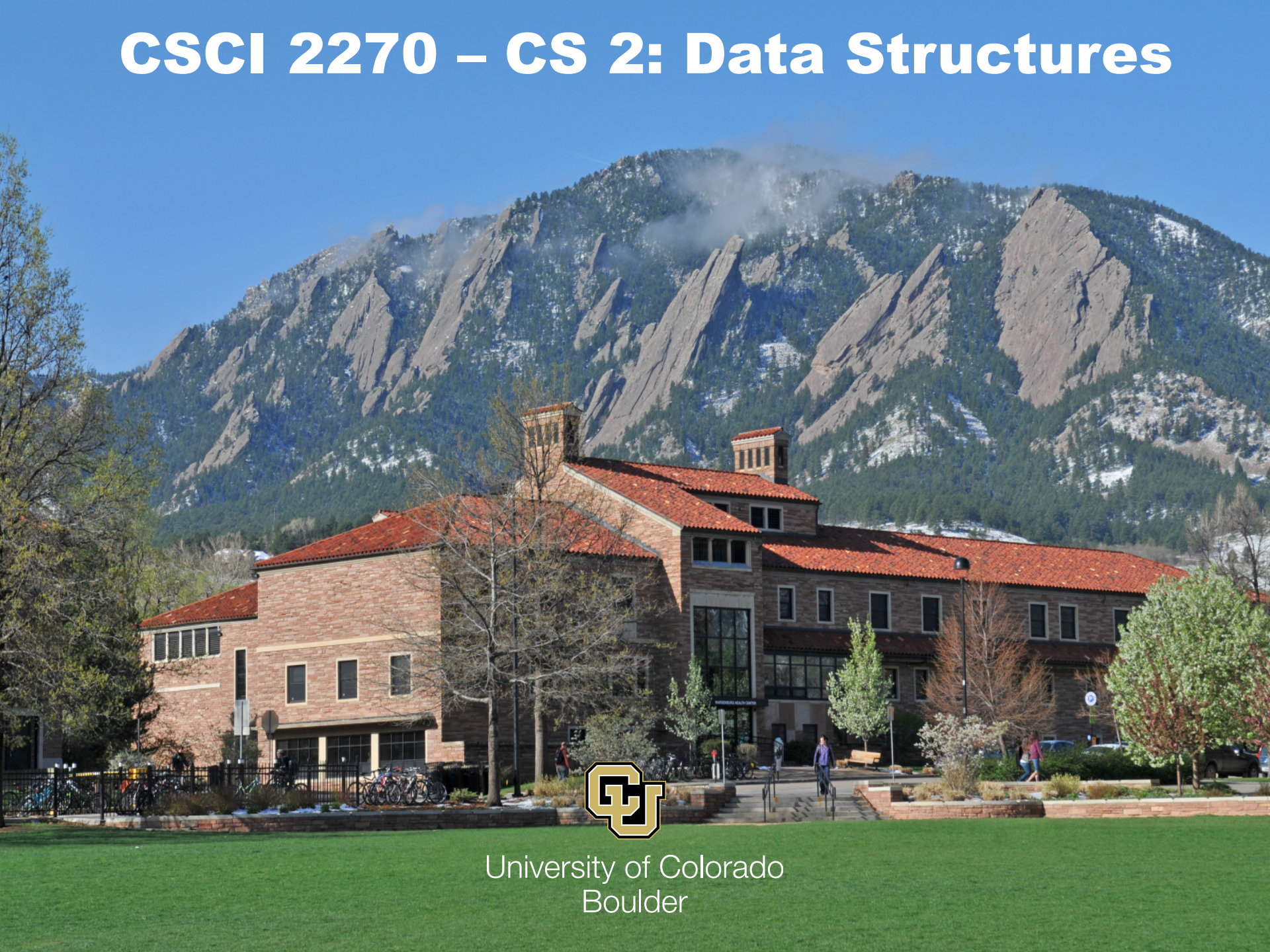
CSCI 2270 – CS 2: Data Structures



University of Colorado
Boulder

Topics

- Git

What is Git?

- Open-source Version Control System (VCS)
- Store content and code in repositories
- Repos are stored in remote server but are locally saved on every team member's computer

Git Account

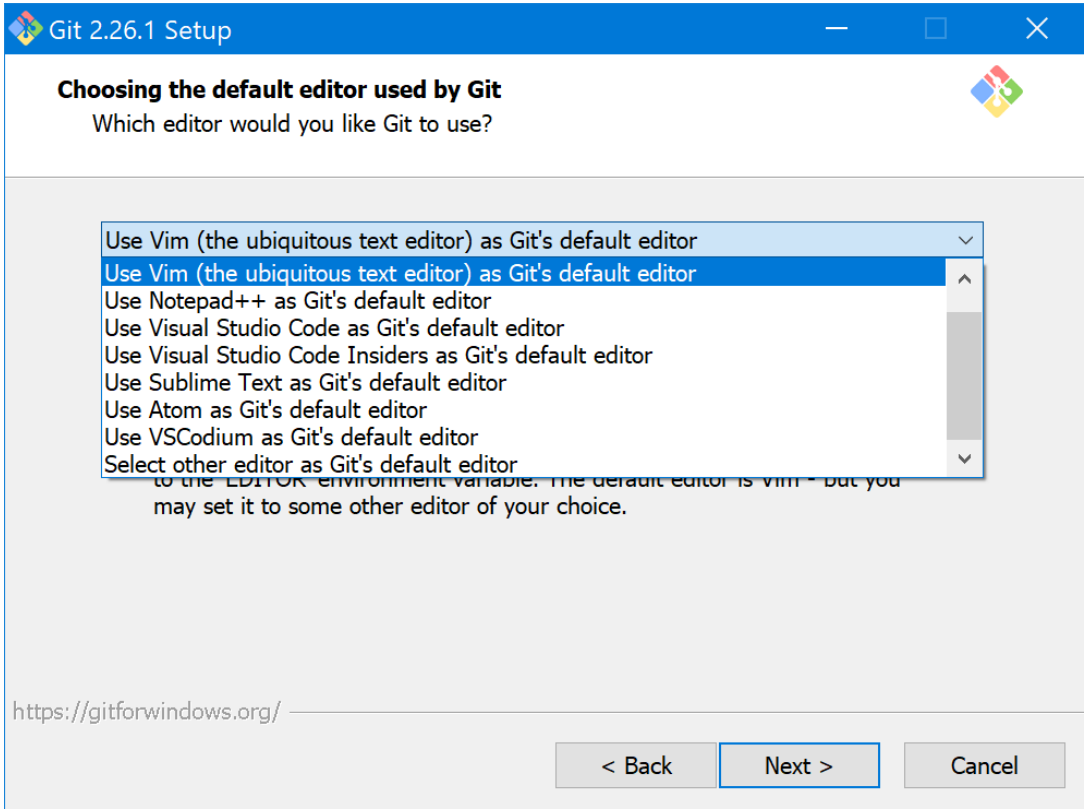
- Sign up for a Git account
  - https://github.com
  - E.g. https://github.com/asaashraf

Git Clients

- Git for Windows
  - https://git-scm.com/download/win
- GitHub Desktop
  - https://desktop.github.com
- TortoiseGit
- SmartGit
- SourceTree
- Mac users
  - https://git-scm.com/download/mac
  - https://git-scm.com/downloads/guis

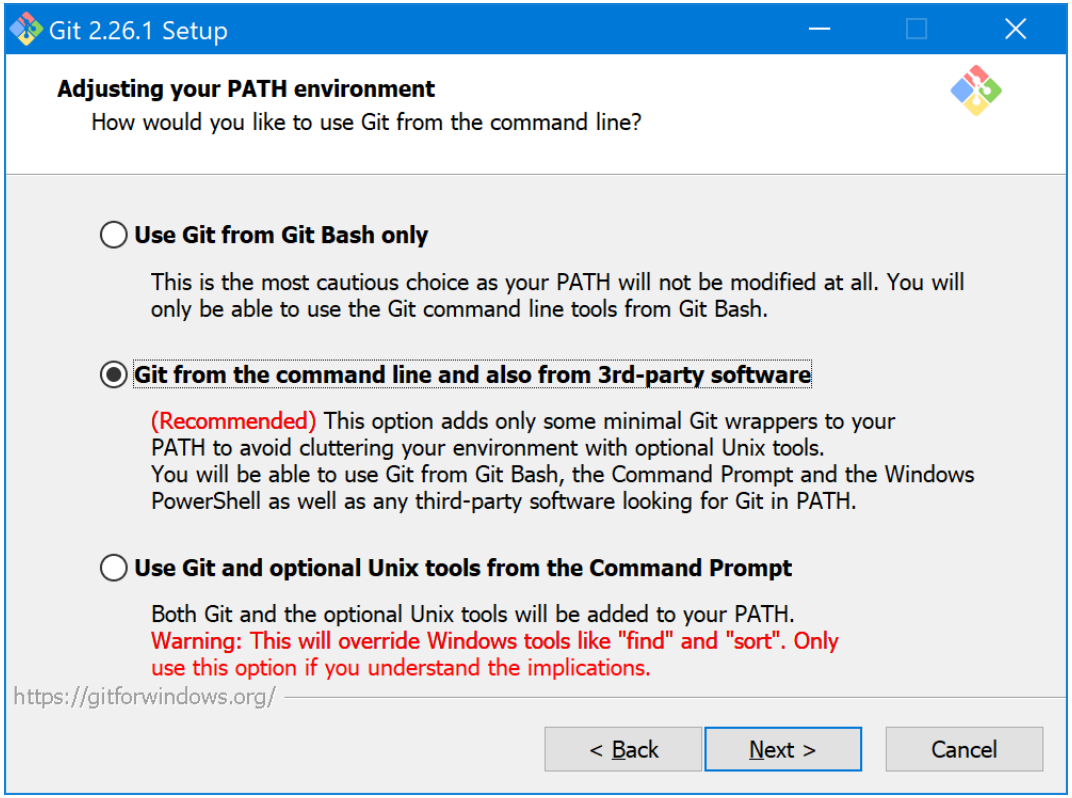
Git for Windows

- Git for Windows installation



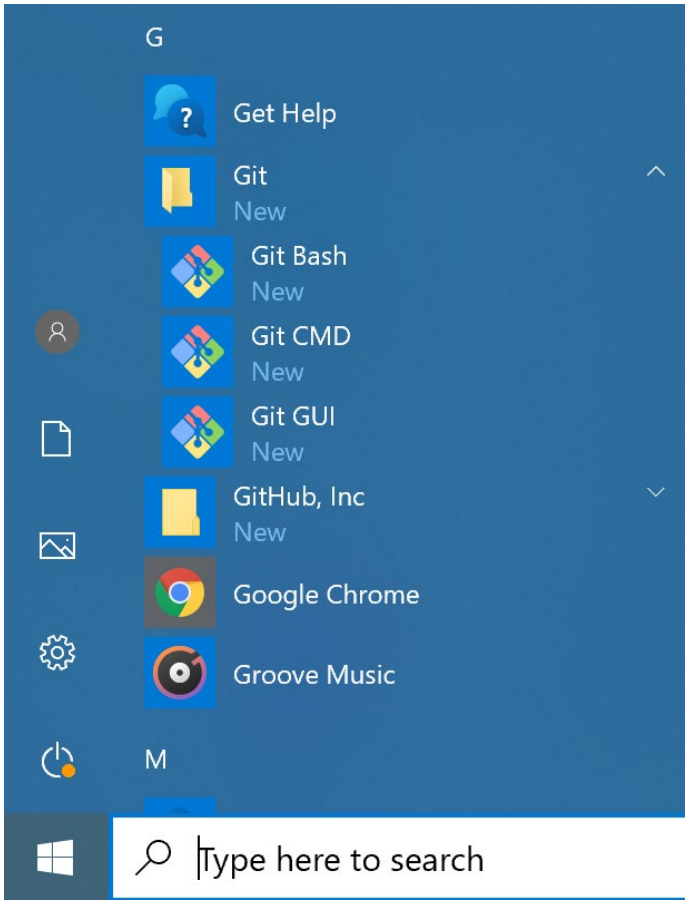
Git for Windows

- Git for Windows installation



Git for Windows

- Open Git Bash



Git Commands

- Use a cheat sheet
- https://git-scm.com/docs
- https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf

- Source: a lot of the examples and screenshots in this slide deck are taken from the pdf listed above.

Git Commands – Config.

- Configure Tooling
  - Run the following commands to configure your Git username and email using the following commands, replacing my name with your own. These details will be associated with any commits that you create:

    $ git config --global user.name “Asa Ashraf”

    $ git config --global user.email “asa.ashraf@colorado.edu"

Git Commands – Create Repo

- Creating repository

  $ git init

  – Turn an existing directory into a git repository

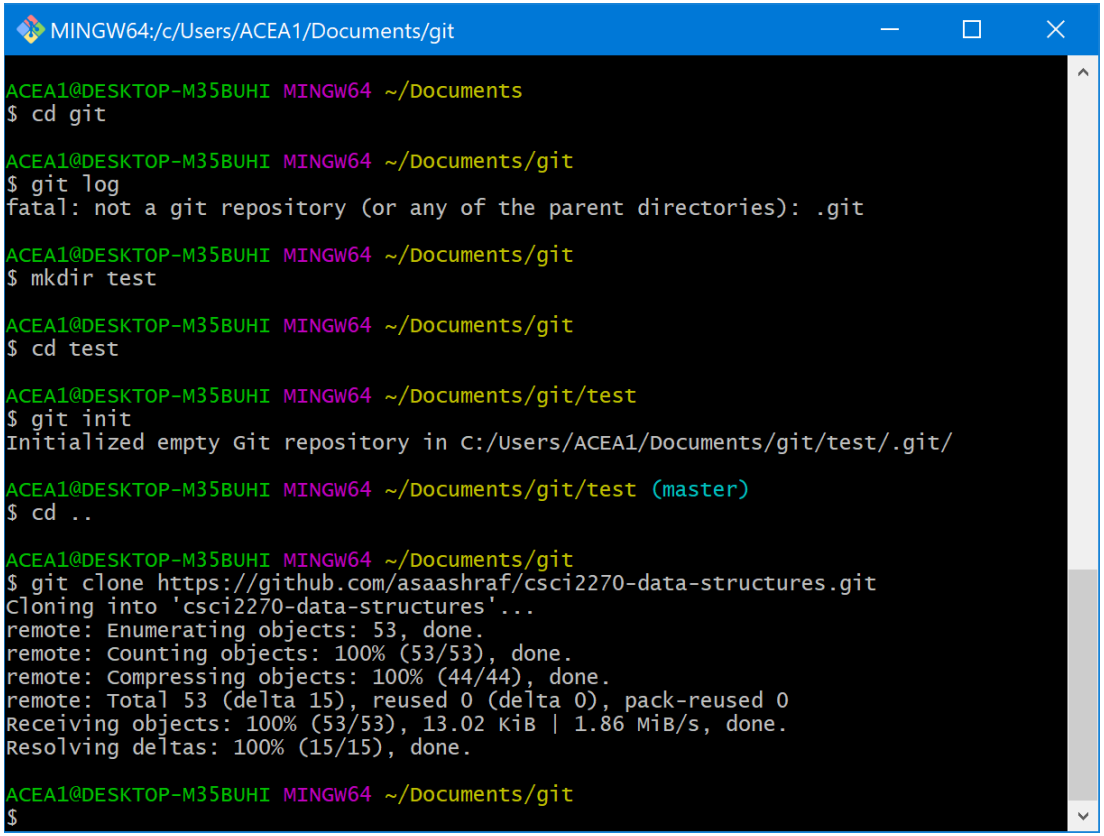
  $ git clone [url]

  – Clone (download) a repository that already exists on GitHub, including all of the files, branches, and commits

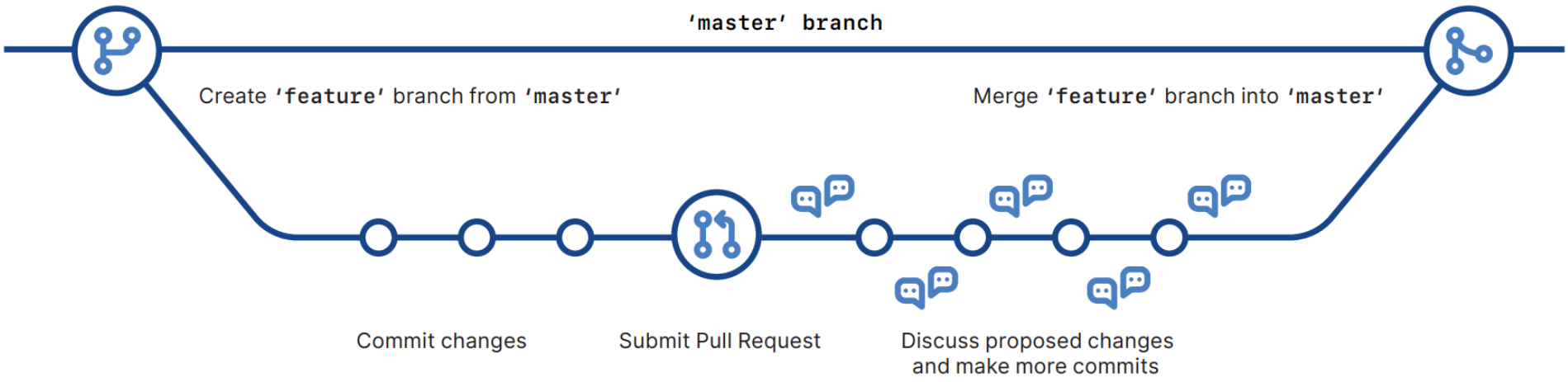
- Note: after the statement(s) above you will be able to find a .git directory that contains information about your current branch. View by typing cat .git/HEAD at the root repo directory.

Git Commands – Create Repo

- Creating repository



Git Branches



Git Branches

- By default, we checked out from the Master branch. We can create our own, “personal” branch to work on.

- Create a new branch
  - git branch NEW-BRANCH-NAME
- Switch to the branch
  - git checkout NEW-BRANCH-NAME

- There’s a shortcut method so that you don’t have to type both statements above. Instead, use –b:
  - git checkout –b NEW-BRANCH-NAME

- I could check out a specific branch by using:
  - git checkout BRANCH-NAME

Git Exercise 1

- Create a directory named test and a main.cpp within it
  - mkdir test
  - vi main.cpp *// write a simple main() function*
- Turn the test directory into a new repository by running git init.
  - cd test
  - git init
- Check with branch you're working on
  - git status *// git branch*
- Add main.cpp to the repository staging area
  - git add main.cpp *// git add --all (add any changed repo files)*
- Create a new commit with a message
  - git commit –m "added main.cpp to the repo"
- Since this is a new project we created locally, it's not tied to a GitHub repo. So, create a new GitHub repo first on GitHub.com.
  - Click the new repository button in the top-right then click Create repository. Name the repo "test."
- Add the repo we just created (test) to the current project as one of the servers
  - git remote add origin https://github.com/asaashraf/test.git
- Verify that it has been added
  - git remote –v *// -v flag simply shows you the git repos that you have configured*
- Push the changes in your local repo to GitHub
  - git push –u <branch we want to push to> <my branch I want to push>
  - git push –u origin master *// push any commits you've done back up to the server*

Git Exercise 2

- Branching and merging
- Inside of a project directory you want to create a new branch for, create a new branch called new_feature and check it out (e.g. inside of C:/…/Documents/git/test)
  - git branch new_feature
  - git checkout new_feature *// don't forget we can use –b flag*
- Check to make sure the new branch was created
  - git branch *// should see master and new_feature*
- Make some changes like adding a file (e.g. prog.cpp)
- Make sure the remote server is where we want it to go
  - git remote –v *// should see the name of your remote repo*
- Let's push the new branch (new_feature) to the repo on GitHub
  - git push –u origin new_feature // origin is our remote repo
- But there's one thing we forgot! We forgot to add prog.cpp so you won't find it on GitHub.
  - git add prog.cpp
  - git commit –m "added prog.cpp to the repo"
  - git push –u origin new_feature

Git Exercise 2 (Continued)

- Go back to the main branch
  - git checkout master
- Make a change to an existing file within master (e.g. add a comment to prog.cpp or add a new file abc.cpp)
- Switch back to new_feature because we will want to merge the changes we just made on master with new_feature
  - git checkout new_feature
  - git merge master
- Assume you make a change to new_feature, which is the current repo. For example, change a file's contents or add a file. Now, merge it into master.
  - git checkout master
  - git merge new_feature

Git Exercise 3

- git clone [url]
  - clone (download) a repo that already exists on GitHub, including all of the files, branches, and commits
- Clone the csci2270-data-structures repo
  - git clone https://github.com/asaashraf/csci2270-data-structures.git
- Make a change to one of the files in one of the directories (e.g. use vi editor to add a comment to BST.cpp)
- Verify the changes
  - git status *// the file we changed should be listed*
- Add the file we changed and commit. There are many options to do this.
  - Option 1
    - git add <file>
    - git commit –m “added new comment”
  - Option 2
    - git commit –a *// adds all the files to your commit*
      - This option will open in whatever editor you configured when you installed git. Type in a comment and save.
- Finally, push the changes to GitHub and the new changes will be reflected on the remote server
  - git push origin master
- Note: use a **staging area** to commit files (research “Staging a file”)

Git Fetch vs Git Pull

- Assume that we want to download new data from a remote repository (e.g. master) but we don't want to integrate the new data into our working files. So, fetch does not manipulate or destroy anything.
  - git fetch origin
  - git branch –a *// will display remotes/origin/master and ../new_feature*
    - ***These are remote tracking branches that are created by the git fetch origin command above***
  - git log origin/master ^master *// let's find out what is different between the two*
  - We can review changes because we might want to review before integration. If we want to actually merge/integrate the two branches, use a git pull (see below)
- To update your current HEAD branch with the latest changes from the remote server, use git pull. It will pull new data and integrate it into your current working copy. Consequently, git pull can result in "merge conflict" because it will try to merge the results with your local ones. To avoid merge conflicts, use git pull only with a clean working copy. This means you shouldn't have any uncommitted local changes before you pull.
  - git pull origin master *// update local master repo with (remote) origin repo*
  - In the following example, create a local repo and then pull a branch from your friend's repo to the one you just created
    - git checkout project3
    - git pull myfriend project3 *// pulls your friend's myfriend repo to your project*

Git Fetch vs Git Merge

- git pull = git fetch + git merge

git pull <remote> <branch> *// e.g. git pull origin master*

is the same as

git fetch <remote> *// e.g. git fetch origin*
git merge <remote>/<branch> *// e.g. git merge origin/master*

- git merge definition and example
  - Assume we're in branch feature1. A git merge will merge any changes that were made to the code base on a separate branch (e.g. feature2) to feature1.
  - git merge feature2

Merge Conflict

- If commits are made on separate branches that alter the same line(s) in conflicting ways, a merge conflict will occur. Git will not know which version to keep. For example:

CONFLICT (content): Merge conflict in abc.cpp Automatic merge failed; fix conflicts and then commit the result

Git will mark the code to indicate the HEAD (master) version of the file and the other version of the file. Notice the direction of the arrows so that you know which section of the code it's referring to. For example:

<<<<<<< HEAD

>>>>>>> OTHER

You can manually fix the conflicts and then re-add and re-commit the changes.

Git Delete

- If you're finished with a branch and decide it's okay to delete it:
  - git branch –d new_feature *// if it was pushed to GitHub, it'll still be there*
- To delete the branch from GitHub:
  - git push <remote_repo> --delete <remote_branch>
  - For example:
    - ***git push origin --delete new_feature***

- **If you can't delete a branch, see "Common Errors" slide for solution.**

- **Note:** one common/simple way to delete a local GitHub repo is to remove the .git file
  - fm –rf .git
  - Then, delete the directory using rm -r

Git Revert

- Git revert HEAD~1
  - Revert the changes to HEAD by 1, meaning the last commit
- Git push origin master
  - Make a new commit that undoes those changes and then push the new commit to the origin branch (in this case the master branch)

Git Security

- SSH vs HTTPS
  - git@github.com:username/new_repo // reference remote repo like this for ssh
  - https://github.com/username/new_repo // and this is how we've been doing it

- The differences is in the way authentication is done: SSH uses keys to authenticate and HTTPS uses a username/password.

- The **SSH** protocol uses encryption to secure the connection between a client and a server. There are several steps involved in setting this up:
  - 1. generate a public/private key
  - 2. publish it on GitHub
  - 3. launch an ssh-agent to enter the passphrase you would have associated with your private key

- https://help.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh

Git Ignore

- The .gitgnore file
  - Sometimes it may be a good idea to exclude files from being tracked with Git. This is typically done in a special file named .gitignore . You can find helpful templates for .gitignore files at github.com/github/gitignore.

Common Errors

- Error: failed to push some refs to <…git>
- Solution: run *git push –u origin master –force*

- Error: Cannot delete branch <branch_name> checked out at …
- *Solution: You cannot delete the branch you're currently on. Try creating a new branch*
  - *git checkout –b new-branch*
  - *git branch –d <branch_name_i_want_to_delete>*

Questions?