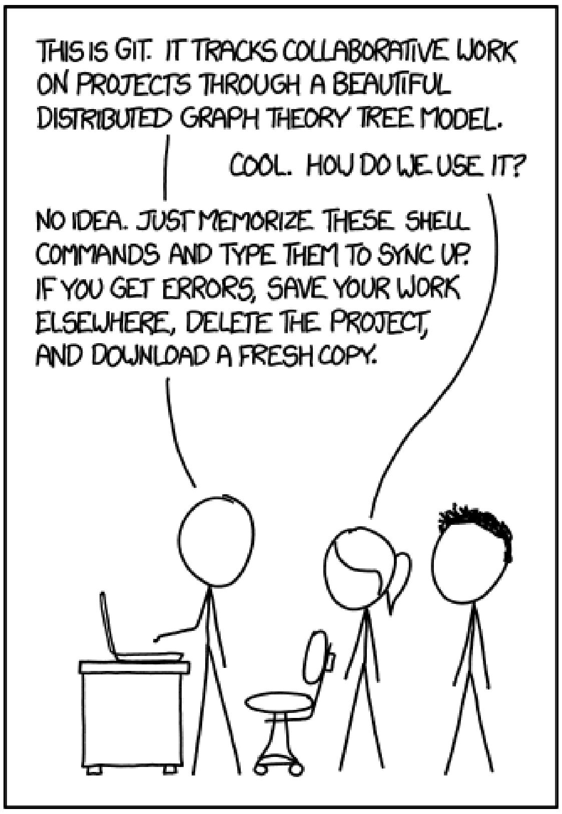
Introduction to Git

Dr. Chris Bourke
cbourke@cse.unl.edu
Department of Computer Science & Engineering
University of Nebraska–Lincoln
Lincoln, NE 68588, USA

August 2015


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.
https://xkcd.com/1597/

Contents

1 Git Overview

As you develop software and make changes, add features, fix bugs, etc. it is often useful to have a mechanism to keep track of changes and to ensure that your code base and artifacts are well-protected by being stored on a reliable server (or multiple servers). This allows you access to historic versions of your application's code in case something breaks or to "roll-back" to a previous version if a critical bug is found.

The solution is to use a *revision control system* that allows you to "check-in" changes to a code base. 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.

You may already be familiar with similar online (or "cloud") storage systems such as

Google Drive or Dropbox that allow you to share and even collaborate on documents and other files. However, a version control system is a lot more. It essentially keeps track of all changes made to a project and allows users to work in large teams on very complex projects while minimizing the conflicts between changes. 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. Otherwise, such a system allows multiple developers to work on a very complex project in an organized manner.

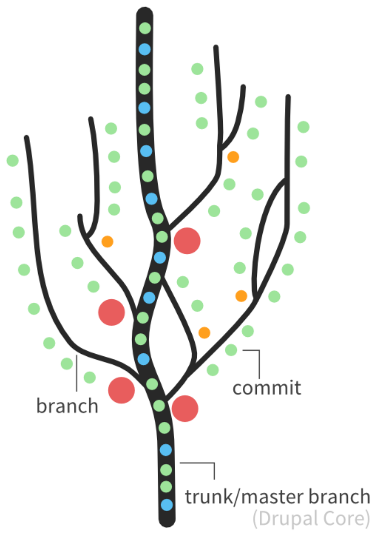


Figure 1: Trunk, branches, and merging visualization of the Drupal project

There are several widely used revision control systems including CVS (Concurrent Versions System), SVN (Apache Subversion), and Git. CVS is mostly legacy and not as widely used anymore. 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). They also check all changes into the main repository.

Git, however, is a *decentralized* system; multiple servers can act as repositories, but each copy on each developer's own machine is *also* a complete revision copy. Code commits are committed to the 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 itself is a version control system that can be installed on any server (UNL has a Git repository setup at `https://git.unl.edu`). However, we'll primarily focus on Github (https://github.com), the main website used by thousands of developers across the globe.

The rest of this tutorial will describe how to use Github for use in your courses and to manage and share your code among your peers for group assignments and projects.

1.1 Registering

You can register for a GitHub account at https://github.com/. However, it is *strongly recommended* that you get a free "student" account. A normal, free GitHub account does not allow you to create "private" repositories. Any code you push to GitHub is automatically public and accessible by anyone. This is okay in general, however many of your courses will have Academic Integrity policies that will require you to *not* share code. A student account allows you up to 5 private repositories (normally $7/month as of this writing) so that you can comply with Academic Integrity policies.

To get a student account first register at GitHub using an email account that ends in `.edu` (to "prove" you're a student). Then go to https://education.github.com/pack and register for a "student pack." Sign up early as some have reported long wait times to receive their student pack. The student pack contains a lot of other free and reduced cost software packages, tools and services that may be of interest.

1.2 Installing Git on Your Machine

If you want to use Git on your own personal machine, then you may need to install a Git client. There are many options out there and you are encouraged to explore them, however the following suggestions are all free and open source.

- Git has released its own graphical user interface clients which are available for free for both Windows and Mac:
  - Windows: https://windows.github.com/
  - Mac: https://mac.github.com

  See section 2 for instructions on using the client.
- If you will be using the Eclipse IDE (http://www.eclipse.org/downloads/) for development, the most recent versions already come with a Git client. Eclipse will work on any system. See Section 4 for using Git with Eclipse.
- If you use Windows and prefer to use a command line interface, you can download and install TortoiseGit (https://code.google.com/p/tortoisegit/) a Windows Shell Interface to Git. See Section 3 for using Git via the command line interface.
- If you use Mac and want the command line version of Git, you can download and

install here: http://www.git-scm.com/download/mac. Alternatively, you can install Git using a tool like MacPorts:
http://iamphioxus.org/2013/04/20/installing-git-via-macports-on-mac-osx/.
See Section 3 for using Git via the command line interface.

1.3 Creating a Repository on Github

You will eventually want to publish ("push") your project code to Github. To do this you'll first need to create a repository on Github's site:

1. Login to Github (https://github.com/) and click on the "repositories" tab.
2. Create a new repository (see Figure 2) with the name that will match your project folder (the names do not *have* to match, but it keeps things organized). Provide a short description and choose whether or not to make it public or private depending on whether or not you are allowed to share your code with your peers.

   You *may* choose to include a README file and/or establish a license (which creates a LICENSE file). However, for this tutorial we will assume that you start with an empty repo on Github. If you choose to create these files some extra steps may be necessary.

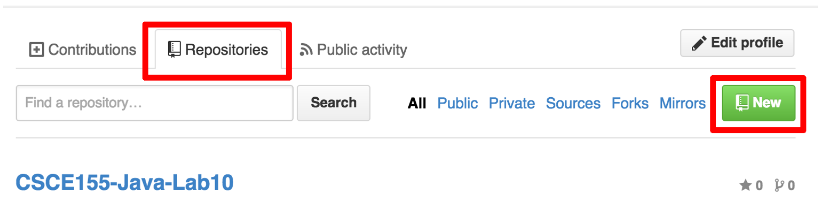


Figure 2: Creating a New Repository on GitHub

2 Using Git via Git's Clients

In this section we'll explore the basic uses of Git by using Git's client which provides a Graphical User Interface (GUI) to Git. A complete online help guide is available here: https://mac.github.com/help.html (Mac) and here: https://windows.github.com/help.html (Windows).

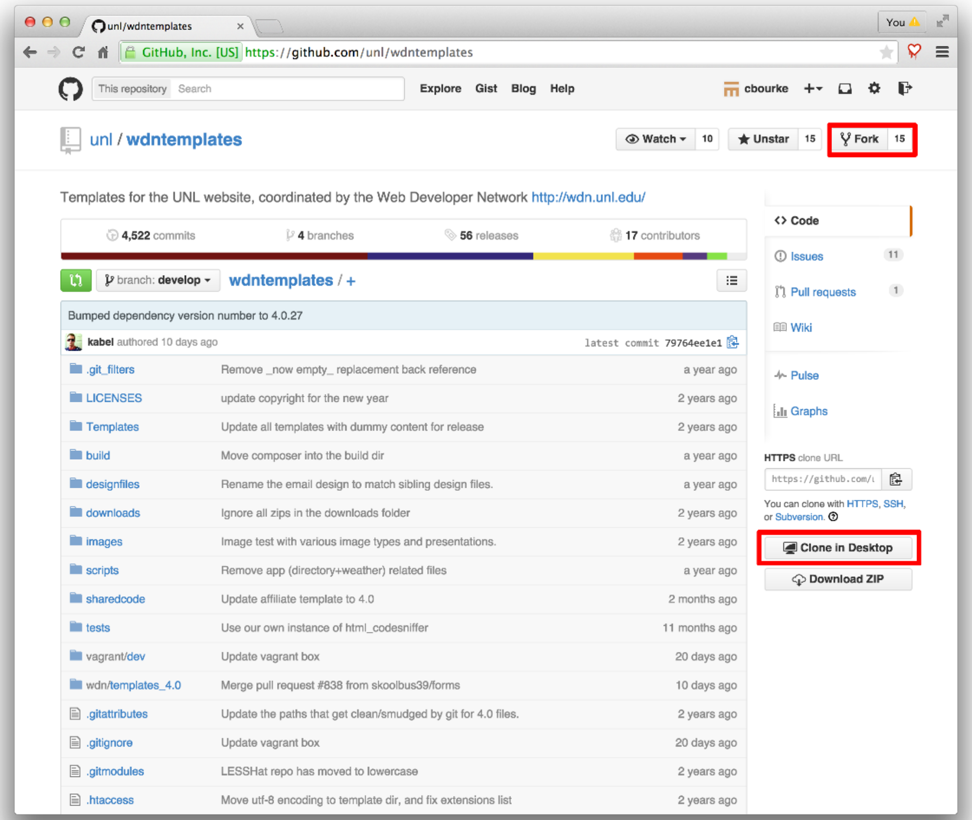
Advantage: nice, clean, intuitive interface with GitHub-style diff markup

Disadvantage: requires a separate client; some tasks are better done on GitHub.com or are difficult to do with the client alone.

Though the clients should be almost identical for Mac and Windows, there may be some slight differences; this tutorial was written using the Mac version.

2.1 Cloning an Existing Repository

To clone an existing repository hosted on GitHub, point your browser to its URL. On its page there will be several options to clone, fork or download the repository (see Figure 3).



Templates for the UNL website, coordinated by the Web Developer Network http://wdn.unl.edu/

Bumped dependency version number to 4.0.27

kabel authored 10 days ago   latest commit 79764ee1e1

.git_filters	Remove _now empty_ replacement back reference	a year ago
LICENSES	update copyright for the new year	2 years ago
Templates	Update all templates with dummy content for release	2 years ago
build	Move composer into the build dir	a year ago
designfiles	Rename the email design to match sibling design files.	a year ago
downloads	Ignore all zips in the downloads folder	2 years ago
images	Image test with various image types and presentations.	2 years ago
scripts	Remove app (directory+weather) related files	a year ago
sharedcode	Update affiliate template to 4.0	2 months ago
tests	Use our own instance of html_codesniffer	11 months ago
vagrant/dev	Update vagrant box	20 days ago
wdn/templates_4.0	Merge pull request #838 from skoolbus39/forms	10 days ago
.gitattributes	Update the paths that get clean/smudged by git for 4.0 files.	2 years ago
.gitignore	Update vagrant box	20 days ago
.gitmodules	LESSHat repo has moved to lowercase	2 years ago
.htaccess	Move utf-8 encoding to template dir, and fix extensions list	2 years ago

Figure 3: Forking and/or Cloning on GitHub.com

If you click the "Clone in Desktop" option, you'll be prompted to allow the GitHub client to open and clone the repository to your local file system (you will be prompted to indicate where unless you've setup a permanent clone path/directory). You will be able to make changes to your local copy but you will not be able to push changes to the original project unless you are a collaborator with write permission. However, you can create a new repository in your GitHub account and push the project back to your own repository.

A "fork" essentially does this in reverse. If you choose this option, a new repository will be created in your account and the project will be copied to this new repository. Then, in your Git client, you can clone it as a local copy to work on by clicking the "Add a repository" button in the Git client as depicted in Figure 4.

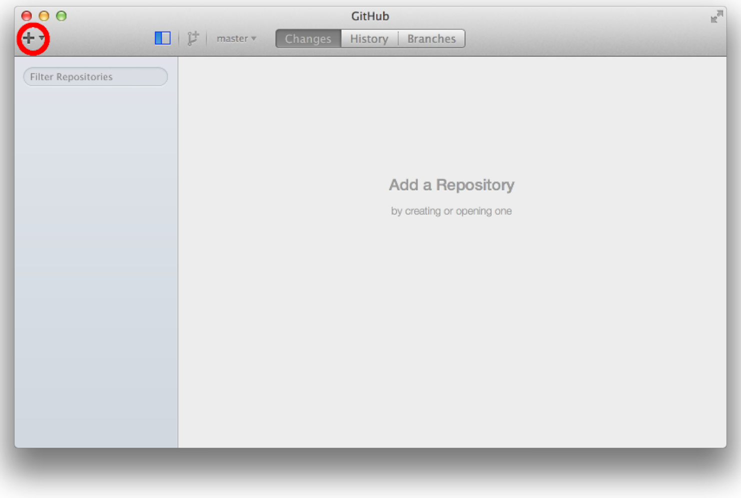
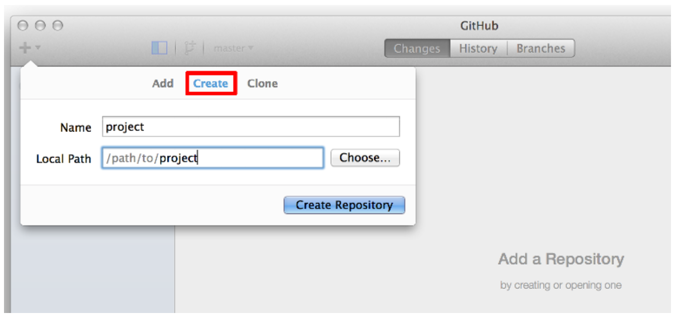


Figure 4: Cloning in the GitHub Client

2.2 Creating & Sharing Your Own Project

To share/publish a project to GitHub, you can start with an existing project or create a repository and then start working on your project.

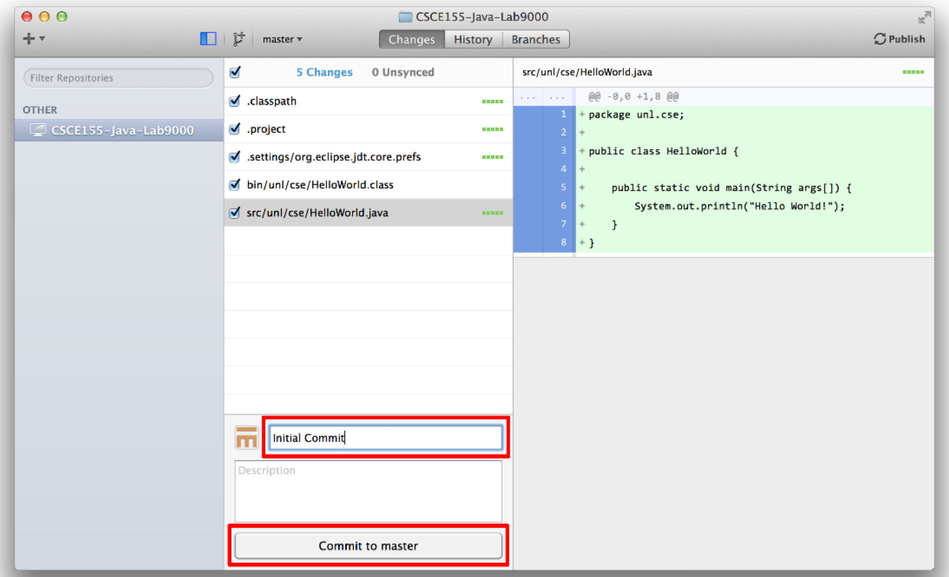
1. Open your Git hub client and click the "Add a repository" button as in Figure 4.
2. Select the "Create" tab and select the directory of the project you wish to create a repository with as in Figure 5.



Name: project
Local Path: /path/to/project

Figure 5: Creating a Repository in the GitHub Client

3. Upon success, the Git client should appear as in Figure 6; you can now make an initial commit by filling in the commit message and description and clicking "Commit to master"



.classpath
.project
.settings/org.eclipse.jdt.core.prefs
bin/unl/cse/HelloWorld.class
src/unl/cse/HelloWorld.java

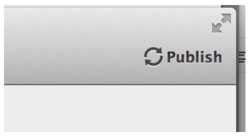
src/unl/cse/HelloWorld.java

```
@@ -0,0 +1,8 @@
+ package unl.cse;
+
+ public class HelloWorld {
+
+     public static void main(String args[]) {
+         System.out.println("Hello World!");
+     }
+ }
```

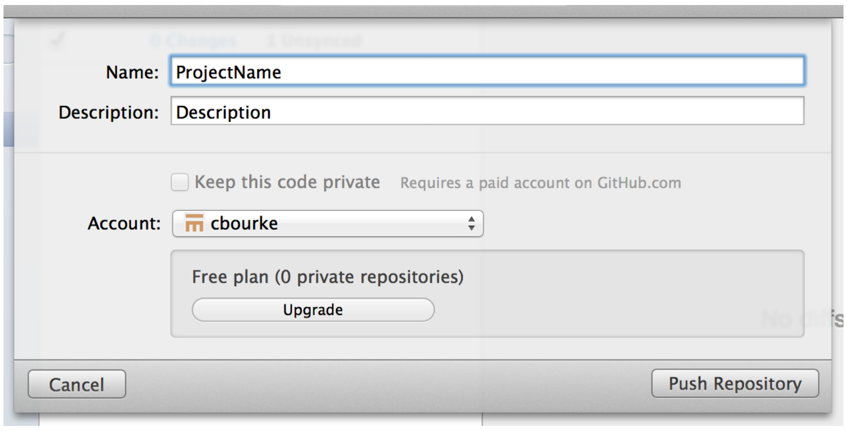
Initial Commit

Figure 6: Committing in the GitHub Client

4. You can now "publish" your repository to GitHub by clicking the "Publish" icon in the top right of the Git client



5. This opens a new dialog where you can specify the name and description of the project as it will appear on GitHub.

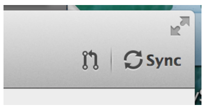


Name: ProjectName
Description: Description
Account: cbourke
Free plan (0 private repositories)

To finish up, click the "Push Repository" button and observe your new project on GitHub

2.3 Making, Committing & Pushing Changes

You can make changes to your local project and the changes will automatically be detected in the Git client. As in the previous step-by-step process, you can select a subset of changes to commit. Once committed, you can push the changes by clicking the "Sync" icon at the top right:



3 Using Git via the Command Line

In this section we'll explore the basic uses of git by using the Command Line Interface (CLI) utilities. This section assumes basic familiarity with the unix command line.

Advantage: quick, straightforward access to git

Disadvantage: requires good working knowledge of the command line; proficiency takes longer

3.1 Cloning an Existing Repository

The first thing you may want to do is to "clone" an existing project that has already been published on Github. You may do this if your instructor has provided some code for assignments or labs.

To start, you can verify that git has been properly installed on your machine by executing the following:

```
git --version
```

which may output something like:

```
git version 1.9.5 (Apple Git-50.3)
```

though your specific version may differ. However, if this command does not work, you will need to troubleshoot your installation before continuing.

1. Move to the directory where you want the project files to be placed. Usually this is your "workspace" folder.
2. Execute the following command:

   ```
   git clone https://github.com/project/url
   ```

where the URL is replaced with the URL of the project that you want to clone. For example, if you wanted to clone Lab 01 for CSCE 155E/H, which has the url https://github.com/cbourke/CSCE155-C-Lab01, you would execute the command:

```
git clone https://github.com/cbourke/CSCE155-C-Lab01
```

3. If successful, you should see a message like the following:

```
Cloning into 'CSCE155-C-Lab01'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 9 (delta 2), reused 9 (delta 2), pack-reused 0
Unpacking objects: 100% (9/9), done.
```

   A new directory/file structure should now exist in your directory and you can start working with/editing the files.

If the owner of the repository that you just cloned ever makes changes, you can "pull" those changes from the repository by using `git pull` to pull all changes.

3.2 Creating & Sharing Your Own Project

1. Before continuing you will need to create a repository on Github. To do this, refer to the steps in Section 1.3.
2. Setup your *local* repository from the command line by going to your project directory and executing the following commands (approximate expected outputs have been included here:
   - Initialize your directory using:

     ```
     git init
     ```

     which should have output similar to:

     ```
     Initialized empty Git repository in /your/directory/foo/.git/
     ```

   - Add all files, directories and subdirectories to your git *index* using:

     ```
     git add --all
     ```

   - Commit your files using the following. The `-m` specifies that a commit **m**essage follows:

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

     Output should resemble:

```
[master (root-commit) 7a3fb99] Initial Commit
 2 files changed, 24 insertions(+)
 create mode 100644 README.md
 create mode 100755 hello.c
```

- Associate your repo with the repo on GitHub using the following command:

  `git remote add origin https://github.com/login/PROJECT.git`

  where the URL is replaced with the URL for your project.

- Push your commit to the remote repository using the following command:

  `git push -u origin master`

  Output should resemble something like:

  ```
  Counting objects: 4, done.
  Delta compression using up to 8 threads.
  Compressing objects: 100% (4/4), done.
  Writing objects: 100% (4/4), 577 bytes | 0 bytes/s, done.
  Total 4 (delta 0), reused 0 (delta 0)
  To https://github.com/login/PROJECT.git
   * [new branch]      master -> master
  Branch master set up to track remote branch master from origin.
  ```

3. Refresh your browser's Github page to verify the changes were pushed remotely

3.3 Making, Committing & Pushing Changes

Now that your code is committed to Github's servers, you'll eventually want to make changes to current files and/or add/remove files and commit these changes. Once you have made your changes, you can essentially repeat part of the process above:

```
git add --all
git commit -m "Update Message"
git push -u origin master
```

Note:

- The `"Update Message"` should be more descriptive: it is used to document the changes you've made for this commit. It is best practice to be as descriptive as possible as to your changes.
- The `git add --all` command adds all files in the current directory as well as all of its subdirectories to the commit index. If you want to be more precise and intentional, you can add individual files using `git add foo.txt`, etc.

4 Using Git via the Eclipse

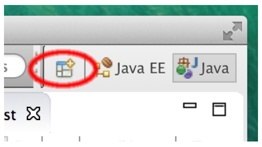
Eclipse is an industry-standard Integrated Development Environment (IDE) that integrates code editors (with markup) and build tools to streamline the development process. There are many plugins and utilities that can be used with Eclipse to interact with git. However, the latest version of Eclipse (Luna as of this writing) supports git natively. The process below describes how to use this functionality.

Advantage: using a single IDE/interface keeps things simple

Disadvantage: interface can be a bit clunky; it is more difficult to see differences

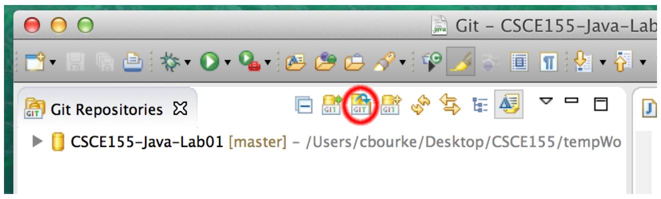
4.1 Cloning an Existing Repository

1. First we need a Git *perspective* (a context in the Eclipse User Interface that will allow us to work with Git). To open the Git perspective, click on the "Open Perspective" tab in the upper right:

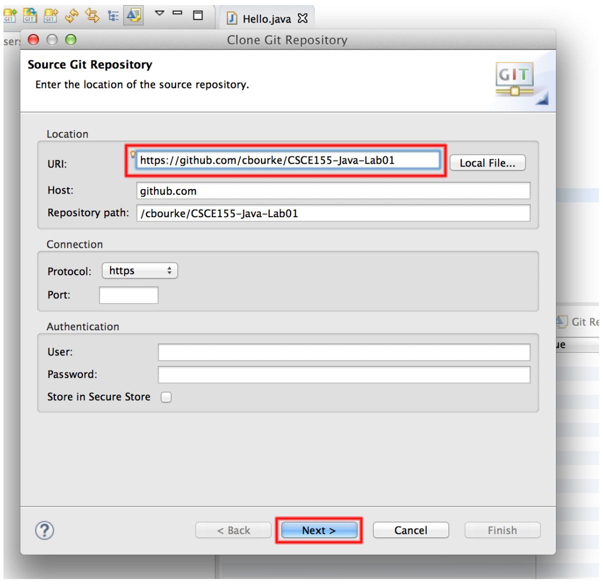
   

   Select "Git" from the menu and click OK

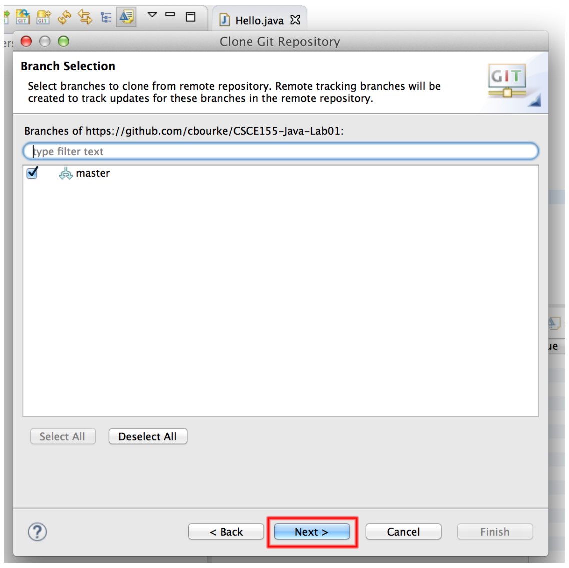
2. Click the "Clone a Git repository" in the Git Repositories navigation menu:

   
   Git – CSCE155-Java-Lab
   Git Repositories
   CSCE155-Java-Lab01 [master] – /Users/cbourke/Desktop/CSCE155/tempWo

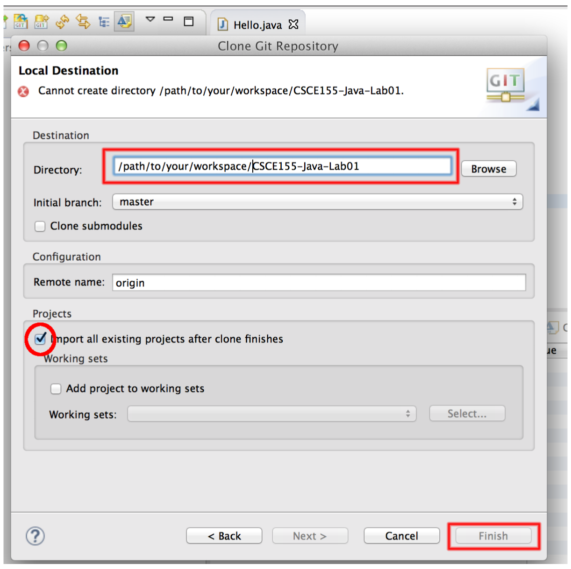
3. Copy/past or type into the URI field, the URL of the project that you want to clone. For example, if you wanted to clone Lab 01 for CSCE 155E/H, you would use the URL `https://github.com/cbourke/CSCE155-Java-Lab01` Then click "Next"



4. Once Eclipse has grabbed the project, the "master" branch should be selected (checkbox); click "Next" again.


Hello.java
Clone Git Repository
Branch Selection
Select branches to clone from remote repository. Remote tracking branches will be created to track updates for these branches in the remote repository.
Branches of https://github.com/cbourke/CSCE155-Java-Lab01:
type filter text
✓ master
Select All
Deselect All
< Back
Next >
Cancel
Finish

5. Select the directory where you want your project to be saved. Caution: the default option may not correspond to your default workspace. You may want to change it to your workspace, but the choice is yours. Also mark the "Import all existing projects after clone finishes" checkbox option or you will need to manually import the cloned project into Eclipse.


Clone Git Repository

Local Destination

Cannot create directory /path/to/your/workspace/CSCE155-Java-Lab01.

Destination

Directory: /path/to/your/workspace/CSCE155-Java-Lab01

Initial branch: master

Clone submodules

Configuration

Remote name: origin

Projects

[✓] Import all existing projects after clone finishes

Working sets

Add project to working sets

Working sets:

6. Switch back to your Java or JavaEE perspective and you can see your cloned project.

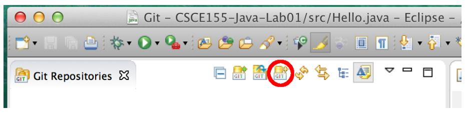
Note: this process assumes that the project you are cloning originated from an Eclipse project. Eclipse expects that files be organized in a particular way and that configuration files are present that describe how the project is setup. If the project was not an Eclipse project, you'll need to clone/setup the project in Eclipse manually.

If the owner of the repository that you just cloned ever makes changes, you can "pull" those changes from the repository by right-clicking the repo in the Git Perspective and selecting "Pull."

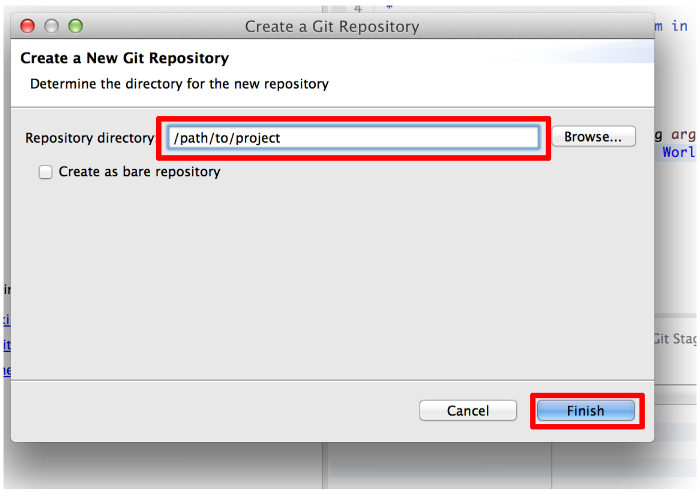
4.2 Creating & Sharing Your Own Project

Create and develop your own project in Eclipse and get it to the point where you want to make an initial commit and push to Github. Then do the following:

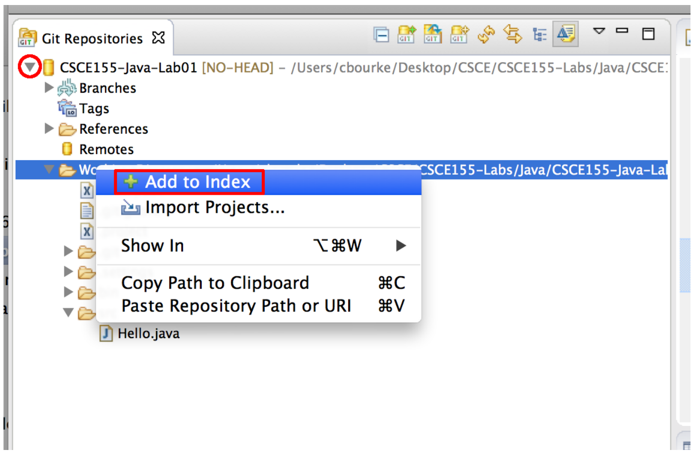
1. Before continuing you will need to create a repository on Github. To do this, refer to the steps in Section 1.3.
2. Open the Git Perspective in Eclipse.
3. Click the "Create a new repository and add it to this view" icon:



4. Select the project folder for the Eclipse project you want to add as a git repo

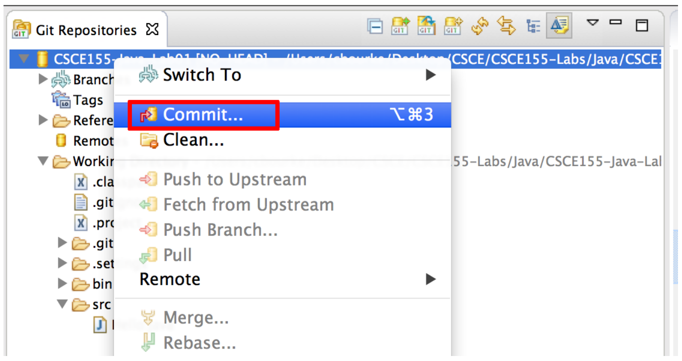

Create a Git Repository
Create a New Git Repository
Determine the directory for the new repository
Repository directory
/path/to/project
Browse...
Create as bare repository
Cancel
Finish

5. Expand the directory structures and select the file(s) you wish to add to the index (that is, the files you want to "stage" for your commit), right click and "add to index".

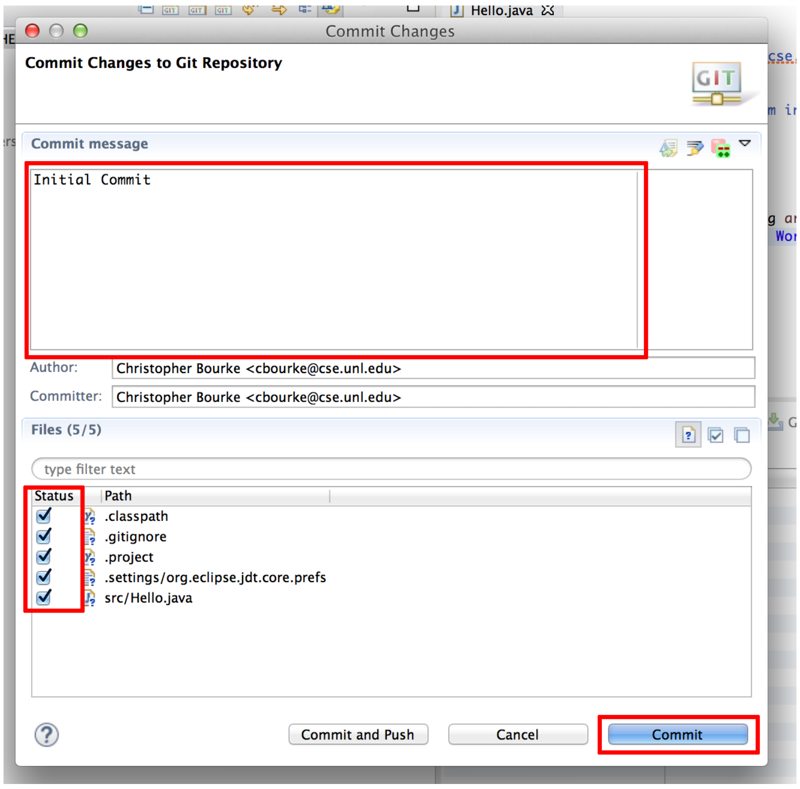

Git Repositories
CSCE155-Java-Lab01 [NO-HEAD] – /Users/cbourke/Desktop/CSCE/CSCE155-Labs/Java/CSCE
Branches
Tags
References
Remotes
Add to Index
Import Projects...
Show In ⌥⌘W
Copy Path to Clipboard ⌘C
Paste Repository Path or URI ⌘V
Hello.java

Note: adding a folder (or the entire working directory) to git's index adds all files and subfolders within that folder. You can instead, highlight individual files if you want to be more precise or intentional with each commit.

6. Right click the repo and select "Commit.."



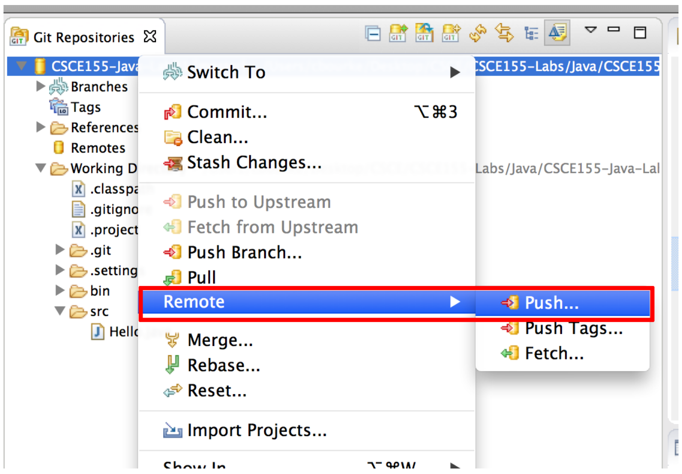
7. Enter a commit message; `Initial Commit` is good for the first commit, but each commit message should be descriptive and document the changes that have been made. Select the checkboxes of all the files you wish to commit. Click "Commit"



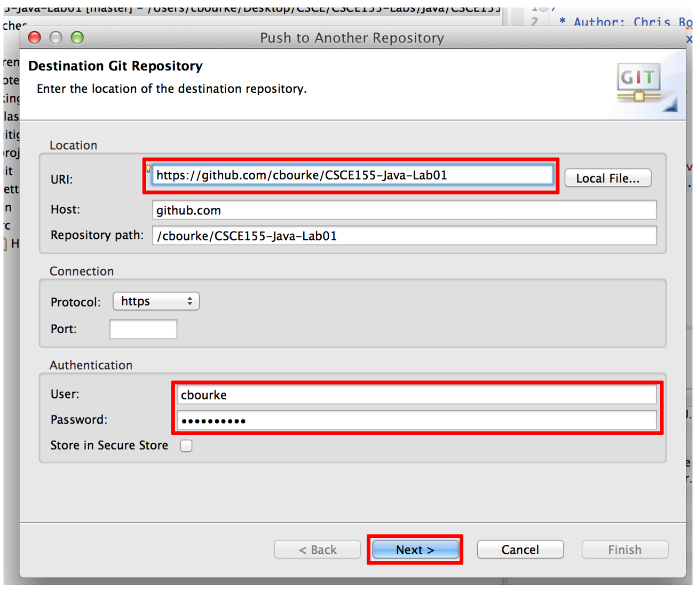
Commit Changes
Commit Changes to Git Repository
Commit message
Initial Commit
Author: Christopher Bourke <cbourke@cse.unl.edu>
Committer: Christopher Bourke <cbourke@cse.unl.edu>
Files (5/5)
type filter text
Status
Path
.classpath
.gitignore
.project
.settings/org.eclipse.jdt.core.prefs
src/Hello.java
Commit and Push
Cancel
Commit

Note: you can see the differences in each file if you double click the file.

8. Right click the repo again and select Remote → Push



9. Enter the URL of the repo you created on http://github.com. Then enter your Github user name and password.


Push to Another Repository

Destination Git Repository

Enter the location of the destination repository.

Location

URI: https://github.com/cbourke/CSCE155-Java-Lab01

Host: github.com

Repository path: /cbourke/CSCE155-Java-Lab01

Connection

Protocol: https

Port:

Authentication

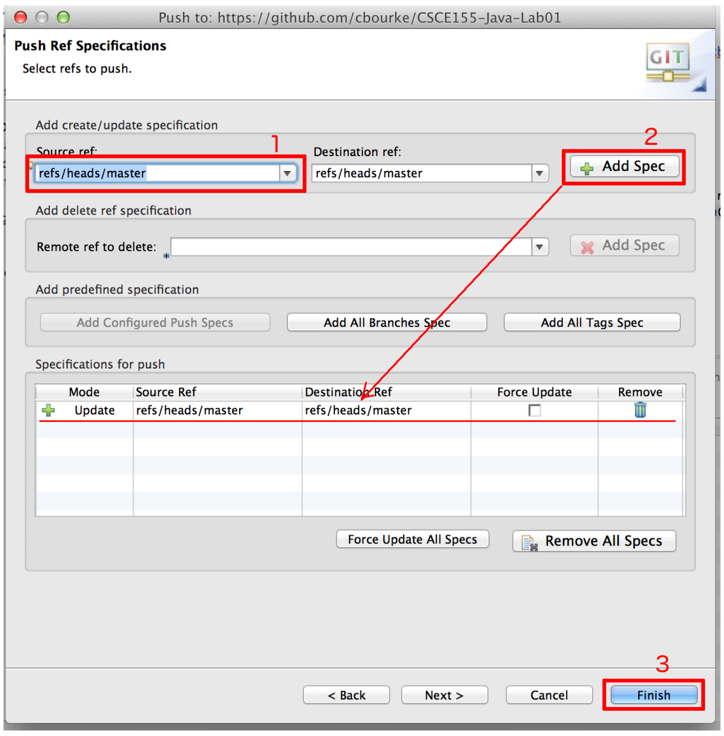
User: cbourke

Password: ••••••••••

Store in Secure Store

Note: this will only work for *your* repositories or repos on which you are a collaborator and have been granted write access.

10. Select "master" from the "Source ref" menu and click "Add Spec". The branch should now appear in the "Specifications for push" menu. You can now click "Finish".


Push to: https://github.com/cbourke/CSCE155-Java-Lab01

Push Ref Specifications

Select refs to push.

Add create/update specification

Source ref: refs/heads/master

Destination ref: refs/heads/master

Add delete ref specification

Remote ref to delete:

Add predefined specification

Specifications for push

Mode	Source Ref	Destination Ref	Force Update	Remove
Update	refs/heads/master	refs/heads/master	☐	

11. If successful, a new dialog confirming the push should appear and your files should be updated on Github.

4.3 Making, Committing & Pushing Changes

Now that your code is committed to Github's servers, you'll eventually want to make changes to current files and/or add/remove files and commit these changes. Once you have made your changes, you can essentially repeat part of the process above; however, steps 1–4 will not be necessary.

Note that you don't need to push every commit to Github. You can make as many local commits as you want. The entire history and all the diffs (differences) are tracked between each commit.

5 Working With Others

As previously mentioned, you will not be able to pull from a private repo. Nor will you be able to push to a repo that you do not own or that you are not a collaborator of. However, you will want to do this when you work with other individuals either as partners for an assignment or as a group in a group project (assuming that your course instructor's policies allow this of course).

To do this, simply create a repository that will be used by the group as outlined by one of the methods above. You can then grant read/write access to your others by making them *collaborators* on the project. You can easily do this in Github by following the instructions at this link:

`https://help.github.com/articles/adding-collaborators-to-a-personal-repository/`

Once you've all been added, each of you should be able to push/pull from the same repository.

6 Resources

This tutorial is only an introduction to get you started. Git is a complex tool that takes a while to master. There are many other issues (branching, resolving conflicts, merging, etc.) that will you eventually encounter. Below are some additional resources that may help you.

- Git Reference: `http://gitref.org/`
- Git Glossary: `https://help.github.com/articles/github-glossary/`
- GitHub's walkthrough for both Windows and Mac:
  `https://help.github.com/articles/set-up-git/`
- GitHub Desktop (released 2015/08/12):

https://github.com/blog/2046-github-desktop-is-now-available

- GitHub's Tutorial/Challenge:
  https://try.github.io/levels/1/challenges/1
- Workflow tutorial using git:
  https://www.atlassian.com/git/tutorials/comparing-workflows/centralized-workflow
- Git from the inside out:
  http://maryrosecook.com/blog/post/git-from-the-inside-out
- Github Video Tutorial: https://www.youtube.com/watch?v=0fKg7e37bQE
- Lifehacker tutorial on using git:
  http://lifehacker.com/5983680/how-the-heck-do-i-use-github
- Eclipse Mars now comes with Git Flow! http://eclipsesource.com/blogs/2015/06/22/git-flow-top-eclipse-mars-feature-3/
- 19 Tips for Everyday Git Use http://www.alexkras.com/19-git-tips-for-everyday-use/

7 Using UNL's GitLab

Git software is free and open source (FOSS), but GitHub.com is a private company that has to pay to keep the lights on and make a profit. The software that GitHub runs on is closed and proprietary; they could choose to shut off access tomorrow. The venture capitalists that back them could decide to call in their chips and completely change their business model. Even as it stands, free GitHub accounts are limited to public repositories only. It is important to consider the alternatives.

- Bitbucket (https://bitbucket.org/) allows free unlimited private repositories but limits the number of collaborators for free accounts (to 5).
- Visual Studio Online (https://www.visualstudio.com/en-us/version-control-vs) offers free, unlimited private repos with additional features from Microsoft.
- GitLab (https://about.gitlab.com/) is a GitHub like solution that offers free, unlimited private repos hosted on their servers.

There are many more alternatives each with their own features and business model.

UNL hosts its own GitLab instance. GitLab is software similar to GitHub but is free and open source, offering unlimited private repositories and unlimited users. UNL's instance can be found at https://git.unl.edu and is free for all students, faculty, and staff.

You can login with your My.UNL login credentials (the same that you use for Blackboard). The look and feel will be very similar to GitHub. To get started, GitLab has an introductory video available to help you get started: https://about.gitlab.com/

2014/02/26/getting-started-with-gitlab/. The full documentation can be found here: http://doc.gitlab.com/ce/.

1. Create a repository on GitLab, make it private if you don't want everyone to be able to access it.

2. Initialize the repository on your local file system and make an initial commit as previously:

   ```
   git init
   git add --all
   git commit -m "Initial Commit"
   git remote add origin git@git.unl.edu:login/projectName.git
   ```

3. You won't be able to push your project to GitLab though as UNL's GitLab does not support password authentication (since to login to GitLab, you use a Single Sign On (SSO) service). Instead, you need to generate an SSH key and add it to your profile on GitLab.

   a) Follow this documentation (Windows and Mac) to generate an SSH key: http://doc.gitlab.com/ce/ssh/README.html.

   b) Go to GitLab and click "Profile Settings" and go to the "SSH Keys" tab. Click "Add SSH Key" and copy and paste the SSH key.

4. Now you can push your repo to GitLab:

   ```
   git push -u origin master
   ```

   likely, it will prompt you for your passphrase (if you established one, which you should have!) and it will push the code.

To collaborate with other students, you can do the following.

1. Navigate to the repository you wish to collaborate on.

2. Click the "Settings" tab.

3. Click the "Members" navigation tab on the left

4. Click "New project member"

5. Add a user by searching for their username or actual name (they'll need to be registered first).

6. Select the appropriate level of access (likely "Master" to allow them to commit changes); full documentation on what each level can do is available here: https://gitlab.com/gitlab-org/gitlab-ce/blob/master/doc/permissions/permissions.md

There are many other alternatives as well; here is a pretty good comparison: http://en.wikipedia.org/wiki/Comparison_of_source_code_hosting_facilities.