

Learning GitHub:

<https://training.github.com/downloads/github-git-cheat-sheet.pdf>

<https://gist.github.com/octocat/9257657>

git init ⇒ this creates an empty repo on the file you selected

git add (filename OR . to add all) ⇒ this adds the files to the staging area :

git status ⇒ shows files ready to commit

git commit -m "message" ⇒ saves changes and adds a message (this will not commit changes on untracked files aka files that we didn't include in git add)

git config -l ⇒ displays current config such as user name

git log ⇒ displays commit messages

git rm file.txt ⇒ removes file from the directory ⇒ u need to commit after

git checkout filename.py ⇒ restores file to the last commit

git checkout -p restores files to the latest stored snapshot, reverting any changes before staging.

if u already staged using git add . you can use **git reset -p** to unstage

if u already committed u can use **git commit - - amend**

git revert HEAD or **commit ID** reverts commits by doing the opposite in a new commit

git branch displays branches and puts * next to current branch

git branch new_feature creates a new branch named new feature

git checkout branch_name switches you to that branch

git checkout -b branch_name creates a new branch and switches you to that branch

git branch -d branch_name deletes the branch

git merge branch_name tries to merge branches if there are conflicts fix them then git add . then commit

git merge - - abort ⇒ aborts the merge and resets to the latest commit

git reset --hard HEAD^ ⇒ restores to the last commit before the successful merge

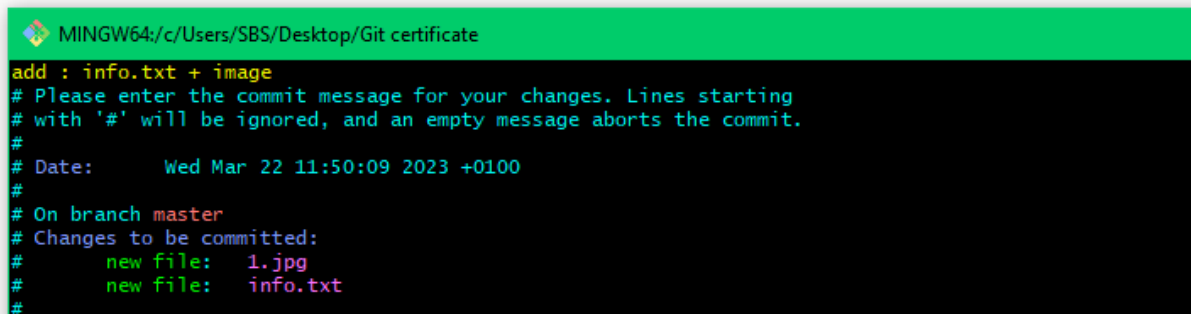
if you get trapped in vim 😊:

▲
304
▼

Had troubles as well. On Linux I used `Ctrl` + `X` (and `Y` to confirm) and then I was back on the shell ready to pull/push.

On **Windows GIT Bash** `Ctrl` + `X` would do nothing and found out it works quite like vi/vim. Press `i` to enter inline insert mode. Type the description at the very top, press `esc` to exit insert mode, then type `:x!` (now the cursor is at the bottom) and hit `enter` to save and exit.

If typing `:q!` instead, will exit the editor without saving (and commit will be aborted)



```
MINGW64:/c/Users/SBS/Desktop/Git certificate
add : info.txt + image
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Wed Mar 22 11:50:09 2023 +0100
#
# On branch master
# Changes to be committed:
#   new file:   1.jpg
#   new file:   info.txt
#
```

Question

What's the main difference between git fetch and git pull?

- ☒ git fetch fetches remote updates but doesn't merge; git pull fetches remote updates and merges.

if someone added a branch in the remote repo and you want to have it on your local
:

```

SBS@DESKTOP-2DL61M3 MINGW64 ~/Desktop/Git certificate (main)
$ git remote show origin
* remote origin
Fetch URL: https://github.com/AymenEssid1/GitCertificateRepo.git
Push URL: https://github.com/AymenEssid1/GitCertificateRepo.git
HEAD branch: main
Remote branches:
  experimental_branch tracked
  main                 tracked
  secondary_branch     tracked
Local branch configured for 'git pull':
  main merges with remote main
Local refs configured for 'git push':
  main          pushes to main          (up to date)
  secondary_branch pushes to secondary_branch (up to date)

SBS@DESKTOP-2DL61M3 MINGW64 ~/Desktop/Git certificate (main)
$ git checkout experimental_branch
Switched to a new branch 'experimental_branch'
branch 'experimental_branch' set up to track 'origin/experimental_branch'.

SBS@DESKTOP-2DL61M3 MINGW64 ~/Desktop/Git certificate (experimental_branch)
$ git branch
* experimental_branch
  main
  new-feature
  secondary_branch

```

In order to get the contents of a remote branch without automatically merging

⇒ **git remote update**

download remote branches from remote repositories without merging the content with your current workspace automatically ⇒ **git fetch**

after creating a new branch in local if you want to push it to remote where the new branch does not exists ⇒ **git push -u origin branch_name**

Command	Explanation & Link
git commit -a	Stages files automatically
git log -p	Produces patch text
git show	Shows various objects
git diff	Is similar to the Linux `diff` command, and can show the differences in various commits
git diff --staged	An alias to --cached, this will show all staged files compared to the named commit
git add -p	Allows a user to interactively review patches to add to the current commit
git mv	Similar to the Linux `mv` command, this moves a file
git rm	Similar to the Linux `rm` command, this deletes, or removes a file

There are many useful git cheatsheets online as well. Please take some time to research and study a few, such as [this one](#).

.gitignore files

.gitignore files are used to tell the git tool to intentionally ignore some files in a given Git repository. For example, this can be useful for configuration files or metadata files that a user may not want to check into the master branch. Check out more at: <https://git-scm.com/docs/gitignore>.

A few common examples of file patterns to exclude can be found [here](#).

Git Branches and Merging Cheat Sheet

Command	Explanation & Link
git branch	Used to manage branches
git branch <name>	Creates the branch
git branch -d <name>	Deletes the branch
git branch -D <name>	Forcibly deletes the branch
git checkout <branch>	Switches to a branch.
git checkout -b <branch>	Creates a new branch and switches to it .
git merge <branch>	Merge joins branches together.
git merge --abort	If there are merge conflicts (meaning files are incompatible), --abort can be used to abort the merge action.
git log --graph --oneline	This shows a summarized view of the commit history for a repo.

Git Revert Cheat Sheet

[git checkout](#) is effectively used to switch branches.

[git reset](#) basically resets the repo, throwing away some changes. It's somewhat difficult to understand, so reading the examples in the documentation may be a bit more useful.

There are some other useful articles online, which discuss more aggressive approaches to [resetting the repo](#).

[git commit --amend](#) is used to make changes to commits after-the-fact, which can be useful for making notes about a given commit.

[git revert](#) makes a new commit which effectively rolls back a previous commit. It's a bit like an undo command.

There are a [few ways](#) you can rollback commits in Git.

There are some interesting considerations about how git object data is stored, such as the usage of sha-1.

Feel free to read more here:

- <https://en.wikipedia.org/wiki/SHA-1>
- <https://github.blog/2017-03-20-sha-1-collision-detection-on-github-com/>

Some important terms

Inside the working tree are the files and folders that make up the project.

The staging area is where changes are prepared to be committed to the repository.
when using `git add` files are added to the staging area

Staging area (index)

A file maintained by Git that contains all of the information about what files and changes are going to go into your next commit

The Git directory contains all the information about the repository, including the commit history and metadata about the files and folders in the project.