presentation is not meant to...

- replace Jae’s Git tutorial or lab submission instructions. Go read them.
- even begin to cover everything Git does and how awesome it is.
- teach you how to use Git commands.
- teach you about GitHub.

This presentation <u>is</u> meant to...

- clarify basic Git concepts needed for this class.
- complement Jae’s Git tutorial and lab submission instructions.
- provide additional resources if you wish to learn more about Git.

So let’s go!

1. Understanding version control, repositories
2. Configuring Git
3. Cloning, adding, committing
4. Submitting your solutions

Version control

“Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.”

http://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

- Git essentially takes snapshots
- It records the entire state of your project at different moments in time

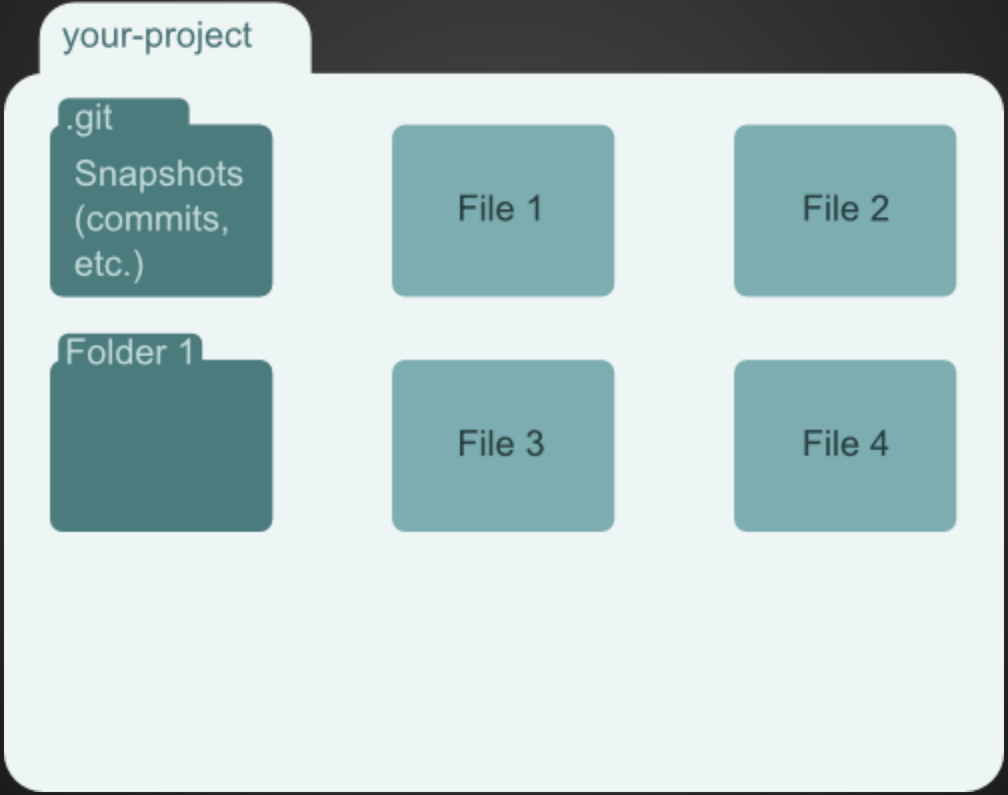
Git repository

- A repository is just a directory that somebody has *initialized* with Git
    - This is done with the simple command `git init`
- If you copy somebody else’s repository, you don’t have to (and should not!) initialize it again
    - Only the original creator needs to `init`

Git repository (repo)

- Let’s say you initialized `your-project`
  - You should see a new subfolder called `.git`
  - That’s the actual repository where Git tracks changes
- The entire folder (`your-project`) is also often referred to as the repository, or repo
- Short version: A repo is a set of files and subfolders that Git is tracking for you

Git repository (repo)



1. Understanding version control, repositories
2. Configuring Git
3. Cloning, adding, committing
4. Submitting your solutions

Configuring Git

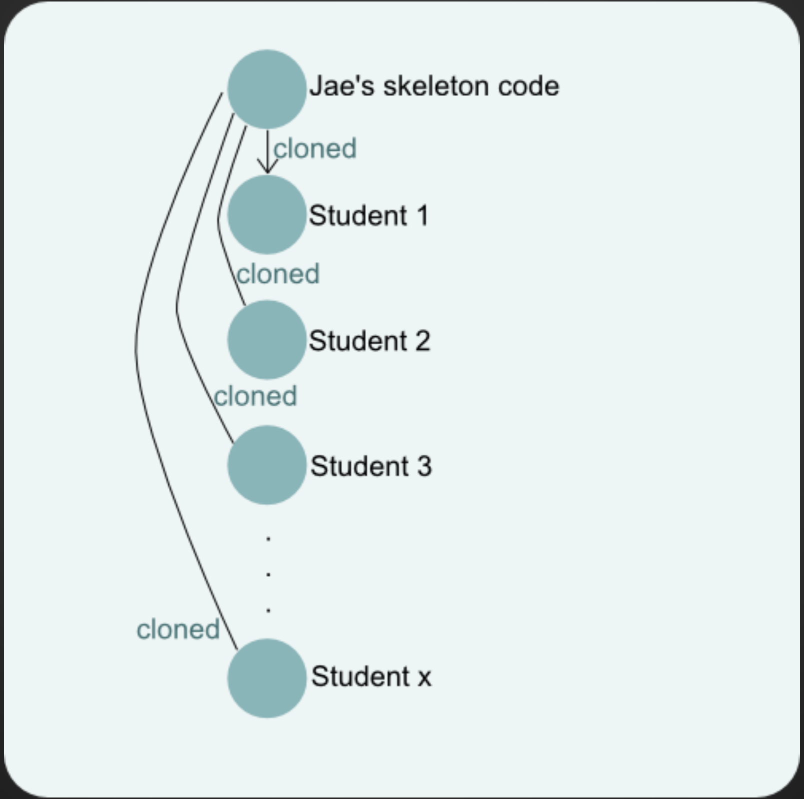
- You need to set up your name and email
- Why? Multiple people can make changes to repos
  - Git needs to keep track of who is making which changes
- This is how we’ll identify your submissions
  - See Jae’s Git tutorial and lab submission instructions
  - Make sure it’s your Columbia email!

1. Understanding version control, repositories
2. Configuring Git
3. Cloning, adding, committing
4. Submitting your solutions

Cloning repositories

- You can clone someone else's repo instead of initializing your own
    - You get copies of their current files
    - You also get their entire history (the `.git` folder)
- This is how you will start all of your labs
    - You will clone Jae's code
    - This means you all get a copy of his repo
    - This means you all start from the same point
    - This is how we grade (more on this later)

Cloning repositories



Adding and committing changes (in imprecise terms)

- There are two parts to keeping track of changes: adding and committing

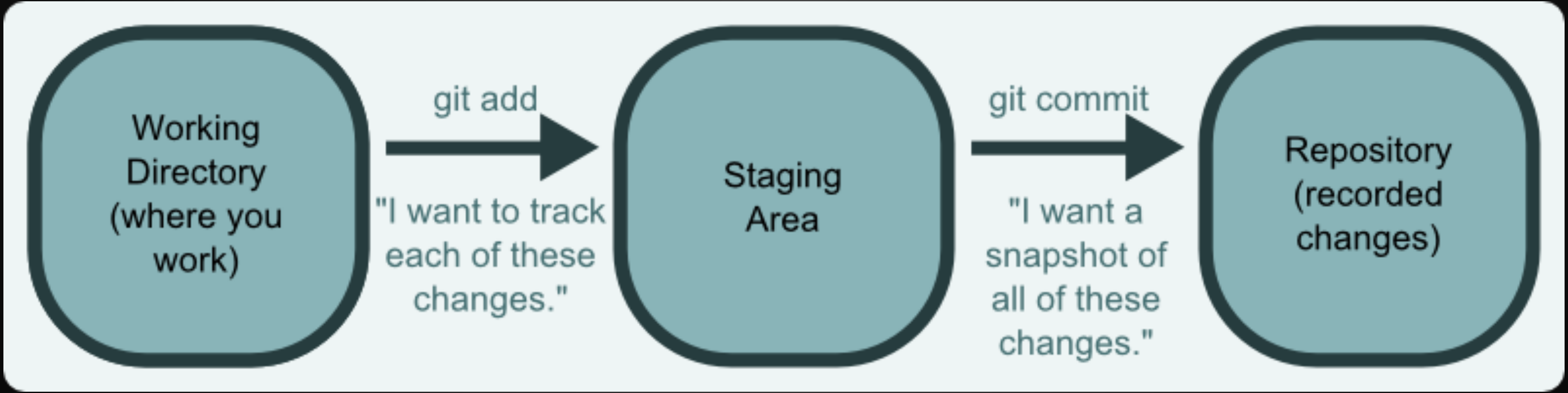
- Adding a file (`git add`) tells Git:
  "Next time I take a snapshot, include these changes."

- Committing (`git commit`) tells Git:
  "Take a snapshot of all the changes I added."

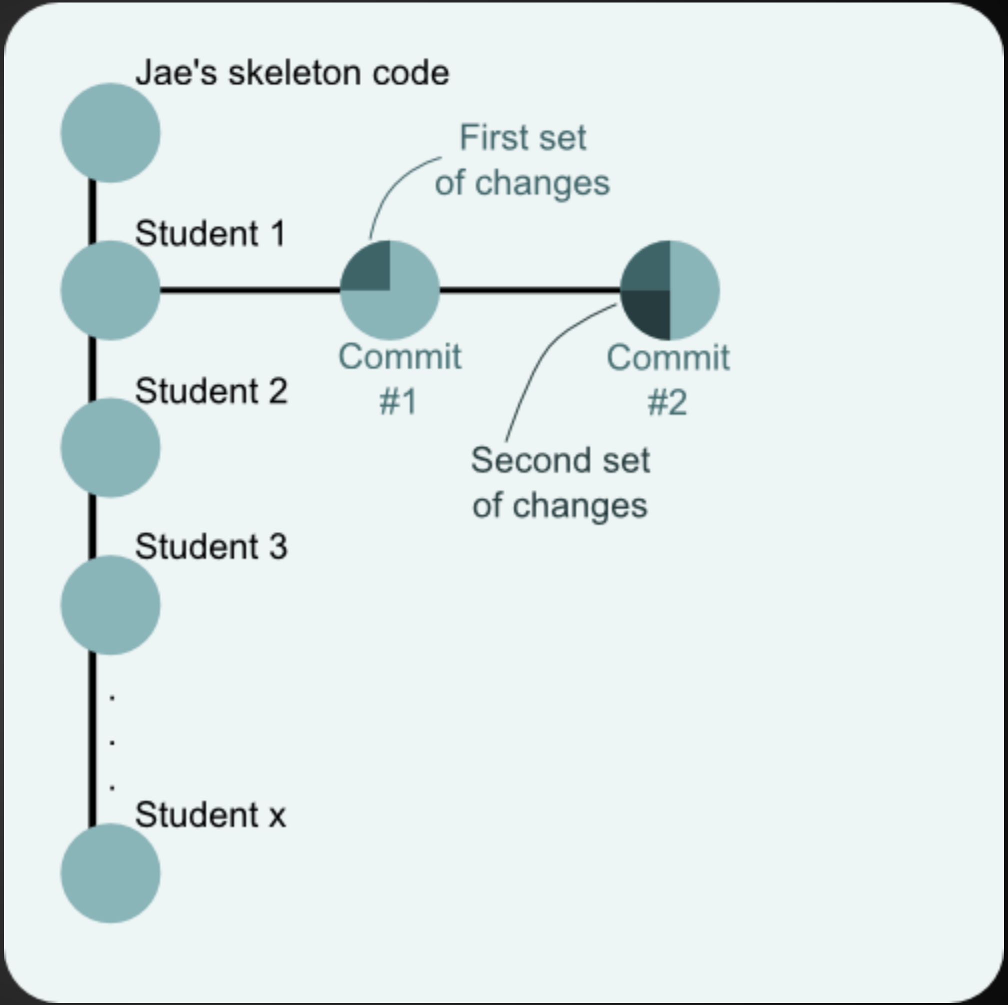
Adding and committing files (in technically correct terms)

- Terminology clarification
  - Working directory: the directory you are currently in (not all files necessarily being tracked by Git)
  - Staging area: where files are stored before commit
- `git add` actually adds files to staging area
- `git commit` moves changes out of the staging area and into the `.git` folder (where all commit info is kept)

Putting it all together


Working Directory (where you work)
git add
"I want to track each of these changes."
Staging Area
git commit
"I want a snapshot of all of these changes."
Repository (recorded changes)

Putting it all together

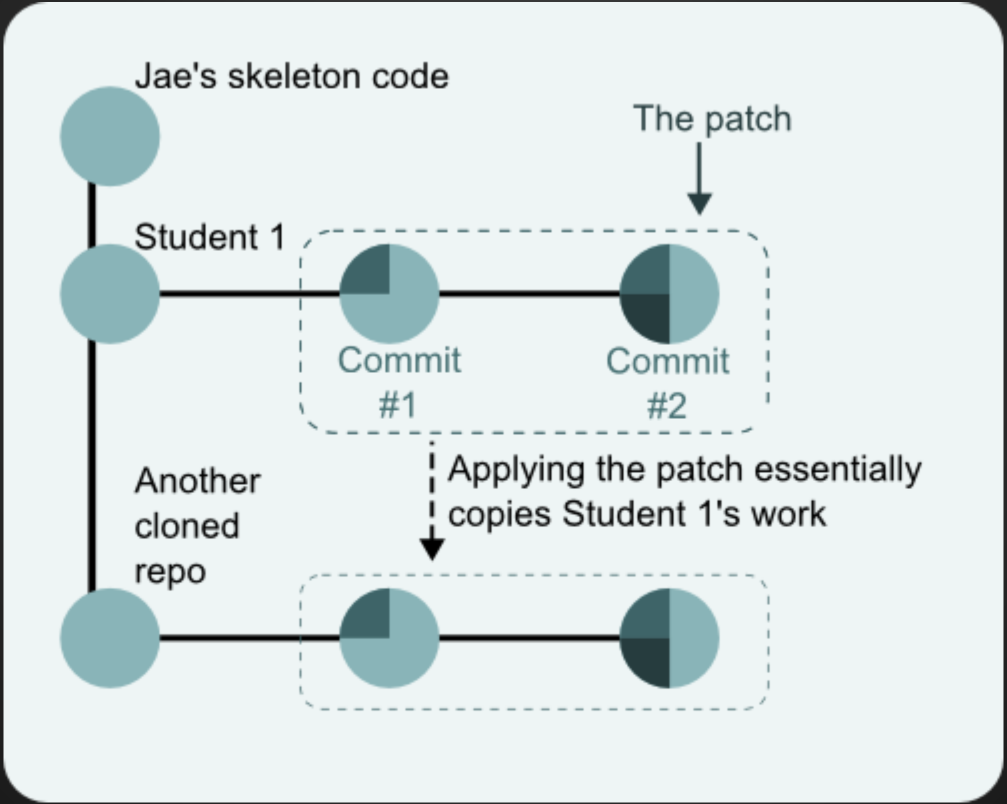


1. Understanding version control, repositories
2. Configuring Git
3. Cloning, adding, committing
4. Submitting your solutions

Submitting your code

- Run Jae's provided submit script
- The script gives us a patch
  - This is a record of all of your changes
- Two things happen with this patch
  - The script sends us (the TAs) a copy of the patch
  - The script clones the skeleton code into a new subfolder called **`labN-TIMESTAMP`** and applies your patch for you

Submitting your code



Wait, why do the TAs get a patch?

- The TAs cannot access your CLIC account directly
  - That would be creepy

- Instead, we just get a copy of your patch
- Then we can see the changes you made without hacking your account

What was that second part with the clone and the patch and the things?

- The script creates a new subfolder in your **`labN`** folder called **`labN-TIMESTAMP`**
- It then clones the skeleton code into that folder
- It then applies your patch to the cloned skeleton code

What was that second part with the clone and the patch and the things?

- Do you see why this works?
    - You start in the same place (Jae’s cloned skeleton code)
    - You apply the same changes (the ones you’ve been committing throughout the lab)
    - Which means you end up in the same place (the work you’re trying to submit)
- This is exactly how we’re going to grade

Quick sidenote: clone vs. init

- You should ALWAYS start labs by cloning Jae’s skeleton code
- Trying to apply a patch to a repository that’s not related doesn’t make any sense
- You should NEVER `git init`
    - Then we cannot apply your patch to Jae’s skeleton code
    - That means you get a zero

Okay, so what happens again?

- Having your patch lets us see your work
- Applying your patch to the **`labN-TIMESTAMP`** folder lets you see exactly what changes you just sent us
- You should always run **`make`** in this folder after submitting
    - If your code does not build for you, *it won't build for us either!* You're going to get a ZERO! Fix it!

So I fixed my mistakes - I’m good?

- Not unless you add, commit, and re-submit your changes
- Do you understand why?
  - We only get your most recently submitted patch
  - We cannot access your CLIC account
  - If you make changes in your directory but do not resubmit, we’ll never see them
  - You need to send us a new patch with your new commits by running the submit script

tl;dr

- ALWAYS clone Jae's skeleton code
- NEVER run `git init` for labs
- ALWAYS commit all changes you want us to see
- ALWAYS run the submit script
- DON'T FORGET to test your code after submitting
    - If `make` fails for any reason, you get a ZERO
- ALWAYS resubmit if you commit new changes
- GO READ the Git tutorial and lab submission instructions carefully
    - Email the listserv or come to office hours with any questions

Additional resources

- http://rogerdudler.github.io/git-guide/
  - “No deep shit” quick guide
- http://blog.interlinked.org/tutorials/git.html
  - More extensive overview, good diagrams
- http://ktown.kde.org/~zrusin/git/
  - Commands cheatsheet

Additional additional resources

- https://guides.github.com/
  - Guides for higher-level Git and GitHub concepts
- https://help.github.com/articles/good-resources-for-learning-git-and-github/
  - Even more resources