**GIT FUNCTIONS**

* git – Shows all the functions in GitHub/Git Bash

**GIT CONFIGURATIONS - CONFIGURING USERNAME & EMAIL**

* git init - Initialises a new empty repo (Creates a brand-new Git repository, locally), no need for it if you have cloned a repo.

Global Option (Only need to set it once on your system):

* git config --global user.name "**[USERNAME]**" - Use Github Username
* git config --global user.email "**[EMAIL]**" - Use the email tied with your Github account

Local Option (Set on a per repo basis):

* git config user.name "**[USERNAME]**" - Use Github Username
* git config user.email "**[EMAIL]**" - Use the email tied with your Github account

To see Git Credentials:

* git config user.name - Shows username
* git config user.email - Shows email

Default Branch Name:

* git config --global init.defaultBranch main - Sets the 'default branch' as the 'main' NOTE: if you don't use this function, Git (by default) will create a branch called 'master' when a new repo is created and initialised by 'git init'

**CLONING A REPO**

* git clone **SSH Key** (or **Web URL**) – Clones a repository into a new directory

- SSH Key: git@github.com:**[Username]**/**[Repo-name]**.git

- Web URL: [https://github.com](https://github.com/%5BUsername%5D/%5BRepo-name%5D.git)/**[Username]**/**[Repo-name]**.git

* git fetch - fetches all branches/commits/files from a remote repo to your local repo.

\*\*\*git clone\*\*\* is a combination of:

* git init (create the local repository)
* git remote add (add the URL to that repository)
* git fetch (fetch all branches from that URL to your local repository)
* git checkout (create all the files of the main branch in your working tree)

**STAGING A CHANGE (TRACKING + CHECKING STATUS)**

* git status - Shows the working tree status (What you need to track)
* git add --all - Stage (Tracks) all file contents to the index
* git add . - Stage (Tracks) all untracked files and adds the changes to modified files.
* git add **[Filename]** - Tracks selected files
* git add \*.txt (or .py or .docx) - Tracks files with similar types of extension
* git rm --cached -r – Unstage/untracks the files/folders that are staged.

**COMMITTING A CHANGE & VIEWING COMMITS**

* git commit -m "**[Commit Message]**" - Saves changes to the repository and allows you to add a commit message (title)
* git commit -m "**[Commit Message]**" -m "**[Commit Message]**" - Second commit message acts as the description
* git commit --amend - Allows you to amend your commit

**PUSHING CHANGES**

* git remote -v - Check origin of remote repo
* git remote add origin **[url]** - Creating a connection between remote repo and local
* git remote set-url origin **[url]** - Setting the local origin to the repo URL
* git push -u **[Local] [Local\_Branch] >aka (**git push -u origin **[Branch])** – Pushes the changes to a remote repo
* git push --delete origin **[Branch]**

**FORKING**

* Go to the repo of choice, click fork, copy HTTPS URL/SSH Key and git clone in terminal
* git remote -v to see if you have cloned the correct repo
* Go to the **ORIGINAL** git repo and copy the repo **URL**
* git remote add upstream **[URL]** - Allows you to sync changes you make in a fork with the original repo.
* Check git remote -v to see if the upstream to the original repo is set
* git fetch upstream - Pulls any changes from original repo into yours
* git merge upstream/**[Name of Main Branch]** - Updating the main branch by merging the **[upstream]** main branch to the local **[origin]** main branch.
* git push **[Local] [Local Main Branch]** > git push **origin master** (or **main**) - After resolving merge conflicts (if any), push to update the main branch (local)

**BRANCHING**

* Two main branches exist: **main** and **develop**;from the **dev branch**, a **feature branch** is created. This is where new app features are tested out and merged back into the **dev branch**.
* git branch - Outputs a list of branches
* git branch **[Branch]** (or git branch -c **[Branch]**) - Creates a new branch
* git checkout **[Branch]** - Changing to a different branch
* git checkout -b **[Branch]** - Creates a new branch & sets it to that branch
* git checkout **[File.txt] –** Reverts the changes in the file to its last commit.
* git branch -d **[Branch]** - Deletes a branch

**MERGING**

* git merge **[Branch]** - Merges selected branch to your current branch.
* git rebase **[Branch]** - Same as git merge, except the commits appear to be in sequential order (when in reality it was in parallel).

**RETRIEVING REMOTE CHANGES (PULLING)**

* git pull **[Remote] [Remote\_Branch] >aka (**git pull origin **main)** – Downloads changes from the remote repo to the local repo. **Merges remote branch** with **current branch**.

**VIEWING COMMITS + REVERTING**

* git log - History of all commits made so far (Newest on top); date, commit message, SHA-1 - First 7 characters of the commit code (Shown clearer via the git log --oneline)
* git log --oneline - Displays output in a summarised, easy-to-read format
* git log --oneline --graph - Detailed version of git log (graphical representation)
* git log --branches=\* - History of all commits for all branches in that repo
* git log **[Branch]** - Log of specific branch
* git revert HEAD - Creates a new commit and undoes the latest commit
* git reset --hard **[SHA-1 Code]** - Returns the state to the selected commit. SHA-1 code is the first 7 characters of the long commit code from the **git log** function. (NOTE: **git log --oneline** better for convenience, only shows the commits first 7 chars)
* git reset --soft - Undoes last commit
* git reset --hard - removes last commit

**FILE PATH**

* pwd – Shows the file path

**IGNORING FILES**

* touch .gitignore - A list of intentionally untracked files that Git should ignore
* \*.py - Git will ignore all files with extension .py
* \*\*/venv - Git will ignore a file/folder with "venv" anywhere in the repo

**CHANGING DIRECTORIES**

* cd - Change directory (Change folders) - Users Home Directory
* cd ./ - Navigates forward to a different directory
* cd .. - Navigates back to the previous directory

**CREATING/OPENING FILES & FOLDERS**

* touch **[File]** - Creates a new file
* touch **[File1] [File2] [File3]** - Creates multiple files
* touch .hidden-**[File]** - Creates a new hidden file
* start **[File] -** Opens the file
* mkdir **[Folder]** - Makes a new directory (Folder)
* mkdir **[Folder1] [Folder2] [Folder3]** - Makes 3 new directories called 'F1' 'F2' and 'F3'
* echo **"[Enter a line of text]"** > **[File].txt** - Creates a text file with the words written in the quotation marks inside the file.
* > **[File].txt** (**.py** or **.docx**) - Creates a Text File/PyCode/Doc in the current directory
* cp **[File] [File]**-copy - Creates a copy of a file
* cp --recursive **[Folder] [Folder]**-copy - Creates a copy of a directory

**DELETING FILES & FOLDERS**

* rm **[File]** - Deletes a file
* rm \* - Deletes all files
* rmdir **[Folder]** - Deletes a directory

**RENAMING FILES & FOLDERS**

* mv **[File].txt [New File].txt** - Renames a file (works with .py and .docx)
* mv **[Folder] [New Folder]** - Renames a directory

**MOVING FILES & FOLDERS**

* mv **[File] [Folder]** - Moves the file into the directory

**PRINT FUNCTION**

* cat **[File].txt** - Displays the '"text" written within a file in the terminal
* cat **[File1].txt [File2].txt** - Concatenates files together; e.g. if **[File1]** has text "**123**", and **[File2]** has text "**abc**", the cat function will display "**123abc**" in the terminal.
* tail --lines=3 (or tail -n 3) **[File].txt** - Shows the last 3 lines of a text file.

**LISTS INFO, SIZE & PERMISSION**

* ls - List contents of a directory (Folder)
* ls --all (or ls -a) - List hidden files
* ls -l --all - More info on all files; L-R: Permissions/Owner of file/File size/Timestamp/Directory or File Name
* ls --size (or ls -s) - Size of the File
* -ls -l **[File]** - View Permission of the text-file; 1st char: - (File), 2nd chars: rw-r--r-- (Permission: User/Group/Other: Read(r)Write(w)Execute(x))

**EXTRAS**

* pwd - View the current directory you are in
* git stash
* git stast pop - Restores stashed changes
* git blame: Commit ID, Name of Committer, Timestamp of changes, Content of line
* git diff - Difference between current state and last commit
* git restore: – staged <file>... > discard changes in the working directory