This lesson is being piloted (Beta version)

Sharing Code

Overview

Teaching: 75 min
Exercises: 25 min
Questions

- How do I share my code with others using git and GitHub?

Objectives

- Initialize a new git repository with existing files.
- Add and commit files to a git repository.
- Create a new repository on GitHub from a local repository.
- Create a README.md file for a project.
- Learn the commands git push and git pull

Putting your code on GitHub

Let's get your project put on GitHub so you can share it with others. In your browser, navigate to github.com (https://github.com/). If you already have a GitHub account, click the Sign In button. If you need to create an account, click the Sign Up button.

Choosing your GitHub username

To create a GitHub account, all you need is a valid e-mail address and to choose a GitHub username, which will be public. It is now *very* common in many technical disciplines for potential research advisors or employers to look at your GitHub page to see what kind of projects you have contributed to and what kind of work you have done. Consequently, you want to choose a recognizable, professional username and always keep your GitHub profile up-to-date and professional.

Creating an online repository

GitHub is a website which provides us with a place to host our code. We are using the software `git` to version control our code. GitHub is providing us a place on the internet to put copies of those repositories. In general, you should have a different repository for each of your projects.

Once you are signed in to GitHub, on the left side of the page, click the green button that says New to create a new repository. On the next page, choose a name for your repository and write a short description of your project. You can choose whether your repository will be public or private. Even if you choose private, you can identify other specific GitHub users who can see your repository and commit to it. However, if you want other people to be able to find your project without you specifically sharing it with them, you should choose public.

For this workshop, make your repository **public.**

Note the last question, "Initialize this repository with a README". We will leave this unchecked in our case because we have an existing repository we are adding to GitHub. If you were creating the repository for the first time on GitHub, you would select this. There are also options for adding a .gitignore file or a license, but you don't have to do that right now.

Click Create repository.

We now have an empty spot on GitHub where we can put a copy of our code. On the next page, GitHub now very helpfully gives us directions for how to get our code on GitHub. The most relevant set of instructions for our current situation is the third set **push an existing repository from the command line**.

📌 Local and Remote Repositories

We are using the word “repository” here to refer to both the local copy of the code on your machine, and to the copy on GitHub. A “repository” simply refers to a directory with code which is being tracked by git. Sometimes, people will shorten “repository” to “repo”.

Since we initialized git in our project using `git init` , it is now a repository. We will enter commands into the terminal to get our local copy onto GitHub (our online repository). That is what is meant by “push an existing repository from the command line.”

Pushing your code online

Before we follow these directions, let’s talk about what the directions mean. First, we have to tell `git` that the online repository we created exists. When you want to be able to put the code that is on your computer (your local repo) into an online repository, you have to add what git calls remotes to your local repository. Think of remotes like roads; if there is a road between two places you can travel between them. If there is a remote between two repos, information can travel between them.

Currently, our repository has no remotes. You can see this by typing

Bash

```bash
$ git remote -v
```

You should see no output from this command.

Now, we will follow the directions given by GitHub to add remotes. In the Terminal window, type

Bash

```bash
$ git remote add origin https://github.com/YOUR-USERNAME/REPOSITORY-NAME.git
```

Note that the URL in your first command will be different because it will contain your own GitHub username and repository name. Just copy the line that GitHub gives you.

Bash

```bash
$ git push -u origin master
```

Note that you will need to enter the first command, press enter, and then enter the second command. The first command adds a remote named origin and sets the URL to our repository. The second command pushes our repo to where we have set as origin. The word master means we are pushing the master branch.

Now if you refresh the GitHub webpage you should be able to see all of the new files you added to the repository.

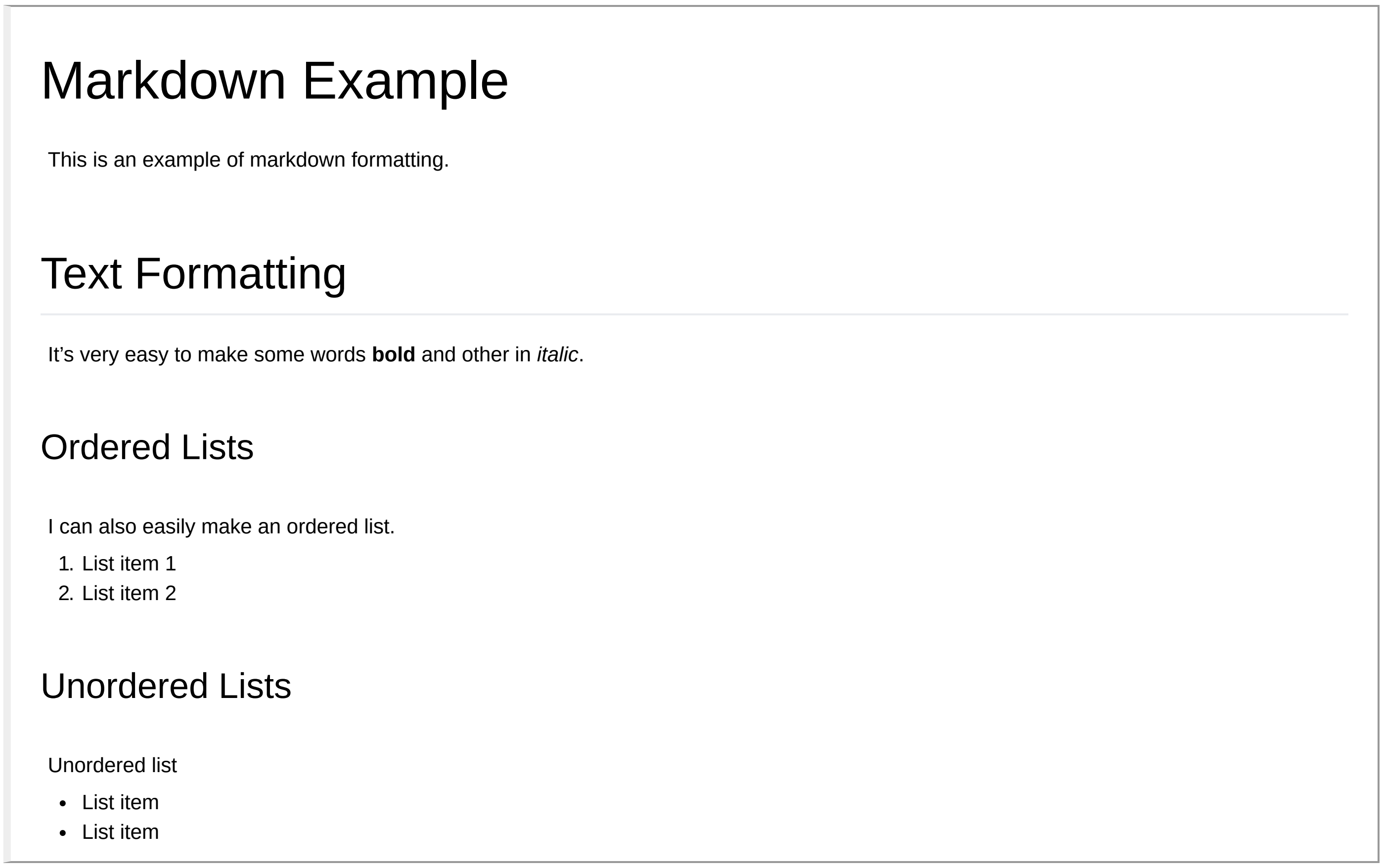
Adding a README.md file

If your repository contains a file named `README.md` then GitHub renders it into a nice description so that anyone who comes across your repo knows what the project is about. The `md` extension on this file refers to “markdown”. Markdown is a simple language which can be used for text formatting.

Here is an example

```
# Markdown Example

This is an example of markdown formatting.

## Text Formatting
It's very easy to make some words **bold** and other in *italic*.

### Ordered Lists
I can also easily make an ordered list.
1. List item 1
1. List item 2
1. List item 3

### Unordered Lists
Unordered list
* List item
* List item
* List item
```

This will make something that looks like this:



Markdown Example

This is an example of markdown formatting.

Text Formatting

It's very easy to make some words **bold** and other in *italic*.

Ordered Lists

I can also easily make an ordered list.

1. List item 1
2. List item 2

Unordered Lists

Unordered list

- List item
- List item

Exercise

In your favorite text editor, create a new file called `README.md` and write a description of your geometry analysis code and how it works. If you want to have section headings in your file, use ## to indicate a section heading. Otherwise, just type in your description.

Make sure you save your file in the folder with the rest of your project. Using the git commands learned above, add the file to your repository, commit it with an appropriate commit message, and push it to the master branch of your GitHub repository.

Solution

Now if you refresh the GitHub page for your project, you will see your project description at the bottom of the page. That's it! Your code is now on GitHub for the world to see, download, and use!

Time Check

If you are running out of time, this is a good place to end the lesson. Especially if the GitHub lesson is the last lesson being covered in the workshop, this is a good place to stop.

Working With Multiple Repositories

One of the most potentially frustrating problems in software development is keeping track of all the different copies of the code. For example, we might start a project on a local desktop computer, switch to working on a laptop during a conference, and then do performance optimization on a supercomputer. In ye olden days, switching between computers was typically accomplished by copying files via a USB drive, or with ssh, or by emailing things to oneself. After copying the files, it was very easy to make an important change on one computer, forget about it, and go back to working on the original version of the code on another computer. Of course, when collaborating with other people these problems get dramatically worse.

Git greatly simplifies the process of having multiple copies of a code development project. Let's see this in action by making another clone of our GitHub repository. For this next exercise **you must first navigate out of your project folder**.

Bash

```
$ cd ../
$ git status
```

Before continuing to the next command, make sure you see the following output:

Output

```
fatal: Not a git repository (or any of the parent directories): .git
```

If you do not get this message, do `cd ../` until you see it.

Next, make another copy of your repository. We'll use this to simulate working on another computer.

Bash

```
$ git clone https://github.com/YOUR_GITHUB_USERNAME/YOUR_REPOSITORY_NAME.git friend
$ cd friend
```

Check the remote on this repository. Notice that when you clone a repository from GitHub, it automatically has that repository listed as `origin` , and you do not have to add the remote the way we did when we did not clone the repository.

Bash

```
$ git remote -v
```

```
origin  https://github.com/YOUR_GITHUB_USERNAME/YOUR_REPOSITORY_NAME.git (fetch)
origin  https://github.com/YOUR_GITHUB_USERNAME/YOUR_REPOSITORY_NAME.git (push)
```

Create the file `testing.txt` in this new directory and make it contain the following.

```
I added this file from a new clone!
```

Now we will commit this new file:

Bash

```
$ git status
```

Output

```
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  testing.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Bash

```
$ git add .
$ git status
```

Output

```
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  new file:   testing.txt
```

Bash

```
$ git commit -m "Adds testing.txt"
$ git log
```

Now push the commit:

Bash

```
$ git push
```

If you check the GitHub page, you should see the testing.txt file.

Now change directories into the original local repository, and check if `testing.txt` is there:

Bash

```
$ cd ../<original clone>
$ ls -l
```

To get the newest commit into this clone, we need to pull from the GitHub repository:

Bash

```
$ git pull origin master
```

Output

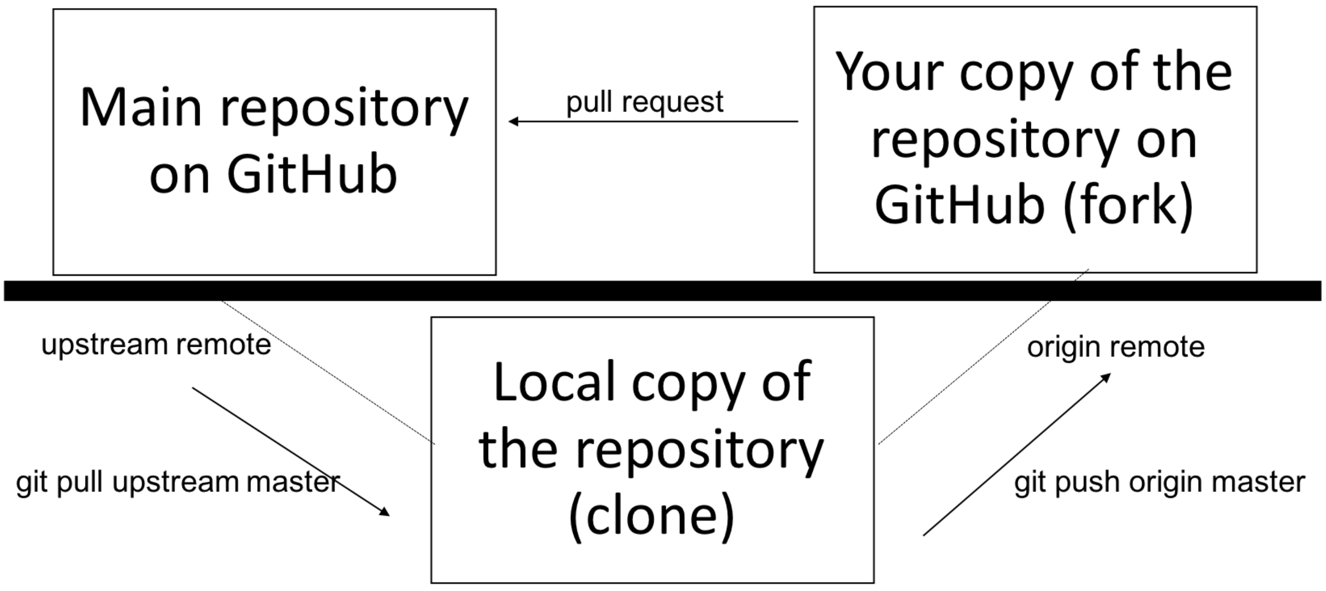
```
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/YOUR_GITHUB_USERNAME/molecool
 * branch            master     -> FETCH_HEAD
   2ac4843..754da2b  master     -> origin/master
Updating 2ac4843..754da2b
Fast-forward
 testing.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 testing.txt
```

Now we can actually see `testing.txt` in our original repository.

Collaborating with others using GitHub

Many software projects, large and small, are hosted on GitHub. If you are working with other people on a project, they may ask you to collaborate and contribute your code contributions through GitHub. To do this, we need to understand a little more the flow of information on GitHub.

As discussed above, a collection of files for a certain project is called a *repository* on GitHub. When you are collaborating with others on a project, there are at least three relevant copies of a repository: the main repository on GitHub, your copy of the repository on GitHub, and the local copy of the repository on your computer.


Main repository on GitHub
pull request
Your copy of the repository on GitHub (fork)
upstream remote
Local copy of the repository (clone)
origin remote
git pull upstream master
git push origin master

Forking a repository

To make your copy of someone else's repository on GitHub, you *fork* their repository.

Find someone else in your class and navigate to the GitHub page of their project. If you are working these lessons on your own, use MolSSI's geometry analysis repository (https://github.com/MolSSI-Education/geometry-analysis-example). In the upper right hand button, click the button that says **Fork**. This will make a copy of the repository on *your GitHub account*.

Cloning a repository

Now you need copy the repository on *your GitHub* to your local computer. This is called a *clone*. Navigate to the GitHub page of *your copy* of the repository on GitHub. Click the green button that says Clone or Download. Copy the link in the box.

Now open a terminal window and navigate to the location where you want the repository to be stored on your local computer.
Type `git clone` and then paste the link you copied.

Bash

```
$ git clone https://github.com/YOUR-USERNAME/REPOSITORY-NAME.git
```

where the username will be your GitHub username and the repository name will be the name of the repository. This command will create a folder with the same name as the repository that will be under git control.

Setting git remotes

Now that you have all the copies of the repository in place, you need to make connections between the copies so you can transfer information. These connections between the copies are called *remotes*. We already discussed remotes a bit above, when we pushed our code to GitHub.

When you clone a repository, it automatically sets up one remote for you called *origin*. To see the remotes for a repository, go to the command line and type

Bash

```
$ git remote -v
```

Output

```
origin  https://github.com/YOUR-USERNAME/REPOSITORY-NAME.git (fetch)
origin  https://github.com/YOUR-USERNAME/REPOSITORY-NAME.git (push)
```

Since we cloned the repository from our copy on GitHub, the *origin* remote is between our copy on GitHub and the local copy on our computer.

We probably also want to connect the repository on our local computer to the main repository on GitHub so that we can get updates that are made to that repository. We need to create a new remote called *upstream* that connects the repository on our local computer to the main repository on GitHub. You can actually name any remote anything you want, but *upstream* is the accepted name for the remote to the original main repository.

To set a new remote, go to GitHub and navigate to the page of the main repository (not your copy, the same page you forked from). Click the green button that says Clone or Download. Copy the link in the box.

Go to the terminal and type `git remote add upstream` and then paste the link you copied.

Bash

```
$ git remote add upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git
```

Now if you use `git remote -v` again to list your remotes, you should see two remotes, origin (to your copy on GitHub) and upstream (to the main repository on GitHub).

Output

```
$ git remote -v
> origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
> origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
> upstream  https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (fetch)
> upstream  https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (push)
```

Information flow using GitHub

The normal workflow when you are working on a collaborative project with git is:

1. Pull changes to the main project from upstream.
2. Work on the project on your local copy of the repository.
3. Commit your changes to your local copy.
4. Push your changes to your copy of the repository on GitHub.
5. Submit your changes to the main developers through a *pull request* so they can consider including them in the main repository.

Getting changes from the main repository

To get changes from the main repository into your local repository, you use the upstream remote and use the `pull` command.

Bash

```
$ git pull upstream master
```

When you enter this command, several things can happen, depending on what changes have been added to the main repository. If the new changes don't conflict with what you have already, then you might see something that says `Fast Forward`. If there are conflicts, you will have a merge conflict that you need to resolve, where you select which version of the file you want.

Pull vs. fetch and merge

The `pull` command is actually a combination of two other git commands `fetch` and then `merge`. The `pull` command is easier because it is just one step, but in complex situations you might want to do a `fetch` so you can see what changes have been made and then decide to `merge` or not.

Using branches to make changes

Within a repository, you can actually create different copies of your code. These are called *branches*. This is particularly useful if you want to work on a new feature, but you want to keep another copy of the code that is working in case you mess something up. In all of the examples above, we have been using the main or *master* branch.

Let's make a branch and make some changes to the classmate's code you forked and cloned. First, make sure you are on your master branch.

Bash

```
$ git status
```

Output

```
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Now, make a new branch for your edits. You can name your branch anything you want, but we will call it edits in this example. Usually you would choose a more informative name, but this is just an example.

Bash

```
$ git branch edits
```

Now we will switch from our master branch to the new edits branch.

Bash

```
$ git checkout edits
```

Output

```
Switched to branch 'edits'
```

Now that we have crated a branch to work on, we can make changes to the files without worrying about messing up anyone else's work.

Exercise

Open the README.md file and make a change to the file.

Solution

Pushing changes to your repository on GitHub

Exercise

Use `git status` to check which files have been modified on this branch. Then use `git add` and `git commit` to commit the changes to the branch.

Solution

Now you can push your changes through the origin remote to your repository on GitHub.

```
git push origin edits
```

This will create a new branch on your GitHub repository called edits and push your changes to that branch. Generally, you can not push your changes upstream into the main repository on GitHub. This is why you need your own fork on GitHub; you can always push to something that belongs to you, but not something that belongs to someone else.

Being a Collaborator

There are ways that you can push upstream to the main repository. The owner of a repository can add you as a *collborator* and give you permission to push changes. For small projects with only a few people, this is fairly common. For large projects with 100's of developers, no one does this, and you must submit your changes through a *pull request* as described below.

Notice that in this flow of information, if you want to pull new changes from the main repository into your repository on GitHub, there is no remote that directly connects those two. To accomplish this, you would do a `pull` from the main repository to the copy on your local computer through the upstream remote and then do a `push` from your local copy to your copy on GitHub through the origin remote.

Submitting your changes to the main developers

Once you have pushed changes to your repository on GitHub, you may want to tell the main developers about your changes and ask them to consider merging them into the main repository. This is called making a *pull request*. Essentially, you are suggesting to the developers that they pull changes from your repository into their repository. To make a pull request, navigate to the GitHub page of the main repository (your classmate's repository or the MolSSI geometry analysis example repository, whichever you are using) and click on Pull Request in the navigation bar across the top. On the next page, click the green button that says New Pull Request.

On the next page, in the left box that says *base*, the main repository should be listed. If it is not, use the dropdown menu to select it. In the right box that says *compare*, use the dropdown menu to select the edits branch of your repository. If you don't see it in the list, then you need to click the blue link above the boxes that says *compare across forks*. Click *Create Pull Request*.

On the next page, you will type a message to the developers telling them what changes you have made. You can also suggest reviewers who you want to check out your changes. When you are finished, click *Create Pull Request*.

The developers will receive a notification of your pull request and then they can review your changes and decide if they are going to merge them into the main repository. You will receive a notification letting you know if/when the pull request has been merged.

Exercise

Go to the GitHub page of *your* geometry analysis project. See if a classmate has submitted any changes through a pull request, review their changes, and merge the changes into your master branch.

Solution

Key Points

- Putting your code on GitHub is the best way to easily share your code, collaborate, and track changes.

(../09-version-control/index.html)

(../)