

This first Lab is the tip of the iceberg for our DevSecOps course, as we will continue building upon our GitLab codebase to create a real DevSecOps pipeline that deploys real infrastructure to Amazon Web Services (AWS). Each Lab that we do in this course augments work from a previous Lab, until we ultimately have a fully-functioning DevSecOps pipeline. As such, it is critically important that you do not miss any Lab deliverables, and proactively contact the professor if you have any trouble following the steps.

In this Lab, we will get acquainted with our GitLab repository (“repo” for short), Visual Studio Code, and push some code to our repo.

Before you begin:

- You should have already accepted the GitLab invitation, sent to your Marymount e-mail, which allowed you to create a GitLab account and become a Maintainer of the repo that contains your name.
- You should already have installed Microsoft Visual Studio Code for free via the link posted on Canvas. Note that this is a different application than regular Visual Studio, which is a heavier, payware application.
- You should already have installed Git for free via the link posted on Canvas. VS Code should automatically detect that you have successfully installed Git.
If your Git installation was unsuccessful, VS Code will display an error in the left sidebar stating that it was unable to find Git on your system.

What you’ll achieve:

- Get a basic feel for using Git to synchronize code changes

What you’ll turn in with this lab:

- Please simply go to the Lab 1 assignment on Canvas, and author a text submission that simply says “Done.” This will notify the professor to check your GitLab repo for satisfactory completion of the Lab, and enter a grade into Canvas for the same.

Task 1: Clone Your Repository

The first step to contributing to a Git repository is to “clone” the repo to your local machine. Unlike other online collaboration services such as Google Docs, edits to code stored in a Git repo are not performed in real-time. Instead, you make changes to a copy of the code on your personal computer, “commit” those changes with a short comment about what you changed, and then “push” those changes back up to the repo.

Git then checks to ensure there are no conflicts between the changes you are pushing and any changes your colleagues may be pushing (if they were using the same repo; in this course, you have a personal repo all to yourself), and either allows the push to complete, folding in your changes comfortably alongside your colleagues' changes, or identifies any conflicting commits where the push can't succeed without someone's code being lost, at which point it's up to you to pick up the phone and work with your colleague to manually fix that line of code. Git also allows you to rollback to previous commits, even ones that weren't yet pushed to the server, giving it powerful versioning to save you from any other mistakes that befall developers.

One final note on our Git crash course before we start the Lab: Conflicts in pushing code are rare if you're following best practices, because *normally* you would cut a "branch" from the "main" branch of the repo before working on a new feature or bugfix (we probably won't do this in this course, since we each have our own repos). This way, only you are pushing code to that branch. Some organizations don't do this as well as they should, and at the end of the day, you still have to "merge" all of your branches back into the main branch before you can deliver your software, so there's always some level of risk that your code might clash with someone else's, which is a big reason why DevOps practitioners like to push their code "early and often" to minimize this risk. Okay, stop talking, Professor, let's do the Lab already!


1. Open a Web browser and go to GitLab. Log in with your account and navigate to your personal repo under the Marymount University IT-351 group.
2. Open Visual Studio Code. On the left sidebar, click "Clone Repository."
3. Back at GitLab, click the "Clone" menu, copy the HTTPS link, and paste it into the "Clone from URL" prompt in VS Code.
4. You will be prompted for a local location for the repo's files. This can be anywhere that you won't accidentally delete. For this reason, I prefer to put it in a subdirectory named "Repos" under my main user directory. You don't need to create an additional directory for each individual repo. Once you are in your main "Repos" directory, click "Select Repository Location."
5. You will be prompted for a username and password, which should be your GitLab credentials.
6. When prompted, you can simply choose to "Open" the repository in VS Code, unless you wish to create a Workspace.
7. When prompted, select "Yes, I trust the authors."

You should now see a default README.md file – the same file visible on GitLab – in the local clone of your repo in VS Code. This means that the clone operation was successful, and this Task is complete!

Task 2: Make and Commit Changes

Now that we've cloned our repository, which shows us the default README.md file created by GitLab, we can now make changes to the files in the repo and commit them. For this Task, we



will simply edit our README file to contain some information about ourselves, and then commit that change.

8. Double click the README.md file to open it in the VS Code editor.
9. Make some changes to your README.md file, using the pre-existing syntax to ensure that the formatting will work correctly. You can retain the main header with your name, but make two sub-headers, one that says “This is my Lab 1 readme” followed by some text, and another that says “Links that I like” followed by 2-3 URLs that you like. Save the file.
10. On the left sidebar of VS Code, click on the  Source Control tab.
11. Make a comment that describes the changes you made, and click the checkmark to commit your changes.

If you return to GitLab, you’ll notice that your README has not changed. That’s because we only committed the changes to our local repo, not the server-side one. Let’s keep working, and later, we’ll push our changes to GitLab.

Task 3: Create and Commit a New File

Let’s also practice the relatively straightforward operation of creating a new file in our repo, and committing it, which is still, technically, a change (to the repo). In a situation when a rollback of our codebase is necessary, this will also cause this change to be reversed, i.e., the file would be deleted.

12. Return to the  Explorer tab of VS Code and click the  new file icon. Name the new file “lab1.txt.”
13. Place any content you want in this file. For example, you can say “This is my Task 3 file for Lab 1 in IT-351.” Save the file.
14. Return to the Source Control tab of VS Code, make a comment that indicates you created a new file, and commit the change.

Task 4: Push All Changes

Even though we’ve been committing the changes from our previous Tasks, you’ll notice that these changes aren’t showing on the GitLab web interface. That’s because the Git service that you installed locally is managing the versioning right now, not GitLab. Right now, all the changes you’ve committed have only been committed locally. To make them take effect within the server-side repo, we have to push those changes.

15. Before we can push our changes, we need to identify ourselves as the authors to our local Git service. Open a terminal window in VS Code by going to View > Terminal.

16. Run the following commands **separately, replacing your name and e-mail address where appropriate:**

```
git config --global user.name "Professor Edgar"
```

```
git config --global user.email "jtedgar@marymount.edu"
```
17. In the bottom left of VS Code, observe that you have two (2) changes to push and zero (0) to pull. Click this bar to sync your local repo with the server-side repo. You will be prompted to push and pull all changes for origin/main (the main repo on GitLab). Click OK and wait for the numbers at the bottom left of VS Code to disappear, indicating that all changes have been synchronized.
18. Check GitLab to ensure that your README from Task 2 now reflects your changes, and that your lab1.txt file from Task 3 is in the server-side repo. If they are, you've successfully completed this Lab!