**1. Configuration**

Use git config command to configure your user name and email for Git. It’s a good idea to set these from the beginning, so that your name will be displayed in the contribution history.

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

git config --global user.email yourname@example.com

**2. Initialization**

To initialize a Git repository run following command in your project directory:

git init

You may notice that a new *.git* directory is created inside your project directory. NOTE: *.git* is a hidden directory so you may want to enable hidden files and folders view on your system.

**3. Cloning**

To clone an existing remote repository use:

git clone https://github.com/username/reponame.git

This command creates a copy of a remote repository by downloading all the files.

To clone a local repository use:

git clone path/to/the/local/repository

**4. Adding/removing files**

After you’ve made changes to your project you can add changed files to a temporary staging with git add command.

git add file1 file2 file3

To add all files use dot:

git add .

To remove a file from a temporary staging use git reset command:

git reset file1 file2 file3

To remove all files use:

git reset .

**5. Checking status and changes**

These commands can help you overview the status of your changes before adding or committing.

After changes are done or the file gets added or removed from temporary staging, you can check the current status of the situation.

git status

This command will show you new, deleted, or modified files as well as if the file is added to the temporary staging.

To see the changes you’ve made use:

git diff

**5.1. Stashing**

If for some reason you need to save changes and apply them later Git allows you to temporarily stash your changes:

git stash

List all stash changes:

git stash list

Apply changes from latest stash:

git stash apply

Apply specific stash from index:

git stash apply stash@**{**0**}**

**6. Committing**

Once you’ve added your files and ready to commit them use the command:

git commit

NOTE: this command will prompt you with a commit message screen, where you have to input your commit message, save it and close the screen. Many beginners [struggle with this screen](https://stackoverflow.com/questions/13239368/git-how-to-close-commit-editor). To make it easier you can write it in one line by adding -m option:

git commit -m "Your commit message"

**7. Undoing/reverting commits**

Before you pushed any commits there’s a possibility to revert or undo them. The git reset command will allow you to roll back to a given commit. Suppose you’ve made several commits already.

To undo/revert the last commit use:

git reset HEAD~

To revert to a specific commit, you’ll need to specify the [commit SHA](https://blog.thoughtram.io/git/2014/11/18/the-anatomy-of-a-git-commit.html) key. To do that you’ll need to view your history of changes with:

git log

It will output a list of your commits, each containing a unique SHA key. Copy that key and run:

git reset 0447816744595c69968bb517b93bc4f5b42df20f

**8. Pushing**

Once you’ve satisfied with your changes and have committed all of them you’re ready to push them to your repo.

**8.1. Syncing**

Before the actual push, you’ll need to synchronize your local project instance with the remote repository changes. To do that you’ll need to fetch and merge the latest changes from the remote repo.

git pull

If you only want to see the changes without merging then use fetch command:

git fetch

To finally push changes run:

git push

**9. Branching**

Branches allow you to work safely and independently on different features. Git allows to have multiple branches. A few commands that will help you quickly get into the issue.

See the list of all local branches and the current active one:

git branch

Create a new branch:

git branch feature-x

Change the branch:

git checkout feature-x

Delete the branch:

git branch -d feature-x

A few useful commands to be more productive.

Checkout to the previous branch:

git checkout -

Create a new branch and immediately checkout to it:

git checkout -b feature-y

Rename current branch:

git branch -m feature-z

**10. Remote repos**

If you’re going to collaborate on project with multiple people, you’ll need to know few commands to set up your local project just for that.

To check all of your [remote repositories](https://docs.github.com/en/github/using-git/managing-remote-repositories):

git remote -v

To connet your local project to a remote repository:

git remote add origin https://path.to/remote/repository.git

creating username

* git config --global user.name <username>

Creating email

* git config --global user.Email <email address>

Display all the configurations

* git config –list

creating a repository

* git init

To see where all your files are in the directory or staging area

* git status

To check any changes that has been committed to the repository

* git log

To add files from the work directory to the staging area

* git add <file name>
* git add.

to add files from the staging area to the repository

* get commit -M <to leave a comment>

to compare files from the working area to the staging area the command is

* git diff
* git diff --stage or git diff --stage HEAD
* git diff HEAD

Compare the 2 comments within the depositor

* git diff <commit ID or the first 6 digits of the ID> <second ID>

To see how many branches there are

Git branch

To create a new branch

Git branch < new branch name >

To switch from one brunch to another brunch

Git checkout < brunch name >

Create and immediately switch

git checkout –b <branchname>

To commit changing to a selected brunch

Git push origin <brunch name >

Delete a branch

git branch –d <branchname>

Merging two brunches

Git checkout <master branch name>

Git merge < local branch name>.

Undo changes from working directory

Git restore <file name>

Undo changes from staging area

Git restore –-staged <file name>

Undo changes from local repository to working directory

Git reset HEAD^

to squeeze all the comments into one comment

git rebase -i HEAD~<number of comics to be squeezed>

a rebase takes the contents of one branch after the “split” and moves them to the end of the other branch

git rebase <basebranch>