How to Git

Table of Contents

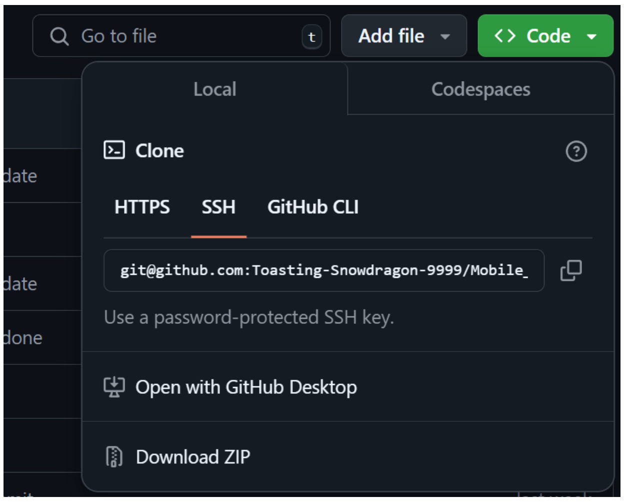
List of commands:

Nice to know commands:

```
$  git clone
$  git checkout
$  git branch
$  git add
$  git commit
$  git push
$  git pull
$  git merge
$  git status
$  git stash
$  git stash pop
```

Initial clone:

Go to the repository on GitHub or other git websites, find this:



```
git@github.com:Toasting-Snowdragon-9999/Mobile_
```

Use a password-protected SSH key.

Make sure to use SSH.

Then type

```
$ git clone <URL>
```

If repository consist of submodules use either of these two options:

1)

```
$ git clone --recurse-submodules <repo-url>
```

2)

This second option should be used if the repo is already cloned without submodules:

```
$ git submodule init
$ git submodule update --recursive
```

Configuring Git

You might need to setup config for email and username:

```
$ git config --global user.name "Your Name"
$ git config --global user.email "your-email@example.com"
```

Creating a branch:

To create a branch, use this method:

```
$  git checkout -b <New-branch-name>
```

this will create a branch and switch to it.

Then to setup upstream to remote repository on git webserver.

```
$  git push --set-upstream origin <New-branch-name>
```

Git push to remote repo:

First add the files you want to push:

```
$  git add <file> # This adds all listed files
$  git add .      # This will add all changes in working directory
$  git add -A     # This will add all changes in entire repository
$  git add -u     # This adds modification and deletions
```

When the files are added commit them:

```
$  git commit -m "commit message"
```

Now it would be wise to see if any changes are not being committed or anything to watch out for, so run a status

```
$  git status
$  git push
```

If any errors occurred, check internet connection, ssh key and that a upstream is setup for the branch, if not see Creating branch but only the git push.

Stashing

Git stash is useful for storing temporary changes, the stash will store you changes, making it possible to checkout without committing your changes then swap branch or do something else before popping the changes back.

```
$  git stash   # allows you to store uncommitted changes temporarily
```

Note: It does not stash untracked or ignored files by default.

```
$  git stash pop # This will pop the stored changes back
$  git stash clear # Deltes all stashes
```

Git merge:

Don’t merge a repo that is not up to date, this means first merge main/master branch into your branch, making sure that your branch consists of everything from the master/main branch, this is where the conflict will happen and should be resolved.

When your branch is then up to date, you can merge it into main/master, this should create no merge conflict because you have already made sure to fix conflict.

ALWAYS TEST THE CODE IN YOUR BRANCH, ONLY MERGE WORKING CODE INTO MAIN/MASTER BRNACH.

First make sure you are on the branch to merge:

```
$  git status                 # Use this to check what branch your on
$  git checkout <branch>      # swap to branch you want to merge
```

Fetch the latest changes to your local repository:

```
$  git fetch origin
```

Now merge main branch into your branch you want to merge:

```
$  git merge <main branch> # merge with main/master branch
```

Fix the merge conflicts.

When they are fixed, add, commit and push the new changes to you branch:

```
$  git add <attribute>
$  git commit -m “Merge message2
$  git push origin <branch>
```

Now switch to main branch and get the latest changes, then merge your branch into main branch:

```
$  git checkout <main branch>
$  git pull origin <main branch>
$  git merge <branch>
$  git push origin <main branch>
```

The merge should have no conflicts here.

Setup SSH key Linux

First make sure that a ssh client is installed:

```
$ sudo apt update && sudo apt install openssh-client
$ ssh -V
```

make sure an agent is running:

```
$ ps -ax | grep ssh-agent   # this wil output if an agent is running
$ eval $(ssh-agent)         # this will start an agent
$ echo 'eval $(ssh-agent)' >> ~/.bashrc    # starts up agent auto
```

.bashrc can be replaces with whatever underlying shell is being used

Now navigate to root, and create ssh-key pair

```
$ cd ~
$ ssh-keygen -t ed25519 -b 4096 -C
  "<username@emaildomain.com>" -f ~/.ssh/<ssh-key-name>
```

Add it to the ssh client:

```
$ ssh-add ~/.ssh/ssh-key-name>
```

If you want to have multiple ssh keys for different git websites use this:

```
$ ls ~/.ssh/
```

if there is no config, make one like this:

```
$ touch ~/.ssh/config
```

Then use nano of vim to add this part to the config fike:

```
Host github.com
     AddKeysToAgent yes
     IdentityFile ~/.ssh/<ssh-key-name>
```

The github.com can be replaces with other git websites.

The ssh-client will then use that key on that website