# Exercise 3: Git and Github

# 

1. Git

Objectives and Outcomes

In this exercise you will learn to install Git on your computer. Git is required for using all the remaining Node.js and Node based tools that we encounter in the rest of the course. At the end of this exercise, you would be able to:

* Install Git on your computer
* Ensure that Git can be used from the command-line or command-prompt on your computer
* Set up some of the basic global configuration for Git

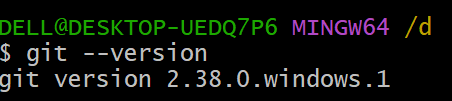
Downloading and Installing Git

* To install Git on your computer, go to <https://git-scm.com/downloads> to download the Git installer for your specific computing platform.
* Then, follow the installation steps as you install Git using the installer.
* You can find more details about installing Git at <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>. This document lists several ways of installing Git on various platforms.
* Installing some of the GUI tools like GitHub Desktop will also install Git on your computer.
* On a Mac, setting up XCode command-line tools also will set up Git on your computer.
* You can choose any of the methods that is most convenient for you.

Some Global Configuration for Git

* Open a cmd window or terminal on your computer.
* Check to make sure that Git is installed and available on the command line, by typing the following at the command prompt:

git --version



* To configure your username to be used by Git, type the following at the prompt:

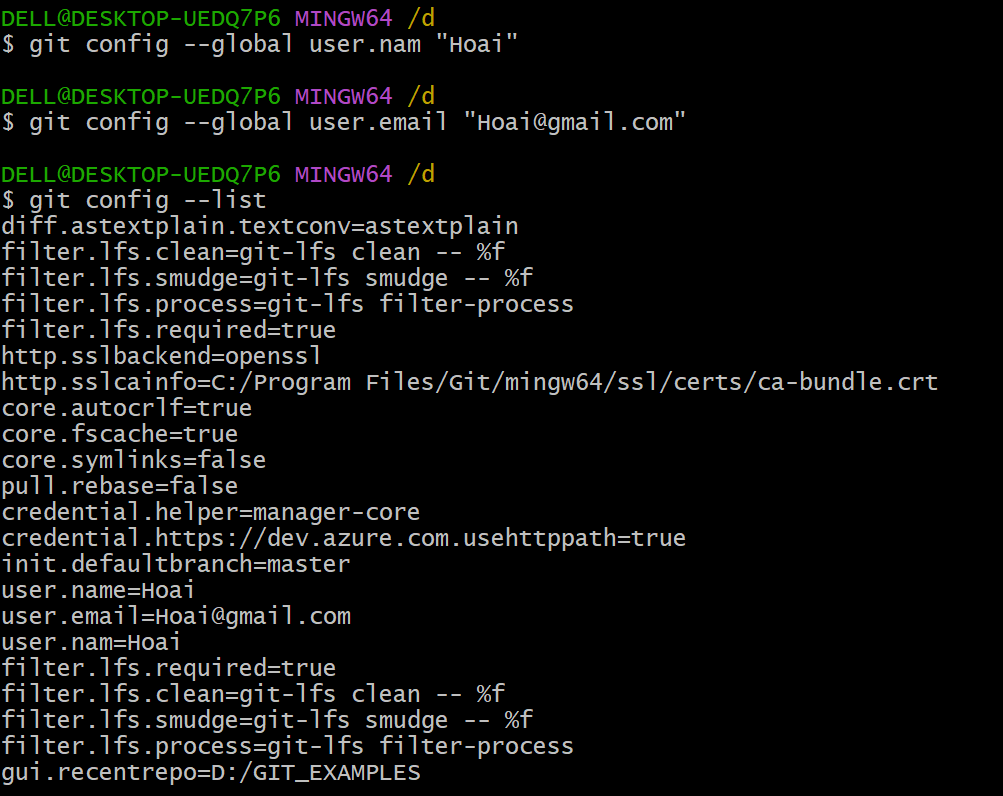
git config --global user.name "Your Name"

* To configure your email to be used by Git, type the following at the prompt:

git config --global user.email <your email address>

* You can check your default Git global configuration, you can type the following at the prompt:

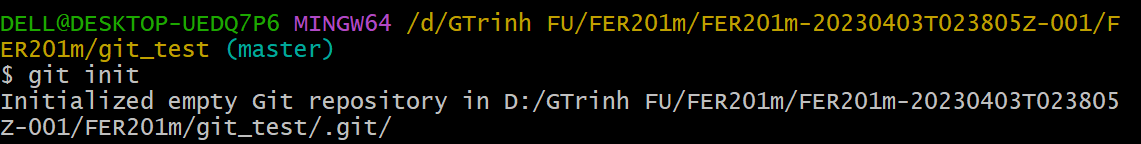
git config –list

****

Git Commands

* Git init

Git init is used to initialize a new Git repository in a directory. When you run git init, it creates a new, empty Git repository or re-initializes an existing one.

* 
* Git branch

Branches are highly important in the git world. By using branches, several developers are able to work in parallel on the same project simultaneously. We can use the git branch command for creating, listing and deleting branches.

****Creating a new branch:****

git branch <branch-name>

This command will create a branch **locally**. To push the new branch into the remote repository, you need to use the following command:

git push -u <remote> <branch-name>

****Viewing branches:****

git branch or git branch --list

****Deleting a branch:****

git branch -d <branch-name>

* Git checkout

This is also one of the most used Git commands. To work in a branch, first you need to switch to it. We use **git checkout** mostly for switching from one branch to another. We can also use it for checking out files and commits.

git checkout <name-of-your-branch>

There are some steps you need to follow for successfully switching between branches:

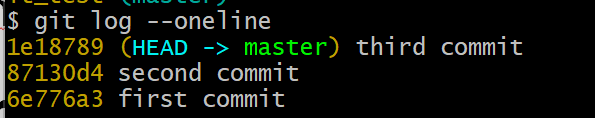
* The changes in your current branch must be committed or stashed before you switch
* The branch you want to check out should exist in your local

**There is also a shortcut command that allows you to create and switch to a branch at the same time:**

git checkout -b <name-of-your-branch>

This command creates a new branch in your local (-b stands for branch) and checks the branch out to new right after it has been created.





* Git status

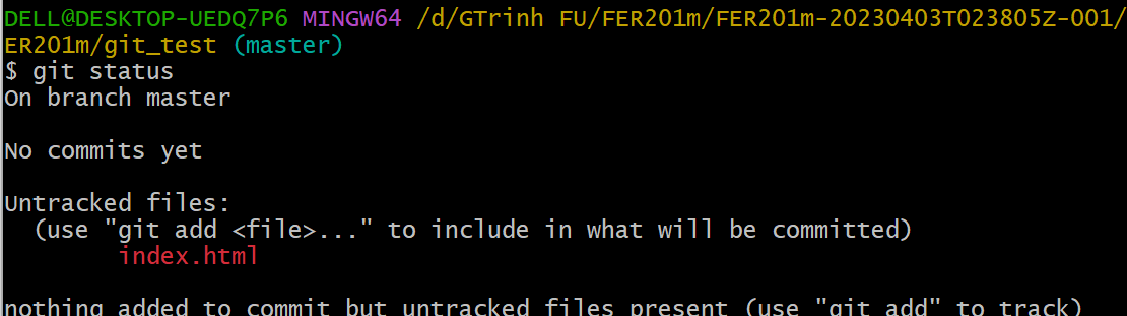
The Git status command gives us all the necessary information about the current branch.

git status

We can gather information like:

* Whether the current branch is up to date
* Whether there is anything to commit, push or pull
* Whether there are files staged, unstaged or untracked
* Whether there are files created, modified or deleted





**Git status gives information about the branch & files**

* Git add

When we create, modify or delete a file, these changes will happen in our local and won't be included in the next commit (unless we change the configurations).

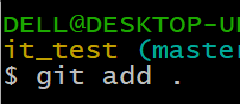
We need to use the git add command to include the changes of a file(s) into our next commit.

****To add a single file:****

git add <file>

****To add everything at once:****

git add -A



When you visit the screenshot above in the 4th section, you will see that there are file names that are red - this means that they're unstaged files. The unstaged files won't be included in your commits.

****To include them, we need to use git add:****

**Files with green are now staged with git add**

**Important: The git add command doesn't change the repository and the changes are not saved until we use git commit.**

* Git commit

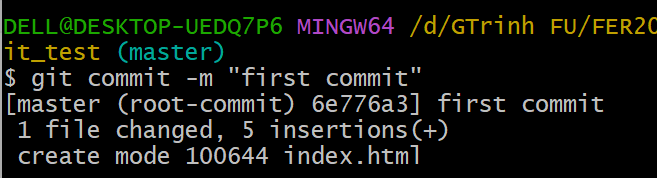
This is maybe the most-used command of Git. Once we reach a certain point in development, we want to save our changes (maybe after a specific task or issue).

Git commit is like setting a checkpoint in the development process which you can go back to later if needed.

We also need to write a short message to explain what we have developed or changed in the source code.

git commit -m "commit message"

**Important: Git commit saves your changes only locally.**



* Git push

After committing your changes, the next thing you want to do is send your changes to the remote server. Git push uploads your commits to the remote repository.

git push <remote> <branch-name>

However, if your branch is newly created, then you also need to upload the branch with the following command:

git push --set-upstream <remote> <name-of-your-branch>

or

git push -u origin <branch\_name>

**Important: Git push only uploads changes that are committed.**

* Git pull

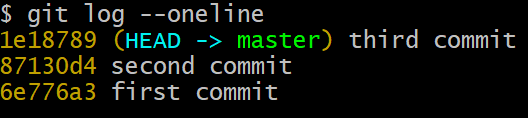
The **git pull**command is used to get updates from the remote repo. This command is a combination of **git fetch** and **git merge** which means that, when we use git pull, it gets the updates from remote repository (git fetch) and immediately applies the latest changes in your local (git merge).

git pull <remote>

**This operation may cause conflicts that you need to solve manually.**

* ****Git log -- oneline****

This command is used to display the commit history in a concise and simplified format, showing each commit on a single line. It provides a condensed view of the commit messages and their associated commit hashes.



* Git revert

Sometimes we need to undo the changes that we've made. There are various ways to undo our changes locally or remotely (depends on what we need), but we must carefully use these commands to avoid unwanted deletions.

A safer way that we can undo our commits is by using **git revert**. To see our commit history, first we need to use

Then we just need to specify the hash code next to our commit that we would like to undo:

git revert 3321844

After this, you will see a screen like below - just press **shift + q** to exit

The advantage of using **git revert** is that it doesn't touch the commit history. This means that you can still see all of the commits in your history, even the reverted ones.

Another safety measure here is that everything happens in our local system unless we push them to the remote repo. That's why git revert is safer to use and is the preferred way to undo our commits.

* Git merge

When you've completed development in your branch and everything works fine, the final step is merging the branch with the parent branch (dev or master). This is done with the **git merge** command.

Git merge basically integrates your feature branch with all of its commits back to the dev (or master) branch. It's important to remember that you first need to be on the specific branch that you want to merge with your feature branch.

For example, when you want to merge your feature branch into the dev branch:

****First you should switch to the dev branch:****

git checkout dev

****Before merging, you should update your local dev branch:****

git fetch

****Finally, you can merge your feature branch into dev:****

git merge <branch-name>

* Push helloapp into Git

You can initialize your project to be a Git repository by typing the following commands at the prompt:

git init

git add .

git commit -m "Initial Setup"

Thereafter you can set up an online Git repository and synchronize your project to the online repository. Make sure that the online Git repository is a **private** repository.

* Conclusions

At the end of this exercise, you should have Git available on the command-line of your computer.

1. GitHub

Objectives and Outcomes

In this exercise you will learn to use GitHub. GitHub has become an essential platform for developers, enabling collaboration, version control, and open-source contributions. It provides a robust infrastructure for managing projects, tracking changes, and facilitating teamwork. Whether you're an individual developer or part of a team, GitHub offers a powerful set of tools to enhance your development workflow:

* Set up the online repository as a remote repository for your local Git repository
* Push your commits to the online repository
* Clone an online Git repository to your computer

Setting up an Online Git repository

* Sign up for an account either at Bitbucket ([https://bitbucket.org](https://bitbucket.org/)) or GitHub ([https://github.com](https://github.com/)).
* Then set up an online Git repository named **git-test**. Note the URL of your online Git repository. Note that private repositories on GitHub requires a paid account, and is not available for free accounts.

Git Commands

* Set the local Git repository to set its remote origin

At the prompt, type the following to set up your local repository to link to your online Git repository:

git remote add origin <repository URL>

* Pushing your commits to the online repository

At the prompt, type the following to push the commits to the online repository:

git push -u origin master

* Cloning an online repository

To clone an online repository to your computer, type the following at the prompt:

git clone <repository URL>

Conclusions

In this exercise you have learnt to set up an online Git repository, synchronize your local repository with the remote repository, and clone an online repository.