

ECSE 321 Introduction to Software  
Engineering  
***Hands-on Tutorials***

McGill University

# Table of Contents

1. Preliminaries .....	1
1.1. Getting Started .....	2
1.2. Project Management Tools for Agile Development .....	3
1.2.1. GitHub Projects .....	3
1.3. Command Line Basics .....	6
1.3.1. Windows prerequisites .....	6
1.3.2. Basic file system operations .....	6
1.3.3. Finding files .....	8
1.3.4. Batch file operations .....	8
1.3.5. Some additional useful commands .....	8
1.4. Git and GitHub .....	9
1.4.1. Installing Git .....	9
1.4.2. Creating a remote git repository on GitHub .....	9
1.4.3. Cloning to a local repository .....	9
1.4.4. Git basics .....	10
1.4.5. Browsing commit history on GitHub .....	13
1.5. Travis CI .....	15
1.6. Gradle: A Build Framework .....	17
1.6.1. Example Gradle application .....	17
1.6.2. Setting up a Spring/Spring Boot backend app with Gradle .....	20
1.7. Heroku .....	22
1.7.1. Preparations .....	22
1.7.2. Creating a Heroku app .....	22
1.7.3. Adding a database to the application .....	22
1.7.4. Extending the build for the Heroku deployment environment .....	24
1.7.5. Supply application-specific setting for Heroku .....	25
1.7.6. Deploying the app .....	25

- [HTML version](#)
- [PDF version](#)

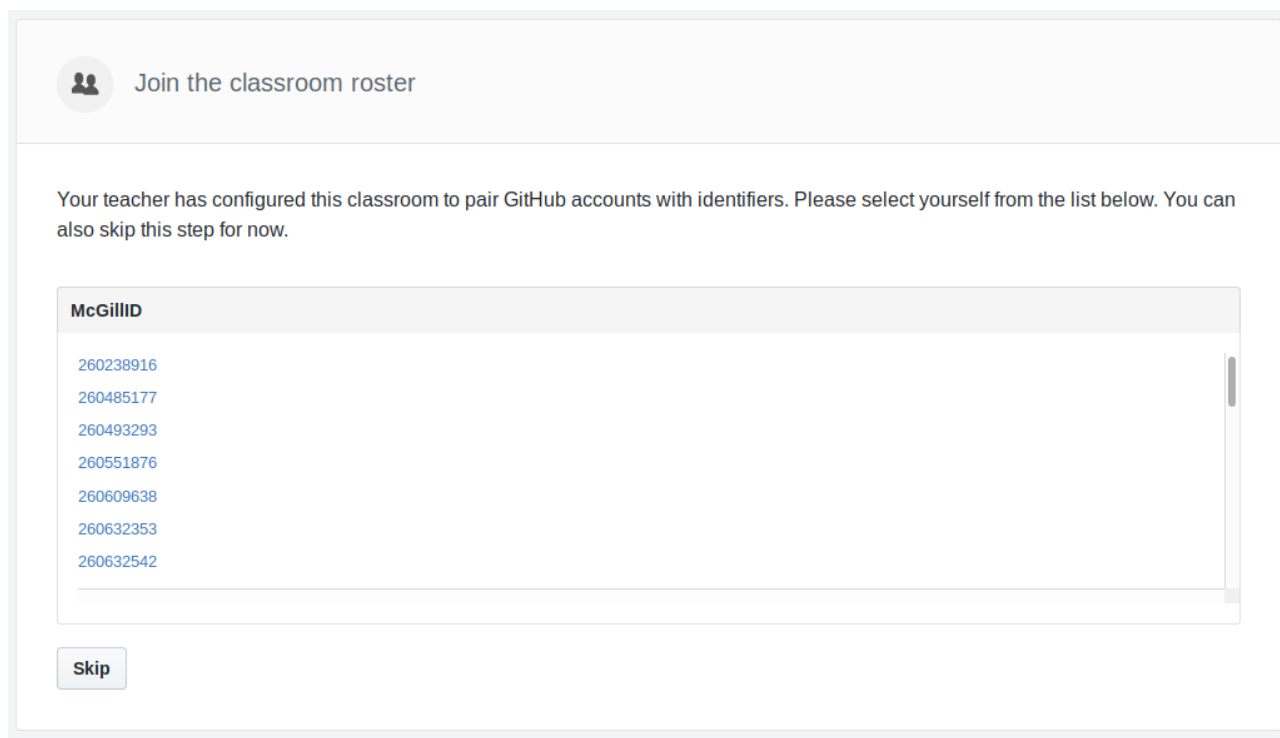
Sections of the tutorial will continuously be published at this web page.

# 1. Preliminaries

# 1.1. Getting Started

Steps for signing up for GitHub classroom:

1. Log in/Register on GitHub.
2. Open link <https://classroom.github.com/g/o9gWNZis>
3. Select your McGill ID from the list



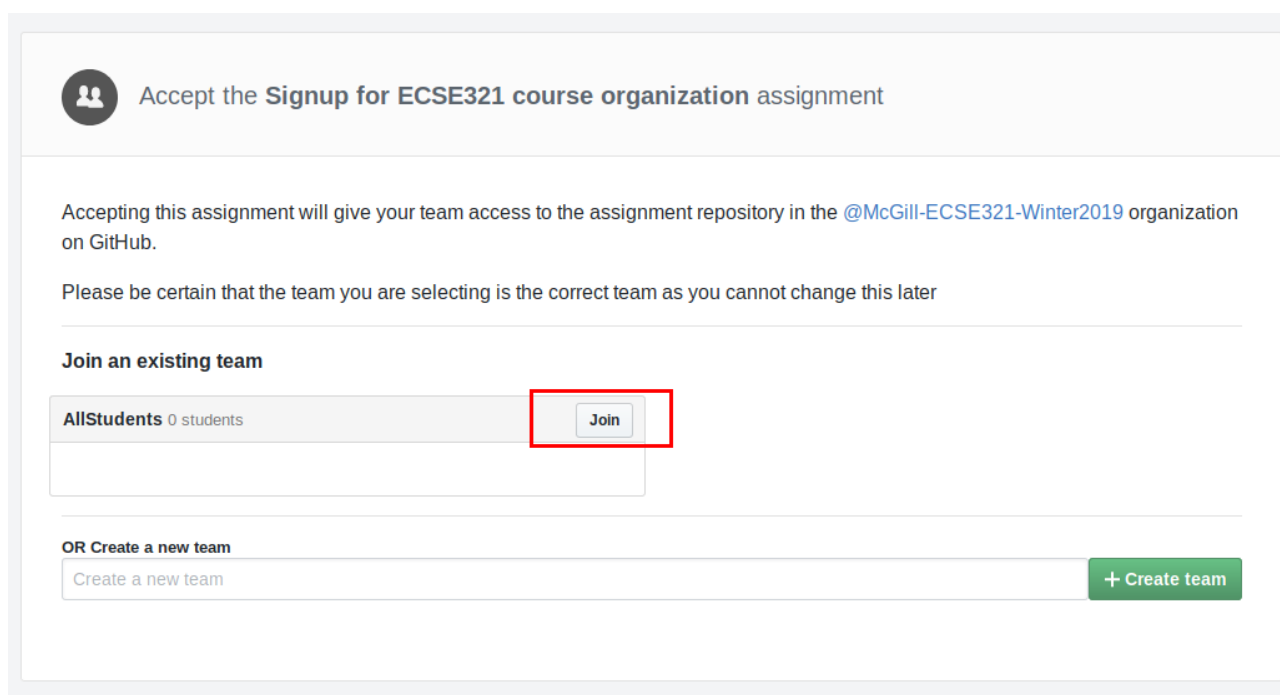
**Join the classroom roster**

Your teacher has configured this classroom to pair GitHub accounts with identifiers. Please select yourself from the list below. You can also skip this step for now.

McGillID
260238916
260485177
260493293
260551876
260609638
260632353
260632542

**Skip**

4. Join team *All students*



**Accept the Signup for ECSE321 course organization assignment**

Accepting this assignment will give your team access to the assignment repository in the [@McGill-ECSE321-Winter2019](#) organization on GitHub.

Please be certain that the team you are selecting is the correct team as you cannot change this later

**Join an existing team**

Team Name	Students	Join
AllStudents	0 students	<b>Join</b>

**OR Create a new team**

Create a new team **+ Create team**

## 1.2. Project Management Tools for Agile Development

### 1.2.1. GitHub Projects


First, we create a new repository under everyone's own account to demonstrate the basic features of "GitHub Projects".

1. Visit <https://github.com/> then click on *New repository* (green button on the right).
2. Set your user as the owner of the repository.
3. Give a name for the repository (e.g., ecse321-tutorial-1), leave it *public*, then check *Initialize this repository with a README*. Click on *Create repository* afterwards. At this point the remote repository is ready to use.

### Create a new repository

A repository contains all the files for your project, including the revision history.


---


Owner	Repository name
 <b>ecse321testuser</b> ▼	/ <b>ecse321-tutorial-1</b> ✓

Great repository names are short and memorable. Need inspiration? How about **furry-octo-journey**.

**Description** (optional)

---


☒  **Public**  
Anyone can see this repository. You choose who can commit.

☐  **Private**  
You choose who can see and commit to this repository.

---

☒ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

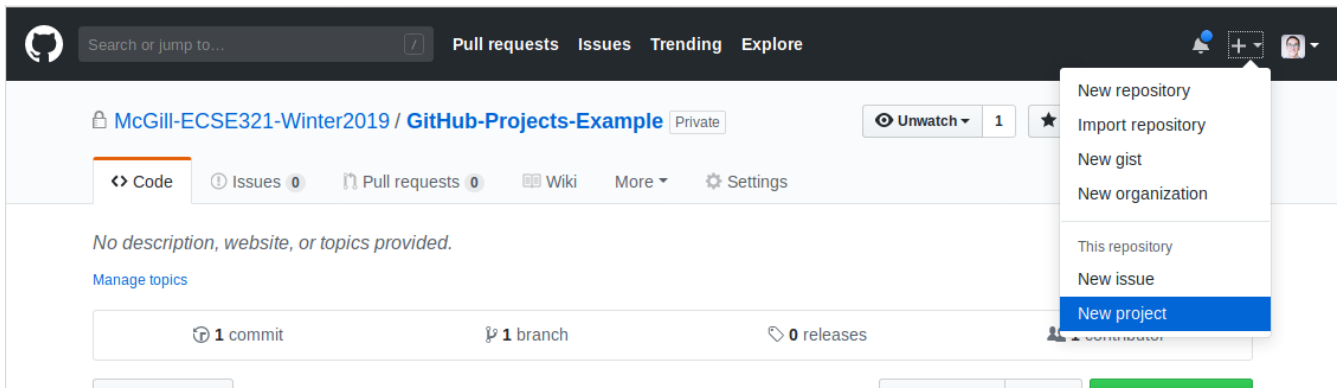
Add .gitignore: **None** ▼

Add a license: **None** ▼ 

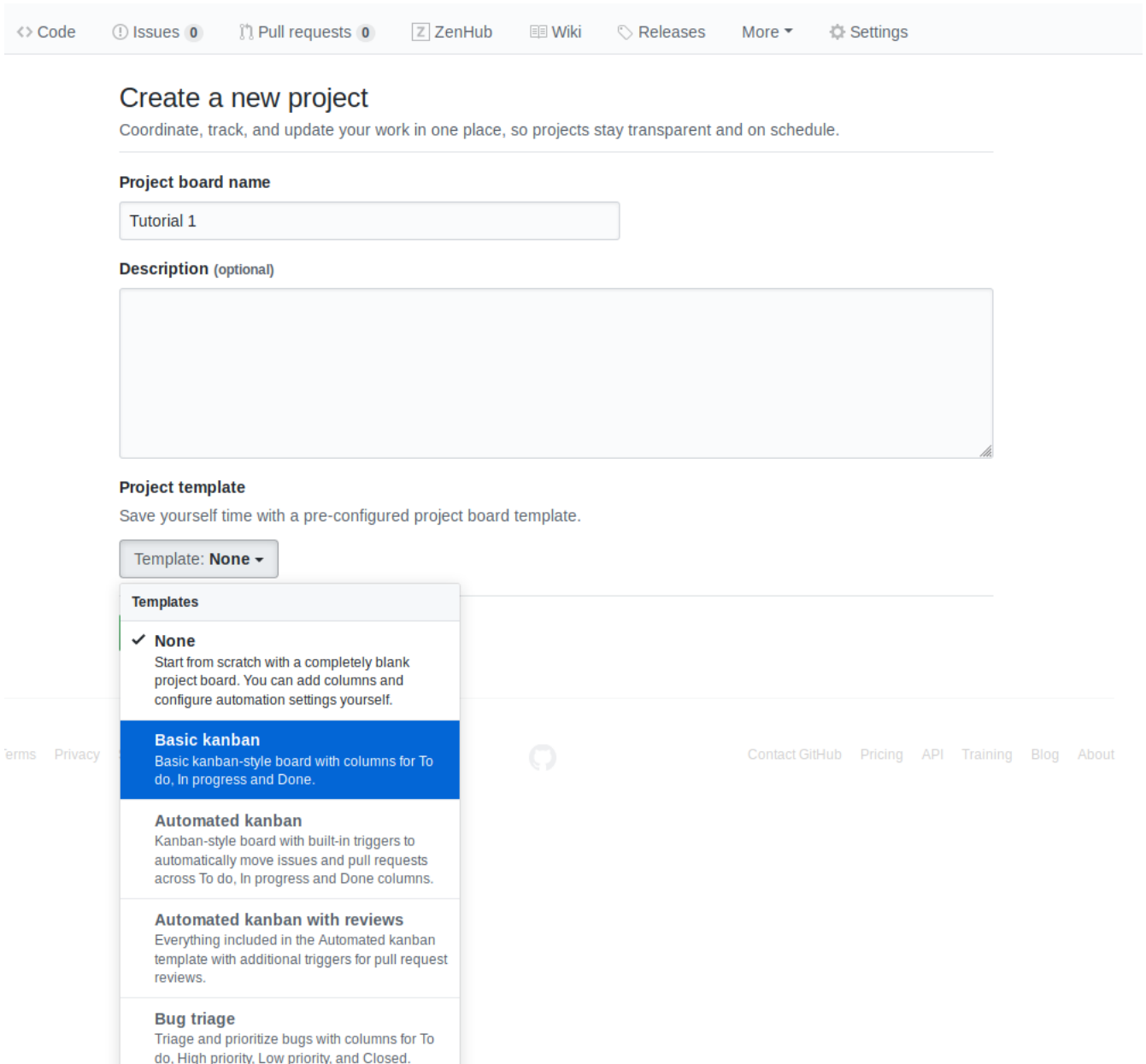
---

**Create repository**

Once the repository is ready, associate a new GitHub Project and see how their features work. Create a project:



Select Basic Kanban project style:



## Tasks to complete:

1. Create a few issues to outline the tasks for the first deliverable. Assign them appropriate labels and add yourself as the assignee!

[Code](#)
[Issues 5](#)
[Pull requests 0](#)
[ZenHub](#)
[Projects 1](#)
[Wiki](#)
[Releases](#)
[More](#)
[Settings](#)

Filters 
[Labels](#)
[Milestones](#)
[New issue](#)

[Clear current search query, filters, and sorts](#)

<input type="checkbox"/>	5 Open	0 Closed	Open All	Author	Labels	Projects	Milestones	Assignee	Sort
<input type="checkbox"/>	<b>Create UML Class diagram in UML Lab</b>								1
	#1 opened 2 days ago by imbur updated 2 days ago								
<input type="checkbox"/>	<b>Add UML Diagram</b>								
	#2 opened 2 days ago by imbur updated 2 days ago								
	<a href="#">documentation</a>								
<input type="checkbox"/>	<b>Create database layer</b>								
	#5 opened 2 days ago by imbur updated 2 days ago								
	<a href="#">epic</a>								
<input type="checkbox"/>	<b>Write project deliverable 1</b>								
	#4 opened 2 days ago by imbur updated 2 days ago								
	<a href="#">epic</a>								
<input type="checkbox"/>	<b>Report individual and teamwork</b>								
	#3 opened 2 days ago by imbur updated 2 days ago								
	<a href="#">documentation</a>								

2. Create a milestone for the issues.

[McGill-ECSE321-Winter2019 / GitHub-Projects-Example](#)
[Private](#)
[Unwatch 1](#)
[Star 0](#)
[Fork 0](#)

[Code](#)
[Issues 5](#)
[Pull requests 0](#)
[ZenHub](#)
[Projects 1](#)
[Wiki](#)
[Releases](#)
[More](#)
[Settings](#)

[Labels](#)
[Milestones](#)
[New milestone](#)

0 Open 0 Closed
 [Sort](#)

3. Create cards from the issues on the project board.

4. See how GitHub track the project progress as you move the cards from the different columns.

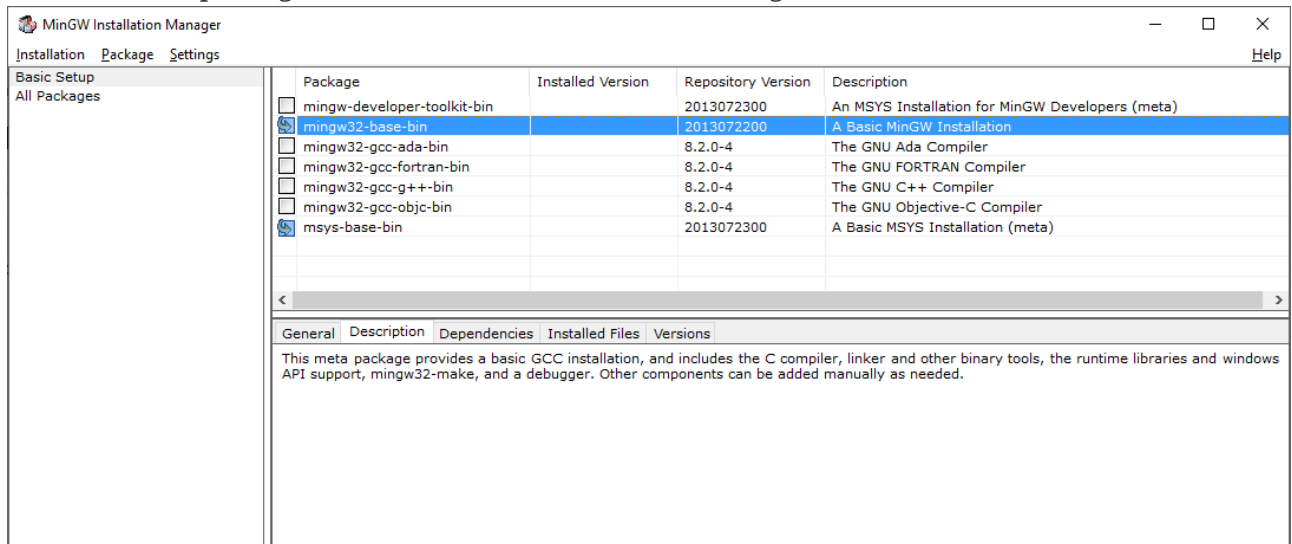
## 1.3. Command Line Basics

This section shows a few handy terminal commands.

### 1.3.1. Windows prerequisites

This step can be skipped if you are using MacOS or Linux. However, if you are using Windows, you need to have a terminal that supports the execution of basic Linux commands. Such programs are Git Bash or MinGW, for example. You can find below a few helper steps to get MinGW running on your system.

1. Get the [MinGW installer from here](#)
2. Install it to wherever you like, the default installation folder is `C:|MinGW`
3. Once the setup finishes, open the MinGW Installation Manager
4. Select the two packages for installation as shown in the figure below



5. Click on *Installation/Apply Changes*. This will take a few moments to fetch and install the required packages.
6. You can open a terminal window by running the executable `C:|MinGW|msys|1.0|bin|bash.exe`

### 1.3.2. Basic file system operations

1. Open a terminal, and try the following commands:

- **pwd**: prints the present working directory

Example:

```
$ pwd
/home/ecse321
```

- **ls**: lists the content of a given folder

Example:



```
$ ls /home
ecse321 guest-user admin
```

- **cd**: navigates the file system

Example:

```
$ cd ..
$ pwd
/home
$ cd ecse321
$ pwd
/home/ecse321
```

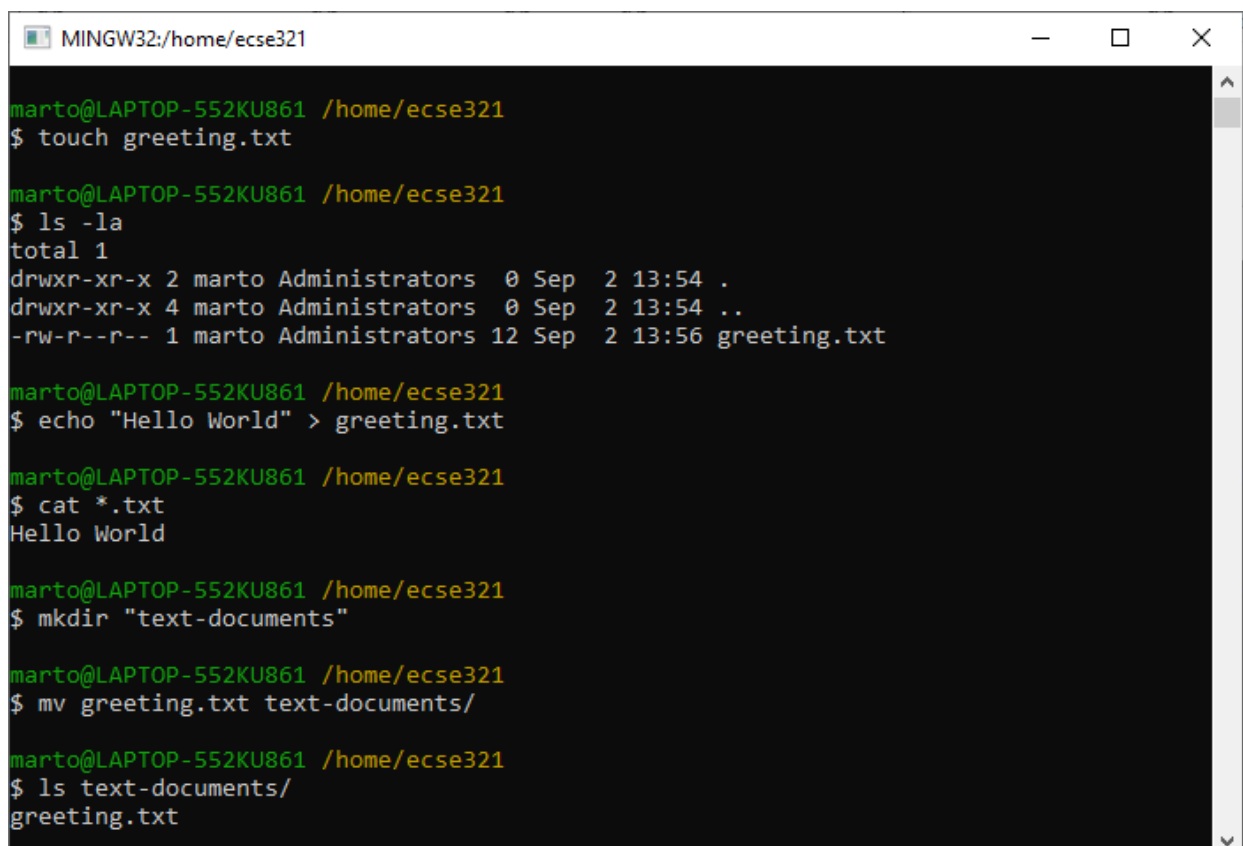
#### NOTE

The following steps will include images that illustrate the commands and their output to prevent easy copy-paste. Sorry! :)

## 2. Creating files and reading/writing their contents

- **touch**: creates a file
- **mkdir**: creates a directory
- **mv**: moves a file (or directory) from its current location to a target location
- **echo**: prints a string
- **cat**: prints the contents of a file

Example:



```
MINGW32:/home/ecse321

marto@LAPTOP-552KU861 /home/ecse321
$ touch greeting.txt

marto@LAPTOP-552KU861 /home/ecse321
$ ls -la
total 1
drwxr-xr-x 2 marto Administrators  0 Sep  2 13:54 .
drwxr-xr-x 4 marto Administrators  0 Sep  2 13:54 ..
-rw-r--r-- 1 marto Administrators 12 Sep  2 13:56 greeting.txt

marto@LAPTOP-552KU861 /home/ecse321
$ echo "Hello World" > greeting.txt

marto@LAPTOP-552KU861 /home/ecse321
$ cat *.txt
Hello World

marto@LAPTOP-552KU861 /home/ecse321
$ mkdir "text-documents"

marto@LAPTOP-552KU861 /home/ecse321
$ mv greeting.txt text-documents/

marto@LAPTOP-552KU861 /home/ecse321
$ ls text-documents/
greeting.txt
```

### 1.3.3. Finding files

The versatile `find` command allows us to find files based on given criteria. Take look at its manual page with `man find`!

Example:

```
MINGW32:/home/ecse321
marto@LAPTOP-552KU861 /home/ecse321
$ ls -la
total 0
drwxr-xr-x 3 marto Administrators 0 Sep  2 23:05 .
drwxr-xr-x 4 marto Administrators 0 Sep  2 13:54 ..
drwxr-xr-x 2 marto Administrators 0 Sep  2 23:05 text-documents

marto@LAPTOP-552KU861 /home/ecse321
$ find ./ -iname *.txt
./text-documents/greeting.txt
```

### 1.3.4. Batch file operations

- `sed`: stream editor; changes a given string to a replacement

Combining `find` with an additional command (e.g., `sed`) can greatly speed up your repetitive tasks.

Example:

```
MINGW32:/home/ecse321
marto@LAPTOP-552KU861 /home/ecse321
$ ls -la text-documents/
total 2
drwxr-xr-x 2 marto Administrators  0 Sep  2 23:26 .
drwxr-xr-x 3 marto Administrators  0 Sep  2 23:05 ..
-r--r--r-- 1 marto Administrators 14 Sep  2 23:26 greeting.txt
-rw-r--r-- 1 marto Administrators 12 Sep  2 23:21 helloworld.txt

marto@LAPTOP-552KU861 /home/ecse321
$ touch temp

marto@LAPTOP-552KU861 /home/ecse321
$ sed "s/World/ECSE321/g" text-documents/greeting.txt temp
Hello ECSE321

marto@LAPTOP-552KU861 /home/ecse321
$ cat temp

marto@LAPTOP-552KU861 /home/ecse321
$ sed "s/World/ECSE321/g" text-documents/greeting.txt > temp

marto@LAPTOP-552KU861 /home/ecse321
$ cat temp
Hello ECSE321

marto@LAPTOP-552KU861 /home/ecse321
$ mv temp text-documents/greeting.txt

marto@LAPTOP-552KU861 /home/ecse321
$ find ./ -iname *.txt -exec sed "s/Hello/Hi/g" {} \;
Hi ECSE321
Hi World
```

**NOTE**      The file *helloworld.txt* in the example is initially a copy of *greeting.txt*.

### 1.3.5. Some additional useful commands

- `rm`: removes a file
- `cp -r`: copies a directory recursively with its contents
- `rmdir`: remove an empty directory

- `rm -rf`: force to recursively delete a directory (or file) and all its contents
- `nano`: an easy-to-use text editor (not available by default in MinGW)
- `grep`: finds matches for a string in a given stream of characters
- `ag`: takes a string as argument and searches through the contents of files recursively to find matches of the given string (this tool is included in the *silversearcher-ag* package)

## 1.4. Git and GitHub

### 1.4.1. Installing Git

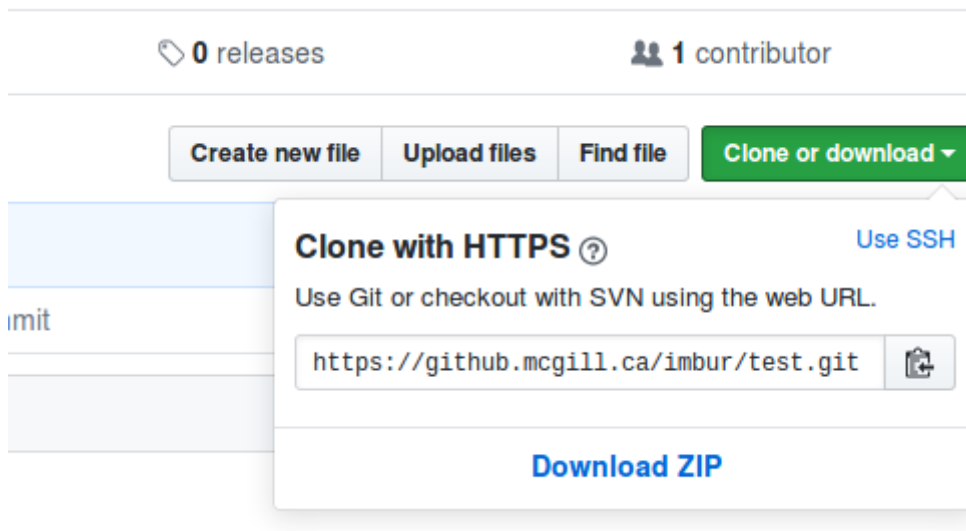
Install the Git version control system (VCS) from <https://git-scm.com/downloads>.

### 1.4.2. Creating a remote git repository on GitHub

1. Go to <https://github.com/new>
2. Set *test* as the name of the repository
3. Check the checkbox *Initialize this repository with a README*
4. Click on create repository

### 1.4.3. Cloning to a local repository

1. Open up a terminal (Git bash on Windows).
2. Navigate to the designated target directory (it is typical to use the `git` folder within the home directory for storing Git repositories, e.g., `cd /home/username/git`).
3. Using a Git client, clone this newly created *test* repository to your computer. First, get the repository URL (use HTTPS for now).



Then, issue `git clone https://url/of/the/repository.git`

You should get an output similar to this:

```
Git Bash
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university
$ git clone git@github.com:mcgill-ecse321/class-notes.git
Cloning into 'class-notes'...
remote: Counting objects: 290, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 290 (delta 0), reused 0 (delta 0)Receiving objects: 96% (279/290), 5.68 MiB | 314 KiB/s
Receiving objects: 100% (290/290), 5.91 MiB | 313 KiB/s, done.
Resolving deltas: 100% (59/59), done.
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university
$
```

4. Verify the contents of the *working copy* of the repository by `ls -la ./test`. The `.git` folder holds version information and history for the repository, while the `README.md` is an auto-generated text file by GitHub.

#### 1.4.4. Git basics

1. Open up a terminal and configure username and email address. These are needed to identify the author of the different changes.

```
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git config --global user.name "shabbir-hussain"

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git config --global user.email shabbir.hussain@outlook.com
```

Glossary — Part 1:

- **Git** is your version control software
  - **GitHub** hosts your repositories
  - A **repository** is a collection of files and their history
  - A **commit** is a saved state of the repository
2. Enter the working directory, then check the history by issuing `git log`. Example output:

```
commit 2a0735092cea1b7f7c850a48b86e8847bf979236
Author: Shabbir Hussain <mohd.husn001@gmail.com>
Date: Thu Aug 28 15:33:09 2014 -0400

    almost finished seat checking

commit 90bfbac1c8134a87d16caf89c9ff66104f8b7fb7
Author: Shabbir Hussain <mohd.husn001@gmail.com>
Date: Thu Aug 28 14:30:07 2014 -0400

    fixed wishlist null ptr exception

commit ca4a6921005e89dace34226560921c9770a82574
Author: Shabbir Hussain <mohd.husn001@gmail.com>
Date: Thu Aug 28 11:03:19 2014 -0400

    grade checker hotfix
```

3. Adding and committing a file: use the `git add` and `git commit` commands.

```
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ touch helloworld.java
```

```

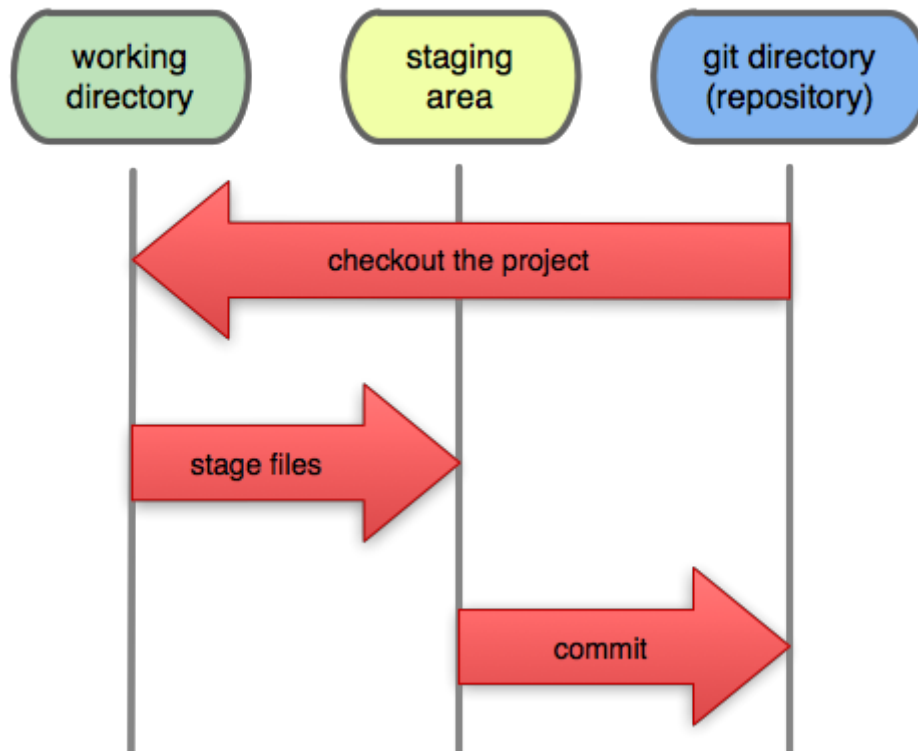
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git add helloworld.java

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git commit -m 'added hello world file to the project'
[master (root-commit) f4a1ddc] added hello world file to the project
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 helloworld.java

```

The effect of these commands are explained on the figure below:

## Local Operations



Glossary — Part 2:

- **Working Directory:** files being worked on right now
- **Staging area:** files ready to be committed
- **Repository:** A collection of commits

4. Checking current status is done with `git status`.

```

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   helloworld.java
#
no changes added to commit (use "git add" and/or "git commit -a")

```

5. Staging and unstaging files: use `git add` to add and `git reset` to remove files from the staging area.

```

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git add .

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   helloworld.class
#       modified:   helloworld.java
#

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git reset helloworld.class
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   helloworld.java
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       helloworld.class

```

**CAUTION** Only staged files will be included in the next commit.

- To display detailed changes in unstaged files use `git diff`, while use `git diff --staged` to show changes within files staged for commit.

```

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git diff helloworld.java
diff --git a/helloworld.java b/helloworld.java
index 28fe9d9..de3a7d2 100644
--- a/helloworld.java
+++ b/helloworld.java
@@ -1,6 +1,6 @@
 public class helloworld{

     public static void main(String[] args){
-        System.out.println("Hello World");
+        System.out.println("Hello World")
     }
 }

```

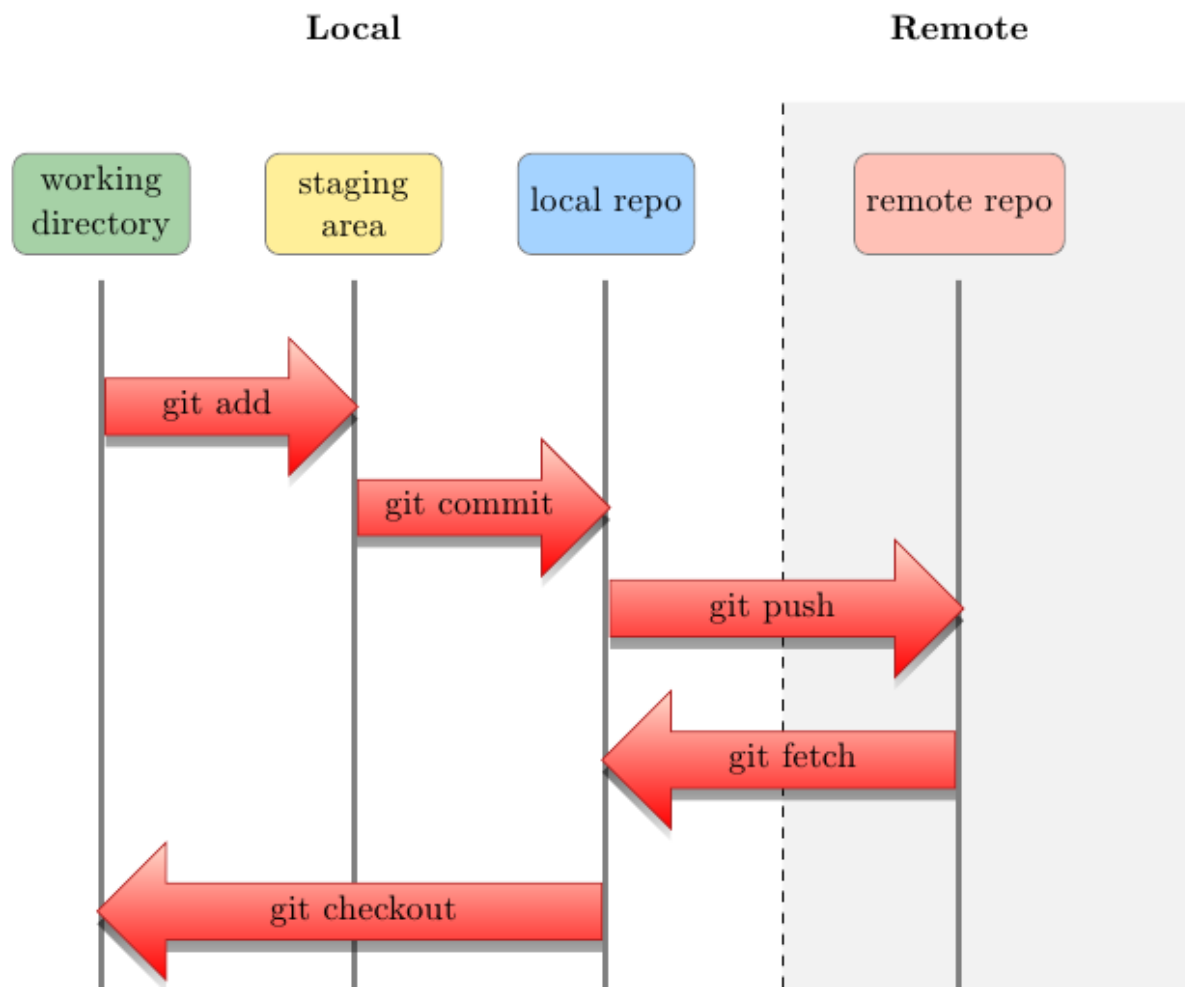
- Reverting to a previous version is done using `git checkout`.

```

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git checkout helloworld.java

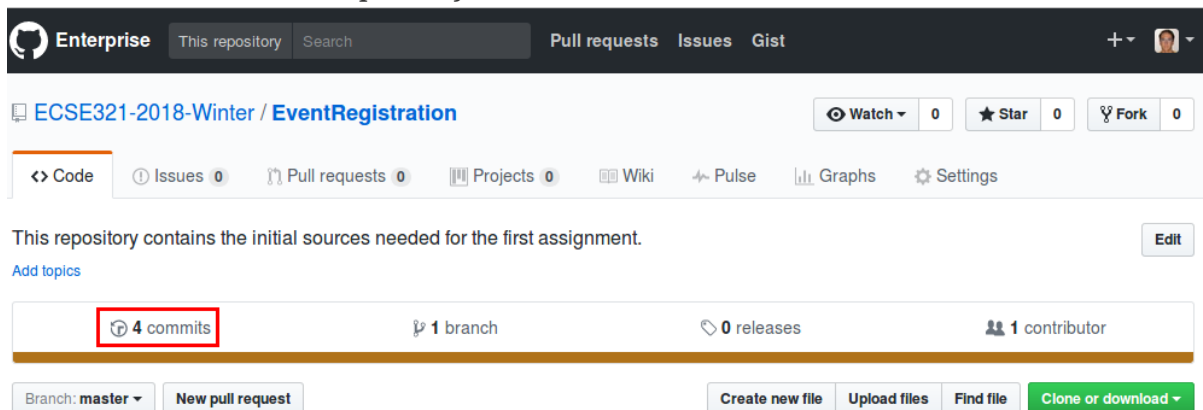
```

- The commands `git pull` (or the `git fetch` + `git rebase` combination) and `git push` are used to synchronize local and remote repositories.



### 1.4.5. Browsing commit history on GitHub

1. You can browse pushed commits in the remote repository online using GitHub. You can select the *commits* menu for a repository.



To get a link for a specific commit, click on the button with the first few characters of the hash of the commit.

The screenshot shows the GitHub interface for the repository 'ECSE321-2018-Winter / EventRegistration'. The top navigation bar includes 'Enterprise', 'This repository', a search bar, and links for 'Pull requests', 'Issues', and 'Gist'. Below the repository name, there are buttons for 'Watch' (0), 'Star' (0), and 'Fork' (0). A secondary navigation bar contains links for '<> Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. The current branch is 'master'. The commit history is displayed with three groups of commits:

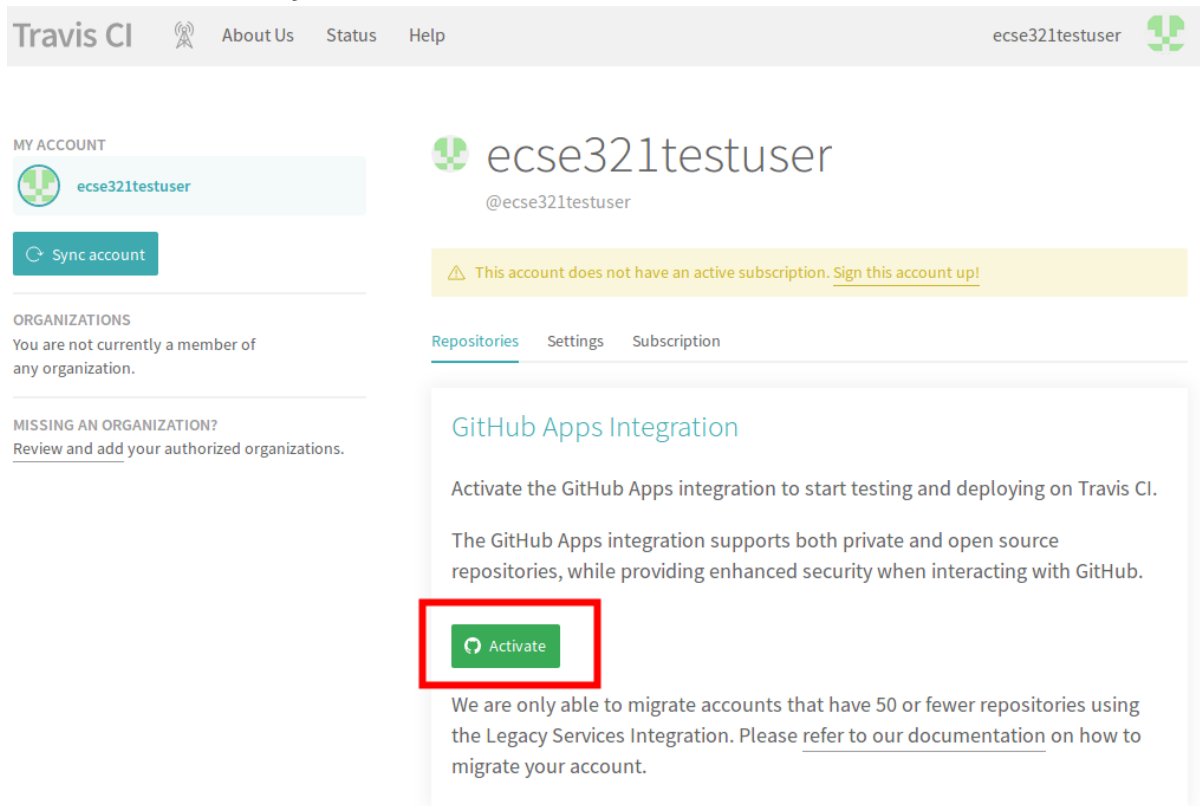
- Commits on Jan 10, 2018**
  - Fixing default Android IP address in the props.** by imbur, committed on GitHub Enterprise 2 hours ago. Commit hash: 4bc0134.
  - Patching URL for CORS mapping** by imbur, committed on GitHub Enterprise 5 hours ago. Commit hash: e8daf92.
- Commits on Jan 8, 2018**
  - Adding initial Java Spring project seed** by imbur, committed 3 days ago. Commit hash: 58c992b (highlighted with a red box).
- Commits on Jan 7, 2018**
  - Initial commit** by imbur, committed 3 days ago. Commit hash: 86170d8.

The source for most of the images in the Git documentation: <https://github.com/shabbir-hussain/ecse321tutorials/blob/master/01-githubTutorial1.pptx>



## 1.5. Travis CI

1. Go to <https://travis-ci.com/>, click on Sign up with GitHub.
2. Click on the green authorize button at the bottom of the page.
3. Activate Travis-CI on your GitHub account



4. Select the repositories you want to build with Travis (make sure to include your repository that you created for this tutorial). You can modify this setting anytime later as well.
5. In your working copy of your repository, create a default Gradle java project.
  - Make sure you have **Gradle** installed (`gradle --version`).
  - Issue `gradle init --type java-library`
  - Add a `.gitignore` to ignore generated resources by Git:

```
.gradle/  
build/
```

- Make sure your application is compiling by running `gradle build`
6. Create a file called `.travis.yml`:

```
language: java  
script:  
- gradle build
```

7. Commit and push your work. If everything is set up correctly, the build should trigger and

Travis should run your build using Gradle.

# 1.6. Gradle: A Build Framework

## 1.6.1. Example Gradle application

This section focuses on writing a Gradle (<https://gradle.org/>) build script that builds a single Gradle project referred to as *Computation*. The source code and tests for a Java application is available here: [Computation.zip](#) (src and tst folders). It is your job to create a folder called *Computation*, move sources and tests into that folder, and produce the Gradle build script *build.gradle* within this folder to automate the software build process for this project.

First, open a terminal, and ensure you have the newest version of Gradle (ver. 5.0+) installed with `gradle --version`.

Follow the steps below and add the snippets listed here to *build.gradle*, one after the other:

1. Create the following folder structure and a new *build.gradle* (empty) file within the *Computation* folder:

```
Computation
├── build.gradle
├── src
│   ├── main
│   │   ├── java
│   │   │   ├── application
│   │   │   │   ├── CompApp.java
│   │   │   │   ├── computation
│   │   │   │   │   ├── Computation.java
│   │   │   │   │   └── view
│   │   │   │   │       └── ComputationPage.java
│   │   └── test
│   │       ├── java
│   │       │   ├── computation
│   │       │   │   ├── AllTests.java
│   │       │   │   ├── ComputationTestAddSubstract.java
│   │       │   │   └── ComputationTestDivideMultiply.java
```

1. Add the `java` and the `application` plugins to the build configuration script *build.gradle*.

```
apply plugin: 'java'
// This plugin has a predefined 'run' task that we can reuse to use Gradle to
// execute our application
apply plugin: 'application'
```

2. Add JUnit libraries to the `dependencies` section.

```
repositories {
    mavenCentral()
}
dependencies {
    testImplementation "junit:junit:4.12"
}
```

3. Add and describe a new task `compile(type: JavaCompile)` to specify all source files (both application and test) and set the *build/bin* as destination dir to put all compiled class files in.

```
task compile(type: JavaCompile) {
    classpath = sourceSets.main.compileClasspath
    classpath += sourceSets.test.runtimeClasspath
    sourceSets.test.java.outputDir = file('build/bin')
    sourceSets.main.java.outputDir = file('build/bin')
}
```

**NOTE** | One can specify source sets and their variables the following way:

```
/*
 * specifying sourceSets is not necessary in this case, since
 * we are applying the default folder structure assumed by Gradle
 */
sourceSets {
    main {
        java { srcDir 'src/main/java' }
    }
    test {
        java { srcDir 'src/test/java' }
    }
}
```

4. Specify the main class and run the application.

```
mainClassName='application.CompApp'
```

In the command line issue `gradle run`

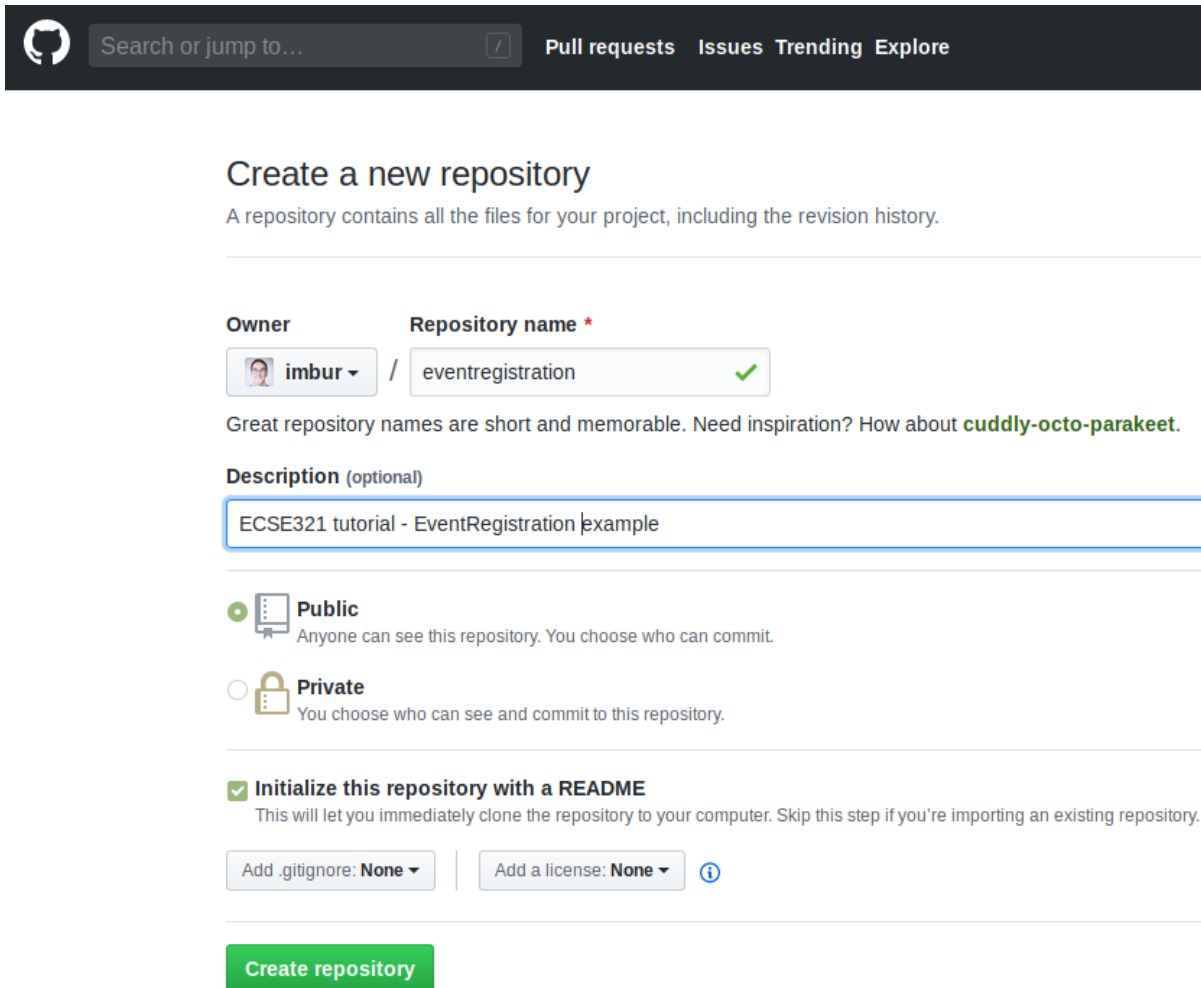
5. Describe the `jar` Gradle task (defined by the `java` plugin) to produce an executable jar file into *distributable/*.

```
jar {
    destinationDir=file('distributable')
    manifest {
        // It is smart to reuse the name of the main class variable instead of
        // hardcoding it
        attributes "Main-Class": "$mainClassName"
    }
}
```

**NOTE** | The `settings.gradle` and its usage is to be shown later.

## 1.6.2. Setting up a Spring/Spring Boot backend app with Gradle

1. Install the [Spring Boot CLI](#)
2. Create a new repository under your account on GitHub for an example application that we are going to develop throughout the semester. Name the repository **eventregistration**. See more on the specification of the application functionality later.



Search or jump to... Pull requests Issues Trending Explore

### Create a new repository

A repository contains all the files for your project, including the revision history.

**Owner** **Repository name \***

imbur / eventregistration ✓

Great repository names are short and memorable. Need inspiration? How about **cuddly-octo-parakeet**.

**Description (optional)**

ECSE321 tutorial - EventRegistration example

☒ **Public**  
Anyone can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None** ⓘ

**Create repository**

3. Clone it somewhere on your disk. We assume you cloned it to `~/git/eventregistration`.
4. Navigate to that folder in the terminal: `cd ~/git/eventregistration`.
5. Create a project for the backend application using Spring Boot CLI in this repository.

```
spring init \  
  --build=gradle \  
  --java-version=1.8 \  
  --package=ca.mcgill.ecse321.eventregistration \  
  --name=EventRegistration \  
  --dependencies=web,data-jpa,postgresql \  
  EventRegistration-Backend
```

### NOTE

Backslashes in this snippet indicate linebreaks in this one liner command typed in the terminal. You can select and copy-paste this snippet as-is.

6. Navigate to the *EventRegistration-Backend* folder

7. For future use, locate the *application.properties* file in the *src/* folder and add the following content:

```
server.port=${PORT:8080}

spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults = false
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQL9Dialect
```

**NOTE**

Source: <https://vkuzel.com/spring-boot-jpa-hibernate-atomikos-postgresql-exception>

8. Locate the Java file containing the main application class (*EventRegistrationApplication.java*) and add the following content

```
package ca.mcgill.ecse321.eventregistration;

import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.SpringApplication;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;

@RestController
@SpringBootApplication
public class EventRegistrationApplication {

    public static void main(String[] args) {
        SpringApplication.run(EventRegistrationApplication.class, args);
    }

    @RequestMapping("/")
    public String greeting(){
        return "Hello world!";
    }

}
```

9. Verify that it builds with **gradle build -xtest**.
10. Commit and push the files of the new Spring project.

```
git add .
git status #verify the files that are staged for commit
git commit -m "Initial commit of the backend application"
git push
```

## 1.7. Heroku

### 1.7.1. Preparations

1. Sign up/log in on Heroku by visiting <https://www.heroku.com/>.
2. Install the command line client for Heroku: [Heroku CLI](#)

#### NOTE

The Travis client might also be useful at later stages of the course, you can install it from here: [Travis CLI](#)

3. Log in to Heroku CLI by opening a terminal and typing: `heroku login`.

### 1.7.2. Creating a Heroku app

We are creating a Heroku application and deploying the *Hello world!* Spring example. Additionally, the steps below will make it possible to store multiple different applications in the same git repository and deploy them individually to Heroku. Steps will be shown through the example EventRegistration application, and should be adapted in the course project.

#### NOTE

All actions described here for configuring Heroku applications using the Heroku CLI could also be done via the web UI.

1. Once you are logged in with the Heroku-CLI, create a new Heroku application: in the root of the git repository of your repository (assumed to be `~/git/eventregistration`), issue `heroku create eventregistration-backend-<UNIQUE_ID> -n` to create an application named "eventregistration-backend-<UNIQUE\_ID>".

#### NOTE

In Heroku, the application name should be unique Heroku-wise, that is, each application in Heroku's system should have a unique name. If you don't provide a name parameter for the command, Heroku will randomly generate one.

2. Add the [multi procfile](#) and [Gradle](#) buildpacks to the app.

```
heroku buildpacks:add -a eventregistration-backend-<UNIQUE_ID>  
https://github.com/heroku/heroku-buildpack-multi-procfile  
heroku buildpacks:add -a eventregistration-backend-<UNIQUE_ID> heroku/gradle
```

#### CAUTION

Order is important.

### 1.7.3. Adding a database to the application

1. Open the Heroku applications web page and go to *Resources*, then add the Heroku Postgres add-on.



HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

Personal
>
eventregistration-backend-123

Open app
More

Overview
Resources
Deploy
Metrics
Activity
Access
Settings

Dynos

This app has no process types yet  
Add a Procfile to your app in order to define its process types. [Learn more](#)

Add-ons

Find more add-ons

The add-on `heroku-postgresql` has been installed. Check out the documentation in its Dev Center article to get started.

Quickly add add-ons from Elements

Heroku Postgres :: Database

Hobby Dev (Free)

Estimated Monthly Cost\$0.00

- Click the entry for Postgres within the list of add-ons, then go to *Settings*. You can see the database credentials there.

DATA

Datastores
>
postgresql-regular-49049

SERVICE heroku-postgresql
PLAN hobby-dev
BILLING APP eventregistration-backend-123

Overview
Durability
Settings

ADMINISTRATION

Database Credentials

Cancel

Get credentials for manual connections to this database.

Please note that **these credentials are not permanent**.  
Heroku rotates credentials periodically and updates applications where this database is attached.

Host	
Database	d57cs3lfifpdhn
User	
Port	5432
Password	
URI	
Heroku CLI	heroku pg:psql postgresql-regular-49049 --app eventregistration-backend-123

**NOTE**

The credentials are periodically updated and changed by Heroku, so make sure that you are using the actual credentials when manually connecting to the database. (E.g., during manual testing.)

#### 1.7.4. Extending the build for the Heroku deployment environment

1. Before deploying, a top level *build.gradle* and *settings.gradle* need to be created in the root of the repository (i.e., in *~/git/eventregistration*)

*build.gradle*:

```
task stage () {  
    dependsOn ':EventRegistration-Backend:assemble'  
}
```

*settings.gradle*:

```
include ':EventRegistration-Backend'
```

2. Generate the Gradle wrapper with the newest Gradle version

```
gradle wrapper --gradle-version 5.6.2
```

3. Create a *.gitignore* file for the *.gradle* folder:

*.gitignore*:

```
.gradle/
```

4. Add all new files to git

```
git add .  
git status #make sure that files in .gradle/ are not added
```

Expected output for **git status**:

```
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)
```

```
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)
```

```
new file:   .gitignore
new file:   build.gradle
new file:   gradle/wrapper/gradle-wrapper.jar
new file:   gradle/wrapper/gradle-wrapper.properties
new file:   gradlew
new file:   gradlew.bat
new file:   settings.gradle
```

Commit changes:

```
git commit -m "Adding Gradle wrapper"
```

### 1.7.5. Supply application-specific setting for Heroku

1. Within the *EventRegistration-Backend* folder, create a file called *Procfile* (**not** Procfile.txt, name it **exactly** Procfile) with the content:

```
web: java -jar EventRegistration-Backend/build/libs/EventRegistration-Backend-0.0.1-SNAPSHOT.jar
```

2. Add the Procfile to a new commit
3. Configure the multi-procfile buildpack to find the Procfile:

```
heroku config:add PROCFILE=EventRegistration-Backend/Procfile --app eventregistration-backend-<UNIQUE_ID>
```

### 1.7.6. Deploying the app

1. Obtain and copy the *Heroku Git URL*

```
heroku git:remote --app eventregistration-backend-<UNIQUE_ID> --remote backend-heroku
```

Output:

```
set git remote backend-heroku to https://git.heroku.com/eventregistration-backend-  
<UNIQUE_ID>.git
```

2. Verify that the **backend-heroku** remote is successfully added besides **origin** with **git remote -v**.  
Output:

```
backend-heroku  https://git.heroku.com/eventregistration-backend-123.git (fetch)  
backend-heroku  https://git.heroku.com/eventregistration-backend-123.git (push)  
origin         git@github.com:imbur/eventregistration.git (fetch)  
origin         git@github.com:imbur/eventregistration.git (push)
```

3. Deploy your application with

```
git push backend-heroku master
```

**NOTE**

If it fails to build, make sure you try understanding the output. Typical issue: buildpacks are not added/are not in the right order.

4. Visit the link provided in the build output. It may take some time (even 30-60 seconds) for the server to answer the first HTTP request, so be patient!
5. Save your work to the GitHub repository, too: **git push origin master**  
Final layout of the files (only two directory levels are shown and hidden items are suppressed):

```
~/git/eventregistration
├── build.gradle
├── EventRegistration-Backend
│   ├── build
│   │   ├── classes
│   │   ├── libs
│   │   ├── resources
│   │   └── tmp
│   ├── build.gradle
│   ├── gradle
│   │   └── wrapper
│   ├── gradlew
│   ├── gradlew.bat
│   ├── Procfile
│   ├── settings.gradle
│   └── src
│       ├── main
│       └── test
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
├── README.md
└── settings.gradle
```