**B.Eng in Software and Electronic Engineering**

**Year 1**

**Web Technologies**

**Lab 6 – Using GIT and GitHub**

This lab will be about getting started with Git. We will use last weeks lectures on version control to get Git running on your system and get it set up to start working with. At the end of this chapter you should understand why Git is around and why you should use it.

You will also be set up to use Git for version control in all or any software modules that you would like.

**Section A – Using Git**

**Installing Git**

We’ll start by checking if you already have Git installed on your computer — and if it isn’t, we’ll install it. Type git --version on the terminal and press enter. If Git is installed, you’ll see the currently installed version, as in the screenshot below.



If you get something like git: command not found, then you will need to install Git. On MacOS, running git --version may already prompt some installation instructions, and you will just need to follow them. If that’s not your case, follow the instructions below, depending on your operating system:

* Windows: install [Git for Windows](https://gitforwindows.org/).
* Linux: run sudo apt-get install git on the terminal.
* MacOS: install [Git for Mac](https://sourceforge.net/projects/git-osx-installer/).

**1: Create a local git repository**

When creating a new project on your local machine using git, you'll first create a new [**repository**](https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository)(or often, '**repo**', for short).

To use git we'll be using “Git Bash”.

To begin, create a folder for todays exercise, namely Lab6\_GIT, go to this folder on windows explorer and open GIT BASH when inside this folder in this folder.

Graphical user interface, application

Description automatically generated

 Or we can open Git Bash and move to the correct location/ project folder as follows:

Text

Description automatically generated

To initialize a git repository in the root of the folder, run the [**git init**](http://git-scm.com/docs/git-init)command:

A screenshot of a computer

Description automatically generated with medium confidence

Next, you need to configure your name and e-mail — they will be used to identify you in your “commits” (I’ll explain what they are later).

Text

Description automatically generated

(If you get a permission error, just remove the word **–global** in the lines above)git

**2: Add a new file to the repo**

Add a new file to the Lab6\_GIT project folder, namely index.html

Graphical user interface, application

Description automatically generated

Once you've added or modified files in a folder containing a git repo, git will notice that changes have been made inside the repo. But, git won't officially keep track of the file (that is, put it in a commit) unless you explicitly tell it to.

 After creating the new file, you can use the [**git status**](http://git-scm.com/docs/git-status) command to see which files git knows exist.

Text

Description automatically generated

What this basically says is, "Hey, we noticed you created a new file called “index.html”, but unless you use the 'git add' command we aren't going to do anything with it."

***An interlude: The staging environment, the commit, and you***

*One of the most confusing parts when you're first learning git is the concept of the staging environment and how it relates to a commit.*

*A*[***commit***](http://git-scm.com/docs/git-commit)*is a record of what files you have changed since the last time you made a commit. Essentially, you make changes to your repo (for example, adding a file or modifying one) and then tell git to put those files into a commit.*

*Commits make up the essence of your project and allow you to go back to the state of a project at any point.*

*So, how do you tell git which files to put into a commit? This is where the*[***staging environment****or****index***](https://git-scm.com/book/en/v2/Getting-Started-Git-Basics)*come in. As seen in Step 2, when you make changes to your repo, git notices that a file has changed but won't do anything with it (like adding it in a commit).*

*To add a file to a commit, you first need to add it to the staging environment. To do this, you can use the*[***git add***](http://git-scm.com/docs/git-add)***<filename>****command (see Step 3 below).*

*Once you've used the git add command to add all the files you want to the staging environment, you can then tell git to package them into a commit using the*[***git commit***](http://git-scm.com/docs/git-commit) *command.*

*Note: The staging environment, also called 'staging', is the new preferred term for this, but you can also see it referred to as the 'index'.*

**3: Add a file to the staging environment**

Add a file to the staging environment using the **git add** command.

If you rerun the git status command, you'll see that **git add** has added the file to the staging environment (notice the "Changes to be committed" line).

Text

Description automatically generated

 To reiterate, the file has **not**yet been added to a commit, but it's about to be.

**4: Create a git commit**

It's time to create your first commit!

Run the command git commit -m "Your message about the commit"

Text

Description automatically generated

The message at the end of the commit should be something related to what the commit contains - maybe it's a new feature, maybe it's a bug fix, maybe it's just fixing a typo. Don't put a message like "asdfadsf" or "fubar". This is no help to others who see your commit.

After committing, run **git log**. This will show your commit history.

Text

Description automatically generated

*Note: you can also run the command git commit without the -m option — this will open a text editor that will ask you to input a commit message. But be careful, because depending on your system this may open a command-line text editor like*[*vim*](https://www.vim.org/)*, which*[*can be hard to exit*](https://stackoverflow.com/questions/11828270/how-do-i-exit-the-vim-editor)*if you don’t know how to use it. You can also*[*change the default text editor*](https://help.github.com/en/github/using-git/associating-text-editors-with-git)*with the command git config --global core.editor <editor> .*

**5: Create a new branch**

Now that you've made a new commit, let's try something a little more advanced.

Say you want to make a new feature but are worried about making changes to the main project while developing the feature. This is where [**git branches**](https://git-scm.com/book/en/v1/Git-Branching-What-a-Branch-Is)come in.

Branches allow you to move back and forth between 'states' of a project. For instance, if you want to add a new page to your website you can create a new branch just for that page without affecting the main part of the project. Once you're done with the page, you can [**merge**](http://git-scm.com/docs/git-merge) your changes from your branch into the master branch. When you create a new branch, Git keeps track of which commit your branch 'branched' off of, so it knows the history behind all the files.

Let's say you are on the master branch and want to create a new branch to develop your web page. Here's what you'll do: Run[**git checkout -b <my branch name>**](http://git-scm.com/docs/git-checkout). This command will automatically create a new branch and then 'check you out' on it, meaning git will move you to that branch, off of the master branch.

After running the above command, you can use the [**git branch**](http://git-scm.com/docs/git-branch) command to confirm that your branch was created:

Text

Description automatically generated

The branch name with the asterisk next to it indicates which branch you're pointed to at that given time. It shows that we are using the branch “feature1”

Add a new file, a CSS file called “style.css”

A screenshot of a computer

Description automatically generated with medium confidence

We can see that git has noticed there is a new file. Add the file and commit

Text

Description automatically generated

Now, if you switch back to the master branch

Text, timeline

Description automatically generated

Notice it is not showing our new file

Graphical user interface, application

Description automatically generated

Switch back to the feature1 branch

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

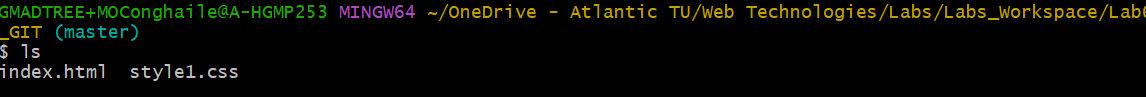
So the changes that you make in the branch are not seen by the master because we have not “merged” them.

To merge commits into the master branch, let's now switch over to the master branch.

Text

Description automatically generated

Now the new file is appearing in the master



Create another branch and then edit the styles1.css by adding some new text, e.g. some property for the h1 element.

Text

Description automatically generated

Commit the changes.

Text

Description automatically generated

Merge with the master like before

Text

Description automatically generated

Now switch to feature1 and edit the same file. This should cause a conflict when we try to merge

Text

Description automatically generated

See in our style1.css, some errors have been added

A screenshot of a computer screen

Description automatically generated

We need to edit this file so we can merge

Graphical user interface, text, application, chat or text message

Description automatically generated

Text

Description automatically generated