13 — Lab 12 — Collaboration with Git

13.1 Introduction

Writing software is no longer the domain of the lone ranger programmer shut up in their office with their computer. Indeed, most projects in industry now require the programmer to work collaboratively in a team, sharing code with other programmers and managers, sometimes globally. Today, programmers generally work on just a small part of a much larger codebase, whether that codebase be a library, API, or custom codebase.

Tools have developed over the years to help programmers work together. One tool, Git, allows programmers to work collaboratively and also allows users to keep versions of their work in order so that previous versions may be restored in case of errors.

13.2 Lab 12 Preparatory Work

13.2.1 Introduction into to VCS

Git is a version (or versioning) control system (VCS, sometimes called revision control system). Version control systems allow you to keep track of changes to groups of files over time in discrete steps called revisions.

Programmers use VCS to keep track of changes to their code base over time, but VCS can usually be used to keep track of any kind of file. For example, authors have been known to use VCS to keep track of versions of a book they are writing.

13.2.2 Introduction to Git, This must be done in your home directory

Git is one of the most popular VCS available today. Git was initially developed by Linus Torvalds to be used to Linux kernel development. There are other VCS available on the market, including commercial versions, each with their own strengths and weaknesses.

Lab 12 References is a good starting point for seeing what is available.

Git is a distributed version control system. This is different compared to Subversion (SVN) , which is a centralized version control system. In a centralized version control system there is only one central repository. All developers work from this repository. When changes are made, the changes are committed to this central repository. A centralized repository has some advantages. For example, everyone for the most part knows what everyone else is doing on the project. Administrators have fine-grained control over who can do what; and it's far easier to administer a CVCS than it is to deal with local databases on every client. However, this setup also comes with some disadvantages. The obvious is the single point of failure that the centralized repository is susceptible to. If the server hosting the centralized repository goes down. No developers can commit and saved their changes to the central repository. If there is some kind of hard disk failure, there is a possibility you could lose everything if there is no proper backup plan in place.

In a distributed environment developers do not just checkout snapshots of the files in the repository, they fully mirror the repository itself. This means they checkout all files and revision history as well. This is called a clone. When a developer clones a repository it essentially can be thought of as a backup of the repository. Committing changes made is a little different between centralized and distributed VCS. For example. with SVN, there is no notion of a local commit. When changes are committed in SVN they are committed to the central repository. On the other hand, Git allows for local commits. What this means is any changes made can be committed locally. Then locally committed changes can be pushed to some remote repository that stores the project's source code.

13.3 Lab 12 In-Class Work

In this section you will be creating and using a Git repository and creating/using GitHub.

13.3.1 Create a GitHub Account

If you already have a GitHub account you may skip to the next section. Click Here

1. First navigate to https://github.com/join to create a Git hub Account
2. Follow the steps for creating a GitHub account. You will need to supply a username and password, and email to create an account. Please note that if you wish to keep this account for later usage (and this is highly recommended), don't use dummy values when creating

the GitHub account.

3. Fill out Account Creation Form

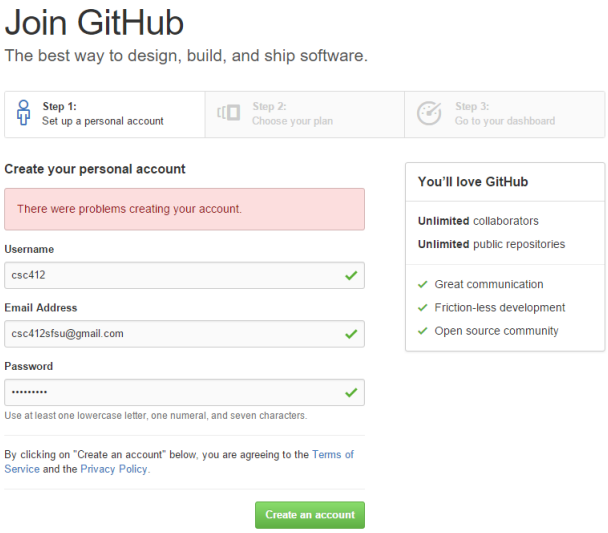

Figure 13.1: Supply Username, Email, and Password

3. Chose the GitHub free plan.

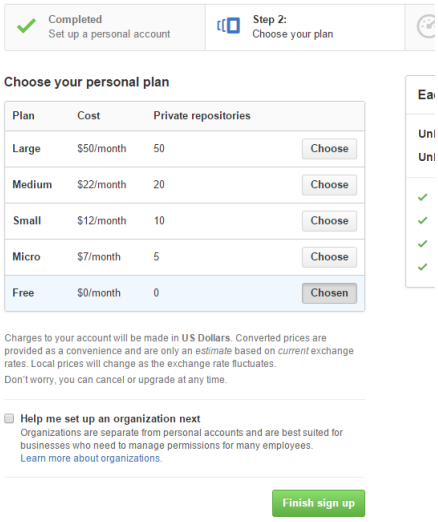

Figure 13.2: Select Free GitHub Account

4. The last step in the account creation process is to verify your email address you used. This can be done by logging into your email account and following the directions in the email from GitHub

13.3.2 Create A Local Repository Directory

Currently what we have is an empty GitHub account with no repositories. We also do not have any local repositories. In the next few steps we are going to create and initialize a local repositories and push them to our GitHub account and then simulate two users using that repository. Login to your account on Amazon-AWS Server. In your home direction (NOT your public_html directory), create a directory called git_test. This can be done by typing the following command:

```
mkdir git_test
```

Next, change your current directory with with the following command:

```
cd git_test
```

During this lab, you will be pretending to be a develop John who is working on a project that he going to host on GitHub:

```
mkdir john
```

As previously mentioned, you must first check out a working version of a repository in order to begin working with the repository. Start by pretending that you are John. Change directories to john `(cd john)`. In this directory we will create another directory called csc412lab using the mkdir command, `mkdir csc412lab`. After creating the directory cd into this directory.

13.3.3 Initializing Local Git Repository

Inside this directory type the following command :

```
git init
```

If done correctly you will see a similar message at the command line:

Initialized empty Git repository in /home/csc412/git_test/john/csc412lab/.git/

Next we need to setup who we are so we can make commits and push commands. To do this type the follow:

```
git config --global user.email "your email"
git config --global user.name "your name"
```

After initializing the local git repository, Git will add a .git folder to the current directory. This can be view with the ls command and the all option (-a). You can peek into this directory by typing `ls .git`

13.3.4 Add Files and Committing

The next step is to create some files. Using any text editor create two text files with the name foo.txt and bar.txt. In the foo.txt file put the following text : "Initial Foo Commit" and in the bar.txt put: "Initial Bar Commit".

Next type the following command :

```
git status
```

This will display the status of the entire repository. It will show you any modified or added

files. Now we will doing a local commit. This commit will add our current changes to the local

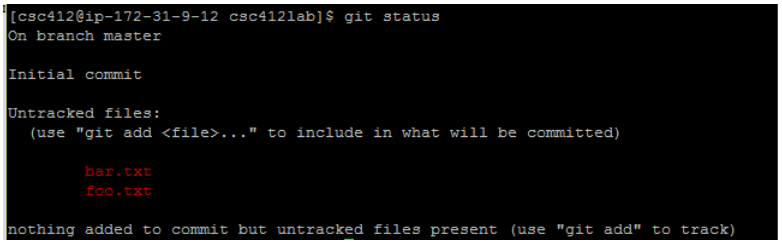

Figure 13.3: Git Status of Johns Directory

repository. First, type the following command:

`git add .` ⏎

This will add all the changes to the commit in the current directory. To see the current status of the commit type `git status .` , including the period. Now we are ready to actually do the

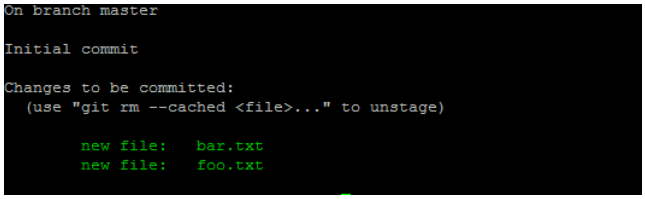

Figure 13.4: Git Status of Johns after git add command.

commit. To do the commit type the following command:

`git commit -m "Johns Initial Commit"` ⏎

This command will commit our local changes to our local repository. If you were to type `git status` you will see that the local git is now current and the changes were added to our git history. If you were to type `git log` you will see our commit. If successful you will see a similar message at the command prompt:

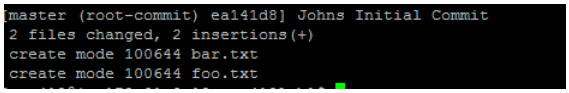

Figure 13.5: Johns Initial Commit.

Now that we have our initial commit for our local repository. it is time to push to a remote repository (GitHub). But now we need to connect our local Git repository to our GitHub repository. To do this we need to create a GitHub repository online with our GitHub account. Navigate to your GitHub account's profile page. Select the repositories tab and the click new. Figure below:

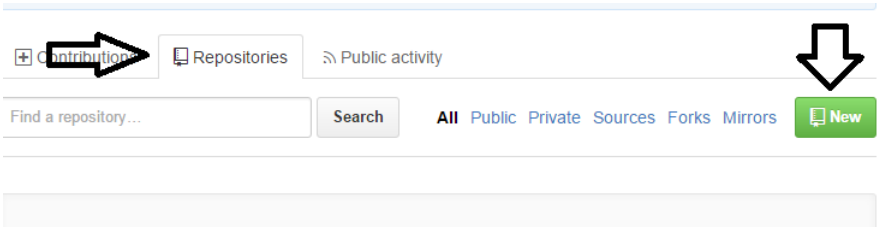

Figure 13.6: Create Remote Repository.

Give the repository the name csc412lab and then select "Create Repository". Next we need the link to the remote GitHub repository. After the remote repository is created, copy the link by clicking on the clipboard icon near the right-hand side of the page. Now switch back to

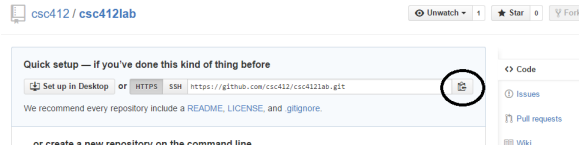

Figure 13.7: Copying GitHub repo Link.

the terminal and type the following command:

```
git remote add origin https://github.com/csc412/csc412lab.git
```

Make sure when you type the above command to use YOUR link and not csc412's link. If there are no errors we are now ready to push or local commits to the remote repository on GitHub. To push our commits type the following command:

```
git push origin master
```

During the commit, if you do not have SSH keys setup it will ask you to provide you GitHub login info to complete the push. Type your username and password to complete the git push. If successful, you will see a similar message at the terminal:

```
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 292 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/csc412/csc412lab.git
 * [new branch]      master -> master
```

Figure 13.8: Successful Git Push.

After the git push look at your GitHub account and see how it has changed. The csc412 remote repository you created should now have 1 commit. There should also be two files inside the repository.

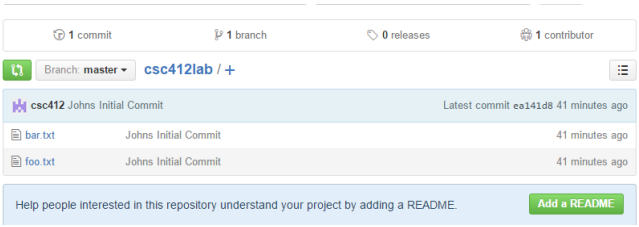

Figure 13.9: Viewing GitHub Repository after push.

13.3.5 Working With branches

Lets say John is working on a big project and he wants to work on a new version of his project. But John does not want to clutter up his main branch with all the commits from the development of the new version. John can create a new branch in his git repository with the following command :

```
git branch v1
```

Then to switch to this branch type :

```
git checkout v1
```

Now John is working on a new branch called v1. Any commits made on this branch will not be seen on his main branch. You should have seen a similar message to the one below.

```
[csc412@ip-172-31-9-12 csc412lab]$ git branch v1
[csc412@ip-172-31-9-12 csc412lab]$ git checkout v1
Switched to branch 'v1'
[csc412@ip-172-31-9-12 csc412lab]$
```

Figure 13.10: Creating and switching to the v1 branch.

Next create two new files with any text editor called verinfo.txt and README.txt. In the verinfo.txt put the following lines "Version 1.0 Author John". In the README.txt file put the following lines "Development version 1.0". After creating and adding the above text to the text files stage another commit by executing `git add .` and then executing a commit command `git commit -m "Version info Commit"`. If the commit was successful you should see something similar to the message below:

```
[csc412@ip-172-31-9-12 csc412lab]$ git add .
[csc412@ip-172-31-9-12 csc412lab]$ git commit -m "Version Info Commit"
[v1 dc924fc] Version Info Commit
 2 files changed, 2 insertions(+)
 create mode 100644 README.txt
 create mode 100644 verinfo.txt
```

Figure 13.11: Committing to the v1 branch.

If there are no errors push to your GitHub account with the `git push origin v1` command

Take a look at your repository on GitHub. You will notice that you will be able to select a different branch. If you switch to the v1 branch it will also tell you how many commits your branch is either ahead or behind.

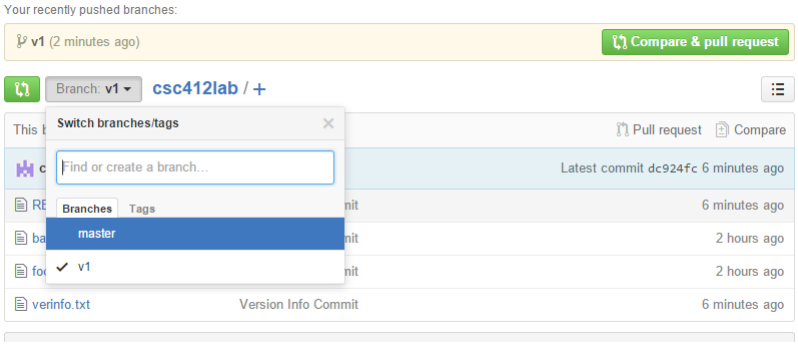

Figure 13.12: GitHub View of multiple Branches.

13.3.6 Merging Branches

For this section of the lab we are going to add a line of text to foo.txt. The line we will add will be "New Version 1.0 Released". After adding this line of text your file should look like the following:

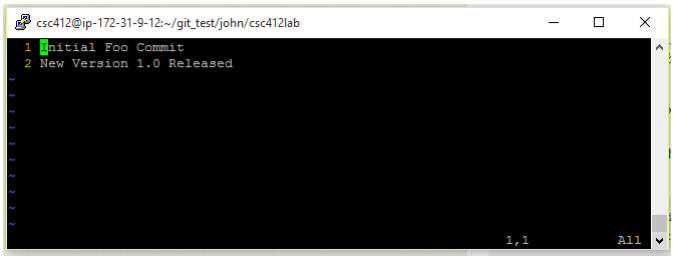

Figure 13.13: Adding second line of text to foo.txt.

After adding the text to foo.txt perform a commit to the v1 branch and push the commit the GitHub repository. Use the following commit message "Adding Release text to foo.txt". If you have done the commit and push correctly you will see a similar message at the terminal :

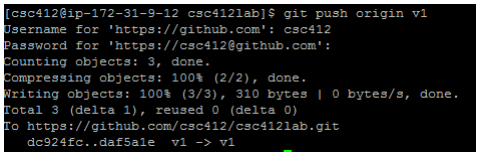

Figure 13.14: Successfully pushed commit to v1 branch

If you notice on the GitHub repository , our v1 branch is now 2 commits ahead of the master branch. This is an example of a forked history. Our commit history is no longer linear.

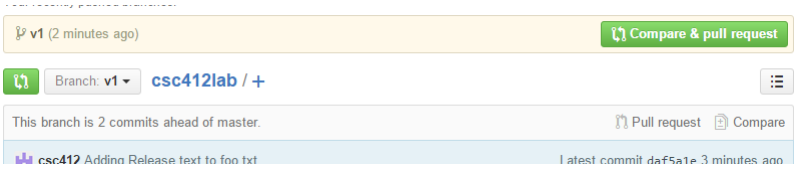

Figure 13.15: SHowing how many commits v1 branch is ahead of the master branch.

Sometimes it is needed to remove this forked history. Next we are going to merge our v1 branch with the master branch. This will removed the forked commit history and will make the master branch and v1 branch even. A merge is a process done in Git that takes two independent lines of development and combine into one single branch. What we are going to do is merge the history of the v1 branch onto the master branch.

To achieve this you will need to type the following commands:

```
git checkout master
```

Note that if you were to type `git log` all the commits we just did are now gone!.
Next type :

```
git merge v1
```

.

If the local merge is successful you will see the following text in the terminal :

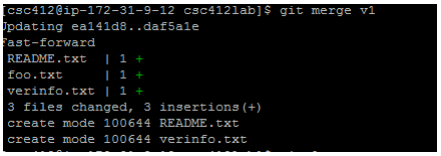

Figure 13.16: A terminal window of a Successful Merge.

Now if you were type `git status` to see that we are in the master branch and then type `git log` you will see now that our master branch contains the history of the v1 branch. Now we can push this local merge to our GitHub repository. Use the following command to push from the master branch:

```
git push origin master
```

If the push is successful you will see similar text in your terminal window:

Figure 13.17: A terminal window of a Successful Merge push.

If the push was successful your GitHub repository will look a little different. The biggest change is now the master branch will contain 3 commits instead of 1. The v1 branch and the master branch are even in terms of commit history.

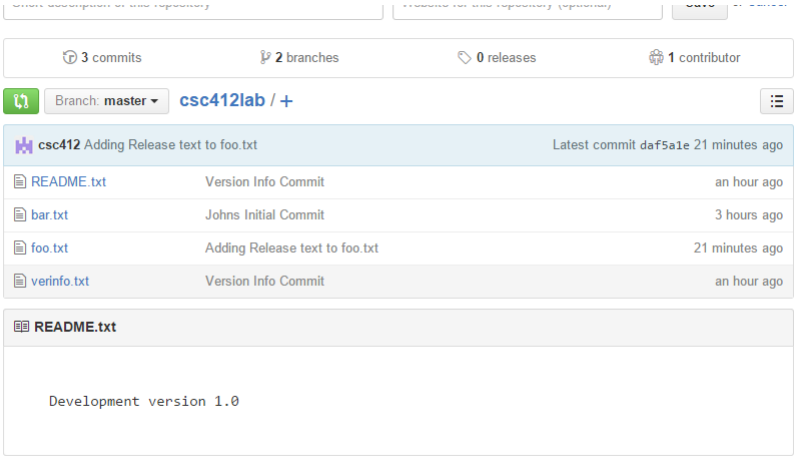

Figure 13.18: GitHub after a Successful Merge push.

Remember

You should always write a unique, descriptive commit message every time you commit.

13.4 References

1. Git Pro Book
2. Git Tutorials
3. Merging-v-Rebasing
4. Revision Control Overview
5. Subversion
6. Comparison of VCS
7. Version Conteol with subversion,a.k.a the Read Bean Book