Exercise path: C:\Users\mp3296\OneDrive - Zebra Technologies\GOALS\IDP\Software Engineering\Git and GitHub\learning-git-github-2421501-01\_02b

|  |  |  |
| --- | --- | --- |
| **Action** | **Description** | **Command** |
| **Add Files to Git (Stage the Files)** | Make sure the files you want to add are in your local project folder. | git add --all |
|  | Adds all files, including new, modified, and deleted files. | git add -A |
|  | Adds all files in the current directory to staging (excluding new or deleted files outside the directory). | git add . |
|  | To add specific files, list them by name. | git add file1.txt file2.jpg |
| **Commit the Changes** | After staging, commit the changes with a descriptive message. | git commit -m "Add initial project files" |
| **Push the Changes to GitHub** | Push your committed changes to GitHub, replace master with main if your repository uses main as the default branch. | git push origin master |
|  | (Alternatively, use main if that is the default branch) | git push origin main |
| **Check Status** | Check the status of your working directory to see which files are staged, unstaged, or untracked. | git status |
| **View Commit History** | View the history of commits in the repository. | git log |
| **View Commit History (One Line)** | View a simplified, one-line history of commits. | git log --oneline |
| **Undo Local Changes** | Discard changes in your working directory for a specific file. | git checkout -- <file> |
| **Undo All Local Changes** | Discard all local changes in the working directory (unstaged changes). | git checkout -- . |
| **Unstage a File** | Remove a file from the staging area, but keep the changes in the working directory. | git reset <file> |
| **Reset to Previous Commit** | Undo the last commit (keeping the changes in the working directory). | git reset --soft HEAD~1 |
| **Remove a File from Git** | Remove a file from the repository and staging area. | git rm <file> |
| **Remove a File but Keep It Locally** | Remove a file from the repository but keep it in your local working directory. | git rm --cached <file> |
| **Create a New Branch** | Create a new branch from your current commit. | git branch <branch-name> |
| **Switch Branches** | Switch to an existing branch. n | git checkout <branch-name> |
| **Create and Switch to a New Branch** | Create a new branch and immediately switch to it. | git checkout -b <branch-name> |
| **Merge Branches** | Merge another branch into your current branch. | git merge <branch-name> |
| **Delete a Branch** | Delete a branch locally (after merging). | git branch -d <branch-name> |
| **List All Branches** | List all branches in your repository. | git branch |
| **List Remote Repositories** | Show the remote repositories linked to your local repository. | git remote -v |
| **Add a Remote Repository** | Add a new remote repository (e.g., on GitHub). | git remote add origin <repository-URL> |
| **Change Remote URL** | Change the URL of an existing remote repository. | git remote set-url origin <new-URL> |
| **Fetch Latest Changes** | Fetch the latest changes from the remote repository without merging them into your local branch. | git fetch origin |
| **Pull Latest Changes** | Fetch and merge the latest changes from the remote repository into your current branch. | git pull origin main |
| **Clone a Repository** | Clone a repository from GitHub (or any other remote). | git clone <repository-URL> |
| **Tag a Commit** | Create a tag for a specific commit. Useful for marking release versions. | git tag -a <tag-name> -m "Release v1.0" |
| **Push Tags to GitHub** | Push local tags to GitHub. | git push origin --tags |

Version control

Centralised (stores the files and history on a server) and distributed (keep copies of the project under local machines/ cloning / working copy, changes are pushed from the copy to the main repository)

Git is a version/source control system. It allows you to crate branches ( a copy of the code).

GitBash to run Linux commands

Node.js

Setting up a project to use git:

Git config to setup a username and email, if you have a GitHub account, use the email address here.

A close up of a message

Description automatically generated

Open a terminal, you can use hyper on a Mac or GitBash on a Windows.

git config --global user.name "mp3296"

git config --global user.email “[maria.pricope@zebra.com](mailto:maria.pricope@zebra.com)”

git config --global user.name "mariapricope"

git config --global user.email “30245077@blackpool.ac.uk”

in VS Code open the files folder, open a new GitBash terminal

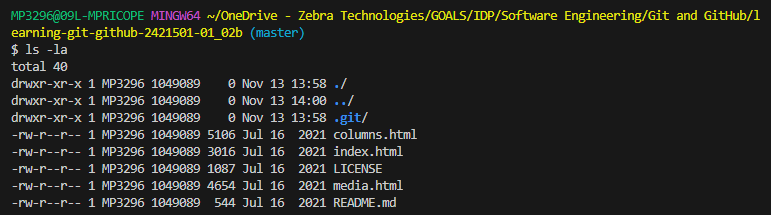
use git init to initialise a new empty Git repository:

A screenshot of a computer

Description automatically generated

This command creates an invisible Git folder where Git stores all the info about the project, which you can see by typing in Linux:

ls -la



You can switch to the git folder by doing this command:

cd .git

then to verify type again:

ls -la

A screen shot of a computer

Description automatically generated

To switch back to the previous directory use

cd ..

A screen shot of a computer

Description automatically generated

To issue a clear command:

Clear

A screen shot of a computer

Description automatically generated

Staging files

Staging is a temporary area that we can store files that we want to commit later on

A white box with black text

Description automatically generated

git add –all will add all files in the project to the staging area, the shortcut for this is git add -A

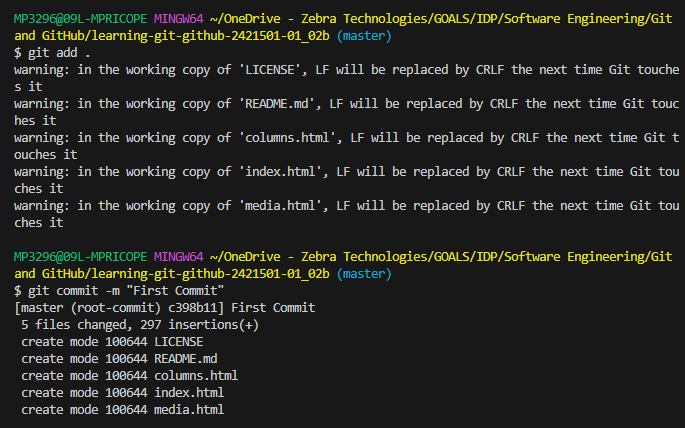
most GitHub commands have a shortcut that’s the first letter of the command, if there’s more than one command with the same letter, some will be capitalised, like this one.

Ther period is a short command for all files in the current directory git add .

Adding files to staging:

Git add .

Git commit -m “Always use comments with your commits”



The above will tell Git that this is one of the main point for our project.

We can now clear this with the clear command.

To check if Git is keeping track of our commands we can use git log command.

A computer screen shot of a computer code

Description automatically generated

Understanding git commands

A close up of a computer screen

Description automatically generated

The above list starts with the commit hash, which is a unique ID for the commit, next the commit shows that the head is in the main branch. Git uses branches to organise projects; each branch is like an alternate reality for the project. The head always points to the current reality which is called a branch. So we’re currently working on the “main” branch. By default, the branch is called main, however older versions of Git use the term master.

Also, the author name and email should be the same as what we’ve configured with the git config command. Next you will see the date and time the commit was made and then whatever message you wrote when you committed the file.

Git environments and states:

1. Working environment

* Files look like what they did during last commit

1. Staging

* Before you create a new commit you can move your files to a temporary location with the add command, this will allow you to queue up changes before you commit

1. Commit

* Once you move files using the git commit a log entry is created with a new hash

File states:

1. Tracked files

* Files that existed in the previous snapshot which is another name of the commit we did.
* Can be in several states:
  + Unmodified: files didn’t change since last commit
  + Modified: files have been changed
  + Staged: files have been moved into the staging environment

1. Untracked files

* Anything else like a new file added since the last commit

You can check the status / states of files by using git status

Clear

Git status

A black screen with yellow and blue text

Description automatically generated

We’re still on branch and have nothing to commit, our working tree is clean.

I’ve updated a file in the folder and it shows as changes are ready to commit, this could be done using the add . but also you can restore files:

A screen shot of a computer

Description automatically generated

After typing git add . we can run another status check and see the changes are ready to be committed:

A computer screen with text on it

Description automatically generated

A close-up of a white background

Description automatically generated

git restore --staged README.md

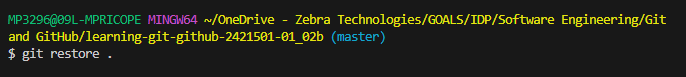
clear

git status

A screen shot of a computer program

Description automatically generated

Git restore has restored the changes made to the file:



You cannot use restore to delete untracked files:

A screen shot of a computer

Description automatically generated

Ignoring file:

A white background with black text

Description automatically generated

You don’t always want to track everything, such as files with sensitive information (authentication tokens, API keys)

A close-up of a computer code

Description automatically generated

Create a file called .gitignore and save there the folders/files you wish to ignore and the files will be ignored:

A screenshot of a computer program

Description automatically generated

You can create your own global ignore file and that way if you have a lot of different projects you can add this as a config variable pointing to a file in your hard drive. Whenever you create a new project, it will pick up the file automatically.

A close up of a logo

Description automatically generated

To clear cache use get rm -r --cached . (period is used for the current folder) and it will delete all the files that are cached recursively

A close up of a sign

Description automatically generated

Deleting and renaming.

To delete the file you can do this from VS or from windows explorer, after that if you run a git status it will show as something you need to add into staging, you can do git add and then

commit files.

A screen shot of a computer

Description automatically generated

You can also restore the file with git restore .

A black screen with yellow text

Description automatically generated

We can also use git rm index.html this deleted the file and automatically moved the deletion into staging ready to commit:

A computer screen with yellow and blue text

Description automatically generated

If we want to restore the index.html, type git restore .

A black background with yellow and blue text

Description automatically generated

To check use git status

A computer screen with text and numbers

Description automatically generated

If you rename a file for example index.html to home.html, it traces the index as deleted and home as addition:

A screen shot of a computer

Description automatically generated

You can do git restore . and delete manually the home:

A computer screen shot of text

Description automatically generated

To rename a file you can use

Git mv <fileexisitngname> <filenewname>

A computer screen with text and images

Description automatically generated

We can also go backwards to restore the name:

A computer screen shot of text

Description automatically generated

To hide path for each line, use PS1='$ '

A screen shot of a computer

Description automatically generated

Differences between files, for example if you remove a paragraph from a file since last commit:

Git diff

It will show you the file where things were deleted and what:

A screenshot of a computer program

Description automatically generated

If you undo your changes, git diff won’t return any changes: A black background with white text

Description automatically generated

Or use git add . to add files to staging

And then use git commit -m “comment” to commit.

To see the log of commits, use git log –oneline

A computer screen shot of white text

Description automatically generated

Use git diff and the has from a previous commit to see the difference:

A screen shot of a computer

Description automatically generated

Changing history:

Amend commits:

A white and black text

Description automatically generated

Git commit --amend (no message)

Git commit --amend -m “message” (with message)

Git commit --amend --no-edit (keeps the same message as the last commit)

If you move multiple files to a different folder and do git diff you will get a long message like this: A screen shot of a computer

Description automatically generated

For multiple files, use the source control button in VS code:

A screenshot of a computer screen

Description automatically generated

git config --global --list (to see the username and email)

git config user.name=”mariapricope”

git config user.email=“30245077@blackpool.ac.uk”

git config user.name=”mp3296”

git config user.email=[maria.pricope@zebra.com](mailto:maria.pricope@zebra.com)

to navigate to a folder:

cd "\OneDrive - Zebra Technologies\GOALS\IDP\Software Engineering\Git and GitHub\learning-git-github-2421501-01\_02b\docs"