DA3330 - Business Application Development

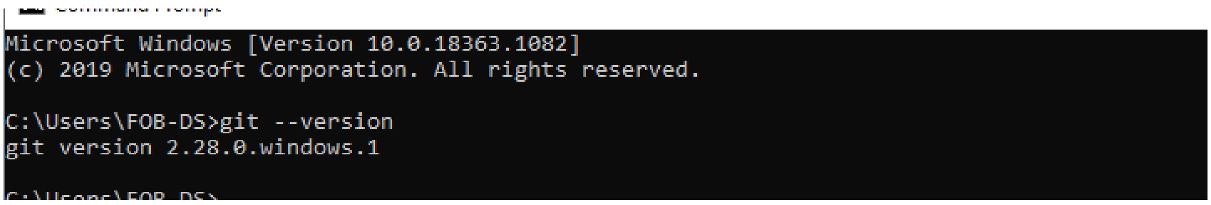
Tutorial – Version Controlling (Git)

1. Installing Git

Before you start using Git, you must make it available on your computer. Even if it is already installed, it is probably a good idea to update to the latest version. You can either install it as a package or via another installer or download the source code and compile it yourself.

Follow this link to download the setup: https://git-scm.com/downloads

After installation open Command Prompt.



```
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\FOB-DS>git --version
git version 2.28.0.windows.1
```

The first thing you should do after you install Git is to set your username and email address. This is important because every Git commit uses this information, and it is immutably baked into the commits you start creating:

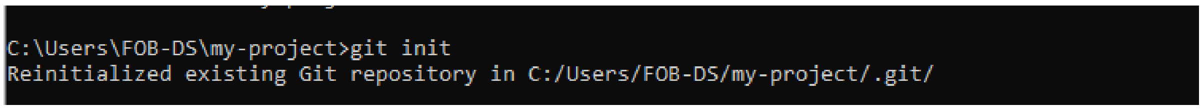
In the Command Prompt enter the following 2 commands with your name and email.

```
git config --global user.name "username"

git config --global user.email user@example.com
```

2. Creating a Repository

Create a new folder and name it my-project. Open a new Command Prompt window from this folder.

Then execute the command: `git init`



```
C:\Users\FOB-DS\my-project>git init
Reinitialized existing Git repository in C:/Users/FOB-DS/my-project/.git/
```

This creates a new subdirectory named *.git* that contains all your necessary repository files — a Git repository skeleton. At this point, nothing in your project is tracked yet. See Git Internals for more information about exactly what files are contained in the *.git* directory you just created.

2.1 Checking the Status of Your Files

The main tool you use to determine which files are in which state is the `git status` command. If you run this command directly after a clone, you should see something like this:

```
git status

On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

This means you have a clean working directory; in other words, none of your tracked files are modified. Git also does not see any untracked files, or they would be listed here. Finally, the command tells you which branch you are on and informs you that it has not diverged from the same branch on the server. For now, that branch is always master, which is the default; you will not worry about it here. Git Branching will go over branches and references in detail.

Go ahead and create a new text file to your project, and name it README.txt. Now, when you run `git status`, you will see your untracked file like this:

```
git status

On branch master

Your branch is up-to-date with 'origin/master'.

Untracked files:

  (use "git add <file>..." to include in what will be committed)

    README.txt

nothing added to commit but untracked files present (use "git add" to track)
```

You can see that your new README.txt file is untracked, because it is under the "Untracked files" heading in your status output. Untracked basically means that Git sees a file you did not have in the previous snapshot (commit); Git will not start including it in your commit snapshots until you explicitly tell it to do so. It does this so you do not accidentally begin including generated binary files or other files that you did not mean to include. You do want to start including README.txt, so let us start tracking the file.

2.2 Tracking New Files

To begin tracking a new file, you use the command git add. To begin tracking the README file, you can run this:

```
git add README.txt
```

Or, if you have lot files to stage you can simply indicate all files by using a . (dot) without specifying each and every file name.

```
git add .
```

If you run your status command again, you can see that your README file is now tracked and staged to be committed:

```
git status

On branch master

No commits yet

Changes to be committed:

  (use "git rm --cached <file>..." to unstage)

        new file:   README.txt
```

You can tell that it is staged because it is under the "Changes to be committed" heading. If you commit at this point, the version of the file at the time you ran `git add` is what will be in the subsequent historical snapshot. You may recall that when you ran `git init` earlier, you then ran `git add <files>` — that was to begin tracking files in your directory. The git add command takes a path name for either a file or a directory; if it is a directory, the command adds all the files in that directory recursively.

Let us change the file that was already tracked. Go ahead and type something in the README.txt and save it. Now let's run `git status` again.

```
git status

On branch master

No commits yet

Changes to be committed:

  (use "git rm --cached <file>..." to unstage)

        new file:   README.txt

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:   README.txt
```

The README.txt file appears under a section named "Changes not staged for commit" — which means that a file that is tracked has been modified in the working directory but not yet staged. To stage it, you have to run the `git add` command again. `git add` is a multipurpose command — you use it to begin tracking new files, to stage files, and to do other things like marking merge-conflicted files as resolved. It may be helpful to think of it more as "precisely add this content to the next commit" rather than "add this file to the project". Let us run `git add` now to stage the README.txt file again, and then run `git status`:

```
git add .

git status

On branch master

No commits yet

Changes to be committed:

  (use "git rm --cached <file>..." to unstage)

        new file:   readme.txt
```

2.3 Committing Changes

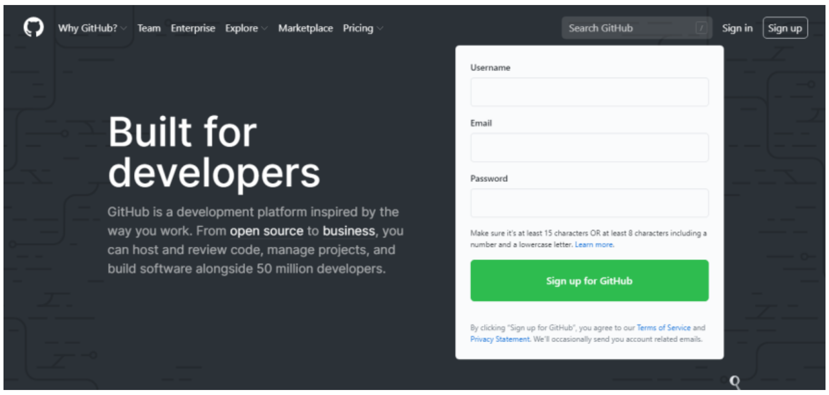
Although we have tracked the README.txt file is not saved to the Git's history of snapshots yet. For that we have to commit.

```
git commit -m "My first commit with readme file"

[master (root-commit) 63b9b4c] My first commit with readme file

 1 file changed, 1 insertion(+)

 create mode 100644 README.txt
```

With that Git will keep the current snapshot of the files in our repository in its history. Still, all the information about this repository is in our own computer. If something happen to our computer our project will be gone. So let's save our repository to GitHub.

3. GitHub

First go to github.com and create an account for free. If you already have an account log in.



Built for developers

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 50 million developers.

Username

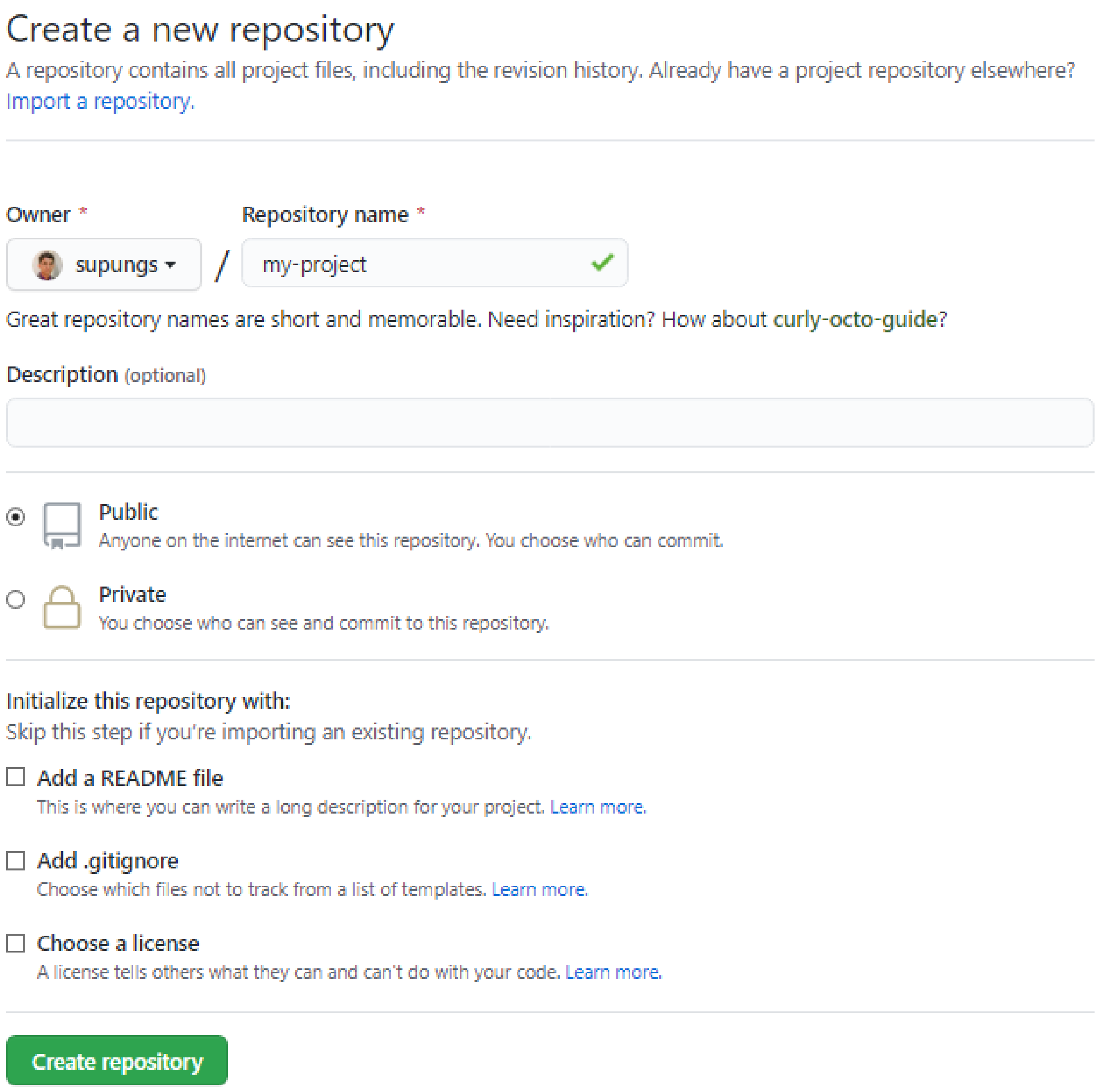
Email

Password

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.

By clicking "Sign up for GitHub", you agree to our Terms of Service and Privacy Statement. We'll occasionally send you account related emails.

Then create a new repository.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * supungs / Repository name * my-project

Great repository names are short and memorable. Need inspiration? How about curly-octo-guide?

Description (optional)

● 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.

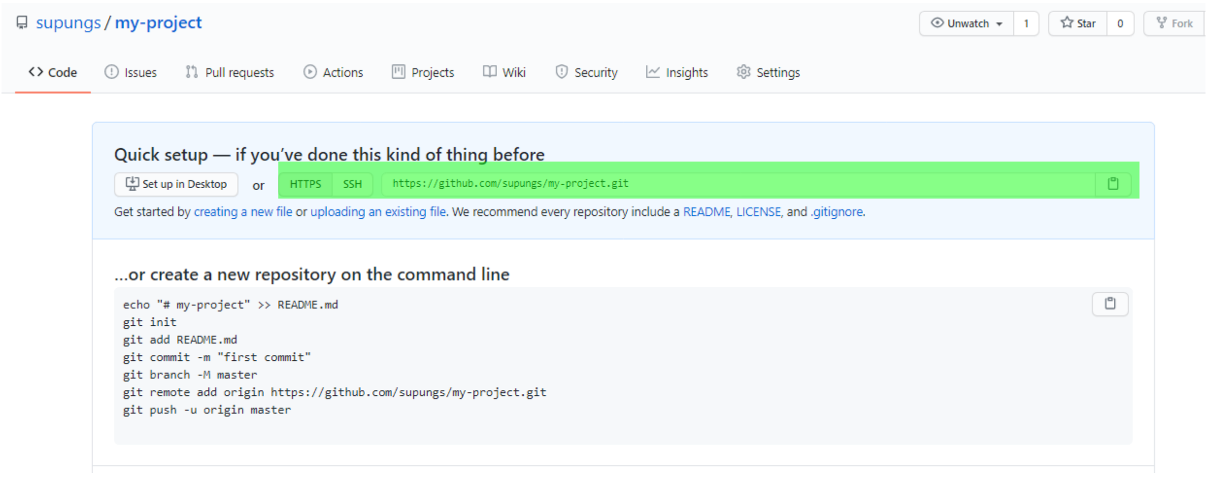
☐ Add a README file
This is where you can write a long description for your project. Learn more.

☐ 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.

3.1 Connecting to GitHub

In the project page, copy the project URL (highlighted in below image).



supungs / my-project

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

Set up in Desktop or HTTPS SSH https://github.com/supungs/my-project.git

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 "# my-project" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M master
git remote add origin https://github.com/supungs/my-project.git
git push -u origin master
```

Now, we need to add this URL as a *remote repository* to the git repository in our computer.

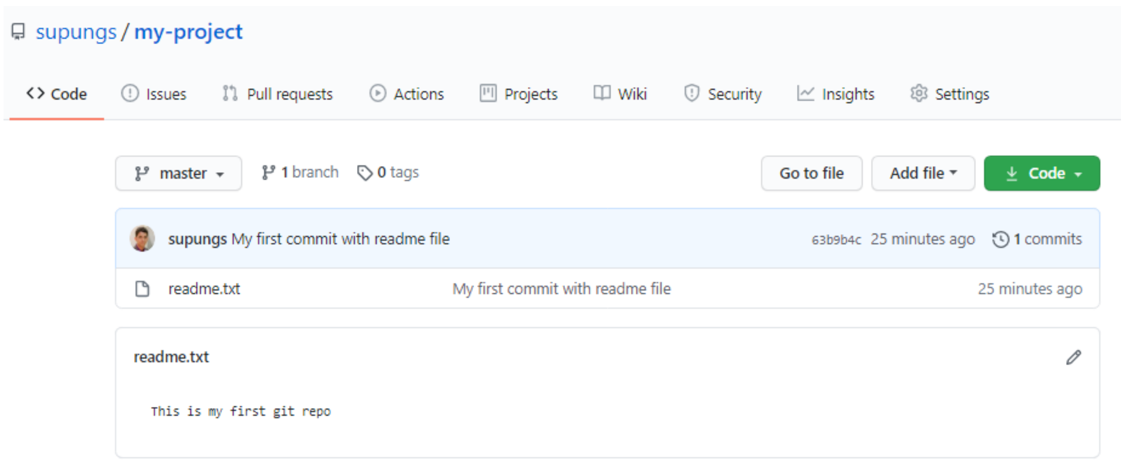
Go back to the command prompt and type the following command to add our GitHub repository as a remote to the local repository (Replace the URL with your own GitHub URL).

```
git remote add origin https://github.com/xxxx/xxxx.git
```

When you have your project at a point that you want to share, you must push it upstream to remote server. The command for this is simple: `git push <remote> <branch>`. Let’s push your master branch to your GitHub server, with the following command:

```
git push -u origin master

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Writing objects: 100% (3/3), 247 bytes | 247.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/supungs/my-project.git

 * [new branch]      master -> master

Branch 'master' set up to track remote branch 'master' from 'origin'.
```

* Note: If you try this in university labs, it might not work due restrictions set by university’s proxy servers.
Now, go back to GitHub in your browser and refresh. You should see the files you created in your local repository now in your GitHub repository.


supungs / my-project

supungs My first commit with readme file 63b9b4c 25 minutes ago 1 commits

readme.txt My first commit with readme file 25 minutes ago

readme.txt

```
This is my first git repo
```

4. Activity Submission

Go ahead and play around with your git repository as you want. Try creating branches, modifying files, merging and push the changes to GitHub. Finally, submit your GitHub repository URL to Moodle.

- Your GitHub repository should be public.
- You should have one or more small python code file/s created in branch/es, modified several times.
- You should have merged the changes back to master branch.

Bonus marks:

- Share your repository with a friend and ask him/her to make some changes to the repository and push back.