CREATE-BEST

Module 1 – Source Control (Git and GitHub) and Open source licensing.

By: Miguel Garzón
http://www.site.uottawa.ca/~mgarz042/

uOttawa

Course Goals and Overview

- **Purpose**:
  - Introduction
  - Inspirations for Further Learning
- **Overview:**
  - Key Concepts
  - Installation and Setup
  - GitHub Introduction
  - Basic Git WorkFlow
  - Utility Commands

Lesson 1: The Basics

Why Git?

- **Distributed** source control system
  - Most operation in Git are local
  - Only a few commands required a network connection
- Massively scales
- **Open Source**
- Developed for Linux project requirements
  - 15 million lines of code
  - 12000 worldwide developers contributing to project
  - Dozens of active branches
- **Fast**, **FREE** and **Open** Source!
- Very active community!

Core Concepts

- **Repository contains:**
  - Files
  - History
  - Special configurations
- **Three (local) States of Git**
  - **Working directory** is the directory or folder on your computer that holds all the project or application files.
  - **Staging Area** (holding area or Git Index): Queues up all changes for the next commit
    - Files in the staging area can be moved in and out
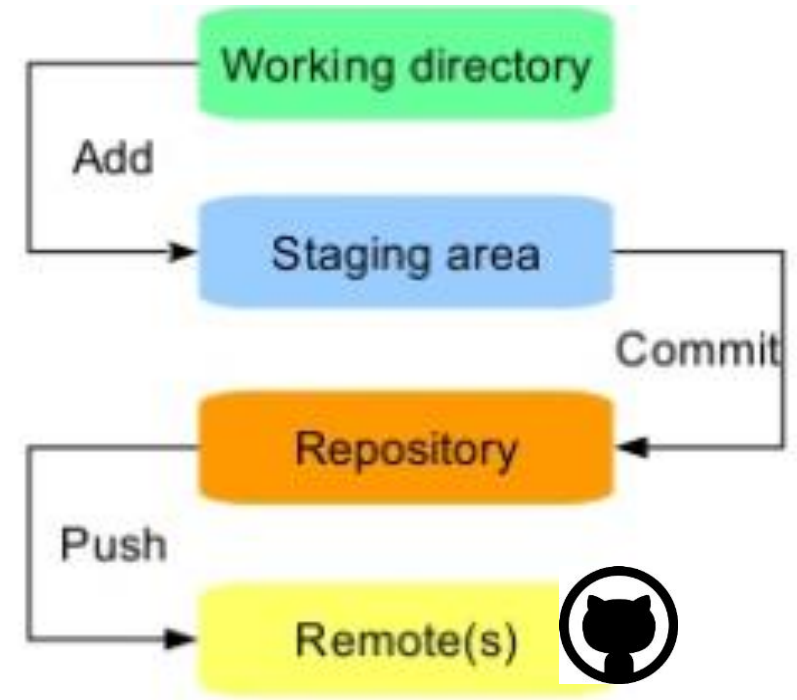 - **Repository** – Git Repository (.git folder)

working directory

git add

staging area

git commit
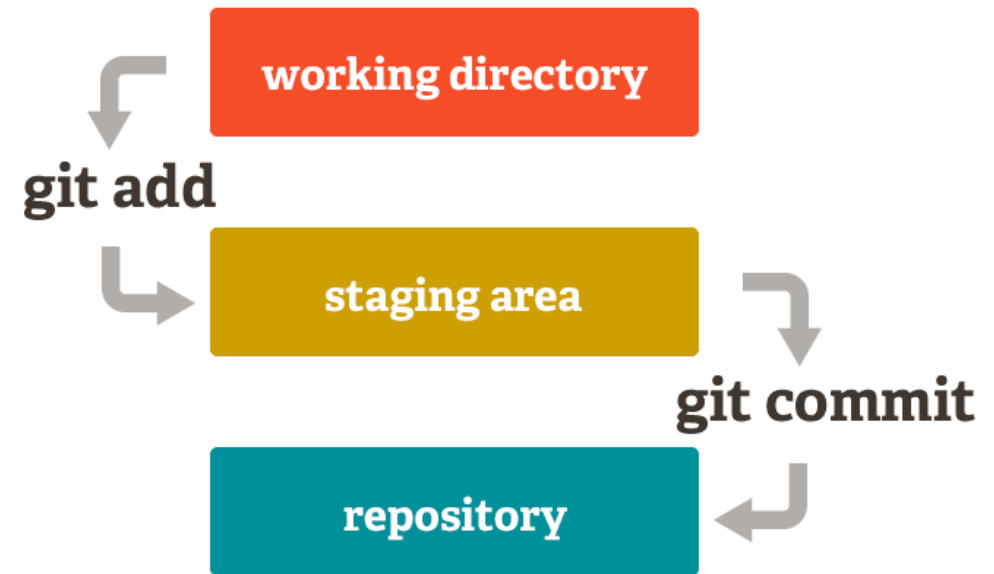
repository

Core Concepts (cont'd)

- **Remote Repository:**
  - The remote repository is just another repository with its own three states
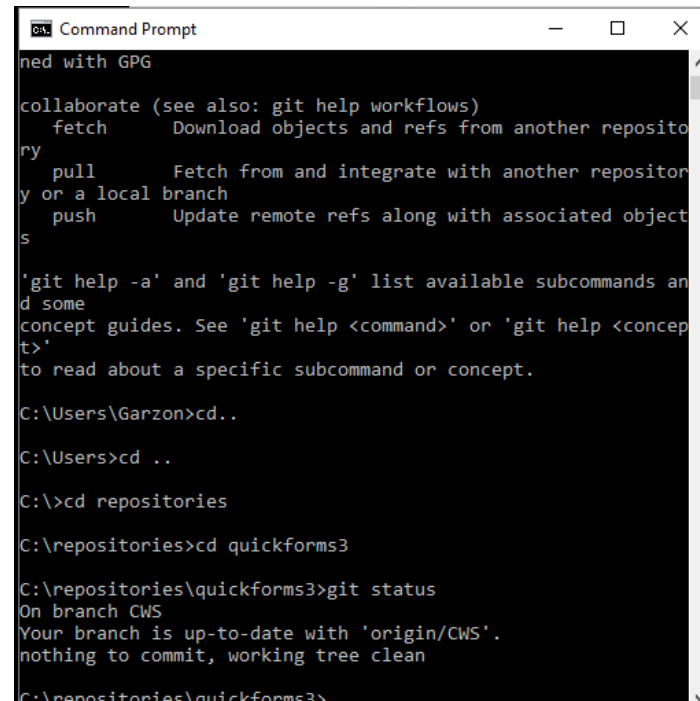 - Last step in the basic Git Workflow

- **Master Branch**
  - Times that contain your changes
  - Git provides a default branch named **master**

Git - Command Line

- Git was originally designed as a command line tool, and graphical support came later.
- Features make it onto the command line before they are integrated into a graphical client
- Documentation
- More power!
- **Consistency**:
  - **Terminal** on Mac/Linux
  - **Git Bash** on Windows

Git Installation

- Windows
  - https://git-scm.com/download/windows

- Mac OS X
  - https://git-scm.com/download/mac
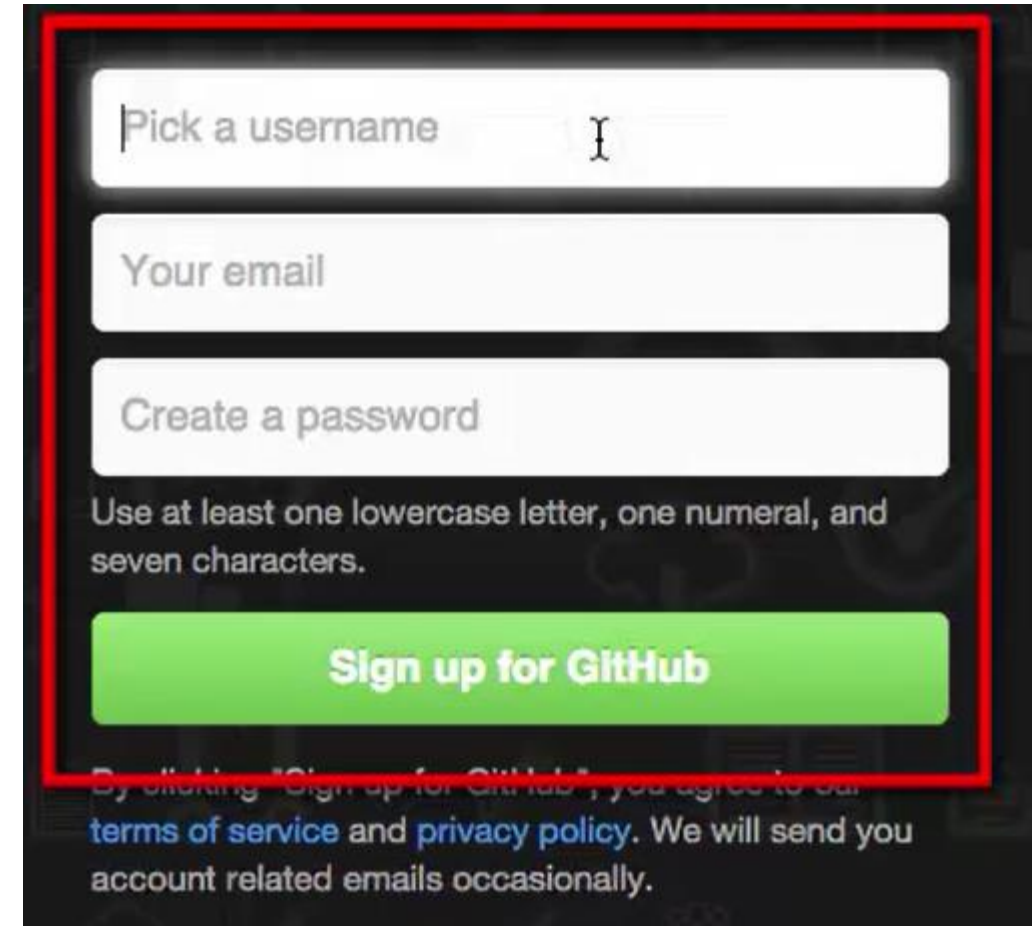 - Or type git version on the terminal

Git Commands will be the same!

Test your installation by typing: **git version**

GitHub – First Repository

- Navigate to GitHub.com
  - You can sign in or sign up.
  - Select the 'Unlimited public repositories for free' plan.
- Create a public repository:
  - **Name**:   **createbest-github**
  - **Description**: A simple demo to show the basic Git Workflow
  - Select the option
  '**Initialize** this repository with a README'

Prepare our local system

- Create a folder called **projects** using the command line
  - Place yourself under c:\
  - Type: mkdir **projects**

- Git requires two pieces of information before we proceed

  - Name: git config --global user.name "Miguel Garzón"
  - Email: git config --global user.email mgarzon@uottawa.ca

Cloning a Repository

- Using the command line, place yourself inside the folder projects (c:/projects)

- Type:

    git clone https://github.com/mgarzon/createbest-github.git

- Confirm that a folder named 'createbest-github' has been created and that you have the README file

- Type

    git status
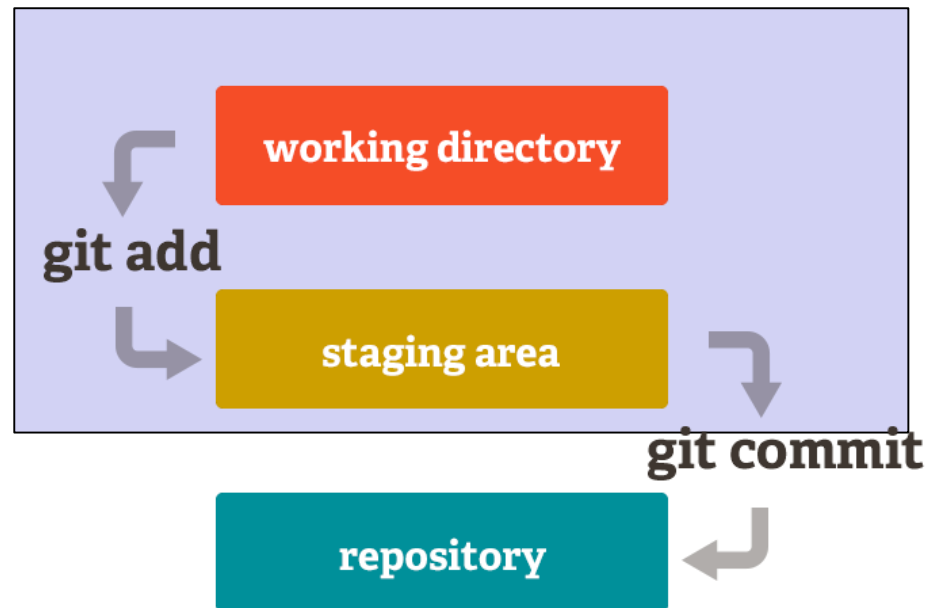
  and pay attention to the information displayed.

Our First commit

1. Add the file game.js to your **working repository** (C:\projects\createbest-github)
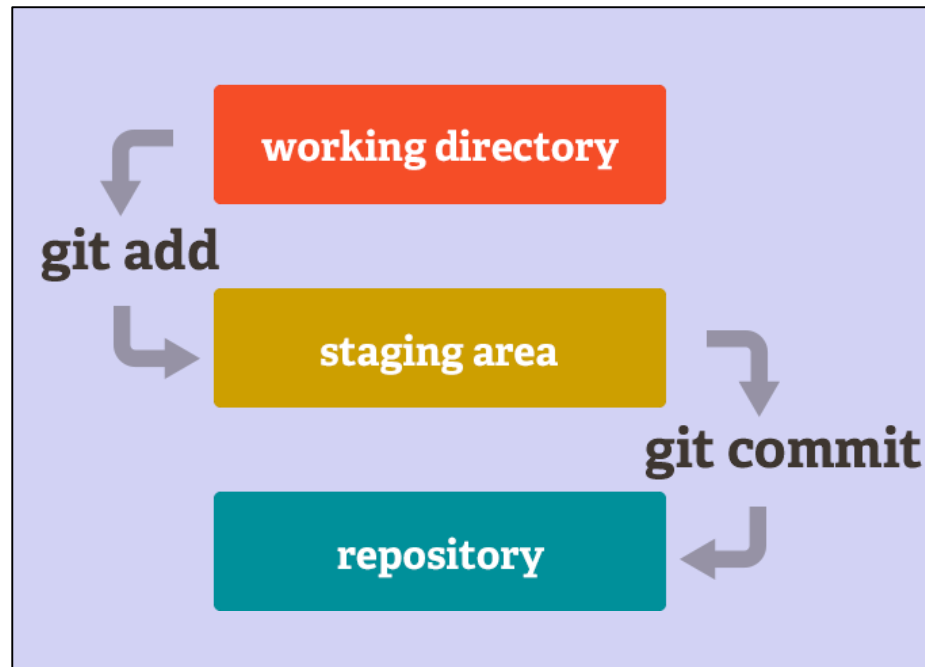2. Type *git status* in the command line. What do you see?
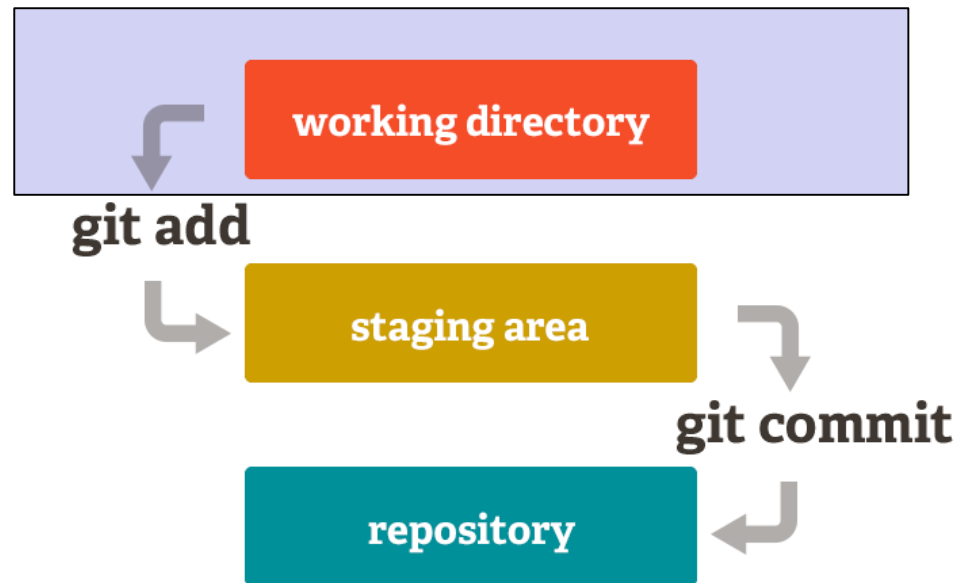
Our First commit (cont'd)

1. Type *git add game.js*
2. Then type *git status* again. What do you see now?

Our First commit (cont'd)

1. Type *git commit –m "first commit"*
2. Then type *git status* again. What do you see now?

Publishing changes to GitHub

- Your previous commit is still a **local** command
- Type ***git push origin master***
  - **origin**: refers to the GitHub copy of our repository
  - **master**: refers to our default and only branch of the repository

- The git push command will prompt you for you GitHub **username** and **password.**

- **Check** your results.
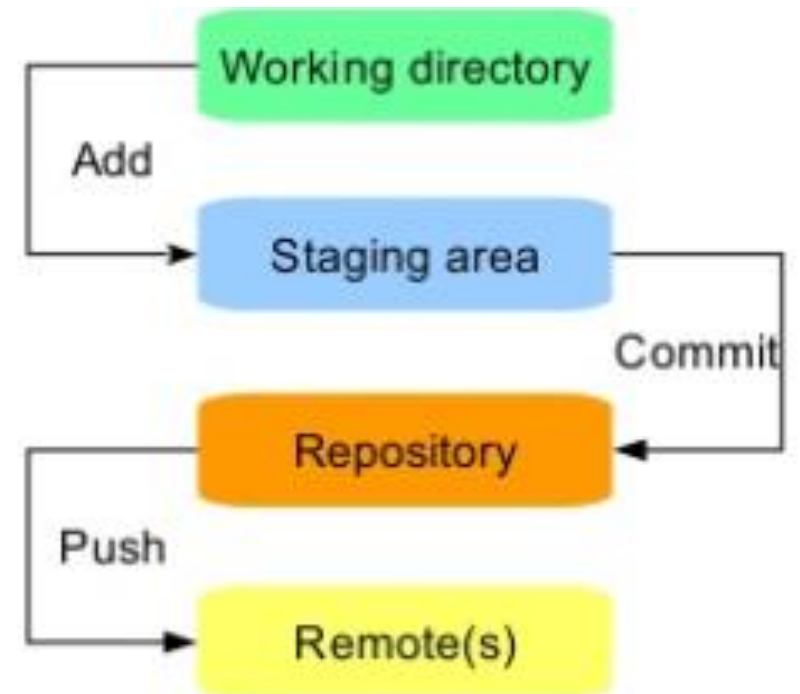
Make Modifications to Files

- **Modify** the Readme.md by adding the following text: « Welcome to our Project ».
- **Modify** the game.js as follows:
  - Key code 37 should be 'right'.
- **Add**, **Commit** and **Push** your changes. Commit message should be 'Second commit'.

Commit multiple files at the same time:

    git add .

    git commit –m "second commit"

    git push origin master

Comparing two commits

Review

- Git Introduction
- GitHub Introduction
- **Basic Workflow:**
  - Clone
  - Add/Edit Files
  - Staging Area
  - Commit
  - Pull & Push

Lesson 2: Intermediate Concepts
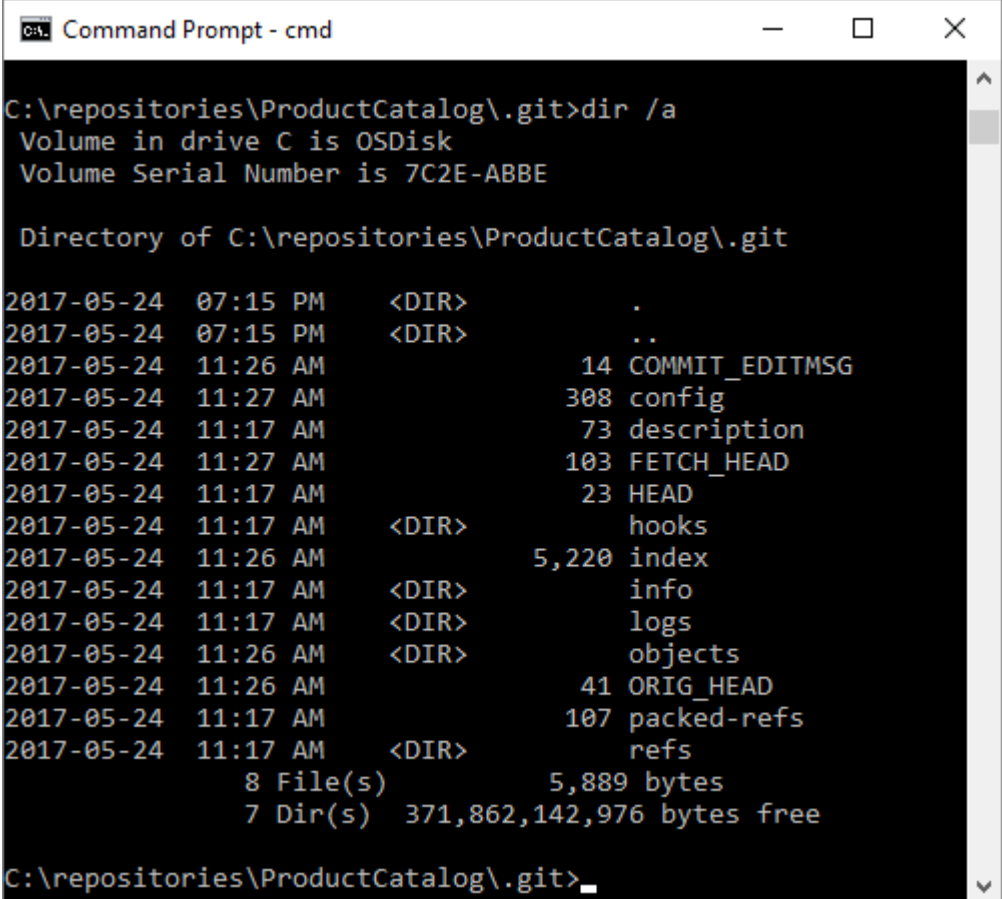
The Repository (local)

- The local repository is simply the **.git** Folder.

- Let's play with it:
    - Delete it by typing

    rmdir .git /s
    or
    rm –rf .git (linux)

Starting With an Existing Project

1. Remember our ProductCatalog Project. We will start with this existing project.

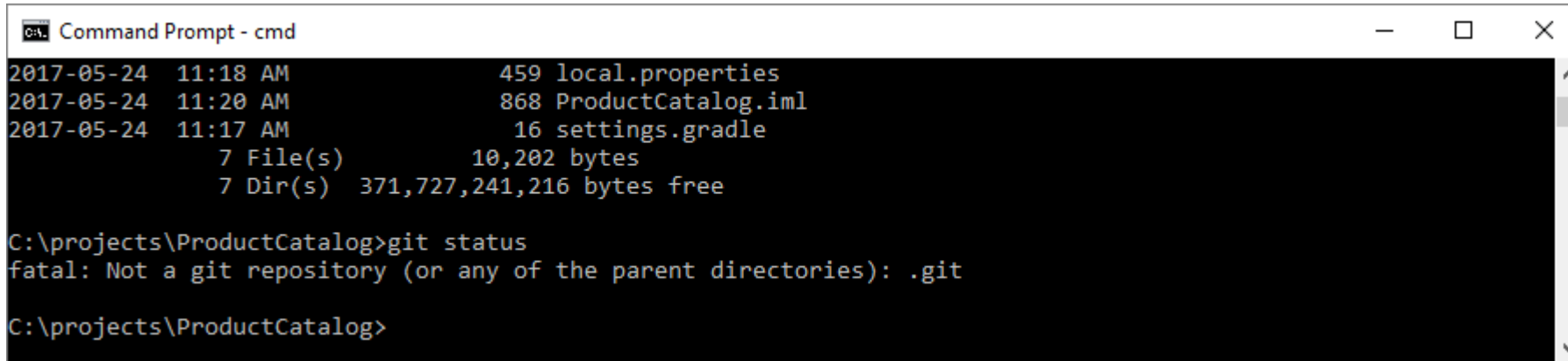2. **Move** this project to the **_projects_** folder we created before

Starting With an Existing Project (2)

Starting With an Existing Project (3)

1. To place this folder under version control with Git, we use the **init** command.
   git init .
   – At this point, Git has initialized an empty Git repositoru in the current folder.
2. Verify that a .git folder exists in the current folder. Type **dir /a** (or ls –al)
3. Type *git status* to check the current state of the (local) repository

Commit Untracked Files

- Before committing our changes let's create a file called **LICENSE.MD**

Commit all files at the same time:

    git add .
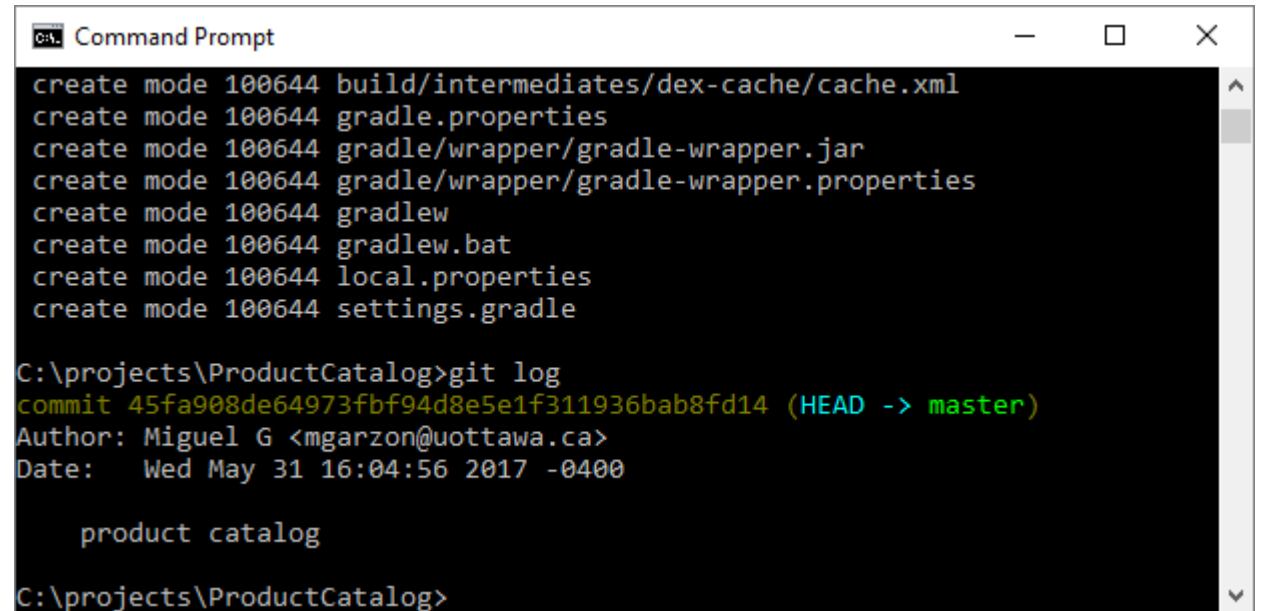
    git commit –m "product catalog app"

<u>Don't push it! We still want to play with the staging area!</u>

Logs!

- Use the *git log* to see the list of commits made locally. The log contains:
  – A long identifier (SHA-1 identifier)
  – Author
  – Timestamp
  – Commit message

- *git show* gives you more details
  Try it!


```
create mode 100644 build/intermediates/dex-cache/cache.xml
create mode 100644 gradle.properties
create mode 100644 gradle/wrapper/gradle-wrapper.jar
create mode 100644 gradle/wrapper/gradle-wrapper.properties
create mode 100644 gradlew
create mode 100644 gradlew.bat
create mode 100644 local.properties
create mode 100644 settings.gradle

C:\projects\ProductCatalog>git log
commit 45fa908de64973fbf94d8e5e1f311936bab8fd14 (HEAD -> master)
Author: Miguel G <mgarzon@uottawa.ca>
Date:    Wed May 31 16:04:56 2017 -0400

    product catalog

C:\projects\ProductCatalog>
```

Making changes to Staging area that will be reverted

- **Make** any changes to the file **build.gradle**

- **Add** the changes to the staging area

  git add build.gradle

- **Unstage** your changes ☺ (Unstage the file)

  git reset HEAD build.gradle

Making changes to Staging area that will be reverted
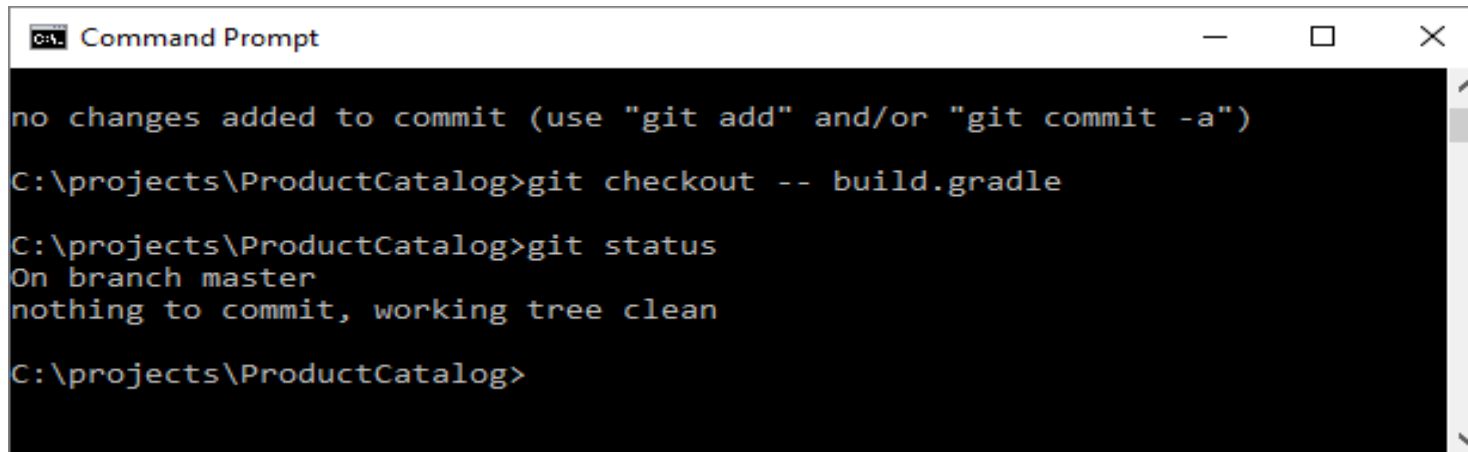
Were the changes reverted?

Push changes to GitHub

- Before pushing any changes remotely, you need to connect first your local copy with you **GitHub** remote repository.

- Type:

  git remote add your_url

  In my case,

  git remote add https://github.com/mgarzon/productcatalog2

- You can now push your changes:

  git push origin master

Reverting Changes

- The changes were **unstaged** but not reverted. To revert:

    – Type *git checkout – build.gradle*

- Check the contents of the file now. What happened?

Lesson 2: Advanced Concepts

Branches

- A branch is a timeline of commits
- Branch names are labels!
  - Deletion removes label only
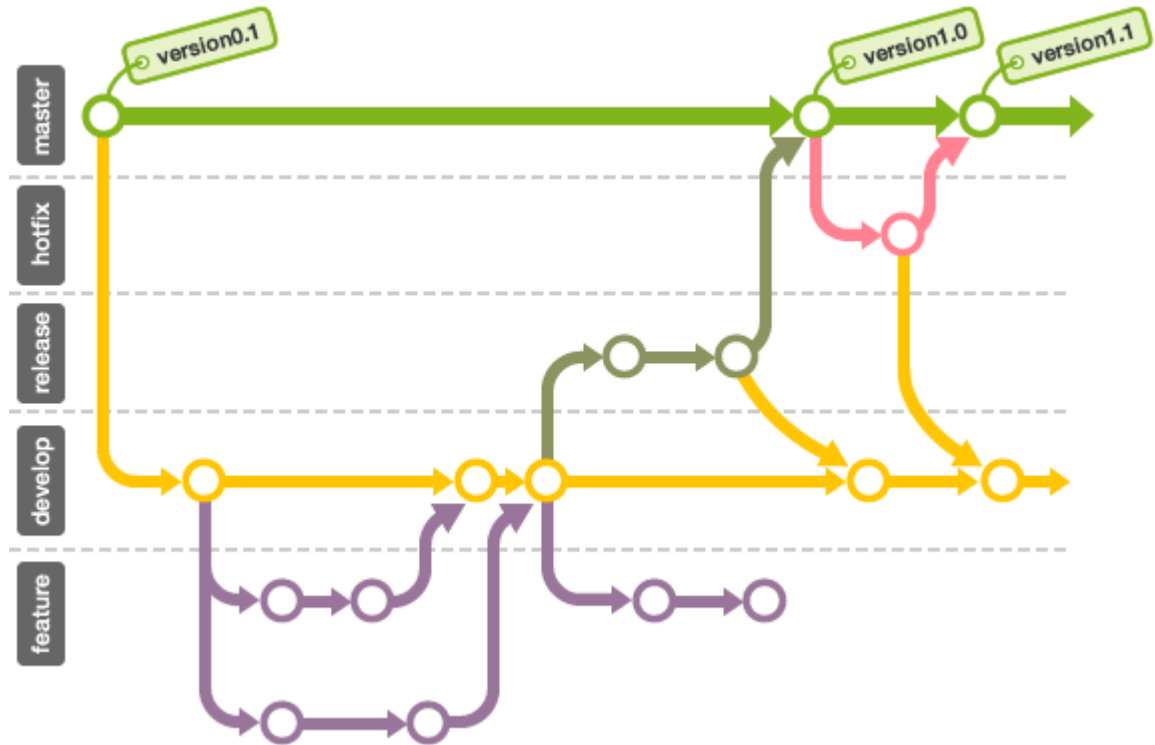
Creating a Branch

- We will rejoin the Master Branch later.
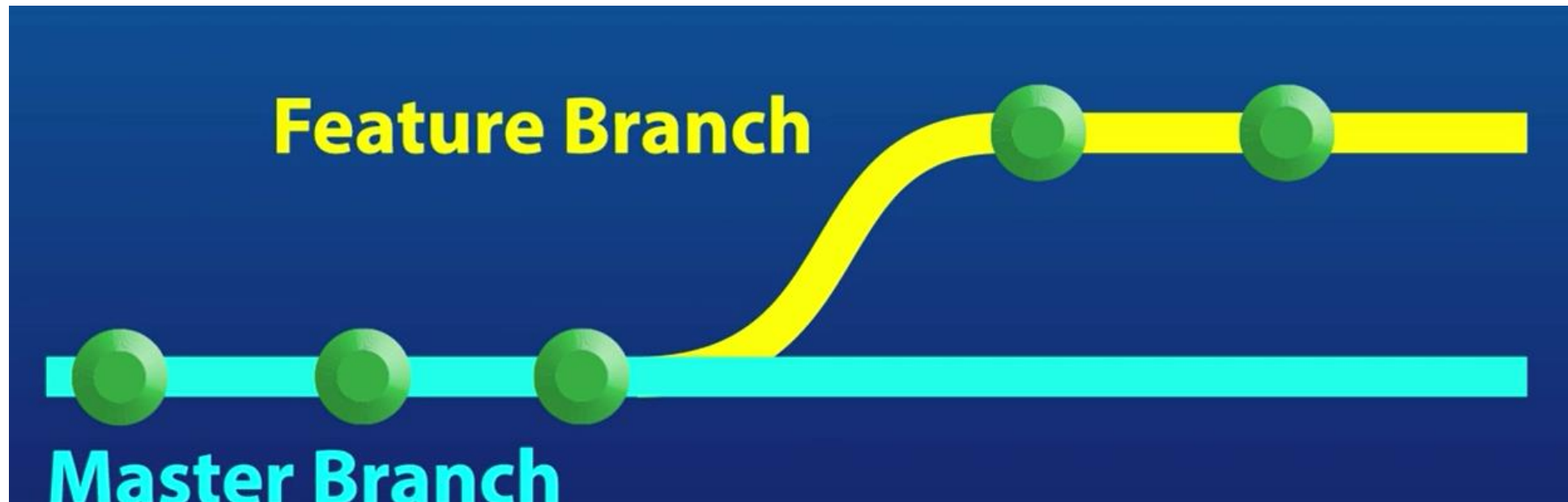
Merging Scenarios

1. **Fast-Forward:**
   - Simplest case
   - No additional work has been detected on the Master (parent) Branch

Merging Scenarios (cont'd)

2. Automated

- Non-Conflicting states
- Preserves both Timelines

Merging Scenarios (cont'd)

3. Manual

- Conflicting Merge State
- Git is unable to automatically resolve any conflicts
- Merge conflicts must be resolved before doing the commit
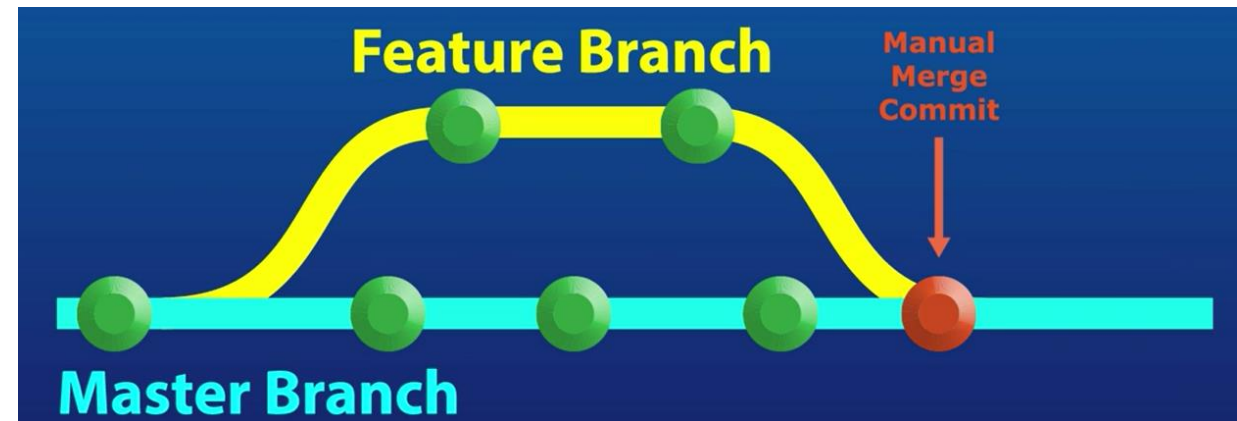
Branching

- Type *git branch* to see the list of branches
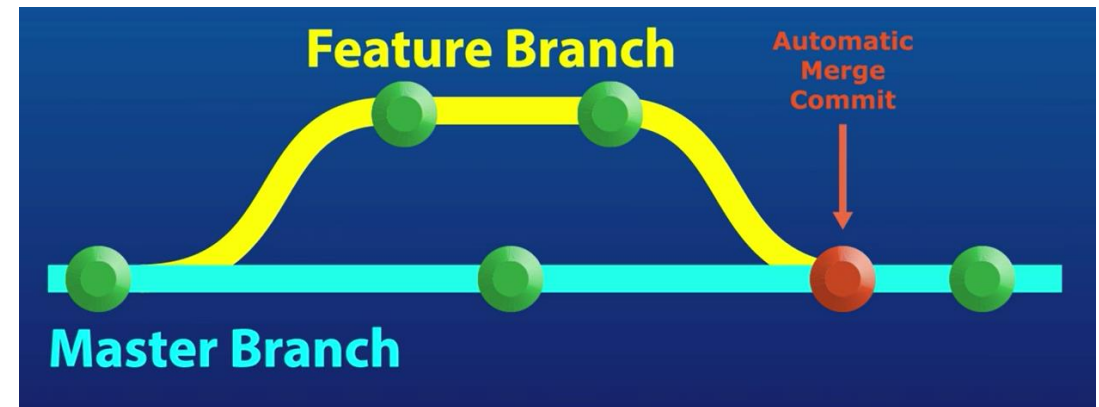
- To **create** the branch and **switch** to that branch, type:

    git checkout –b devbranch

- Now, modify the README file and add, commit and push it.
  git add .
  git commit –m "committing to another branch"
  git push origin devbranch

- You can compare the branches by typing:
  git diff master devbranch
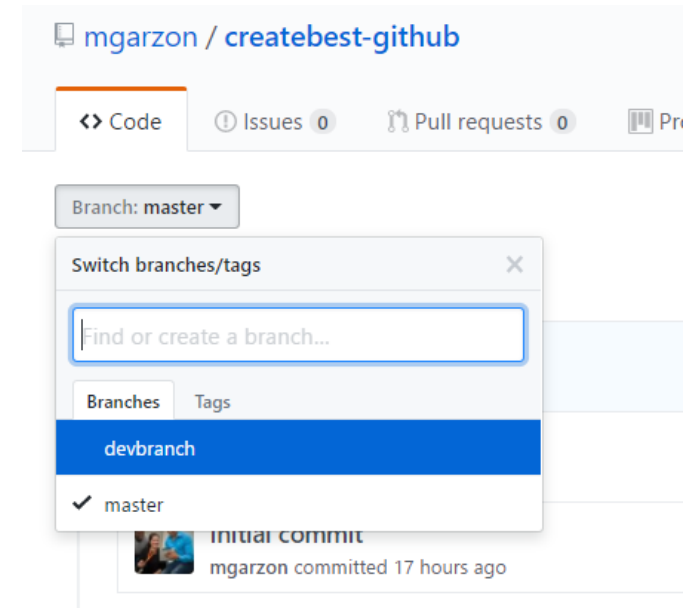
Merging

- To integrate the changes into the master branch:

1. **Switch** to my parent branch (master): *git checkout master*

2. **Type** *git log –all.* What do you see?

3. **Merge :** *git merge devbranch*

4. **Type** *git log –all.* What do you see now?

5. Now delete the branch
   git branch –d devbranch

Lesson 4: Open Source Licenses
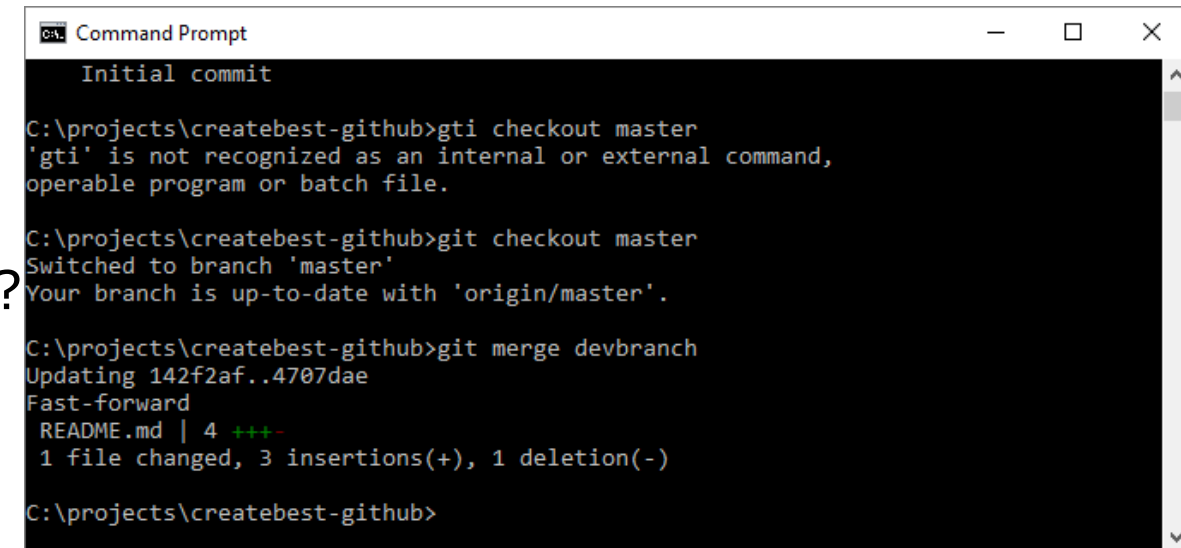
Open Source Licenses

- Open source licenses are licenses that comply with the Open Source Definition — in brief, they allow software to be freely used, modified, and shared.

I want it simple and permissive.

The **MIT License** is a permissive license that is short and to the point. It lets people do anything they want with your code as long as they provide attribution back to you and don't hold you liable.

jQuery, **.NET Core**, and **Rails** use the MIT License.

I'm concerned about patents.

The **Apache License 2.0** is a permissive license similar to the MIT License, but also provides an express grant of patent rights from contributors to users.

Android, **Apache**, and **Swift** use the Apache License 2.0.

I care about sharing improvements.

The **GNU GPLv3** is a copyleft license that requires anyone who distributes your code or a derivative work to make the source available under the same terms, and also provides an express grant of patent rights from contributors to users.

Bash, **GIMP**, and **Privacy Badger** use the GNU GPLv3.

Open source licenses

- MIT: https://choosealicense.com/licenses/mit/
- Apache: https://choosealicense.com/licenses/apache-2.0/
- GNU: https://choosealicense.com/licenses/gpl-3.0/

How to apply the license?

Create a text file (typically named LICENSE or LICENSE.txt) in the root of your source code and copy the text of the license into the file.

More info: https://choosealicense.com/licenses/

THANK YOU!