

CSCI 121: Computer Science II

Terminal and Git

Prerequisites:

1. You have `git` installed on your computer.
2. You have DrJava installed on your computer.
3. You have a folder named CS121 in your home folder.

Version Control

In large-scale software projects, the team of developers use *version control* software to coordinate their collaborative work on the files in the project. In this course we will introduce you to github, a popular version control system. You will obtain copies of the base files for labs and projects, and you will also submit your solutions. In this lab we will introduce you to the processes needed to interact with github.

Figure 1 shows the *workflow* of an assignment for this course. Files will move between various *repositories* and your *sandbox*. A repository is a location that holds all the versions of a file along with the timestamp and username of the person who edited the file. Your sandbox is where you do your actual work, editing and compiling files. One of the benefits of this system is that it provides a backup system – if you accidentally delete a file or make a change you want to undo, you can go back to previous versions from a repository.

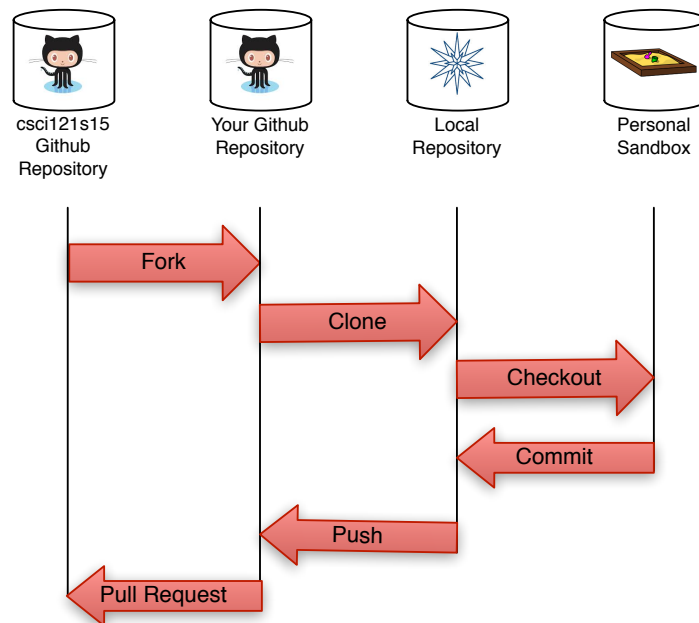


Figure 1: The workflow for assignments in CSCI 121.

The Steps of Our Workflow

1. **Fork the assignment repository:** *Fork* refers to the FIXME “fork in the road,” and means that we are creating an exact duplicate of the repository. As Figure 1 indicates, we will use github to hold the base repository and your forked version. These repositories will be stored “in the cloud” on github’s servers.
2. **Clone your github repository:** *Clone* is another way to create an exact duplicate of a repository. In our workflow, we will clone repository in your personal github account to the computer where you are working. This can be an iMac in the computer lab or your personal laptop.
3. **Checkout a copy of the files:** The repository on your computer tracks all the changes to the files. To do work, you must *checkout* a copy of the files. These files are in your personal *sandbox*, and you can now make whatever changes are necessary to complete the assignment.
4. **Commit your changes:** Since the goal of a repository is to track the changes, we *commit* our work to the local repository at regular intervals. You should commit work multiple times during an assignment to document your progress and to create backups.
5. **Push your changes to github:** When you complete the assignment, you *push* it from your local repository to the one on github. All the commits you performed will be copied to github.
6. **Make a pull request:** To submit your work, you perform a *pull request*. This will send a message to your instructor asking him to pull a copy of your github repository for grading. Notice that your instructor will have a copy of the entire repository, not just the final version of your files. This means that he will be able to see how your files changed over time!

In the remainder of this lab we will walk you through a simple example of using this workflow.


Github

Before you begin, you must create a github account. You will use this account throughout your entire academic career, and it will document all the projects you work on. Because this is a public record, many employers ask for your github username so they can see examples of your work. Remember this as you select a username.

Go to github.com and create an account. When prompted, select the (default) free personal plan.

Fork, Clone, and Checkout

The assignments for this course will be posted at github.com/csci121s15. The following steps will allow you to fork today’s lab to your new github account, clone that repository to your computer, and then checkout a copy so we can do some actual work.

1. On the csci121s15 github page, click on the Git-Lab. Then, at the top of the project page click on the  button. This will create an exact copy of the repository in your github account.

2. This takes you to your copy of the repository, which you can see both from the URL and from the repository name at the top of the webpage. In the lower-right, click in the text box labeled “HTTPS clone URL” and copy the text (CMD-C).
3. Open a terminal and then type `cd cs121` to move to the directory for this class.
4. In the terminal, type `git clone` and then hit CMD-V to paste the HTTPS clone URL you copied earlier. This command will clone your github repository to your computer.
5. `git` assumes you want a copy of the current version of the files, so it automatically performed a checkout. If you want, you can type command `git checkout`, and it will tell you that you have the up-to-date version of the files.

You are now ready for the rest of the lab.

Moving around in the Terminal

To be effective using the terminal, it is important that you start to develop a mental picture of how files are organized on disk. The following exercise gives you an opportunity to practice with the terminal.

In the terminal, we always have a *current directory*, which is the directory (also called a folder) where we are currently working. You can always see your current directory by typing the command `pwd`. To move to a different directory, we use the command `cd`, which stands for “change directory.” To list the contents of the current directory, we use the command `ls`. Please do the following:

1. You should be in the `cs121` directory currently. If not, type `cd`, which takes you to your home directory, and then `cd cs121`.
2. Use `cd` to go to the `Git-Lab` directory, and then use it again to go to the `Sample` directory¹.
3. Type `ls` to see the files in the folder. You should see four files lists. One of the files is a folder. To see that, type `ls -F`. The `-F` is a **command line switch** that tells the `ls` command to put a `/` after any file that is a directory.
4. Use `cd` to go to the `Data` folder and `ls` to see the files in the folder.
5. Type `cd ..` to go “up” one directory – that is, back to the `Sample` directory.

We can visualize the directory structure of this folder as:

```
Sample
├── Data
│   ├── gigantic.dat
│   └── medium.dat
```

¹The terminal has an auto-complete feature where you can type the start of a name, hit the tab key, and it will complete the name for you. You can also change multiple directories at a time. A shortcut for what we just did is to type `cd` and the letter `'G'` and hit tab to complete the name `Git-Lab`. The terminal adds a `/` to the end of the name because it is a folder, so you can now type `'S'` and hit tab again to add `sample`. You now have `cd Git-Lab/Sample/`, and when you hit return, you go to the proper directory.

```

├── tiny.dat
├── Histogram.java
├── Main.java
└── Simulation.java

```

Question 1. The **Git-Lab** directory also contains a directory named **Exercise**. Using only the commands `cd` and `ls` in the terminal, explore the structure of the **Exercise** directory and draw a picture similar to the one above.

Before you continue, feel free to explore the **Git-Lab** directory. A copy of this lab is included in that folder along with the files used to create the lab.

DrJava

For the first half of the semester, we will use a programming environment called DrJava. Please open DrJava and type the following program (changing the name to your name):

```

public class First
{
    public static void main(String[] args)
    {
        System.out.println("Hello, my name is George.");
    }
}

```

Save the code in the **Program** directory inside the **Git-Lab** directory.

You can compile your code by pressing **F5**, and you can run your program by pressing **F2**.

Commit, Push, and Pull Request

When you have a working program, you are ready to submit your program. During this process you will specify which files to commit and then add a message saying what changes you made.

First we will configure **git** to auto-complete some values for you and to use a friendly editor:

1. Type `git config --global core.editor nano` to make **git** use the **nano** editor instead of **vi**.
2. Type `git config --global user.name "John Doe"` with your name in quotes to have commits tagged with your name.
3. Type `git config --global user.email johndoe@example.com` with your email address to have commits tagged with your email address.
4. Change directory to **Git-Lab/Program** and list the files. You should see three files, **First.java**, **First.java** , and **First.class**. The file ending in a tilde is a backup file, and the **.class** file is the compiled version. **bf We only put source files in repositories.**
5. Type `git add First.java` to tell **git** that the changes to this file should go into the next commit. If you type `git status` it will show you the status of the files in your repository.

6. Type `git commit` and an editor window will open. This is where you write a log message that will be permanently associated with the changes to the file. In this case, we can simply say that we created this file to practice with Java.
7. Your changes you added are now in the repository on your computer. To get them to github, type `git push`. This will prompt you for your github username and password.
8. On the github you can now see the changes that you made. Navigate into the Program folder and click on `First.java` to see how github shows changes.
9. To send your code to your instructor you will submit a pull request. On the main github page for this repository, click on the link “Pull Requests” on the right side of the page. Click the green “New pull request” button, and make sure the proper changes are being sent. Click “Create pull request,” fill in the title, and comment, and then click “Create pull request” to finalize your request.

Congratulations! You have successfully completed the first lab.

More information

If you want to learn more about using `git` at the command line, try the online tutorial at <https://try.github.io/>.