Week 10

Diana Gage
www-personal.umich.edu/~drgage

Terminal, Git

Git Slides by Tara Safavi

How was Exam 2? Any questions before we get started? Final Projects?

Command Line Review

ls: list the items in the folder

pwd: print the current working directory

cd: change directory
navigate into a new folder

mkdir: make a new directory (new folder)

cd

Change Directory

cd /	go to root directory
cd directory	go to directory
cd ..	go up a folder
cd OR cd ~	go to home directory

Terminal

- Lecture slides are very useful for terminal!
- You get faster with practice
- What does cd do?
- What does ls do?
- What does pwd do?
- What does mkdir do?
- What does g++ *.cpp do?

Using git

- important for Final Projects
- You will use git through Terminal commands
- Git provides:
  - Version Control
  - Source Management
  - Automatic Merge Conflict resolution
  - Attribution

Version control

- Revert back to the previous version of your code before you made that silly mistake
- Get it back to how it was (x) changes ago so that it works again
- It does this by keeping a list of old versions of the code that you can switch back to

Source Management

- Instead of having multiple people (your team) working on different version of the code, you have one cohesive, **online** place to store the code and modify it
- What's the main way there could be an issue with this?

Source Management

- Instead of having multiple people (your team) working on different version of the code, you have one cohesive, **online** place to store the code and modify it
- What's the main way there could be an issue with this?
  - Two people change the same part of the code, and push those changes
  - What is this called?

Source Management

- Instead of having multiple people (your team) working on different version of the code, you have one cohesive, **online** place to store the code and modify it
- What's the main way there could be an issue with this?
  - Two people change the same part of the code, and push those changes
  - What is this called? **Merge Conflict**

Merge Conflict Resolution

- Git will automatically merge changes whenever possible – but if the same code is changed within the same window of time, there is a conflict – who's changes are correct? Maybe both work, which is better?
- You are responsible for resolving the merge conflict by deciding which change should stick

Attribution

- Git tracks authorship of every line of code, so it is easy to see how much each team member contributed
- Staff members can see your code in case there's some crazy issue we can help with
- This is important for your grade on the Final Project – all team members should be participating – staff can keep track

Review: Why Git?

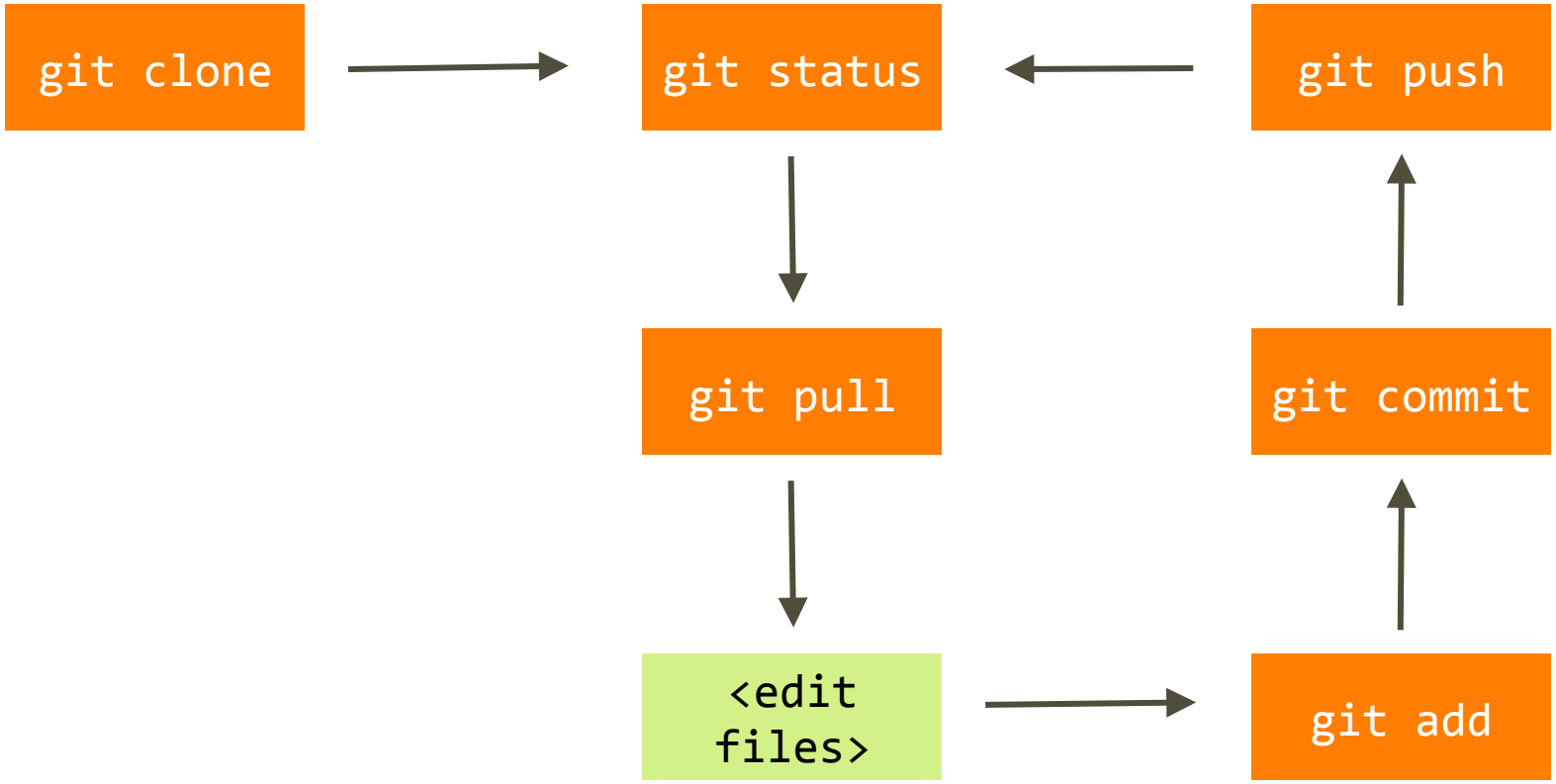
Git provides:

- Version Control
  - We can backtrack instantly if needed
- Source Management
  - Code is transferred instantly between computers
- Automatic Merge Conflict Resolution
  - Code is merged instantly whenever possible
- Attribution
  - If you write code, you are guaranteed credit for it

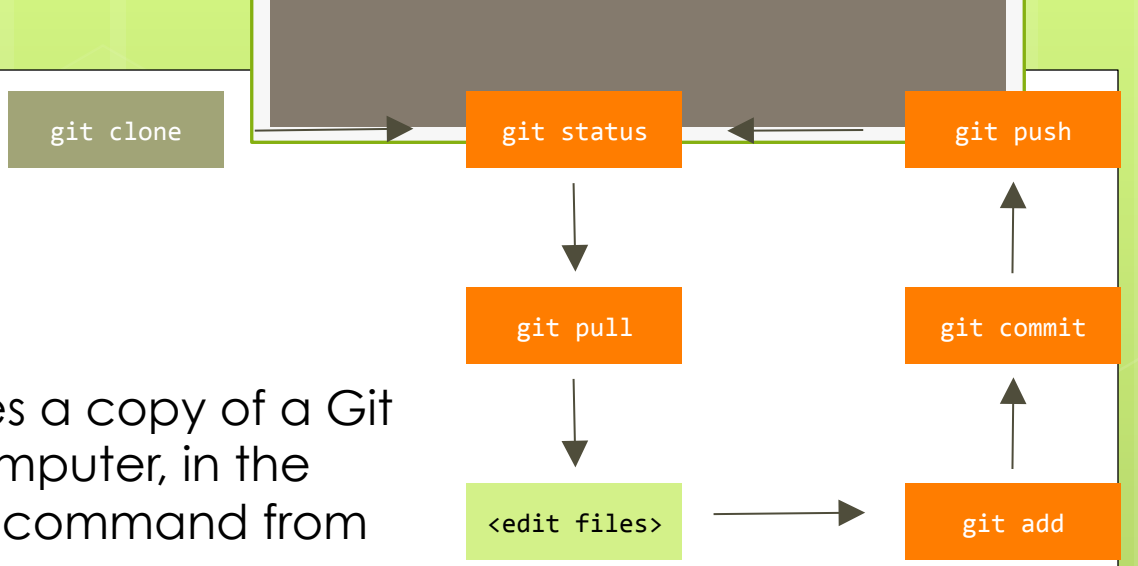
Why Git?

- All about improving efficiency and sanity
  - You know if you break the project, you can easily go back to how it was before it was broken
- You can place more trust in your teammates and experiment more with different ideas for the project without risks

Gitting started



git clone

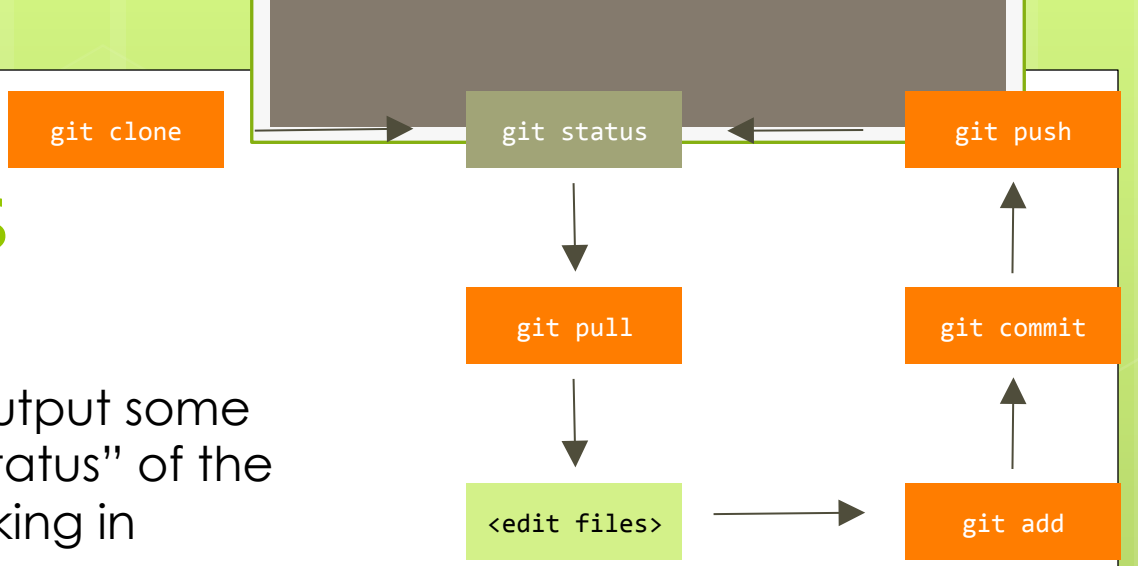


- This command makes a copy of a Git repository on your computer, in the directory you ran the command from
- For example:

```
> git clone https://github.com/tsafavi/wild-style.git
```

- If you ran this command from your Desktop, you would have a copy of the https://github.com/tsafavi/wild-style repository called `wild-style` on your Desktop
- Your `wild-style` repo on your Desktop is not the same copy as the repo you cloned from GitHub
- However, the other Git commands allow you to sync changes between the GitHub repository and your own

git status



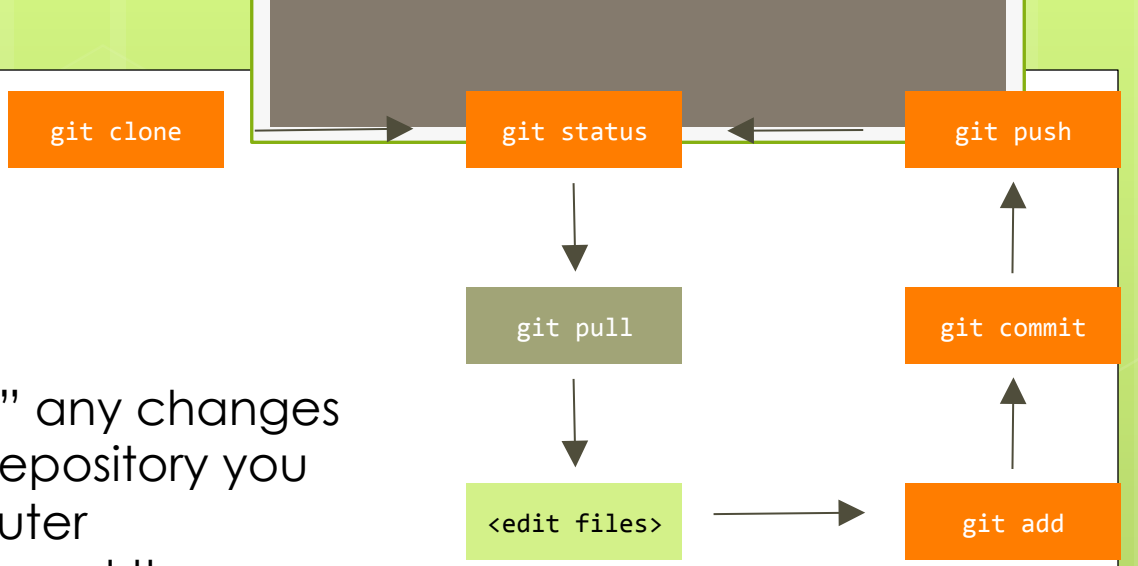
- This command will output some information on the "status" of the repository you're working in

```
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in
working directory)

        modified:   everythingIsAwesome.py

no changes added to commit (use "git add" and/or "git commit
-a")
```

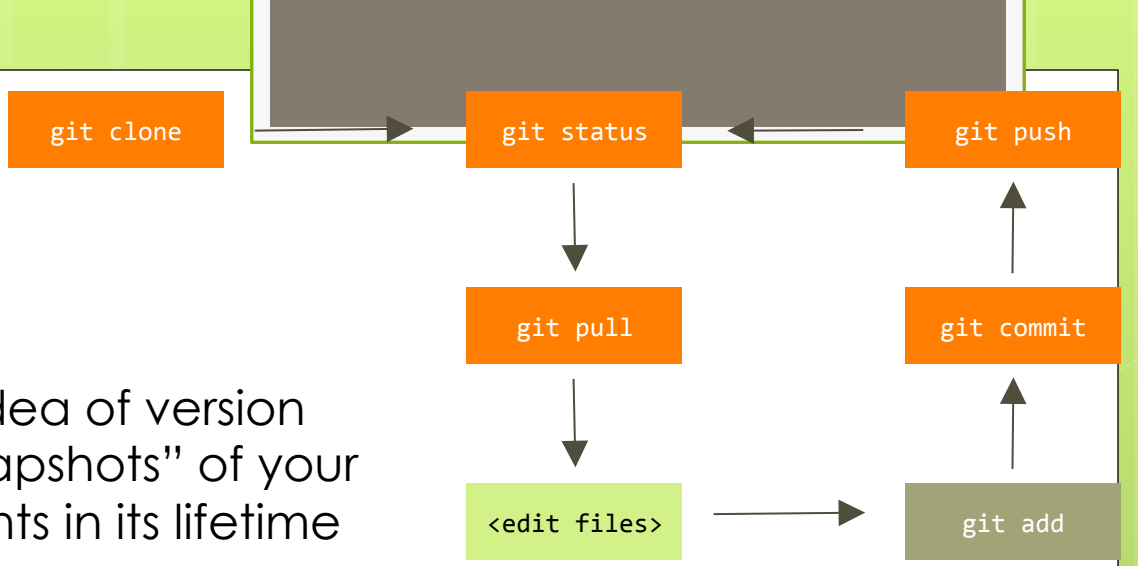
git pull



- This command "pulls" any changes from the centralized repository you cloned to your computer
- If someone else changed the repository that you cloned from, you need to be able to get the latest version of that repository
- **`git pull`** will make your local copy of the code up to date with whatever changes are in the repository you cloned from

```
> git pull
```

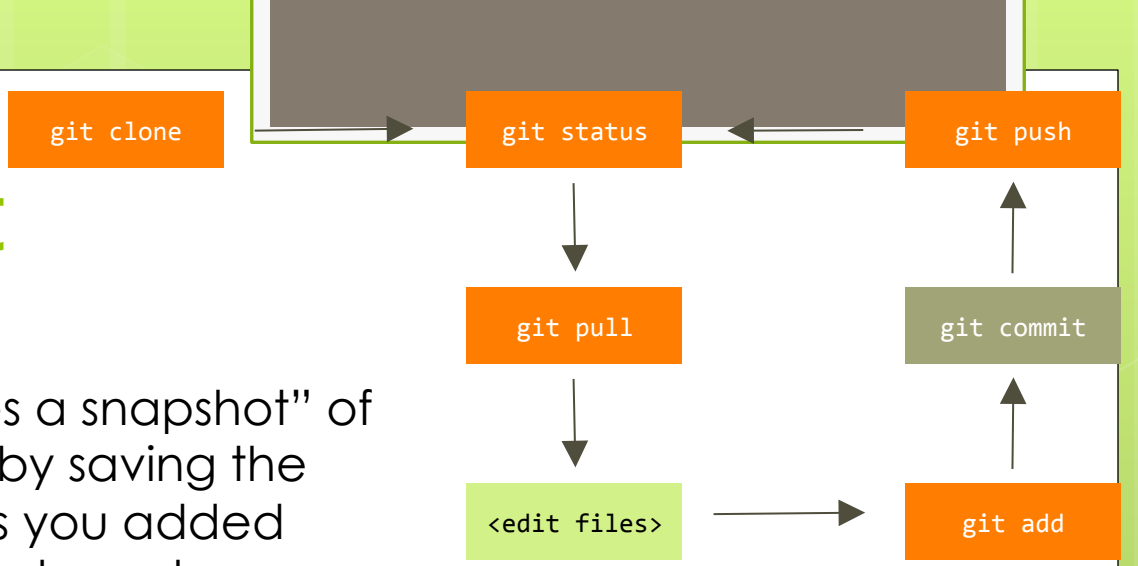
git add



- Going back to the idea of version control: you take "snapshots" of your code at different points in its lifetime
  - `git add` is the first step to taking those "snapshots"
- It makes Git aware that you changed some files
- Say you edited `everythingIsAwesome.py` in your `wild-style` repository. You want to include your changes to `everythingIsAwesome.py` in your next "snapshot" of your code:

```
> git add everythingIsAwesome.py
```

git commit

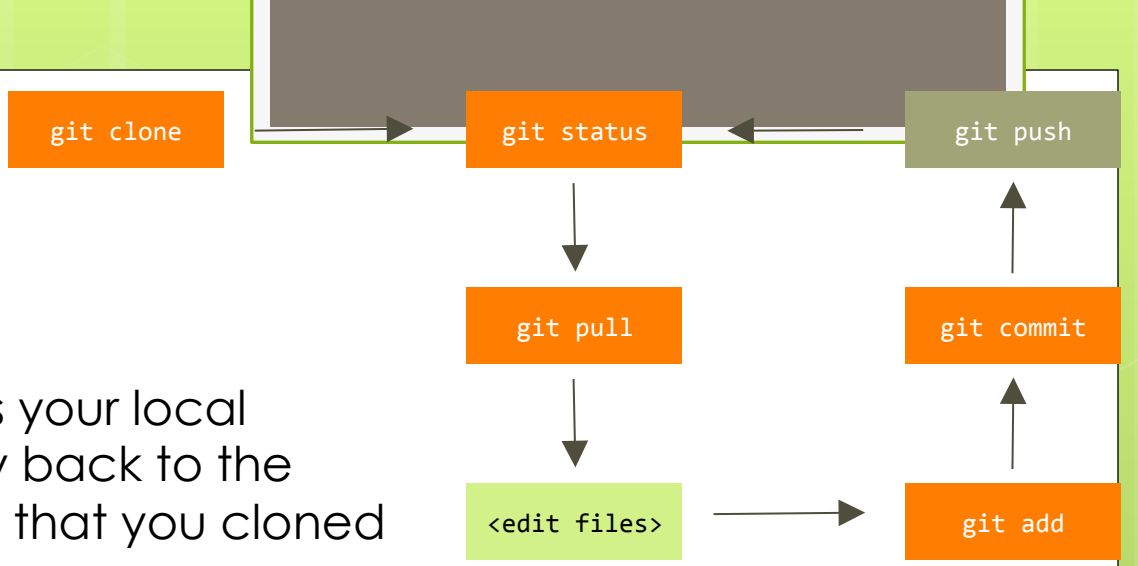


- This command "takes a snapshot" of your local repository, by saving the state of whatever files you added
- **`git commit`** allows you to make versions of your code

- You can only commit files that you have first added using **`git add`**
- When you commit your code, you must always include a commit message that explains briefly what updates you have made to the code

```
> git commit -m "Added main function"
```

git push



- This command sends your local version of a repository back to the centralized repository that you cloned from
- Say you cloned from a repo on GitHub and committed some changes
- **`git push`** sends the changes you committed back to the centralized repository hosted on GitHub

```
> git push
```

Potential Git Problems

and how to deal with them

Merge conflicts

- Let's say you and your teammate both edited line 1 of **`everythingIsAwesome.py`**. Your teammate pushed her changes to the centralized repository first, and then you committed your changes and pulled from your centralized repository
- Now Git is confused because there are two versions of the same code, and Git doesn't know which one is right

```
everything = 'extremely awesome' // your code
```

```
everything = 'super awesome' // your teammate's code
```

Merge conflicts

- This is called a merge conflict
- When this happens, you will see a message that looks like this:

```
Auto-merging everythingIsAwesome.py
CONFLICT (content): Merge conflict in
everythingIsAwesome.py
Automatic merge failed; fix conflicts and then commit the
result.
```

Merge conflicts

- Your `everythingIsAwesome.py` file will look like something like this
- Don't be scared by these symbols! Git just puts them there to differentiate between the two versions of the code it's looking at
- The top part above ===== is your version of the code, the bottom part is the version that you pulled

```
<<<<<<< HEAD
  everything = 'extremely awesome'
=======
  everything = 'super awesome'
>>>>>>> 48991968b0d802c345e8c2bb8845258613fcd01e
```

Merge conflicts

- To fix a merge conflict, delete all the symbols Git added along with the version of the code you don't want to keep
- In this example, everything in red will be deleted

```
<<<<<<< HEAD
  everything = 'extremely awesome'
=======
  everything = 'super awesome'
>>>>>>> 48991968b0d802c345e8c2bb8845258613fcd01e
```

Merge conflicts

- `git add` the file after you delete the symbols, and `git commit` to "resolve" the merge conflict
- That's all you need to do to fix a merge conflict

Vim

- If you don't enter a message when you `git commit`, Git will take you to a text editor called Vim to type your commit message
- There are a lot of different things you can do with Vim
- However, the easiest thing to do is just quit Vim and re-try your `git commit`
- To quit, type `:q`

“fatal: not a Git repository”

- If you ever get this error when running a Git command, you’re likely in the wrong directory
- Type `pwd` to print the directory that you’re currently in