CPE-2600 **FALL 2024**

Systems Programming

Lab 6 Git and GitHub

Topical Concepts

Source Version Control

The concept of tracking source code is simple enough. Over the lifetime of a software project, source code files will change frequently as features are added, tested, debugged and released. Without a tool of some sort to manage the changes to the source files, you invariably end up with discrete copies of the entire set of source files for each milestone. Not only is this inefficient storage, but it is difficult to track particular features or changes. It can also be easy to experience “regression” where features or fixes are made in one set of project source files but not in others. This approach can be made to work for individual developers working in isolation, but in a team environment, failure is inevitable.

Enter version control systems. Many systems exist that range from simple programs that create “versioned” copies of the controlled files within the same directory or subdirectory suitable for a single developer or very small team to elegant systems that might present a particular version of a set of source code as separate networked file systems and can support hundreds of individual developers. These systems exist as free and open source to proprietary systems costing thousands of dollars per developer.

There are three general categories of version control systems. The simplest might be considered “local data model” in that the revision information is stored in the local filesystem – albeit, this could be shareable on a multiuser system.

The next category might be considered “client-server” where the server maintains the source files and version information, aka the repository. A client then requests access to particular files of particular versions and is able to edit and update only those files. The server may control access so that only one developer may be able to edit a particular file at a time and hence developers are frequently requesting and releasing write access to files as they edit a project.

The most modern approach is known as “distributed.” In a distributed system, a server maintains the repository, much like the “client-server” model, but when a developer wishes to work with the project, the developer will create a local copy of the entire repository. This allows each developer to work in isolation and gives each developer local access to all version information.

Regardless of category, all version control system have to deal with updates to the repository. If the updates were completed by a single developer, no problem – simply update the repository with the changes and bump the version. The biggest issue arises when multiple developers make changes to the same file at the same time. When this occurs, a manual “merge” may be required. This is an advanced topic we will not address at this time.

Git

Interestingly, git has evolved to be the distributed VCS tool of choice. It is free and open-source, highly reliable, and trusted. Given these properties, git has also evolved to be the foundation for various online services that offer repository hosting services, notably GitHub, GitLab, BitBucket, and others. As a distributed system, it can copy (clone) remote repositories to the local system for development, and then update the remote repository (push) when work is completed. Interestingly, a remote repository is not needed at all if you are not sharing code with other developers. You can create a local repository for personal code management using a similar workflow.

GitHub Classroom

GitHub Classroom is a feature of GitHub that allows an instructor to create a “template” repository (with starter code and/or instructions). When a student starts the assignment, a new remote repository is actually created (not the usual GitHub behavior) which is a copy of the template assignment, then, that new remote repository is clones to the student’s local system with the normal Git workflow. The new remote repository that is created is owned by the instructor and the student is made a developer. So, the instructor can now easily view source updates that are committed to the remote repository.

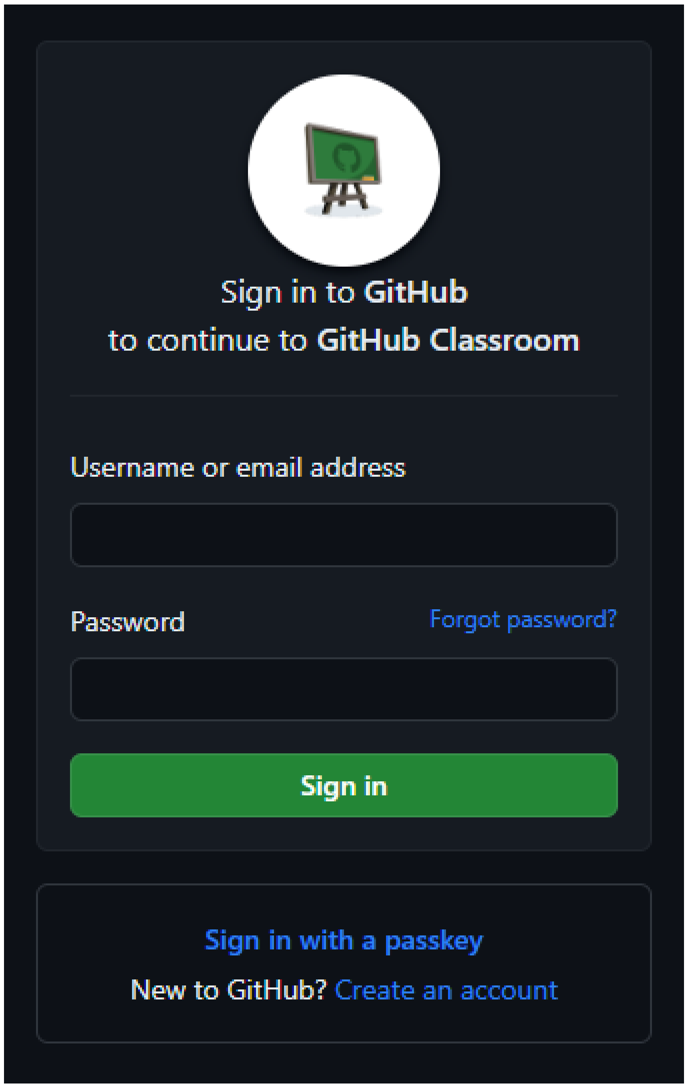
The Exercise

Overview

The goal of this exercise will be to create a GitHub repository (via GitHub classroom) and add your current project source code. This exercise will be largely prescriptive – simply follow each step and make observations along the way.

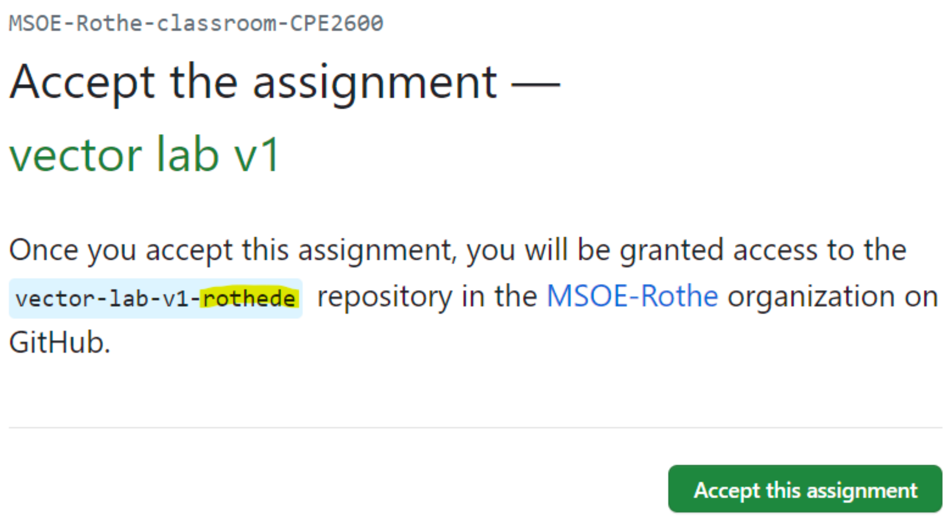
Step 1 – Sign in to GitHub and start assignment.

Go to this link: https://classroom.github.com/a/HPnQGMuD


Sign in to **GitHub**
to continue to **GitHub Classroom**
Username or email address
Password
Forgot password?
Sign in
Sign in with a passkey
New to GitHub? Create an account

If you have an existing GitHub account you may use that and login. If not, create a new account. In either case, ensure your username is the same as or similar to your MSOE username otherwise I will not know who's repository I am looking at.

Once logged in, you should be greeted with this screen:



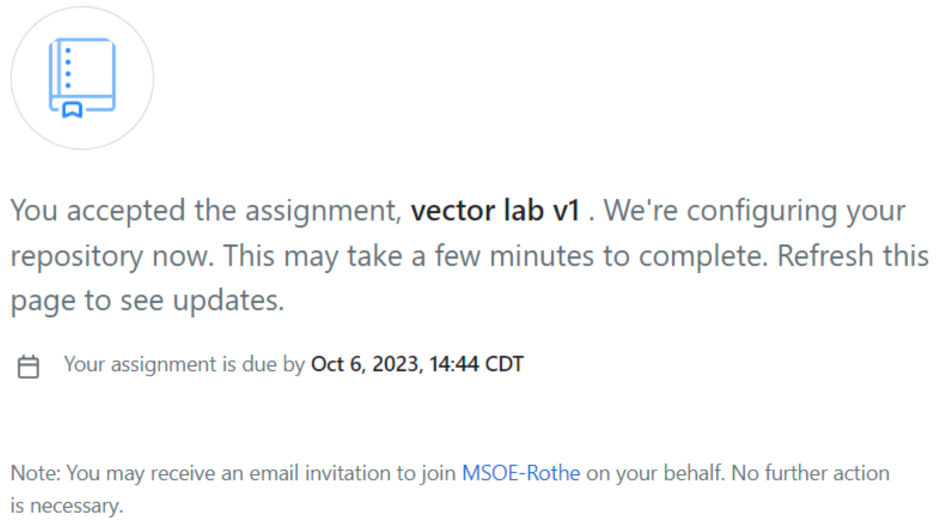
MSOE-Rothe-classroom-CPE2600

Accept the assignment — vector lab v1

Once you accept this assignment, you will be granted access to the `vector-lab-v1-rothede` repository in the MSOE-Rothe organization on GitHub.

except the repository should have your username at the end (instead of mine – highlighted in yellow).

Then you may see this:

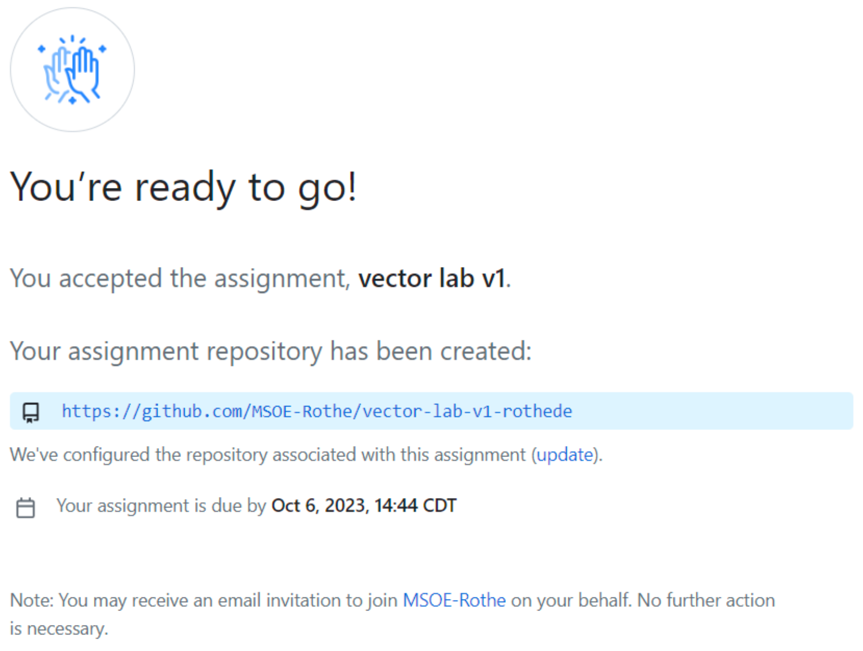


You accepted the assignment, **vector lab v1** . We're configuring your repository now. This may take a few minutes to complete. Refresh this page to see updates.

Your assignment is due by **Oct 6, 2023, 14:44 CDT**

Note: You may receive an email invitation to join MSOE-Rothe on your behalf. No further action is necessary.

and finally, if you refresh the page after a bit, you should see this:


You're ready to go!

You accepted the assignment, **vector lab v1**.

Your assignment repository has been created:

https://github.com/MSOE-Rothe/vector-lab-v1-rothede

We've configured the repository associated with this assignment (update).

Your assignment is due by **Oct 6, 2023, 14:44 CDT**

Note: You may receive an email invitation to join MSOE-Rothe on your behalf. No further action is necessary.

Again, your username will be part of the repository name (where rothede is). This has only created the repository at GitHub, and you can check it out via the GitHub web interface if you wish. The template project was empty, so there will be no files (except perhaps a README.md file).

Step 2 – Setup GitHub Command Line Interface

GitHub has adopted some strict policies for accessing repositories in terms of authentication. Unfortunately, this means that your GitHub credentials cannot be used directly from the git command line utilities. There are a number of authentication mechanisms but they all require some setup. This setup seems OK…

First, install the GitHub CLI (GitHub, not git – we already have git). The package is not in a normal Ubuntu package repo, so follow instructions here: https://github.com/cli/cli/blob/trunk/docs/install_linux.md

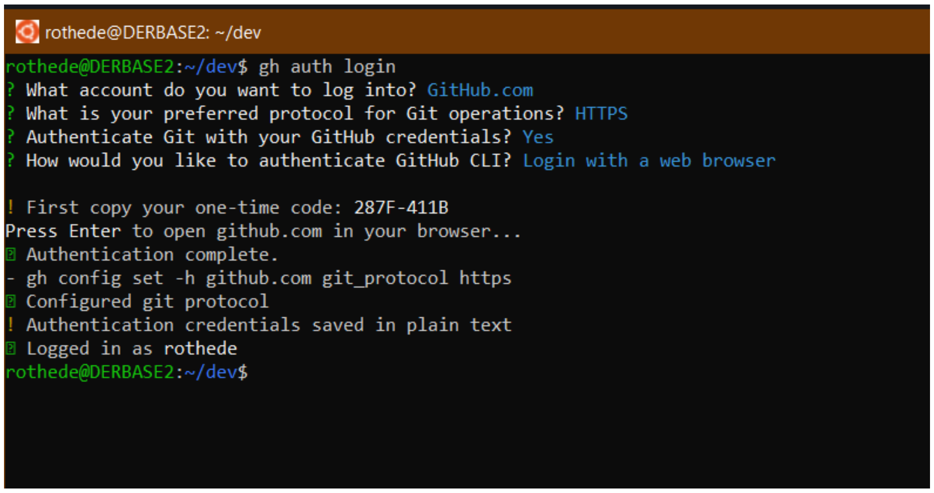
These instructions include the following command:

```
type -p curl >/dev/null || (sudo apt update && sudo apt install curl -y)
curl -fsSL https://cli.github.com/packages/githubcli-archive-keyring.gpg | sudo dd
of=/usr/share/keyrings/githubcli-archive-keyring.gpg \
&& sudo chmod go+r /usr/share/keyrings/githubcli-archive-keyring.gpg \
&& echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/githubcli-archive-
keyring.gpg] https://cli.github.com/packages stable main" | sudo tee
/etc/apt/sources.list.d/github-cli.list > /dev/null \
&& sudo apt update \
&& sudo apt install gh -y
```

Step 3 – Authenticate

Now, you can authenticate with GitHub using the utility 'gh' you will need to do this whenever you wish to use git command line commands to interact with the remote repository.

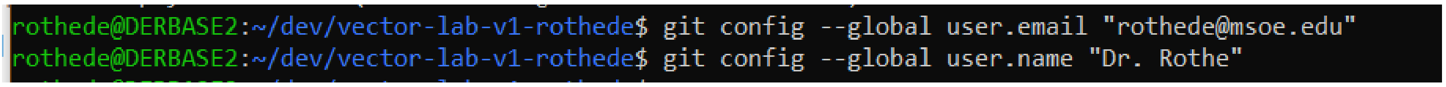
The following shows me authenticating with GitHub:



```
rothede@DERBASE2:~/dev$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: 287F-411B
Press Enter to open github.com in your browser...
 Authentication complete.
- gh config set -h github.com git_protocol https
 Configured git protocol
! Authentication credentials saved in plain text
 Logged in as rothede
rothede@DERBASE2:~/dev$
```

note that this did pop a web browser window to complete authentication for this session.

You will also need to configure git with your email and a username which it will use to tag commits. Do this with 'git config' commands, as follows:

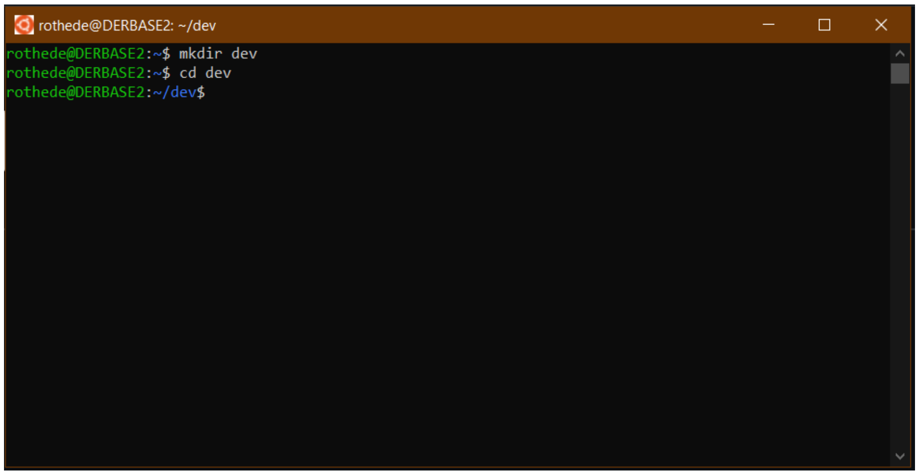


```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git config --global user.email "rothede@msoe.edu"
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git config --global user.name "Dr. Rothe"
```

Step 4 – Clone the Repo

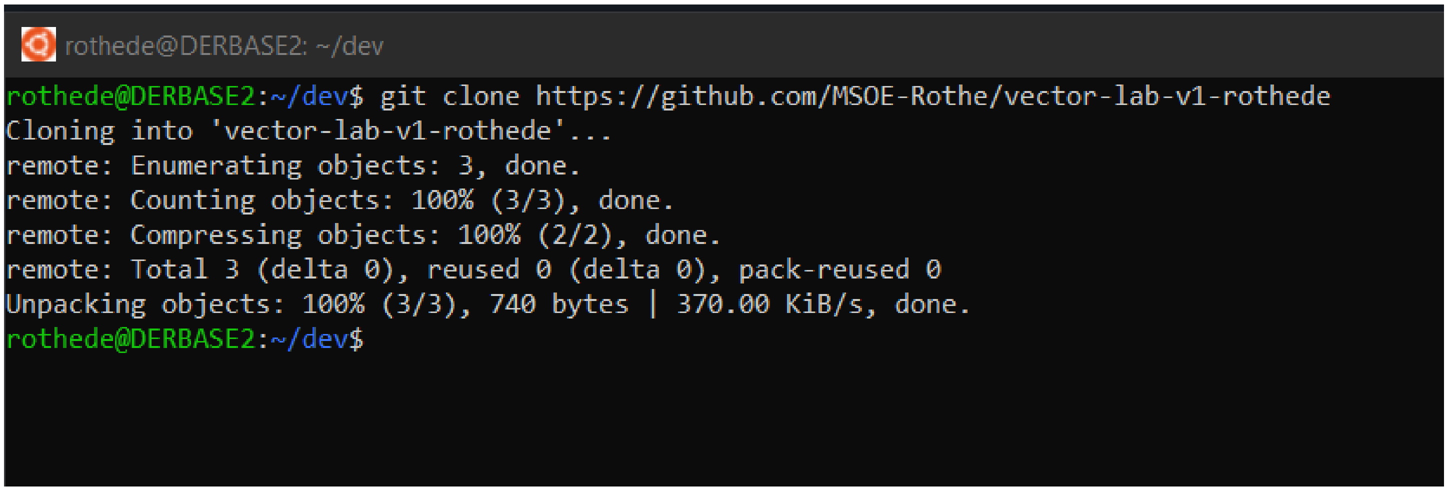
We want to populate the repository, and there are a number of ways to do that. We could actually upload files through the web interface, but that is not a very productive prospect. Instead, we will use the git command line interface to clone the remote repository to your local machine, add files to the local repository, and then eventually push the new files back to GitHub. This sounds more complicated, but it more accurately represents a normal workflow when you are developing.

So, run WSL/Ubuntu. From the command line, create and/or change to whatever directory in which you wish to locate the local copy of the repository. Note that cloning the repo will create another directory, so you can be in a generic (non-project specific) directory. Here I am ready to start:



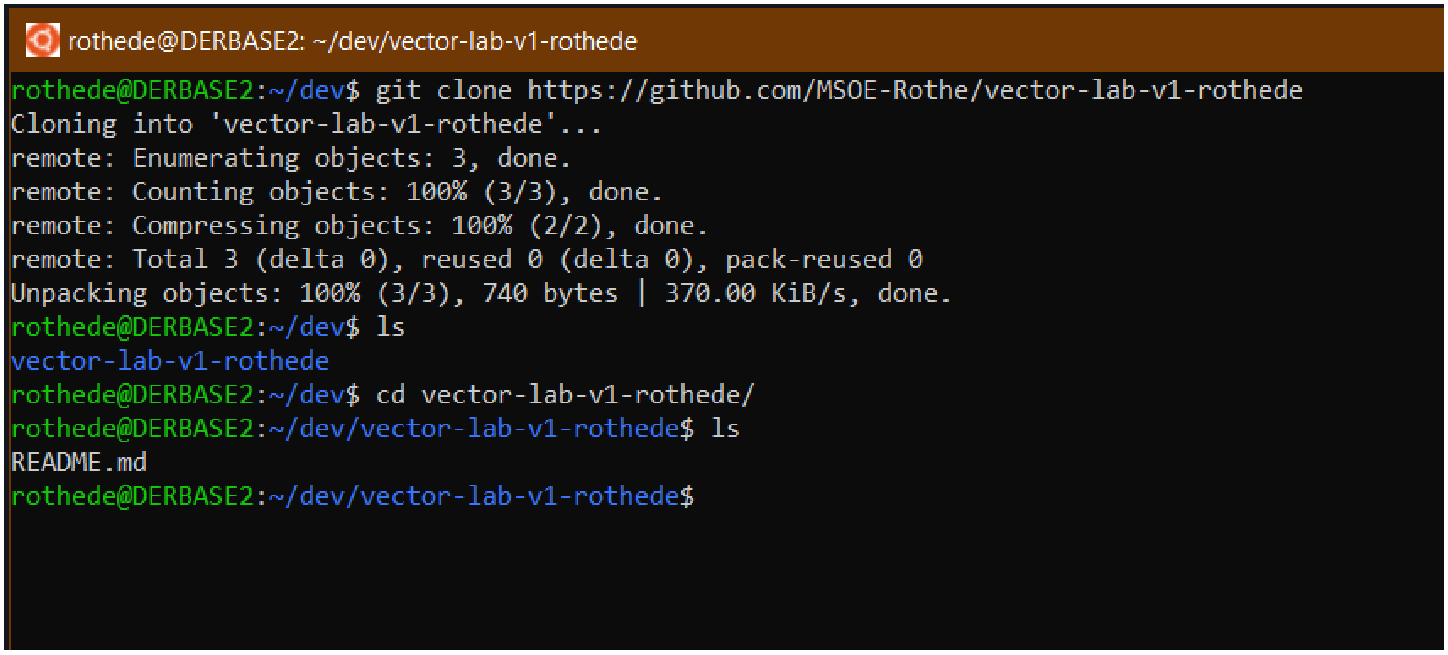
```
rothede@DERBASE2:~$ mkdir dev
rothede@DERBASE2:~$ cd dev
rothede@DERBASE2:~/dev$
```

Now issue the command: 'git clone https://github.com/MSOE-CPE2600/vector-lab-turney-rothede' where rothede at the end is replaced with your username. You should see something to the effect:



```
rothede@DERBASE2: ~/dev
rothede@DERBASE2:~/dev$ git clone https://github.com/MSOE-Rothe/vector-lab-v1-rothede
Cloning into 'vector-lab-v1-rothede'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 740 bytes | 370.00 KiB/s, done.
rothede@DERBASE2:~/dev$
```

Note that there is a new directory with the name of repository. Moving into that directory.



```
rothede@DERBASE2: ~/dev/vector-lab-v1-rothede
rothede@DERBASE2:~/dev$ git clone https://github.com/MSOE-Rothe/vector-lab-v1-rothede
Cloning into 'vector-lab-v1-rothede'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 740 bytes | 370.00 KiB/s, done.
rothede@DERBASE2:~/dev$ ls
vector-lab-v1-rothede
rothede@DERBASE2:~/dev$ cd vector-lab-v1-rothede/
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ ls
README.md
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$
```

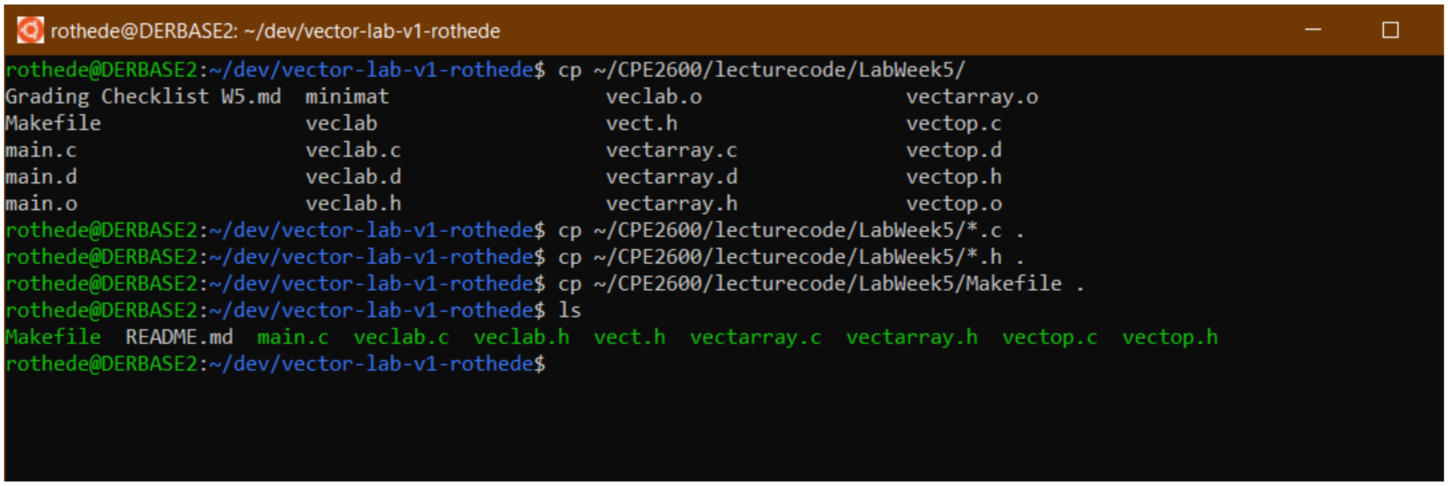
This should match what you saw in the repository through the web browser interface. This is now the working directory for the repository.

Step 5 – Populating the Repository

We can now add our source code to the working directory. Copy all of your source file to this working directory. I suggest copying files and not moving them so you can try again if something get messed up. Eventually you will likely get rid of the original files once under repository control.

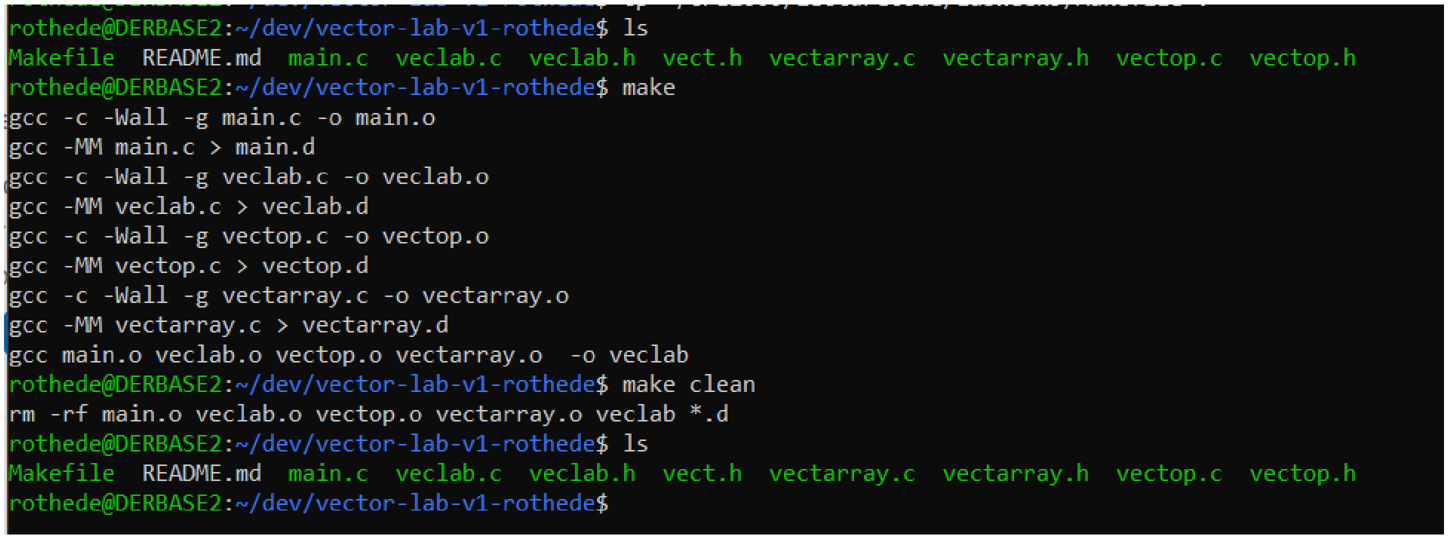
If you are literally starting a project from scratch, you would create the new files here. Another time to start the repository is when you have some working code. In either case, the process is about the same.

I am going to create a repository with the initial version of my "vector lab" project. I need all of the source files (.c), header files (.h) and the Makefile. Of course, if there were any other dependencies that need to be tracked (data files, README files, etc) they should be copied over too.



```
rothede@DERBASE2: ~/dev/vector-lab-v1-rothede
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ cp ~/CPE2600/lecturecode/LabWeek5/
Grading Checklist W5.md  minimat                  veclab.o                 vectarray.o
Makefile                 veclab                   vect.h                   vectop.c
main.c                   veclab.c                 vectarray.c              vectop.d
main.d                   veclab.d                 vectarray.d              vectop.h
main.o                   veclab.h                 vectarray.h              vectop.o
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ cp ~/CPE2600/lecturecode/LabWeek5/*.c .
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ cp ~/CPE2600/lecturecode/LabWeek5/*.h .
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ cp ~/CPE2600/lecturecode/LabWeek5/Makefile .
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ ls
Makefile  README.md  main.c  veclab.c  veclab.h  vect.h  vectarray.c  vectarray.h  vectop.c  vectop.h
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$
```

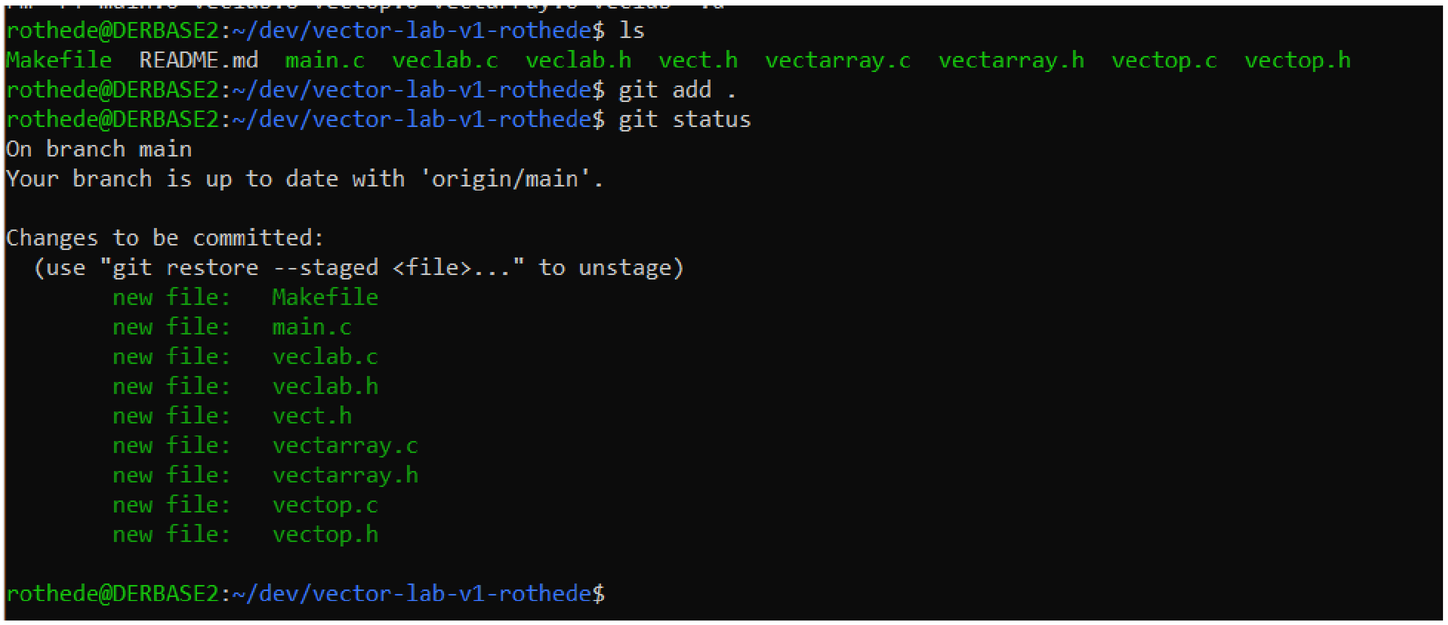
We never want to track build products – that is .o files, .d files or then finished executable. It would not hurt to run make to ensure everything is working and present and then make clean to get rid of the build products.



```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ ls
Makefile  README.md  main.c  veclab.c  veclab.h  vect.h  vectarray.c  vectarray.h  vectop.c  vectop.h
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ make
gcc -c -Wall -g main.c -o main.o
gcc -MM main.c > main.d
gcc -c -Wall -g veclab.c -o veclab.o
gcc -MM veclab.c > veclab.d
gcc -c -Wall -g vectop.c -o vectop.o
gcc -MM vectop.c > vectop.d
gcc -c -Wall -g vectarray.c -o vectarray.o
gcc -MM vectarray.c > vectarray.d
gcc main.o veclab.o vectop.o vectarray.o  -o veclab
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ make clean
rm -rf main.o veclab.o vectop.o vectarray.o veclab *.d
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ ls
Makefile  README.md  main.c  veclab.c  veclab.h  vect.h  vectarray.c  vectarray.h  vectop.c  vectop.h
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$
```

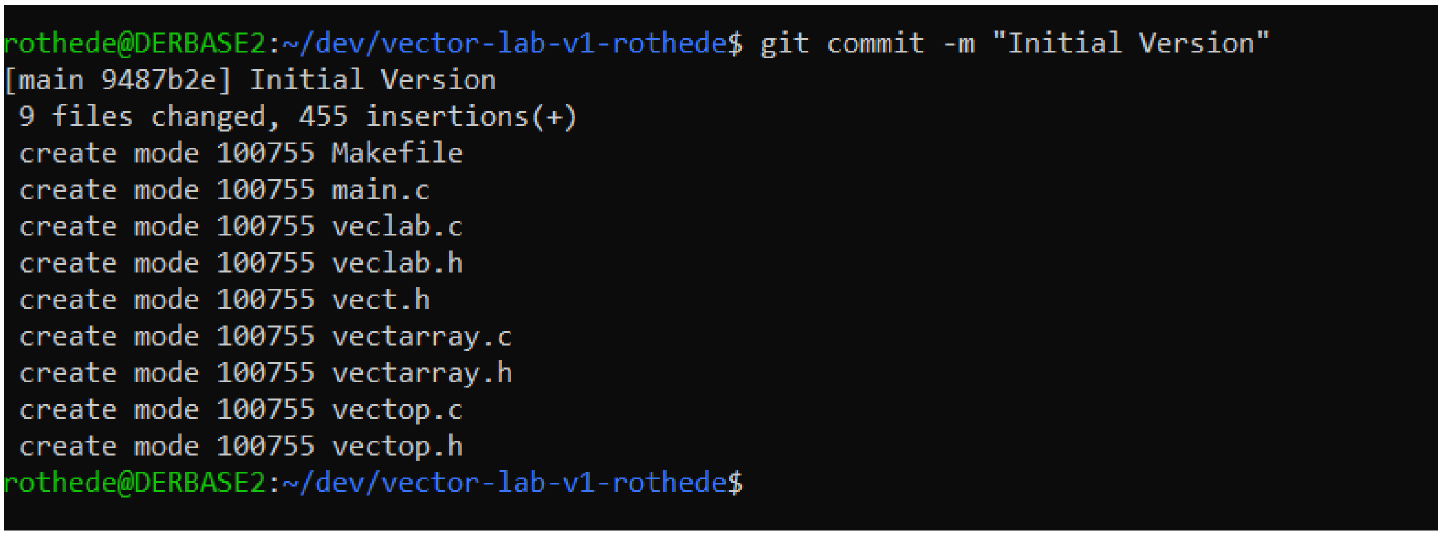
Now, these files are not yet part of the repo and are not being tracked yet. We need to "add" files to the staging area, which in the case of new files will cause git to start tracking them. After that, we can "commit" the files to the local repository.

Making sure there are no unwanted files in the working directory, you can issue the command 'git add .' to add all of the files in the directory to the staging area. Alternatively, you can issue 'git add *.c', 'git add *.h', and 'git add Makefile'. After issuing your 'git add' command(s). Type 'git status' to see what git is up to.



```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ ls
Makefile  README.md  main.c  veclab.c  veclab.h  vect.h  vectarray.c  vectarray.h  vectop.c  vectop.h
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git add .
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Makefile
        new file:   main.c
        new file:   veclab.c
        new file:   veclab.h
        new file:   vect.h
        new file:   vectarray.c
        new file:   vectarray.h
        new file:   vectop.c
        new file:   vectop.h

rothede@DERBASE2:~/dev/vector-lab-v1-rothede$
```

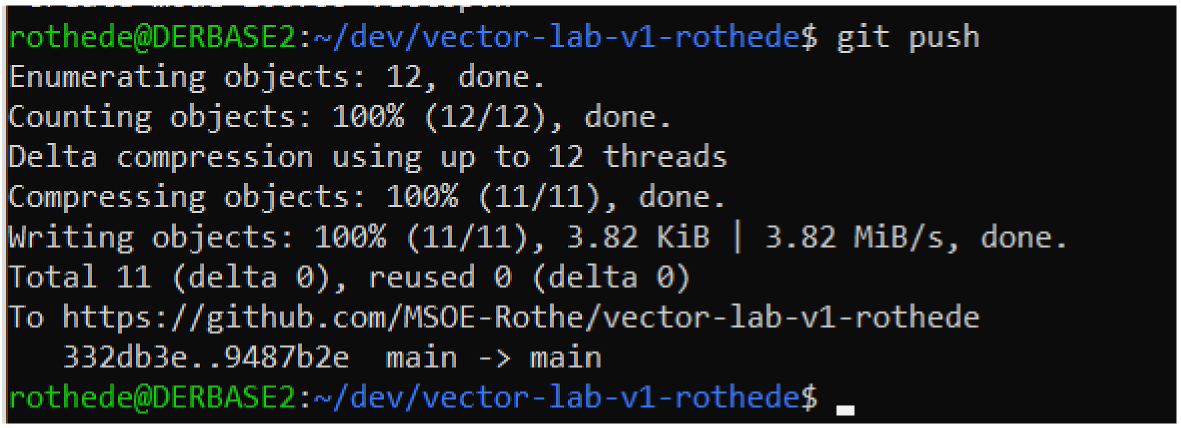
You are now ready to commit to the local repo. Git commit will require a comment which it will attach to the tracked version. It is easiest to provide the comment when git commit is invoked. See below:



```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git commit -m "Initial Version"
[main 9487b2e] Initial Version
 9 files changed, 455 insertions(+)
 create mode 100755 Makefile
 create mode 100755 main.c
 create mode 100755 veclab.c
 create mode 100755 veclab.h
 create mode 100755 vect.h
 create mode 100755 vectarray.c
 create mode 100755 vectarray.h
 create mode 100755 vectop.c
 create mode 100755 vectop.h
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$
```

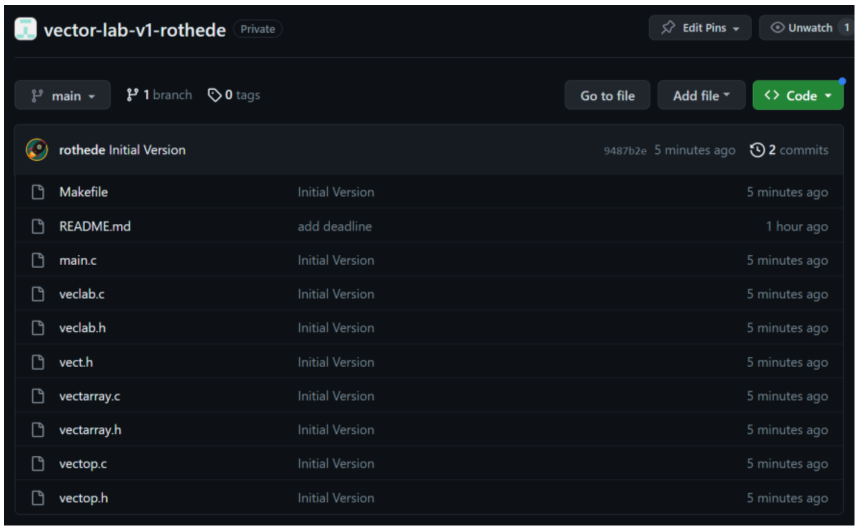
Step 6 – Pushing to Remote

Git is now tracking these files, but they are still only in the local repo. While we are connected to the remote repo, we have to manual push our latest changes when we wish to. When should that be? It depends. Let's do it right now. The command is simple: 'git push':



```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git push
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 12 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (11/11), 3.82 KiB | 3.82 MiB/s, done.
Total 11 (delta 0), reused 0 (delta 0)
To https://github.com/MSOE-Rothe/vector-lab-v1-rothede
   332db3e..9487b2e  main -> main
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ _
```

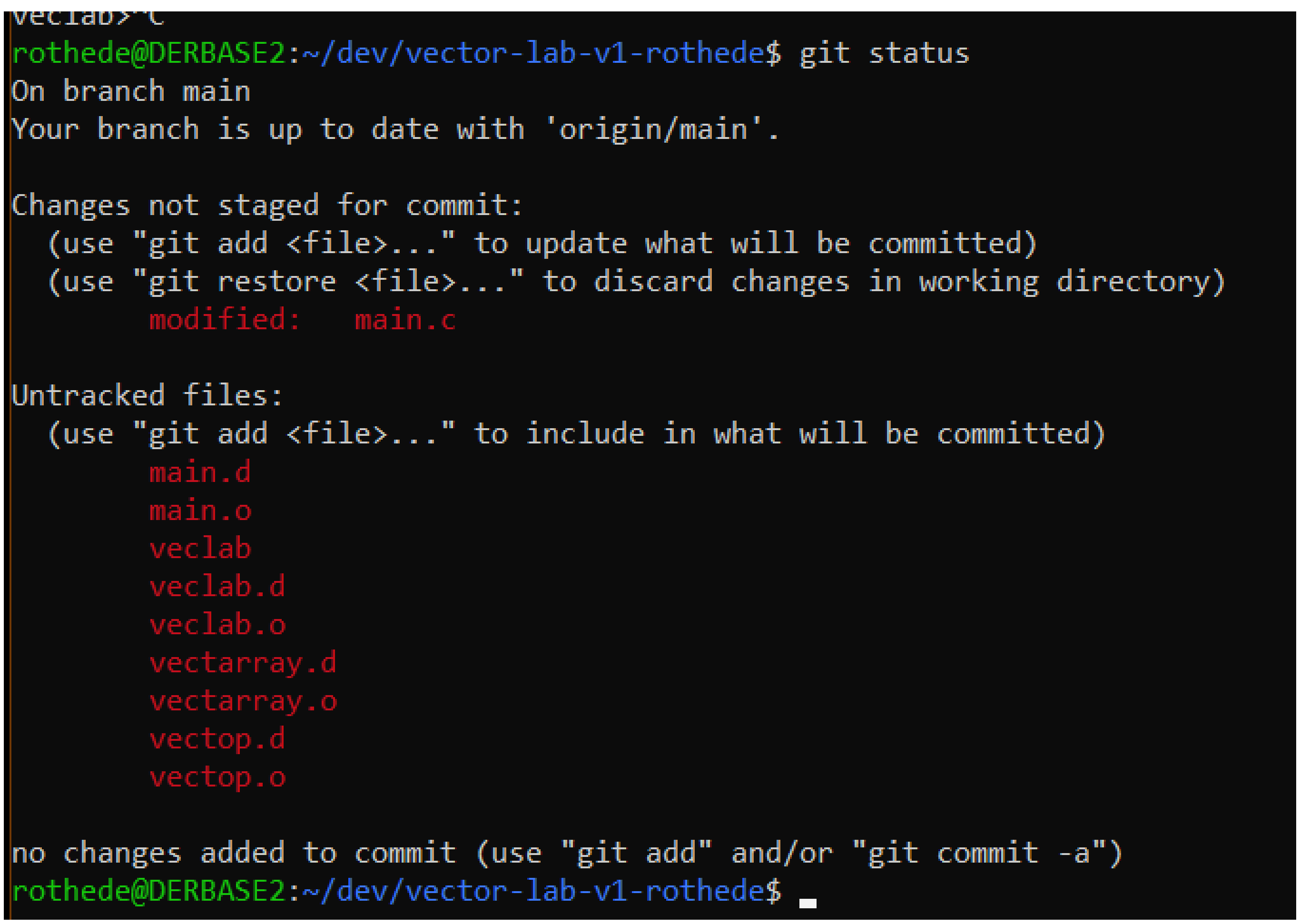
Now you should be able to see your project in GitHub's web interface



```
vector-lab-v1-rothede  Private                                   Edit Pins   Unwatch 1

main   1 branch   0 tags                                 Go to file   Add file   Code

rothede Initial Version                              9487b2e 5 minutes ago   2 commits
Makefile          Initial Version                                  5 minutes ago
README.md         add deadline                                     1 hour ago
main.c            Initial Version                                  5 minutes ago
veclab.c          Initial Version                                  5 minutes ago
veclab.h          Initial Version                                  5 minutes ago
vect.h            Initial Version                                  5 minutes ago
vectarray.c       Initial Version                                  5 minutes ago
vectarray.h       Initial Version                                  5 minutes ago
vectop.c          Initial Version                                  5 minutes ago
vectop.h          Initial Version                                  5 minutes ago
```

Step 7 – Editing Files

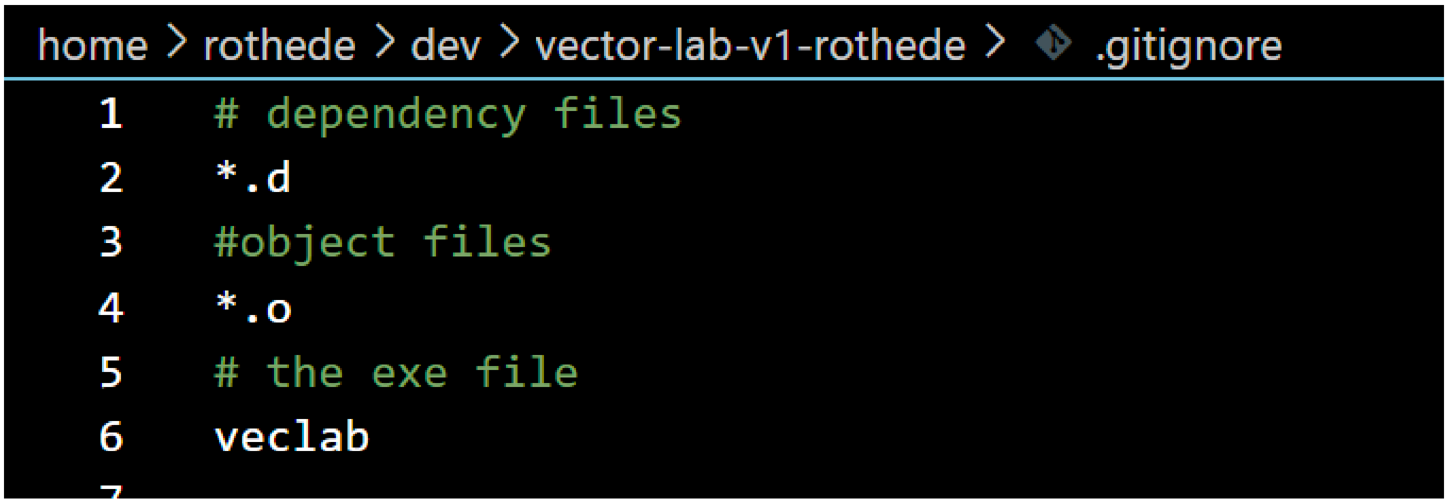
Develop a bit on your project. Perhaps update main() to enhance the help message. You will do this the same as always withing the working directory. Make your project and test your changes. Do not 'make clean' this time. Issue 'git status':



```
veclab>^C
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.c

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        main.d
        main.o
        veclab
        veclab.d
        veclab.o
        vectarray.d
        vectarray.o
        vectop.d
        vectop.o

no changes added to commit (use "git add" and/or "git commit -a")
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$
```

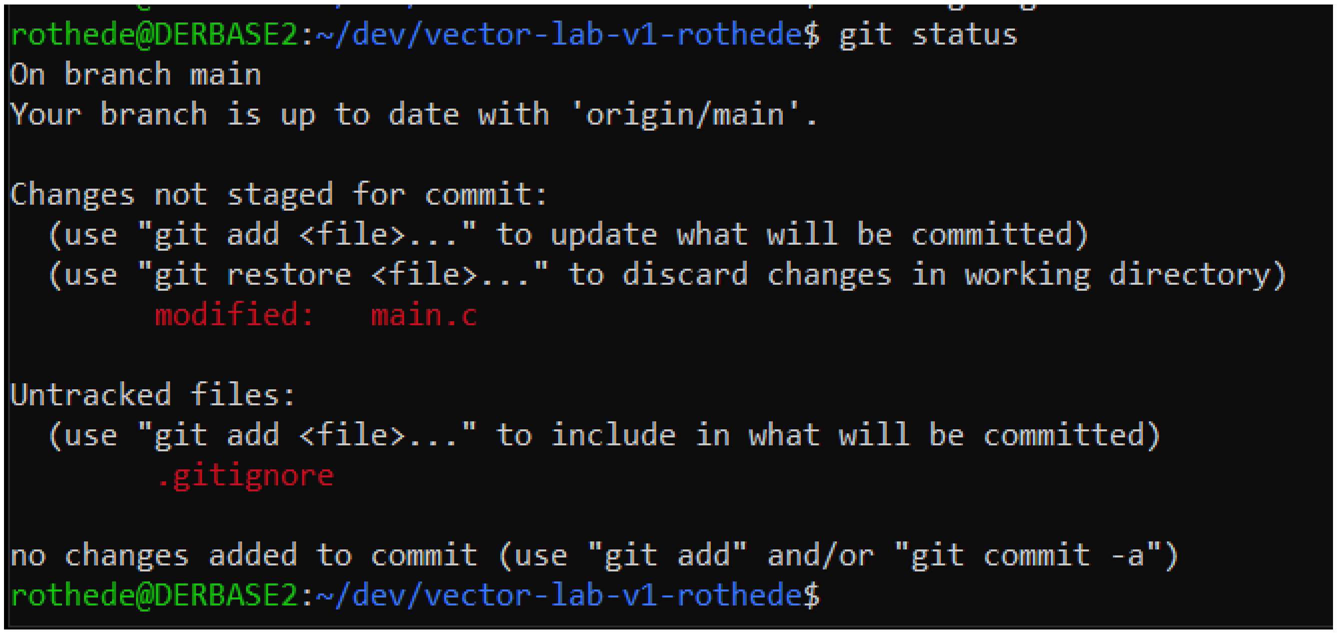
Notice that git sees the changes to main(), but it also sees the .o, .d, and exe files. We can simply ignore these extra files because we know we do not want them tracked. But, it might be nice if git ignored them too since we will never want them in our repo. To make that happen, create a new file called .gitignore and list all of the files you want got to ignore.

Here is mine:



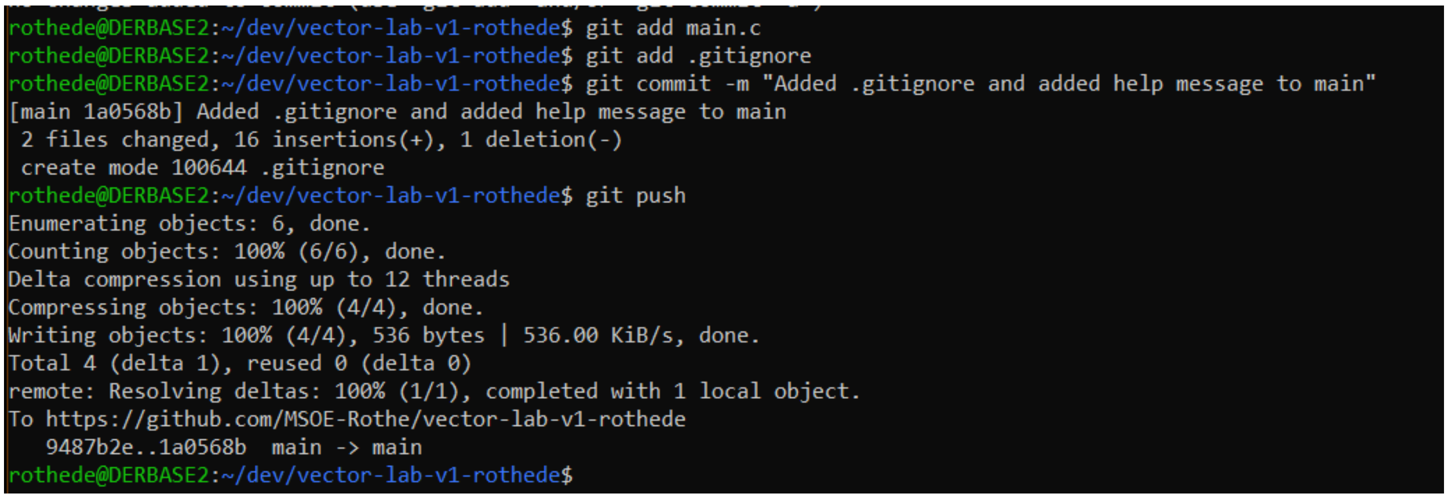
```
home > rothede > dev > vector-lab-v1-rothede > .gitignore
# dependency files
*.d
#object files
*.o
# the exe file
veclab
```

Run ‘git status’ again:



```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.c

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

no changes added to commit (use "git add" and/or "git commit -a")
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$
```

Lets add the new .gitignore file to be tracked. We also need to stage the changes to main.c. We use git add for both purposes. This will be followed by a commit:

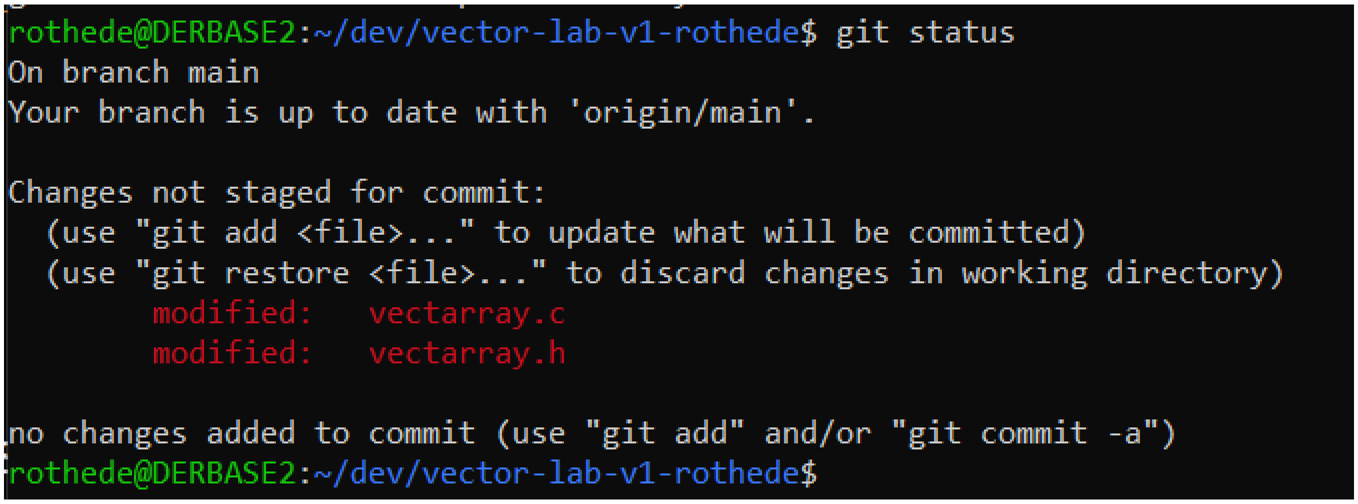


```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git add main.c
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git add .gitignore
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git commit -m "Added .gitignore and added help message to main"
[main 1a0568b] Added .gitignore and added help message to main
 2 files changed, 16 insertions(+), 1 deletion(-)
 create mode 100644 .gitignore
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 536 bytes | 536.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/MSOE-Rothe/vector-lab-v1-rothede
   9487b2e..1a0568b  main -> main
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$
```

Note that the comment encompasses two somewhat different reasons for committing changes. There is a lesson here. It would be wise to make separate commits for each sort of change made to the project.

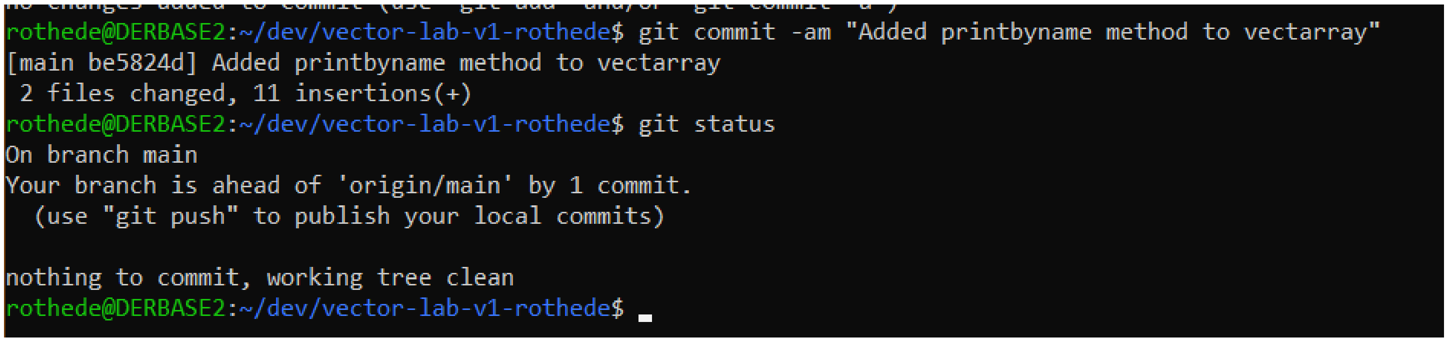
Step 8 – Another Edit

Edit another file or two. Here I edited vectarray.c and vectarray.h:



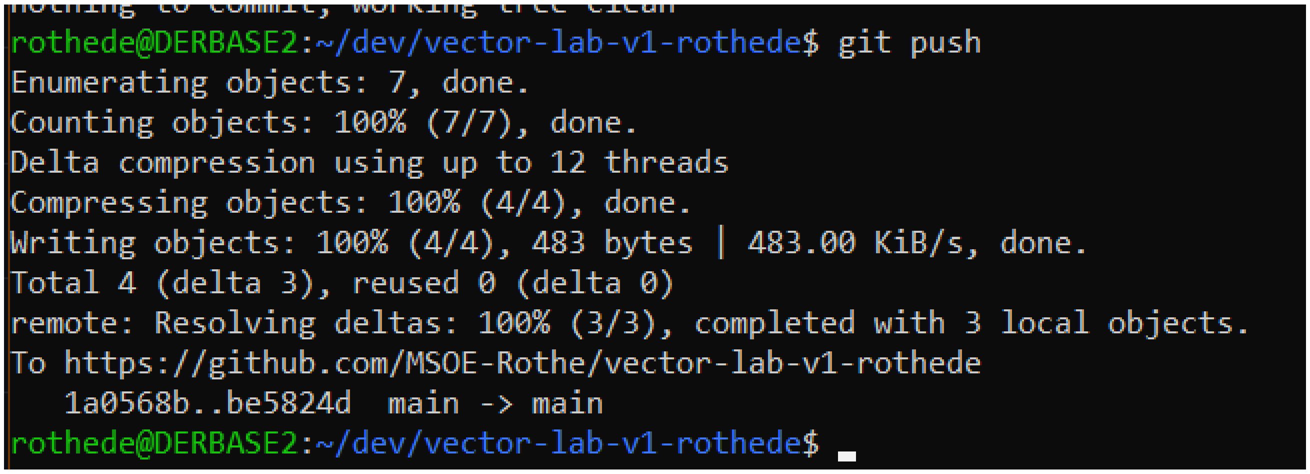
```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   vectarray.c
        modified:   vectarray.h

no changes added to commit (use "git add" and/or "git commit -a")
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$
```

Perhaps not obvious before is that committing changes to the local repository is a two-step process – git add and git commit. There is a good technical reason for this but we do not always want this extra step. We can shortcut is a bit by asking git commit to also perform the add/staging function. Add -a to the command line to automatically stage any tracked files that have changed and commit them with the provided message. See below:



```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git commit -am "Added printbyname method to vectarray"
[main be5824d] Added printbyname method to vectarray
 2 files changed, 11 insertions(+)
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ _
```

The last step of pushing to remote is always separate and does not need to be done with every tracked change. It would be a good practice to push to remote whenever you are finished with a work session.



```
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 483 bytes | 483.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/MSOE-Rothe/vector-lab-v1-rothede
   1a0568b..be5824d  main -> main
rothede@DERBASE2:~/dev/vector-lab-v1-rothede$ _
```

Future Topics

We are really just scratching the surface of version control. The following topics will be explored in future assignments:

1. Tagging (milestones or releases)
2. Branching
3. Merging

Deliverable

By the end of the lab period today, you will have:

1. “Accepted” the GitHub Classroom assignment and created your repository.
2. Cloned and populated the repository with the current / initial version of your “vector” lab (including pushed initial version to remote).
3. Made at least one local change to your source code and added a .gitignore file to your repository and pushed those changes to remote.

Show the screen captures of each of the steps and submit the report to Canvas.

Note, you will be expected to complete Lab 6 independent of Lab 5 status.