

GRAD 778 - Elements of Research Computing

Module 11 - Version Control

November 20, 2021

Instructor: Jonathan Greenberg (jgreenberg@unr.edu)

Contributors: Gunner Stone and Akshay Krishna

Overview:

Introduction to using the version control system git. Git is one of the most popular version control systems. It is used to help teams collaborate on a shared codebase. Students will learn basic git workflows, including the basics (push, pull), branches, merges, and integrating with GitHub.

Zoom: <https://unr.zoom.us/j/85258652647>

Table of Contents

Chapter 1: Introduction	1
Prerequisites:	1
Key websites	2
What is version control?	2
Chapter 2: Introduction to Version Control and GitHub	3
Step 1: Create your first repository	3
Step 2: Branching	5
Step 3: Making and committing changes	5
Step 4: Pull requests	6
Chapter 3: Introduction to Git	7
Step 1: Configuring git	7
Step 2: Create a repo using the Git CLI ("Command Line Interface")	8
Step 3: Add files to your repo and commit them.	9
Step 4: Cloning an existing repository.	9
Step 5: Forking	10
Step 6: Pull changes.	11
Step 7: Exploring and reverting to previous commits.	12
Chapter 4: Conclusions and Being a Coder of the World	13

Chapter 1: Introduction

Prerequisites

Please get a github account from: <https://github.com/>

Windows:

Install GIT: <https://git-scm.com/downloads> (use installation defaults)

Install GitHub CLI: <https://github.com/cli/cli/releases/tag/v2.2.0>

- You probably want the "gh_2.2.0_windows_amd64.msi" version

Mac:

We are going to do this using "HomeBrew" to do our installations. If you've already installed HomeBrew previously, you don't need to do this step.

Please first install HomeBrew by opening Terminal and:

```
/bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

If it asks you for a password, this is your local Mac password.

Next, update Homebrew:

```
brew update
```

If you get an error, try running these lines:

```
git -C /usr/local/Homebrew/Library/Taps/homebrew/homebrew-core fetch  
--unshallow  
git -C /usr/local/Homebrew/Library/Taps/homebrew/homebrew-cask fetch  
--unshallow  
brew update
```

Now install git:

```
brew install git
```

And finally the GitHub CLI:

```
brew install gh
```

Other OSs: Please check

<https://github.com/cli/cli#installation>

Key websites

- This tutorial is heavily cribbed from <https://docs.github.com/en/get-started/quickstart/hello-world>
- Git download: <https://git-scm.com/downloads>
- GitHub CLI: <https://github.com/cli/cli/releases/tag/v2.2.0>

What is version control?

Scene:

Have you ever worked on a collaborative paper before, where you end up doing the following:

myawesomepaper.docx

Which you then send to three people, "A", "B" and "C". A and B both give you comments and return the documents as:

myawesomepaper_A.docx
myawesomepaper_personB.docx

You then need to incorporate changes and re-send the paper out to person A, but you want to make sure the document appears different, so you send out:

myawesomepaper_v02.docx

But meanwhile person C finally gets around to sending you comments, so you now have:

myawesomepaper_c.docx

and then person A sends you back their edits:

myawesomepaper_v02_A.docx

Which you need to now merge with the edits person C gave you... But wait, what version/edit was that? Did you already change the file so much that the edits from C are totally irrelevant now? AIGH.

End scene.

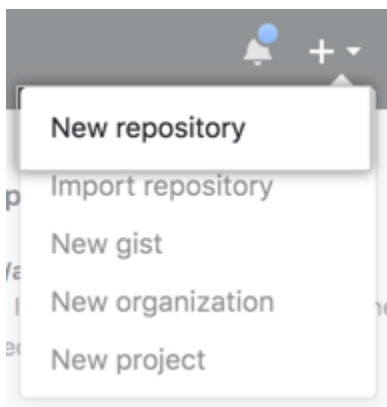
Version control are the methods by which you can cleanly make and track changes to your code and help collaborate on projects, create new versions of software based on an existing code base, handle conflicts and merge different solutions to the same problem.

Chapter 2: Introduction to Version Control and GitHub

Step 1: Create your first repository

A "repository" (aka "repo") is the organizing unit of a project. The repo typically contains all the files and folders your project needs. These repos are hosted both locally (on your computer) and in the cloud (e.g. on github.com). We are going to create a remote repo first.

1. Sign in to github.com.
2. At the top right, create a new repository:





3. The "owner" should be your username, the repository name: "grad778-f21-w11". In the description, write "GRAD 778 Workshop 11 Repo", let it be public, but click "Add a README file". Leave everything else alone.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Owner * **Repository name ***


 jgm307 / grad778-f21-w11 

Great repository names are short and memorable. Need inspiration? How about [psychic-enigma?](#)

Description (optional)

GRAD 778 Workshop 11 Repo

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.


☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

Create repository

4. Click "Create repository".
5. Congrats! You have made your first (remote) repo! Make sure you properly created a "README" by clicking "Code" at the top of the next screen. You should see "README.md".
6. (Optional) If you don't see the README.md file, you probably forgot to click "Add a README file". Please go to Settings -> Options -> Delete this repository. Follow the instructions to delete the repo (you will have to type in the repo name) and repeat steps 2-5.

Step 2: Branching

When coding, you sometimes want to create a new set of functions but retain a previous version (e.g. one you KNOW works correctly) until you are sure your new additions work. We use a concept of "branching" to work on new functions/features, and once we are satisfied with the new features, we can merge the branch back into the "main" trunk.

In our example above, the edits from A, B and C would be our branches.

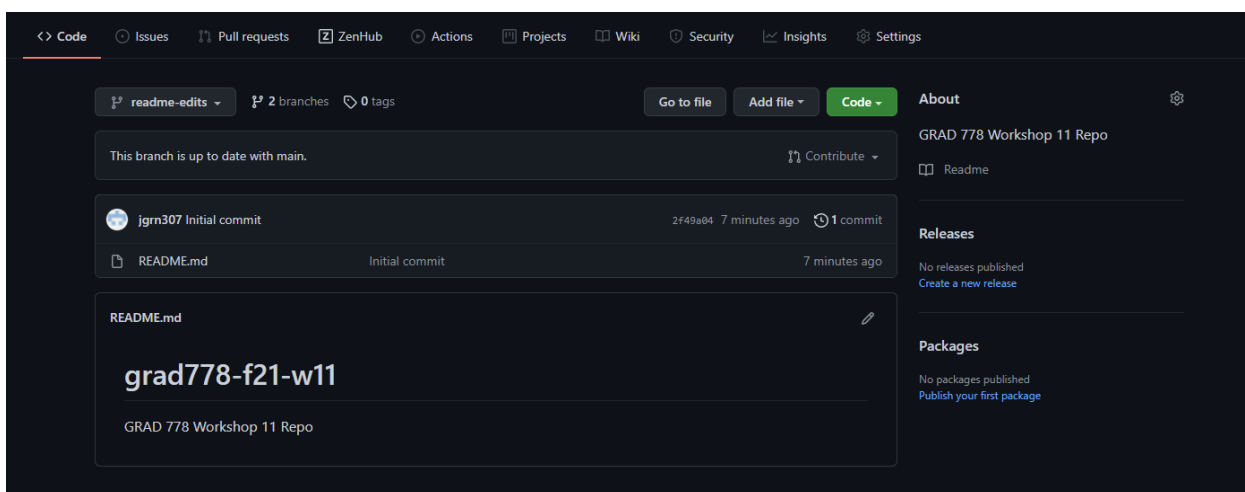
As a standard, we name our primary branch "**main**". Branches off of **main** can take any name you feel they need. Let's make some new branches:

1. Click "Code" at the top of your repo.
2. You will see "main" at the top left(ish). Click that, and in the branch name type "readme-edits" and click "Create branch: readme-edits from 'main'". At this point you have created a new branch, but they are identical (since you haven't changed anything).

Step 3: Making and committing changes

We are going to edit the README.md file that was automatically created for you when you made the repo. Once we do this, we are going to "Commit" the changes which means we are going to create a new version of our file (while preserving the original one). Usually you will edit code in your own developer environment, but we can do simple edits on the github website for now.

1. Click "Code".
2. Make sure you are in the "readme-edits" branch shown at the top left. If you are still in "main", drop that down and select the "readme-edits" branch:



3. Click the README.md filename, then click the "Edit this file" icon (the little pencil).

4. Write a quick sentence about yourself, and write who your favorite superhero is (if you have one). Quick note: this file is public! Anyone on the internet can see it! So nothing too personal :)
5. Once you are done, you are going to save/commit changes (simultaneously for now, when we start doing real coding these will be separate steps). Write something helpful (but short, under 50 characters) in the Commit changes boxes, like "README.md edit to tell the world about myself". Click "Commit changes".
6. Congrats, you have committed your first change in git! Let's check what this did. First, go back to "Code" at the top left.
7. Confirm you are still in the "readme-edits" branch, and click the README.md file. Check to make sure it has your changes.
8. Now change the branch back to "main". Check the README.md file again and you should see the original file! You have preserved both versions of the file without having to do weird naming/organization things.

Step 4: Pull requests

Pull requests are used to merge changes back into the main branch so you don't end up with wildly diverging versions. They are often used by teams of programmers who are tackling different aspects of a code base, but they can also be used by individual programmers to keep track of their own changes over time. We are going to create a "pull request" to merge our new README.md into the main branch and then merge the pull request into our main branch.

1. At the top of your repo, click "Pull requests".
2. Create a "New pull request".
3. At the top, set the "base" to be "main" and the "compare" to be "readme-edits".
4. Take a look at the comparison. It should show additions and subtractions from the original README.md.
5. Click "Create pull request".
6. Title your pull request, e.g. "Dr. Greenberg's proposed changes to the README.md" and write a description of the changes, e.g. "Wrote a quick sentence about myself and wrote who my favorite superhero is." Note this pull request is basically a mini-website editor: you can add formatting, images, etc. to it if need be.
7. Click "Create pull request".
8. If you were collaborating, your collaborators could examine and test your proposed edits, make suggestions (potentially "rejecting" the merge).
9. Now we'll merge the changes. Start by clicking the "Pull requests" at the top. You will see your pull request show up. Click it.
10. Click "Merge pull request", and then "Confirm merge".
11. Now that a merge has occurred, it is good practice to delete the branch. You can always create a fresh branch for new features. Do this now, "Delete branch".

Chapter 3: Introduction to Git

In the previous chapter, we did all of our version control via a website. This isn't how we will typically interact with version control. We will typically develop code on our personal computers, test it in a variety of different ways, and use a command line version control system known as "Git" to perform all of the version control steps. There are other version control systems out there, including subversion ("SVN").

Step 1: Configuring git

Before you came to class, you should have installed Git on your Windows or Mac. We are going to get it configured properly first.

1. On Windows, launch "Git CMD". On Mac open your terminal.
2. We are going to set our username. In the command prompt, type the following (replace "Your Name" with your actual first and last name -- KEEP THE QUOTATION MARKS:

```
git config --global user.name "Your Name"
```

3. Confirm your name was set properly:

```
git config --global user.name
```

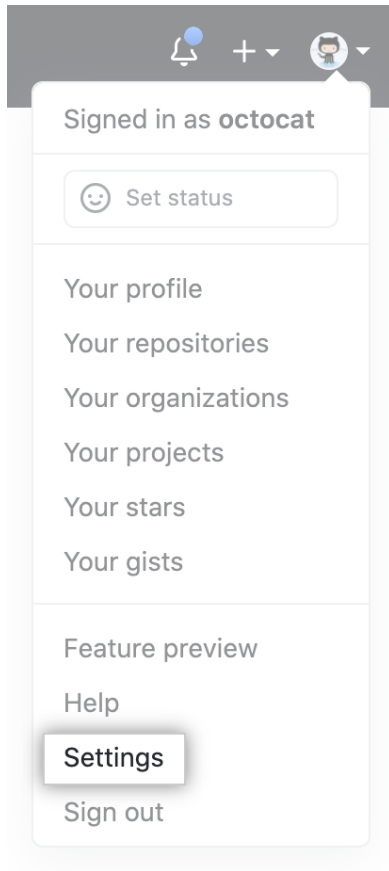
4. Next, we want to set our commit email address. We will do this in two locations, via command line Git and via the GitHub website. First, command line git (replace "email@example.com" with your own email address, keeping the quotes):

```
git config --global user.email "email@example.com"
```

5. Confirm with:

```
git config --global user.email
```

6. On the GitHub website, click the top left icon -> Settings.



7. Click Emails. In "Add email address" enter the same email address you used in the previous steps (the command line Git steps), and set it as your primary email address. You may need to verify the email address.
8. We will now setup authentication with the GitHub website. In Windows, load Git CMD. Mac load the Terminal. Type:

```
gh auth login
```

9. Hit "Enter" to select GitHub.com.
10. Select "HTTPS".
11. When asked if you want to "Authenticate Git with your GitHub credentials? (Y/n)" type "Y".
12. Select "Login with a web browser". It will ask you to copy a one-time code then "Press Enter" to open github.com in your browser..." Do so, type in the code, and then authorize github.

Step 2: Create a repo using the Git CLI ("Command Line Interface")

We are going to repeat the process we did in the first chapter but entirely using the command line.

1. In Windows, make sure you are using the Git CMD window. Mac use the terminal.
2. Create a new repo "grad778-f21-w11-cli":
`gh repo create grad778-f21-w11-cli`
3. Make it Public. When asked, say "N" about creating a .gitignore and license. Then select Y when it asks if you want to create it.
4. When it asks to create a local project directory, also select "Y". Make a note of where those files are stored. We'll need this for the next step.

Step 3: Add files to your repo and commit them.

1. Open up your favorite text editor (e.g. Wordpad or TextEdit) and create a new file. Type in some info about who you are and what department you are in, and who your favorite supervillain is. Save the file as "README.md" to the local git folder you made in the previous step. Make sure the file saves as a ".txt" file.
2. In your command line, navigate to the top level of your git folder e.g.:

```
cd C:/somepathto/grad778-f21-w11-cli
```

3. To see if there are new files (there should be, the new README.md) type:

```
git status
```

4. You should see:

```
Untracked files:
```

```
(use "git add ..." to include in what will be committed)
```

```
README.md
```

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

5. We are now going to "stage" and commit the file. Note that this commits to your LOCAL directory, not to github (yet).

```
git add README.md && git commit -m "Add README using the CLI"
```

6. Now we are going to PUSH the changes to a specific branch on GitHub:

```
git push --set-upstream origin HEAD
```

7. If you navigate to the repo on GitHub.com, you will see your new file added to the main branch!

Step 4: Cloning an existing repository.

In the step before, we created a new repo from scratch, but what if we want to grab a repo that already exists? We are also going to make our workspace a bit cleaner and choose where the git repos live.

1. Go ahead and make a folder somewhere in your user directory called "git". We are going to store all our repos there, from here on out.

2. In Git CMD, let's change our directory to this new folder. On my computer it is:

```
cd C:\Users\gears\git
```

3. Now, we are going to clone the FIRST repo we made from the website. Let's figure out the right way to call it. Navigate to github.com. At the top right click "Your repositories". Click on "grad778-f21-w11".

4. Click Code -> and then look for the green "Code" in the middleish of the screen. Click it. You should see a small window that says "Clone". Make sure it is set to HTTPS, and copy the URL you see inside of it. It should be something like

```
"https://github.com/[username]/grad778-f21-w11.git"
```

```
git clone https://github.com/[username]/grad778-f21-w11.git
```

5. Once it's done, type "dir" (Windows) or "ls" (Mac) to see the directory contents. You will see a new directory with the repo name. Enter that directory with `cd` and look at the contents. You should see your README.md there!

6. Since you own this repo, let's make a quick change to the README.md with a text editor, save it, and commit it back:

```
git add README.md && git commit -m "Mod README from cloned repo"
```

```
git push --set-upstream origin HEAD
```

7. Check the github.com website (remember you are looking at the original "grad778-f21-w11" repo, not the "grad778-f21-w11-cli" one). Check to make sure your change to README.md came through!

Step 5: Forking

Forking is used to either propose code changes to an existing repo, or to use someone else's repo as the basis for your own work. A fork is a bit like a branch, except you are creating a brand new repo (not just a branch within a repo). That new repo is connected, in some ways, to the original repo. You are going to fork a repo I created within an "organization" (the repo is not under my username), and then add your favorite superhero to a subfolder. The repo can be found at: <https://github.com/unr-grad778/grad778-f21-w11-bestsupe>

1. First, make sure you are at the top level of your "git" folder you created in the previous step.

2. Fork the repo:

```
gh repo fork
```

```
https://github.com/unr-grad778/grad778-f21-w11-bestsupe.git
```

3. When it asks if you want to clone it, say "Y". This will put a copy on your local machine.

4. Check your repos. You should now see a new repo with a note "Forked from unr-grad778/grad778-f21-w11-bestsupe".

5. Now use whichever technique you'd like to create a subdirectory called "candidates" within your local repo, then create a text file with your favorite superhero name + .txt (e.g. "gnort.txt") and within the text file, write one sentence on why you like that superhero.

6. Commit and push the new file to your repo. Hint, you can use a wildcard ("*") with git add to add all files/folders. I won't hold your hand this time. Refer to previous steps to get this working.
7. Now, let's propose your candidates as a change to the original repo (mine)! Using the web browser, go to your new github repo that is the fork. In the middle of the screen, it says "Contribute"- click that and "Open Pull Request".
8. Create a pull request, and in the title of the pull request include the name of your superhero. Copy/paste your reason into the box, and select Create pull request.
9. Once everyone is done, as the original repo owner I will show you how to accept (and reject) changes.

Step 6: Pull changes.

Git does NOT auto-update with changes from the remote server. To update your local repository with the latest changes on the remote repository, you need to pull changes from the remote server. We need to understand two situations -- one without conflicts and one with conflicts.

1. To begin with, let's clone the "grad778-f21-w11-cli" to our official git directory (you probably have this folder someplace else on your hard drive. You can safely delete the other copy now). Switch to your git folder in the command line and do a clone of grad778-f21-w11-cli.
2. Use the github website to edit the README.md from your grad778-f21-w11-cli repo -- maybe include your favorite food in the README.md. Don't forget to commit the change (with an appropriate title) before proceeding. At this point, the website will have a newer version of README.md than you have on your local drive. Confirm by looking at the README.md on the website and in your git/grad778-f21-w11-cli folder. CLOSE YOUR LOCAL README.md FILE BEFORE PROCEEDING.
3. Let's now update our local copy with the one on github using git pull. First, in your command line, change into the git/grad778-f21-w11-cli folder

```
git pull https://github.com/[username]/grad778-f21-w11-cli.git
```
4. Now, check your local copy -- it should now be updated to the one on the website. Yay! But...
5. What if we have a conflict? Let's do two new edits, one on the github.com website, and a DIFFERENT edit on the local copy. On github.com, edit the README.md to include your favorite color. On your local copy, edit it to include whether you prefer coffee, tea, or neither.
6. Commit your local copy, but don't push it yet:

```
git add README.md && git commit -m "Favorite drink"
```
7. Let's pause and think about this for a minute. Both our remote and local copies are one version later than the most recent pull. So what happens if we try to push or pull them? Let's find out. First pull from the remote server. Next, push to the remote server. Nope! At this point, git creates a version of the conflicted file that contains information on what the conflict is. Don't worry, you can get your original file back if you need to, but let's resolve the conflicts. Open the README.md file and you'll notice some new text in

there. "<<<<<< HEAD" is the beginning of the conflict, "======" divides the conflict between the two files, above is what is on the remote server, and ">>>>>> [a bunch of letters and numbers]" is the local server.

8. Edit this file locally, remove the conflict info but make it so both the favorite color and drink are both in the same file, and save it. Try recommitting and pushing the local file, now that the conflict has been fixed:

```
git add README.md && git commit -m "Merge drink and color"
git push --set-upstream origin HEAD
```

9. Check github.com and see if the merged file is now there.

Step 7: Exploring and reverting to previous commits.

Remember: every version of your code that has been committed is still available in github. You can explore older commits and even restore them if you aren't thrilled with the changes. Say you decided you didn't want to hear what people liked to drink and what their favorite color was. We want to find and then revert to the README.rd that preceded that.

1. Navigate to your "grad778-f21-w11-cli" repo on github.com. In the middleish, you should see a clock with an arrow next to "5 commits" (or something like that). Click the clock.
2. You will now see every commit you have done! Let's back up to the commit that had the Favorite food update. Click the description of that update. You should see some information on it, and also a "version" next to where it says X comments on commit -- mine said "c251437". Make a note of this code.
3. We now have two choices -- we can blow away all changes since that commit (probably not the best idea), or we can create a new branch based on this older version. Let's do the latter:

```
git checkout -b old-version-branch [revision_id]
```

4. Check your local folder and README.md -- you should see the older version of that file appear (the folder will contain whichever is the current branch). Close the file.
5. Let's switch back to our main branch:

```
git checkout master
```

6. Re-open README.md. Note you have the later version (since we now have two branches).
7. At this point, we could merge the README.md file, but let's just delete the branch we just created. List all branches first (notice the "active" branch is highlighted):

```
git branch
```

8. Now delete the branch:

```
git branch -d old-version-branch
```

9. Just for good measure, commit and push all changes back to the server to make sure both local and remote are in sync. In theory, you shouldn't have anything to commit or push.

Chapter 4: Conclusions and Being a Coder of the World

Collaborative coding and version control go hand-in-hand. Whether your project is open source (public) or closed source (private), the same principles are at play. After the initial code has been committed and pushed, any future changes should often first be created as a branch. This preserves the "main" (default) branch, and allows you and/or collaborators/testers to review your changes before a merge occurs. Commits should have descriptive names so you can quickly find a branch with a specific change that occurred (i.e. don't use "new branch" or something equally useless to name the branch).

Once your code is public, be aware people can post "Issues". These are bug and feature requests typically. If you go to your repo, you can create your own issues to help you track things that need to be fixed. Try this now! On other people's repos, if you use someone's code a lot and find a bug, let them know! Many repos will have "rules" for submitting bug and feature requests, so make sure you follow those.

Exercise: please submit the github repo website you created as your assignment. It should look like:

`https://github.com/\[username\]/grad778-f21-w11-cli`