



Version Control and Code Repositories

Version Control and Code Repositories

- **Version Control:** a system that records changes to a file or set of files over time so that you can recall specific versions later. **Git** is an example of version control.
- **Code Repository:** a file archive and a web hosting facility (in this context) where source code is kept, either publicly or privately. A code repository includes version control. **GitHub** is an example of a Code Repository.

Do not confuse version control utilities such as Git with repositories such as BitBucket and GitHub. Git is version control. GitHub and BitBucket are repositories where you archive your code and work collaboratively.

This tutorial assumes you start with a local project you want to add to a remote repository. You can also go the reverse, remote to local. You need to know both! In class we will usually do remote to local, but we will start here with local to remote.

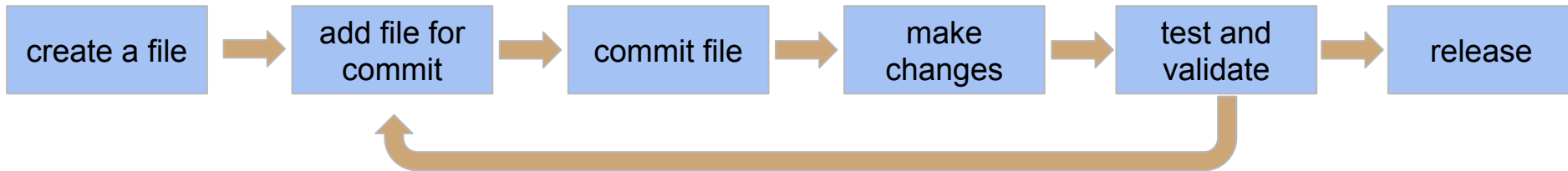
Version Control - Git

- Version control software allows you to have “versions” of a project, which show the changes that were made to the code over time, and allows you to backtrack if necessary and undo those changes.
- Git is one of the most popular version control systems.
- Git is *distributed* version control in a *client-server* model (there are other kinds).
- Developers run a local git instance which *may* also be connected to a central git repository (usually this is the case, but doesn't have to be).
- In this *distributed client-server* model, clients don't check out the current version of the files – you will mirror the entire version history.
- Each developer always has a complete copy of everything.

What's the Purpose? How does it work?

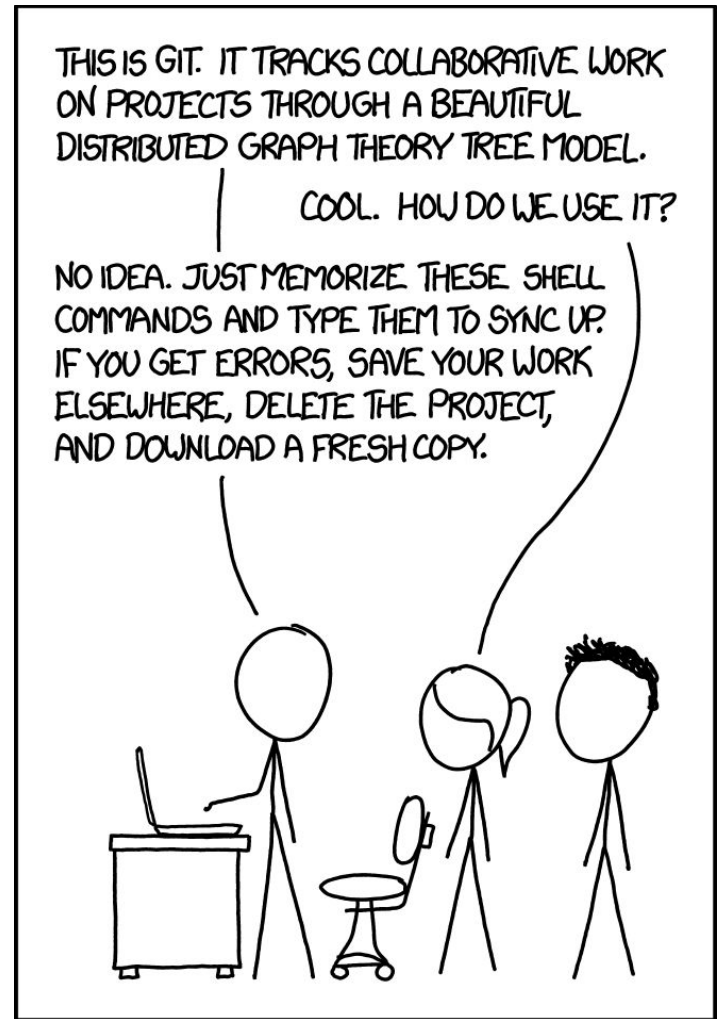
Once your project is set up (after this tutorial) you will have...

- A system to track your changes and see where you were at the last step.
- A copy of all your work in a safe place where you can always see your history.
- A place where you can work together with other developers.
- A way you can work in multiple locations (like school labs and home) without having to carry files and move files around.
- Once you are set up, you will work like this...



It's funny because it's true.

- Git and GitHub can be very complex when used fully.
- However, for simple operations and most work, you only need a few simple commands and a little knowledge.
- If you want to eventually become a Git Master... Good luck (you will need it).



Tutorial and Examples

This tutorial assumes the following:

- You have a local project and you want to add Git and a remote repository.
- The examples show using the command prompt with some *optional* IDE screenshots for reference (but you do **NOT** need an IDE- see below).
- All examples are in Linux because... Linux.
- Git is a command line tool, however most IDEs will have a full suite of Git and repository tools built-in. You can work 100% at the command prompt or 100% in an IDE or mix-and-match them as you like.
- You do NOT need an IDE for this tutorial and if you are a beginner, you probably should NOT use an IDE. The IDE views shown here are only for reference. If you are not using an IDE, just skip the slides that say "IDE view."

Getting Started

- If you are working on your own computer, you need to [install git](#) first.
- This tutorial assumes you have local files and want to set up a remote Repo. In other words, you started a project on your own computer (local) and you want to create a Repo on GitHub (remote) to track, store, and manage your project.
- Start by making a directory (folder) on your computer somewhere you can easily access from the command prompt.
- Go to that directory within a command prompt.
- Make code files in that directory. For this tutorial, you can just put any few text files in there, it doesn't matter what they are or what they contain.
- The rest of the tutorial assumes you are at the command prompt inside the root of your project (i.e. the folder you just made).

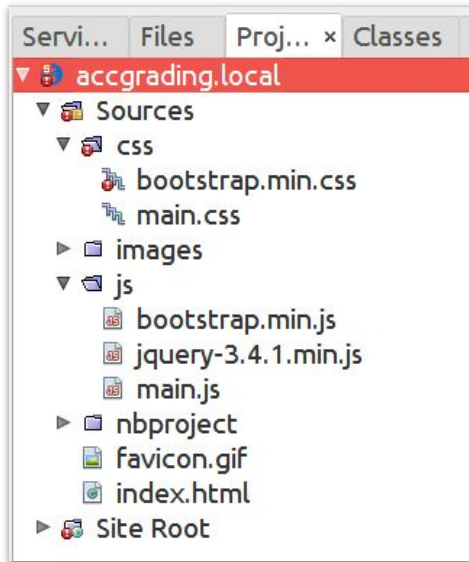
Initialize Git

- Inside the **root** of your project, at the command prompt, type ***git init***
- This creates an EMPTY local version control system.
- You will now see a **.git** folder. This is your local version control system.
- Notice in the listing to the left the difference between before and after ***git init***.
- Do not touch **.git/**, it's not for you, it's for Git.

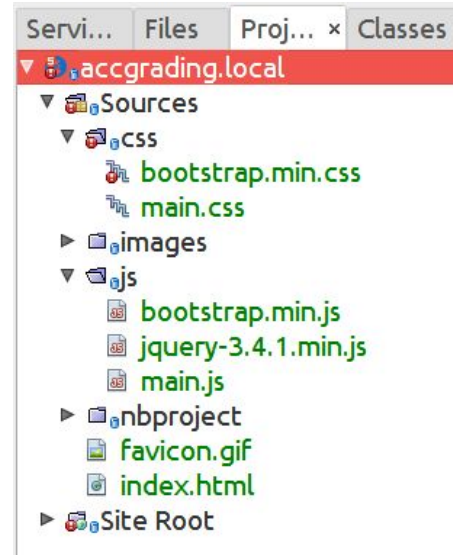
```
alex@alex-laptop:~/Desktop/demo$ ll
total 92
drwxrwxr-x 4 alex alex 4096 Jan 12 10:50 ./
drwxr-x--- 4 alex alex 4096 Jan 12 10:49 ../
-rwxrwxr-x 1 alex alex 17320 Jan 12 10:38 a.out*
-rw-rw-r-- 1 alex alex 84 Jan 12 10:37 main.cpp
-rw-rw-r-- 1 alex alex 43 Jan 12 10:38 main.h
drwxrwxr-x 3 alex alex 4096 Jan 12 10:39 nbproject/
-rw-rw-r-- 1 alex alex 35 Jan 12 10:37 sometextfile.txt
alex@alex-laptop:~/Desktop/demo$ git init
Initialized empty Git repository in /home/alex/Desktop/demo
alex@alex-laptop:~/Desktop/demo$ ll
total 92
drwxrwxr-x 4 alex alex 4096 Jan 12 10:50 ./
drwxr-x--- 4 alex alex 4096 Jan 12 10:49 ../
-rwxrwxr-x 1 alex alex 17320 Jan 12 10:38 a.out*
drwxrwxr-x 8 alex alex 4096 Jan 12 10:54 .git/
-rw-rw-r-- 1 alex alex 84 Jan 12 10:37 main.cpp
-rw-rw-r-- 1 alex alex 43 Jan 12 10:38 main.h
drwxrwxr-x 3 alex alex 4096 Jan 12 10:39 nbproject/
-rw-rw-r-- 1 alex alex 35 Jan 12 10:37 sometextfile.txt
alex@alex-laptop:~/Desktop/demo$ █
```


IDE view

- If you look at your project in an IDE you will see the files are now color coded. Each IDE will have different colors. At this point, probably red or green.
- This indicates they have **not** been added to version control.
- **DO NOT ADD THEM** at this point. You want to create a **.gitignore** first because there are some files you do not want to add to the VCS.



git init



Create .gitignore

- .gitignore is just a text file. Make it with any text editor but it must be named with a leading . and no extension.
- You must have a .gitignore to tell Git what **NOT** to track. You almost never want to track project files, files with sensitive information (eg .htaccess files), and other files that are not part of your application's code.
- There two ways to do this - **pick ONE method**
 - Ignore everything, then add back what you want to track.
 - Just list what you want to ignore directly.
- This step is very important. Once something is in a repo, it's ***forever***.

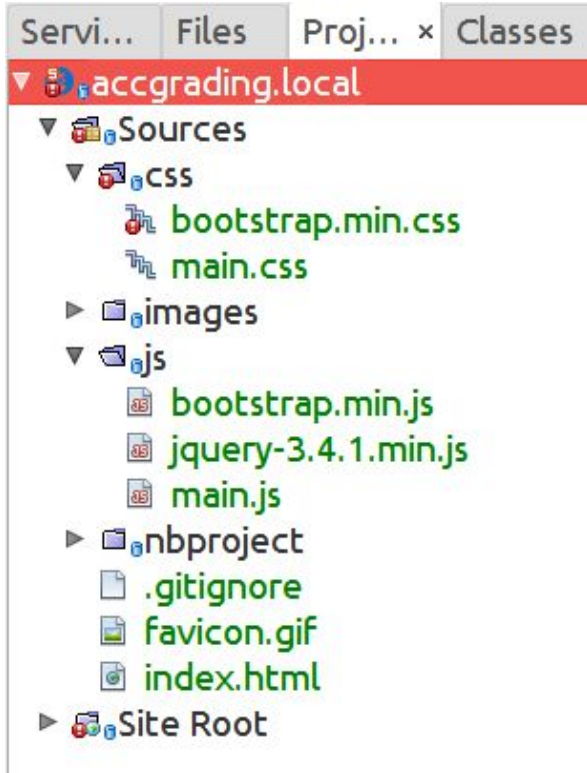
Two .gitignore approaches that do the same thing

```
1 #ignore everything
2 *
3
4 #unignore what we want to track
5 !.gitignore
6
7 !*images/
8 !*css/
9 !*js/
10
11 !*.png
12 !*.jpg
13 !*.gif
14 !*.ico
15
16 !*.html
17 !*.css
18 !*.js
```

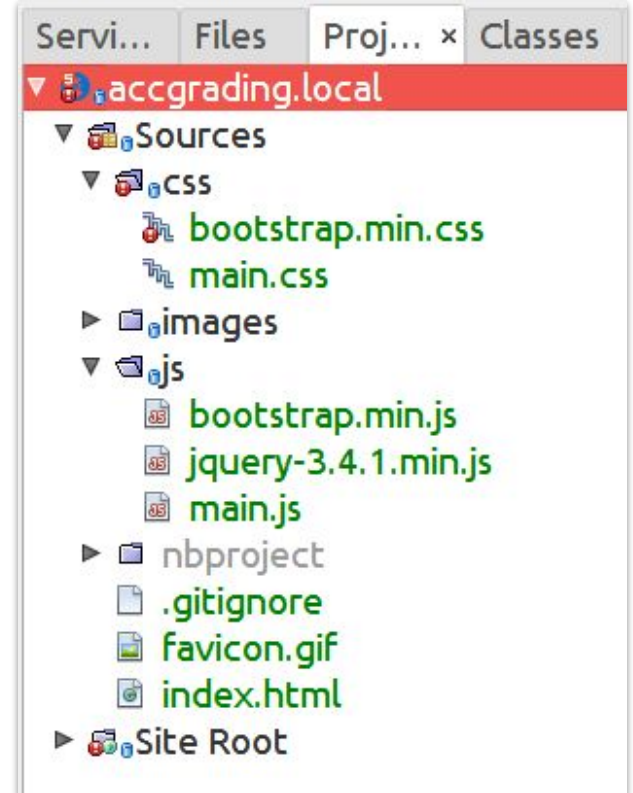
```
1 # directly ignore what we do NOT want to track
2
3 nbproject/
4 nbproject/*
5 *.out
6
7 # this seems easier but it's not necessarily
8 # easier. it depends on project complexity
9 # and person preference. usually this method
10 # starts out easier, and gets more and more
11 # complex, but the other method starts out
12 # complex, and then is easier to maintain
13 # in the long run.
```

Note: you may or may not want to track .gitignore itself. This is up to you.

IDE View



Notice the nbproject directory is now ignored. This is an IDE directory and we do not usually commit those.



Add files with 'git add'

- Now you can add files to your version control.
- Files must be added before they can be committed to be tracked in version control. Example command: **git add file1.ext file2.ext**
- Only add the files you know you want to commit
- NEVER use complete wildcards like **git add .** or **git add ***
- This example shows adding the files and then listing them with **git ls-files** to make sure we added what we think we added. Also, **git status** is shown.

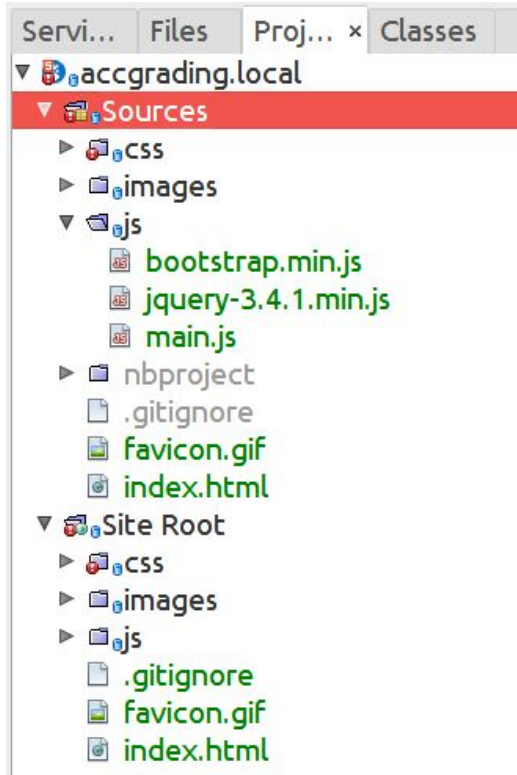
```
alex@alex-laptop:~/Desktop/demo$ ll
total 88
drwxrwxr-x 3 alex alex 4096 Jan 12 11:21 ./
drwxr-x--- 5 alex alex 4096 Jan 12 11:15 ../
-rwxrwxr-x 1 alex alex 17320 Jan 12 11:19 a.out*
drwxrwxr-x 7 alex alex 4096 Jan 12 11:21 .git/
-rw-rw-r-- 1 alex alex 15 Jan 12 11:20 .gitignore
-rw-rw-r-- 1 alex alex 84 Jan 12 11:19 main.cpp
-rw-rw-r-- 1 alex alex 43 Jan 12 11:19 main.h
-rw-rw-r-- 1 alex alex 35 Jan 12 11:19 sometextfile.txt
alex@alex-laptop:~/Desktop/demo$ git add main.cpp main.h
alex@alex-laptop:~/Desktop/demo$ git ls-files
main.cpp
main.h
alex@alex-laptop:~/Desktop/demo$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   main.cpp
    new file:   main.h

alex@alex-laptop:~/Desktop/demo$ █
```

IDE View



This view does not look any different because in Netbeans (shown here) *not-added* files versus *added-but-not-committed* files both show in green. Most other IDEs will show red for *not-added* and green for *added-but-not-committed*.

Commit Files

- The commit is what locks in changes. You will do this each time you are done with some logical portion of code.
- Commit in small batches corresponding to one logical change.
- Always include an intelligent commit message explaining the reason for the commit. In this case our reason is “initial commit.”
- Example: **git commit -m “initial commit”**

```
alex@alex-laptop:~/Desktop/demo$ git commit -m "initial commit"
[master (root-commit) cbd2cb1] initial commit
 2 files changed, 12 insertions(+)
 create mode 100644 main.cpp
 create mode 100644 main.h
alex@alex-laptop:~/Desktop/demo$ □
```

Master versus Main

At this point you *may* have to do something unusual. First check your branch with the command **git branch**. If you see *** master** as the response you need to fix this by renaming “*master*” to “*main*”. To do this execute the command **git branch -M main**. Then execute the command **git branch** again to check it and you should get *** main** as the response.

This extra step is because the default name for the main branch has recently been deprecated from “*master*” to “*main*.” You can [read more about this change](#). This will cause some confusion but you can always check your branches by using the **git branch** command and fix it. Over time, all “*master*” branches will be renamed to “*main*” and this step will go away.

Commit Files - small, often, and smart!

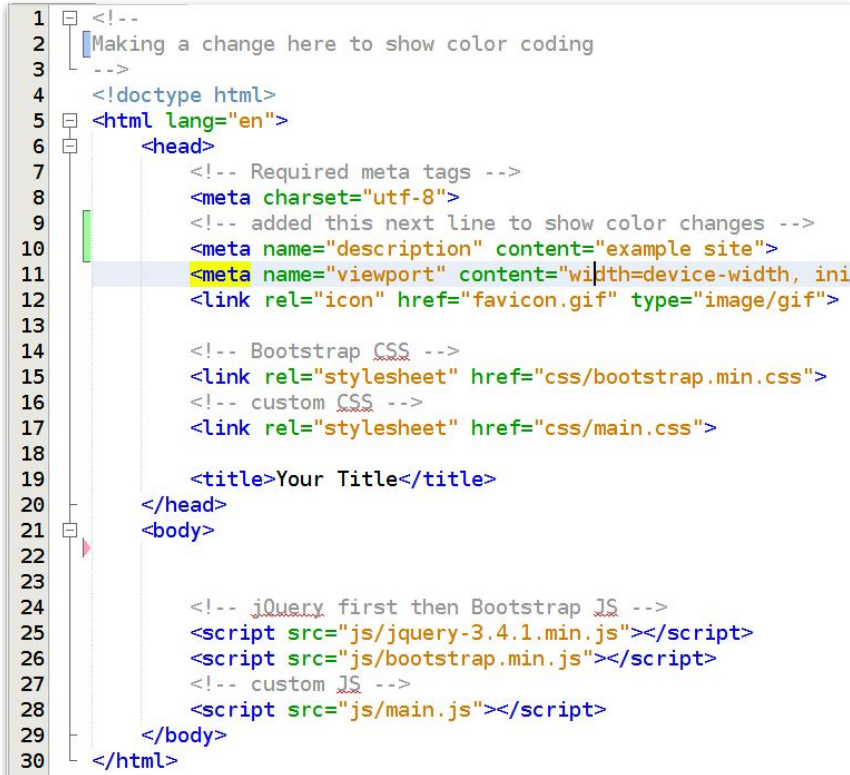
- **Small:** Commit with each small, self contained, logical change. For example:
 - one function
 - one bug fix
 - one logical self contained portion of code
 - one file or function or ADT stub
- **Often:** Since you commit small, you will commit often. Get used to it.
- **Smart:** Make descriptive commit messages that explain what has changed and what you did since the last commit of that file(s).
 - **Example:** *"Fixed one-off error in the loop in the displayArray() method. On 34, changed <= to <"*
 - **Do not ever** make commit messages like this:
 - *"done"*
 - *"bug fix"*
 - *"turning in"*

Local Done!

- At this point your local version control is set up. You could stop here and only work locally. Even if you don't need a remote repo, you should ALWAYS do at least this much.
- This will allow you to track changes, revert code if needed, and compare versions (known as a *diff*).
- However this will not give you a backup (remote repo) or allow you to collaborate with others. For that you need a remote repo like [Github](#).

The next three slides will show you how to work with local only set-up, then subsequent slide will discuss remote repositories.

IDE view (as you code)



```
1 <!--
2 Making a change here to show color coding
3 -->
4 <!doctype html>
5 <html lang="en">
6   <head>
7     <!-- Required meta tags -->
8     <meta charset="utf-8">
9     <!-- added this next line to show color changes -->
10    <meta name="description" content="example site">
11    <meta name="viewport" content="width=device-width, ini
12    <link rel="icon" href="favicon.gif" type="image/gif">
13
14    <!-- Bootstrap CSS -->
15    <link rel="stylesheet" href="css/bootstrap.min.css">
16    <!-- custom CSS -->
17    <link rel="stylesheet" href="css/main.css">
18
19    <title>Your Title</title>
20  </head>
21  <body>
22
23
24    <!-- jQuery first then Bootstrap JS -->
25    <script src="js/jquery-3.4.1.min.js"></script>
26    <script src="js/bootstrap.min.js"></script>
27    <!-- custom JS -->
28    <script src="js/main.js"></script>
29  </body>
30 </html>
```

- Now you can work with an IDE that will show you git tracking your files (if you choose).
- This image shows what an IDE will look like as you code with a local VCS setup properly.
- Changes are tracked in color.
- **Green** for added code.
- **Blue** for changed code.
- **Red** arrow showing deleted code.
- You can click the colors to see previous versions of code.

Committing Changes (as you code)

As you code, when you complete a section of code that you are sure is correct and you want to commit the changes, do the following:

- For all the files you want to commit execute **git add file1.ext file2.ext**
Note that add is for *“adding a file to staging to be committed.”* Meaning that you are adding files to be ready for a commit. For example if you have changed 5 files, but only want to commit one, you only “add” that one.
- Now commit those file(s) with **git commit -m “message”** where the *message* is what you want to note about your changes.
- At any time you want to check what files are tracked, modified, ready for committing, etc. you can use **git status** and/or **git ls-files**

IDE view

```
1 <!--
2 Making a change here to show color coding
3 -->
4 <!doctype html>
5 <html lang="en">
6   <head>
7     <!-- Required meta tags -->
8     <meta charset="utf-8">
9     <!-- added this next line to show color changes -->
10    <meta name="description" content="example site">
11    <meta name="viewport" content="width=device-width, initial-s
12    <link rel="icon" href="favicon.gif" type="image/gif">
13
14    <!-- Bootstrap CSS -->
15    <link rel="stylesheet" href="css/bootstrap.min.css">
16    <!-- custom CSS -->
17    <link rel="stylesheet" href="css/main.css">
18
19    <title>Your Title</title>
20  </head>
21  <body>
22
23
24    <!-- jQuery first then Bootstrap JS -->
25    <script src="js/jquery-3.4.1.min.js"></script>
26    <script src="js/bootstrap.min.js"></script>
27    <!-- custom JS -->
28    <script src="js/main.js"></script>
29  </body>
30 </html>
```

- The colors are gone now because we committed the file.
- As you change more things, the colors will come back to show you your latest changes.

Add Remote Repository

- Go to <https://github.com/> and create an account (if you don't have one).
- Make a repository.
- **DO NOT make a Readme or .gitignore or a license file.** The repo **MUST** be empty for this particular example and case.
- After this step you will have an empty repository on GitHub where you can sync your local code.

Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Repository template

Start your repository with a template repository's contents.

No template ▾

Owner *

 alexander-katrompas ▾

Repository name *

demo ✓

Great repository names are short and memorable. Need inspiration? How about [silver-fortnight?](#)

Description (optional)

demo repo for presentation

Public



Anyone on the internet can see this repository. You choose who can commit.

Private



You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file

This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

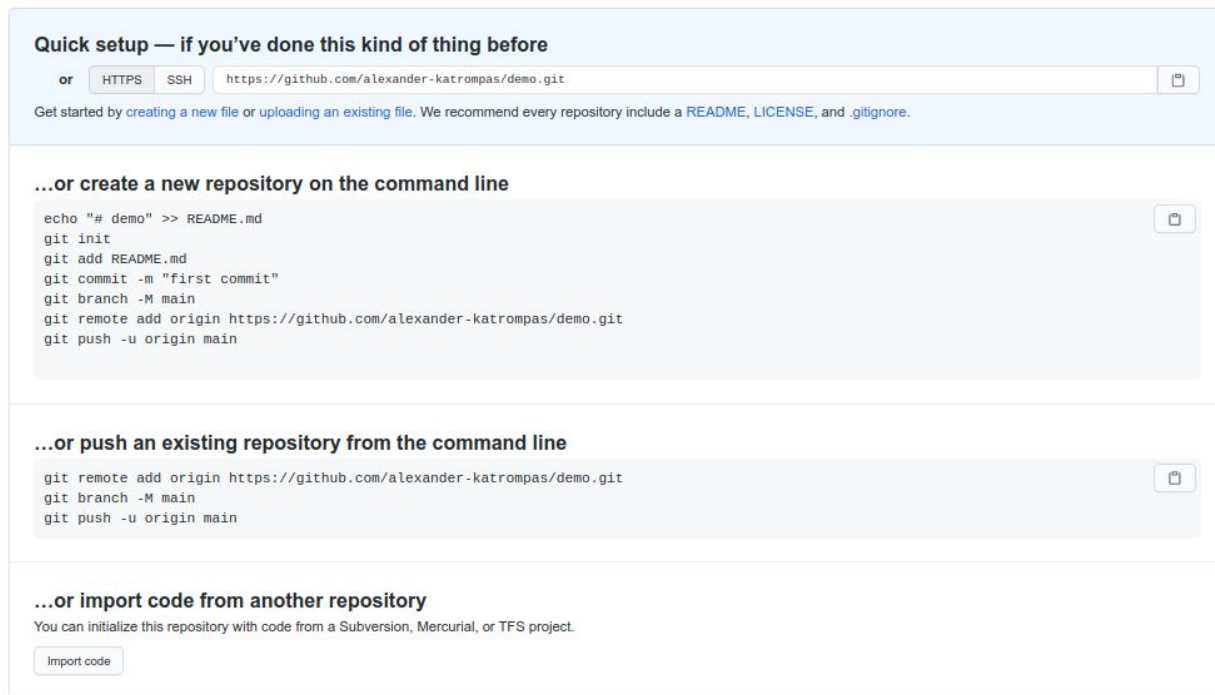
Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository


Now you will see the screen below. **STOP**. If this is your first time using GitHub you will need to set up a SSH key to interact with GitHub.

If you already use GitHub and know what you are doing and can already push/pull/clone to GitHub with HTTPS or SSH skip 1 slide ahead. Otherwise, go to the next slide and keep going.



The screenshot shows the GitHub 'Quick setup' interface. It features a header section with a title and a URL input field. Below this are three main sections, each with a title and a code block containing terminal commands. The first section is for creating a new repository on the command line, the second is for pushing an existing repository, and the third is for importing code from another repository. Each section has a copy icon to its right.


Quick setup — if you've done this kind of thing before

or HTTPS SSH 

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).


...or create a new repository on the command line

```
echo "# demo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/alexander-katrompas/demo.git
git push -u origin main
```



...or push an existing repository from the command line

```
git remote add origin https://github.com/alexander-katrompas/demo.git
git branch -M main
git push -u origin main
```



...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Setting Up a SSH Key (you only need to do this once per account)

- In a *different* browser tab go to [these instructions](#) and follow them. Don't lose the previous screen with the instructions for connecting your repo. If you do, you can get back to it by going to your repos and clicking the link for it.
- **BUT READ THIS FIRST:** Extra Information to go with those instructions:
 - Skip the first step "Create a repo." It says you need a repo with a file in it, but you don't. The repo you just made will work for this example.
 - The commands you are told to execute are done in your command prompt.
 - Make sure to use your email in the example commands.
 - When asked for a file name for your SSH key, just accept the default by pressing enter.
 - When asked for a passphrase, skip it by pressing enter (twice).
 - When you execute the copy command, the key will automatically be in your clipboard and ready for pasting into GitHub.
 - When you do the "test key" step it will warn you, ignore it. It will also ask you if you want to add the key, say yes.
 - **STOP at the step "Test the SSH key"**. Do the test, but *do not* go any further.
 - If the test works you will get a "Hi <username>" message.

Go back to your repo screen.

Quick setup — if you've done this kind of thing before

or



Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:ak-iqumulus/test.git
git push -u origin main
```

FIRST
Click the
SSH tab



...or push an existing repository from the command line

```
git remote add origin git@github.com:ak-iqumulus/test.git
git branch -M main
git push -u origin main
```

SECOND
Do these commands in
your project directory
using YOUR commands
from YOUR screen.



...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Connect your local and remote, then push.

In the previous slide you see GitHub gives you the commands to use to connect your repo and push your code. Cut and paste them one at a time. See below for each command used in practice (the first two lines)

```
alex@alex-laptop:~/Desktop/test$ git remote add origin git@github.com:alexander-katrompas/test.git
alex@alex-laptop:~/Desktop/test$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 220 bytes | 220.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com-alexander-katrompas:alexander-katrompas/test.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Click on your repo name on GitHub to view it.

alexander-katrompas / demo Private

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

main

1 branch 0 tags

Go to file

Add file

Code



greco-roamin initial commit

7bf058d 39 minutes ago 1 commit



main.cpp

initial commit

39 minutes ago



main.h

initial commit

39 minutes ago

Add a README with an overview of your project.

Add a README

Add Files, Change Files, Commit, Push or Pull

- You can add/change files locally, and push them to your remote.
- You can add/change files remotely, and pull them to your local.
- For example, assume main.cpp is changed again by adding a comment. This can now be committed and pushed from the command line or from the IDE.

The next slide shows listing files that are changed with **git ls-files -m**, the status of git with **git status**, adding with **git add**, and committing with **git commit**

```
1
2  #include "main.h"
3
4  int main(){
5      // added this comment
6      cout << "hello world" << endl;
7      return 0;
8  }
9
```

```
alex@alex-laptop:~/Desktop/demo$ git ls-files -m
main.cpp
alex@alex-laptop:~/Desktop/demo$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.cpp

no changes added to commit (use "git add" and/or "git commit -a")
alex@alex-laptop:~/Desktop/demo$ git add main.cpp
alex@alex-laptop:~/Desktop/demo$ git commit -m "added a comment to main.cpp on line 5"
[main 2b74965] added a comment to main.cpp on line 5
 1 file changed, 1 insertion(+)
alex@alex-laptop:~/Desktop/demo$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 359 bytes | 359.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/alexander-katrompas/demo.git
 7bf058d..2b74965  main -> main
alex@alex-laptop:~/Desktop/demo$
```

Add Files, Change Files, Commit, Push or Pull

WARNING!

Never change *both* remote and local files without syncing them with a pull or push. If you do, you will not be able to pull or push until you resolve the conflict (which may not be easy).

- If you change something local (i.e. on your computer), *and* you want to change something remote (i.e. on GitHub), make sure you do a **push** first to sync remote to local.
- If you change something remote (i.e. on GitHub), *and* you want to change something local (i.e. on your computer), make sure you do a **pull** first to sync local to remote.

View code pushed to remote repository.

alexander-katrompas / demo Private

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

main

1 branch 0 tags

Go to file

Add file

Code



greco-roamin added a comment to main.cpp on line 5

2b74965 6 minutes ago 2 commits



main.cpp

added a comment to main.cpp on line 5

6 minutes ago



main.h

initial commit

1 hour ago

There it is.

Add a README with an overview of your project.

Add a README

Complete Set-up Done!

- Your local version control is set-up.
- Your remote repository and version control is set-up.
- You have connected them.
- Now you can work locally and when you have completed and tested a section of code that you want to commit, you commit it locally, and push it to the remote repo.
- You can also change your code in the remote repo and pull the changes.
- Others can now also clone your remote repo, work in their local copy, and push their changes to the shared repo. You can then pull their changes back to your local repo so you keep up with other developer's changes.

Command Summary

The following are some useful commands to remember:

- **git init** initializes version control. You do this once per project (i.e. assignment)
- **git branch** will show you your branches and which one you are on
- **git remote add origin <repo-link>** adds a remote repo to your local git
- **git add file1.ext file2.ext** adds the files listed for staging to be committed
- **git commit -m "some message"** will commit the staged files with "some message"
- **git push -u origin main** will push changes to the remote repo
- **git pull origin main** will pull changes to the local repo
- **git ls-files** will list files being tracked
- **git status** will tell you git's current status
- **git remote -v** will tell you the remotes you have connected