Hack 1.0

Computer Science I

Department of Computer Science & Engineering
University of Nebraska–Lincoln

Introduction

Hack session activities are small weekly programming assignments intended to get you started on full programming assignments. Collaboration is allowed and, in fact, *highly encouraged*. You may start on the activity before your hack session, but during the hack session you must either be actively working on this activity or *helping others* work on the activity. You are graded using the same rubric as assignments so documentation, style, design and correctness are all important. This activity is due at 23:59:59 on the Monday following the hack session in which it is assigned according to the CSE system clock.

Problem Statement

An essential tool when developing software is a *version control system* (VCS). As you develop software you will make changes, add features, fix bugs, etc. and it is necessary to keep track of your changes and to ensure that your code and other artifacts are backed up and protected by being stored on a reliable server (or multiple servers) instead of just one machine.

A *version control system* allows you to "check-in" or *commit* changes to a code project. It keeps track of all changes and allows you to "branch" a code base into a separate copy so that you can develop features or enhancements in isolation of the main code base (often called the "trunk" in keeping with the tree metaphor). Once a branch is completed (and well-tested and reviewed), it can then be *merged* back into the main trunk and it becomes part of the project.

These systems are not only used for organizational and backup purposes, but are absolutely essential when developing software as part of a team. Each team member can have their own working copy of the project code without interfering with other developer's copies or the main trunk. Only when separate branches have to be merged into the trunk do conflicting changes have to be addressed. Such a system allows multiple developers to work on a very large and complex project in an organized manner.

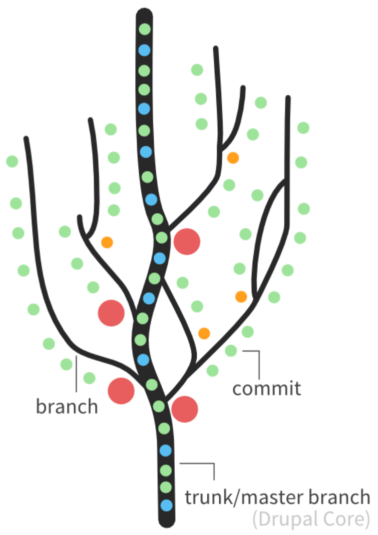


Figure 1: Trunk, branches, and merging visualization of the Drupal project

There are several widely used revision control systems including CVS (Concurrent Versions System), SVN (Apache Subversion), and Git. SVN is a *centralized* system: there is a single server that acts as the main code repository. Individual developers can check out copies and branch copies (which are also stored in the main repository).

Git is a *distributed* VCS meaning that multiple servers/computers act as full repositories. Each copy on each developer's machine *also* contains a complete revision history. This makes git a decentralized system. Code commits are committed to a local repository. Merging a branch into another requires a push/pull request. Decentralizing the system means that anyone's machine can act as a code repository and can lead to wider collaboration and independence since different parties are no longer dependent on one master repository.

Git has become the de facto VCS system in software development. We have provided several external resources below, but this Hack will walk you through the basics of getting started. You will setup a project with git using GitHub (`https://github.com`) as your remote server. You will then collaborate with someone else to commit changes.

1 Installation

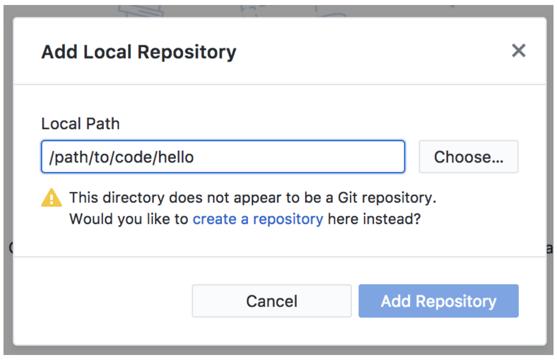
All of the instructions in this Hack will involve using GitHub's Desktop Client. The screenshots were from the Mac version of this program so it may look slightly different on other systems. There are other alternatives and if you wish to use them you are welcome to do so.

1. Go to https://github.com/ and sign up/register with an account if you have not already done so.
2. Download and the client here: https://desktop.github.com/.

2 Creating a Repository

To focus on the git process, you will create and work with a simple "Hello World"-style program but instead of printing "Hello World", it will print your name.

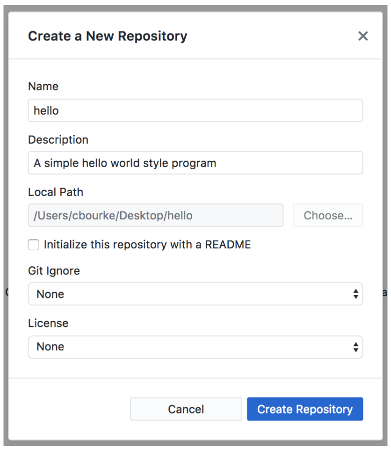
1. Create a new folder (call it `hello`) somewhere on your computer and within it, create a `hello.c` source file with code in it that prints your name.
2. Open your GitHub Desktop Client (you may need to sign in with your new GitHub credentials).
3. Drag and drop your `hello` folder to the GitHub Desktop Client. It will look something like the following.



/path/to/code/hello

This directory does not appear to be a Git repository.
Would you like to create a repository here instead?

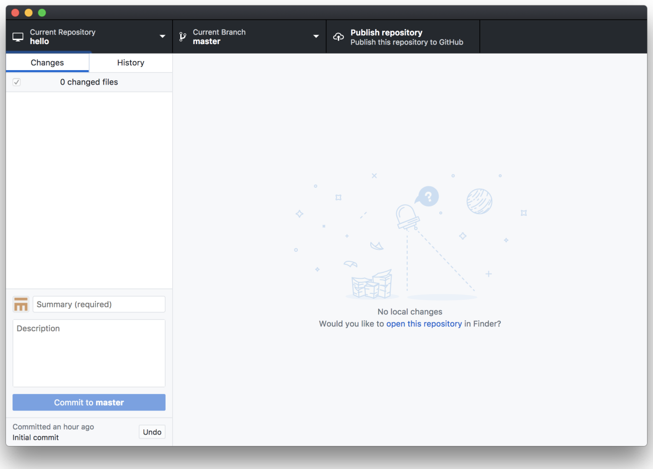
Click `create a repository` and it should look something like the following.



Name: hello
Description: A simple hello world style program
Local Path: /Users/cbourke/Desktop/hello
Initialize this repository with a README
Git Ignore: None
License: None

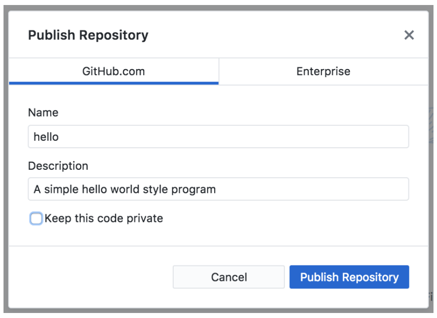
Provide a simple description and click `Create Repository`

4. If everything is successful, it should look something like the following.



The GitHub client will have made an initial commit of the existing files for you.

5. Push/publish your repository to GitHub.com by clicking `Publish repository` at the top right. It should look look something like the following.



Name
hello

Description
A simple hello world style program

☐ Keep this code private

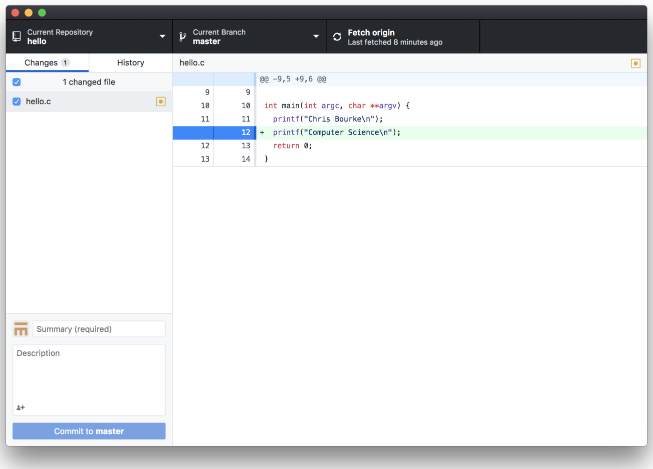
Be sure that `Keep this code private` is *not selected* and click `Publish Repository`

6. If successful, your repository should now be on GitHub. Point your web browser to `https://github.com/login/hello` where `login` is replaced with your GitHub user name. You can browse your repository, view its history, etc.

3 Making Changes

You'll often make changes to you code that you should *commit* to your repository. Though you may make changes and save them to a file, the changes are not saved to the repository's history. Committing is the action that does this. Committing only changes your *local* repository, the changes will still need to be *pushed* to GitHub. In this activity you'll make changes, commit them and then push them to GitHub.

1. Open the `hello.c` file and add a line that prints your major. Be sure to save the file.
2. The GitHub Client should "see" these changes. It should look something like the following.

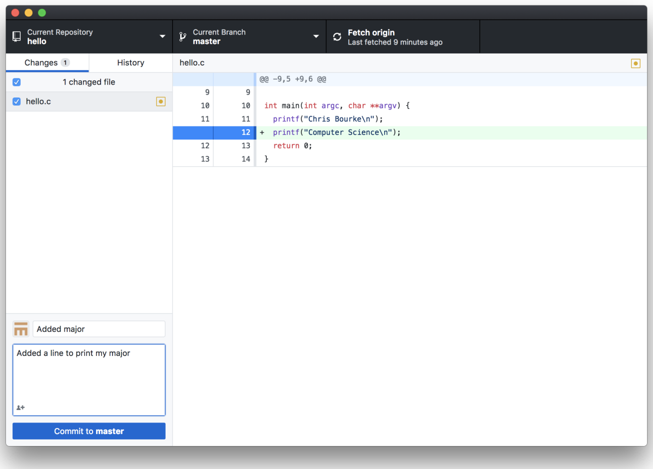


hello.c

```
@@ -9,5 +9,6 @@

 int main(int argc, char **argv) {
   printf("Chris Bourke\n");
+  printf("Computer Science\n");
   return 0;
 }
```

3. Commit your changes by filling out the summary (required) and description. This is your chance to *document* why you made changes to your code. Then click `Commit to master`. Your changes have now been committed to your repository.



Added major

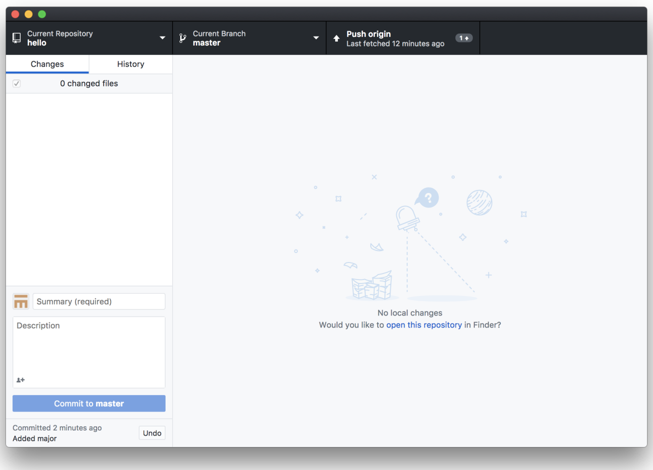
Added a line to print my major

hello.c

```
@@ -9,5 +9,6 @@

 int main(int argc, char **argv) {
   printf("Chris Bourke\n");
+  printf("Computer Science\n");
   return 0;
 }
```

4. Push your changes to GitHub by clicking `Push origin` at the top right. You can verify that your changes have been published to GitHub by refreshing your web browser.



4 Collaborating With a Team

In this exercise, you'll need to team up with at least one other person. You'll make them a collaborator on your project so they can make changes and commit/push them to *your* repository on GitHub. Alternatively you can have them make a pull request, but these instructions do not cover that; refer to one of the resources below for how how to make push/pull requests.

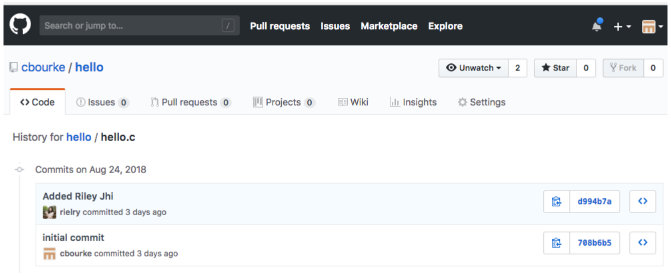
1. On GitHub.com, click `Settings` in your project.
2. In the left menu, click `Collaborators`
3. Type in your partner's GitHub user name and click `Add collaborator`

4.1 What your collaborator needs to do

Together with your partner, walk through the following steps. These steps should be done on *their* computer.

1. Once you've sent an invite to collaborate, they need to accept it.
2. Have them navigate their web browser to your GitHub repository and click `Clone or download` and select `Open in Desktop`. This will launch their GitHub Desktop Client and check out *your* code.
3. Have them add 2 lines of code to print their name and their major.
4. In the GitHub Client, follow the same procedure to commit and push their changes to your repository using the same procedure as above.
5. Verify their changes by refreshing your repository. You can click on the `hello.c`

and if you both did everything correctly, you'll see multiple commits by multiple people:



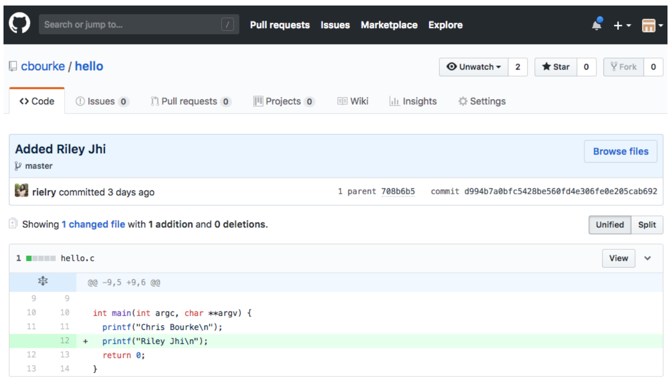
History for hello / hello.c

Commits on Aug 24, 2018

Added Riley Jhi
rielry committed 3 days ago
d994b7a

initial commit
cbourke committed 3 days ago
708b6b5

If you click on `History` you can see the changes for each commit:



Added Riley Jhi
master
rielry committed 3 days ago   1 parent 708b6b5   commit d994b7a0bfc5428be560fd4e306fe0e205cab692

Showing 1 changed file with 1 addition and 0 deletions.

```
1 hello.c
@@ -9,5 +9,6 @@

 int main(int argc, char **argv) {
   printf("Chris Bourke\n");
+  printf("Riley Jhi\n");
   return 0;
 }
```

You are *highly encouraged* to start using git/GitHub (or something similar) for all of your future assignments but be sure to commit code to a *private* repository so that you do not violate the department's academic integrity policy.

4.2 Finishing Up

1. Put *your* GitHub URL into a plain text file named `readme.md`. Turn this file in using webhandin. Each individual student will need to hand in their own copy and will receive their own individual grade.
2. Verify what you handed in by running the webgrader which will display the contents of your file.

Additional Instructions

- You are encouraged to collaborate with any number of students before, during, and after your scheduled hack session.
- Each student is responsible for *their* repository, so if/when you team up with a partner, you'll need to go through this Hack at least twice: once as the primary repository owner and once as a collaborator.

Additional Resources

- Video tutorial on Github Desktop: `https://www.youtube.com/watch?v=kFix7UDJ7LA`
- Interactive git tutorial: `https://try.github.io/levels/1/challenges/1`
- Pro Git, free online book: `https://git-scm.com/book/en/v2`