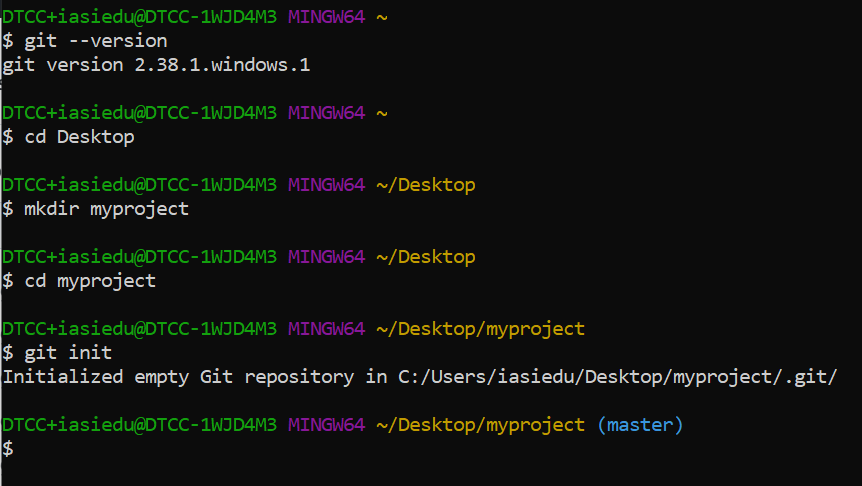
Notes:

* Get the version of git.
* Changed current directory to Desktop.
* Created a new directory(mkdir) ‘myproject’.
* Changed directory into ‘myproject’.
* Created git repository(git init).



* List files in the directory

Text

Description automatically generated

* Create html file in sublime text3 and save it as index.html

Text

Description automatically generated

* List files
* Check git status and see if it is part of the repo.
* Git is aware of the file but has not yet added it to the repo.
* Notes.doc is this word document saved in the repo. AS I update it, different versions will be created.

Text

Description automatically generated

Git staging environment.

Staged files are ready to be committed to the repo.

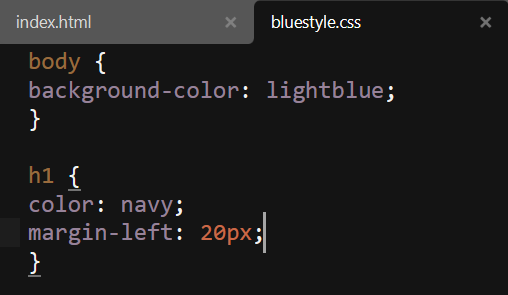
* Git add to add the html file to the staging area.

Text

Description automatically generated

Add more Files.

* A README.md file to describe the repo.(use a txt doc)
* This word document.
* A css file
* An updated html file.

 Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Git Commit

Adding commits keeps tracks of your progress. Git considers each “commit” as a save point. It is a point in the project where you can go back to if you find a bug or want to make a change. When we commit we should use a message to tell everyone what changes have been made.

The commit command performs a commit, and the -m “message” adds a message.

Text

Description automatically generated

Commit Without Stage

It is possible to commit changes directory without staging. The -a option will stage every changed and tracked file.

Let’s update the index.html and this document which is always been updated.

Use –short to check status, then commit.

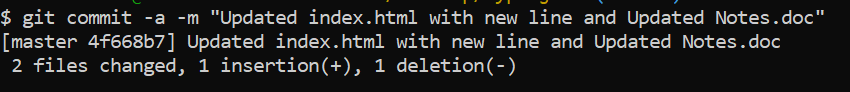
The index.html and the constantly updated notes have been updated.

Text

Description automatically generated with medium confidence

Now commit directly without staging.

Do not always do this as it can cause troubles.



Git Help:

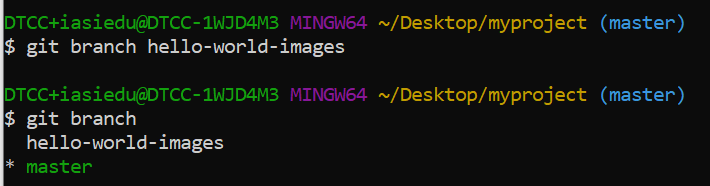
* git ‘command’ -help :will give you all available options for the specific command.
* git help –all :will provide all commands.

Git Branch

* A branch allows you to work on different parts of a project without impacting the main branch.
* You can have several branches and merge them with the main branch later.
* You can switch between different branches and work on different parts of the project.
* Branching is fast.

Now assume we do not want to disturb the main project: Create a new branch called “hello-world-images”

* Git branch <branch name>
* Git branch: to confirm the created branch.
* The \* beside the master means we are currently on the master.



* Use git checkout to move from the master to the branch that we currently created.

Text

Description automatically generated

Now we are on the new branch so we open a new text editor and make some changes and save. Let’s add a hello world image to the folder and modify the index.html file.

Text

Description automatically generated

Git status will tell us the current state of our documents on the new branch.

Text

Description automatically generated

Now add the files to the staging area and find the status.

Text

Description automatically generated

Now Commit them to the branch since everything is as required.

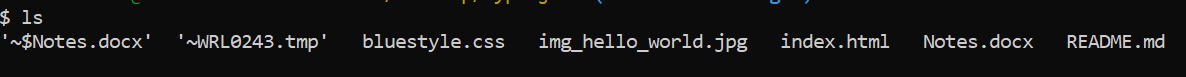
Text

Description automatically generated

We now have a branch different from the master. Using the -b option on checkout will create a new branch and move to it, does not exist.

**Switching Between Branches**

List the files in the current hello-world-images branch.



When you open the index.html file, you will see that the code has been modified to include the image.

The Notes file is also the latest update.

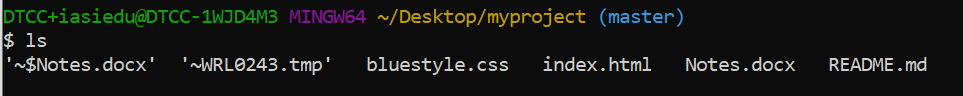
Now change the branch to master.

Text

Description automatically generated

NB: Closing the Notes.doc document could resolve the above issue.

We are now on master branch. And you can see that the image file is not here.



**Emergency Branch**

Imagine that we are not yet done with hello-world-images, but we need to fix an error on master.

We do not want to mess with master directly and we do not want to mess with the hello image since it is not yet done. In that case, we create a new branch to deal with the emergency.

Text

Description automatically generated

Now we have created a new branch that allows us to fix the error without disturbing the other branches.

We will make changes to the html file from the master branch on the new branch and get those changes to the master. Below are the changes.

Text

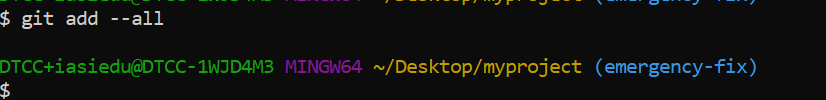
Description automatically generated

Git status

A screenshot of a computer

Description automatically generated with medium confidence

git add index.html (-all)



Now we have a fix ready for the master and we need to merge the 2 branches.

**Merge Branches**

We have the emergency fix ready, so we merge the master and the emergency-fix branch.

First we need to change to the master branch.

Text

Description automatically generated

Since the emergency-fix branch came directly from the master, and no other changes have been made to the master while we were working, Git sees this as a continuation of master.

We can now delete the emergency-fix branch.

Text

Description automatically generated

**Merge Conflict**

Now we can move to hello-world-images and keep working on it.

Add another image img\_hello\_git.jpg and change index.html to show the image.

First we need to change to branch master. Close the open document if ‘unlink’ pops up.

Text

Description automatically generated

Add a new image to the folder- img\_hello\_git.jpg and update the index.html file to show it:

Now stage and commit for this branch.

Text

Description automatically generated

We see some files changed. We are now ready to merge hello-world-images to the master branch. What will happen to the changes that we made recently?

Text

Description automatically generated

The merge failed as there is a conflict between the Notes.docx files and also the index.html files.

Check the status.

Text

Description automatically generated

The above confirms that the image files are ready to be staged but the docx and html files are conflicted.

Fix the conflicts. : You can use git add --all

Text

Description automatically generated with medium confidence

Git status shows the files have been added to the staging area and the conflicts resolves. We can use commit to conclude the merge.

Text

Description automatically generated

Merge the branch and master.

Text

Description automatically generated

Delete the branch since it is not needed now. You may leave it though.

Text

Description automatically generated

Before creating a local repo on Github, you should first configure your local repo to sync with the cloud.

Text

Description automatically generated

**Create a Repository on GitHub**

Graphical user interface, text, application

Description automatically generated

**Push Local Repository to GitHub**

Copy your Github url and use it in the below command:

git remote add origin https://github.com/iodsghana/git-lesson

**Git Pull from GitHub**

Pulling to Keep up-to-date with Changes.

When working as a team on a project, it is important that everyone stays up to date.

Any time you start working on a project, you should get the most recent changes to your local copy.

With Git, you can do that with pull.

pull is a combination of 2 different commands:

* fetch
* merge

Let's take a closer look into how fetch, merge, and pull works.

**Git Fetch**

fetch gets all the change history of a tracked branch/repo.

So, on your local Git, fetch updates to see what has changed on GitHub:

Text

Description automatically generated

Now that we have the recent changes, we can check our status:

Graphical user interface

Description automatically generated with low confidence

**Git Merge**

merge combines the current branch with a specified branch.

We have confirmed that the updates are as expected, and we can merge our current branch (master) with origin/master:

**Git Pull**

But what if you just want to update your local repository, without going through all those steps?

pull is a combination of fetch and merge. It is used to pull all changes from a remote repository into the branch you are working on.

Make another change to the Readme.md file on GitHub.

Use pull to update our local Git:

Text

Description automatically generated

That is how you keep your local Git up to date from a remote repository. In the next chapter, we will look closer at how push works on GitHub.

**Git Push to GitHub**

Push Changes to GitHub

Let's try making some changes to our local git and pushing them to GitHub.

Text

Description automatically generated

**Commit the changes:**

Text

Description automatically generated

And check the status:

Text

Description automatically generated

**Now push our changes to our remote origin:**

Text

Description automatically generated

Go to GitHub, and confirm that the repository has a new commit: