Git : version control system

Git bash is also a good terminal to use python.

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video

$ git config --global user.name Imran

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video

$ git config --global user.email imrantipu90@yahoo.com

Initialization:

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video

$ git init

Initialized empty Git repository in C:/Users/Imran/OneDrive/Desktop/git one video/.git/

**ls** -**lart** # List files in the current dir by the last time they were modified. (-l = long, -a = all, -r = reverse, -t = order by mod time) shows also hidden files

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ ls -lart

total 12

drwxr-xr-x 1 Imran 197121 0 May 1 22:13 ../

drwxr-xr-x 1 Imran 197121 0 May 3 14:31 ./

drwxr-xr-x 1 Imran 197121 0 May 3 14:31 .git/

(I am now in .git folder)

Status check:

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

index.html

nothing added to commit but untracked files present (use "git add" to track)

untraced > staged > commit > modified > Again staged

staged :

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git add index.html

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: index.html

Commit ( to save):

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git commit

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

#

# On branch master

#

# Initial commit

#

# Changes to be committed:

# new file: index.html

Now press I to insert( Initial commit)

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

Initial commit

# On branch master

#

# Initial commit

#

# Changes to be committed:

# new file:

<esktop/git one video/.git/COMMIT\_EDITMSG[+] [unix] (15:33 03/05/2022)4,15 All

**-- INSERT --**

Press Esc then write :wq (shows me the changes I make)

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

Initial commit

# On branch master

#

# Initial commit

#

# Changes to be committed:

# new file:

<esktop/git one video/.git/COMMIT\_EDITMSG[+] [unix] (15:33 03/05/2022)4,15 All

**:wq**

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git commit

[master (root-commit) cb8ed88] Initial commit

1 file changed, 1 insertion(+)

create mode 100644 index.html

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git status

On branch master

nothing to commit, working tree clean

To make blank file( touch)

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ touch about.html

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ touch cotact.html

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ touch monuments.html

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git status

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

about.html

cotact.html

monuments.html

nothing added to commit but untracked files present (use "git add" to track)

Staging area for all files:

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git add -A

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: about.html

new file: cotact.html

new file: monuments.html

Different way to commit all files:

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git commit -m "Added more htmls"

[master 3b03894] Added more htmls

3 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 about.html

create mode 100644 cotact.html

create mode 100644 monuments.html

clear the terminal :

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ clear

To check last file change or last commit

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git checkout contact.html

$ git checkout -f ( for all last checked files)

To see all commit: use log

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git log

commit 3b0389499cfb016fe7577ca1c8c5c22bc7f642ba (HEAD -> master)

Author: Imran <imrantipu90@yahoo.com>

Date: Tue May 3 16:11:02 2022 +0200

Added more htmls

commit cb8ed8819f8b5763cc34d576052f41fb208e98cd

Author: Imran <imrantipu90@yahoo.com>

Date: Tue May 3 15:33:18 2022 +0200

Initial commit

Out of 100 if I want to see only 1 commit

Imran@DESKTOP-V3O6BGB MINGW64 ~/OneDrive/Desktop/git one video (master)

$ git log - p -1

$ git diff ( working tree compared with staging area) (differenc

$ git diff -–staged ( staging area compared with last commit)

$ git rm –-cached waste.html ( file will be in staging area)

$ git rm waste.html ( file will be completely deleted)

FROM BOOK HEAD FIRST;

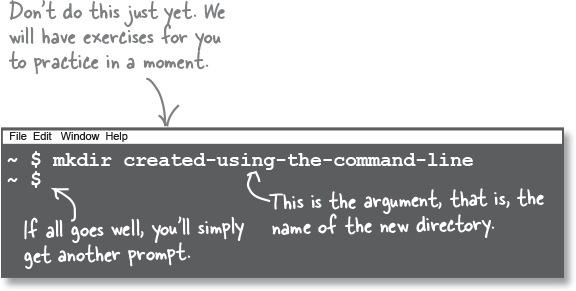
Creating a Git repository involves running the git init command inside the top folder of your project.

creating a Git repository, you really can’t do much with Git.

the top (or root) folder of that project needs to have git init run to get things started with Git.

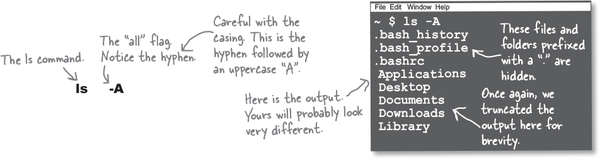
Type pwd and hit return; pwd stands for “print working directory” and it displays the path of the directory the terminal is currently running in

Speaking of new folders, the command for creating new folders is mkdir, which stands for “make directory.” which is the name of the directory you wish to create( folder name)

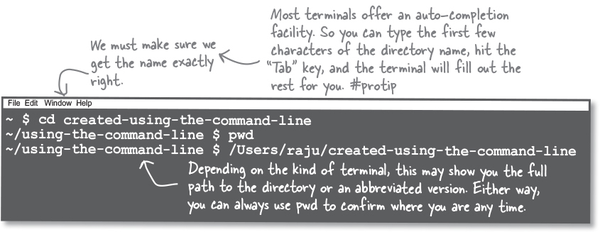


you can list all the files in the current directory. The listing command is named ls (short for list).

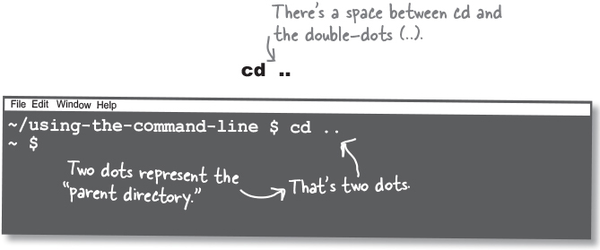
To see hidden files and folders as well. To do that, you can supply ls with a flag. Flags, unlike arguments, are prefixed with a hyphen (to differentiate them from arguments)



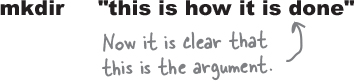
Next, moving around! We created a new directory, but how do we navigate to it? For that, we have the cd command, which stands for “change directory.



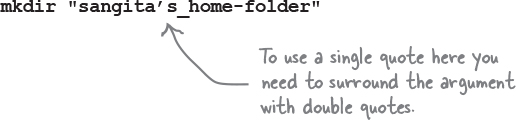
cd navigates to a subdirectory under the current directory. To hop back up to the parent directory, we can also use cd, like so



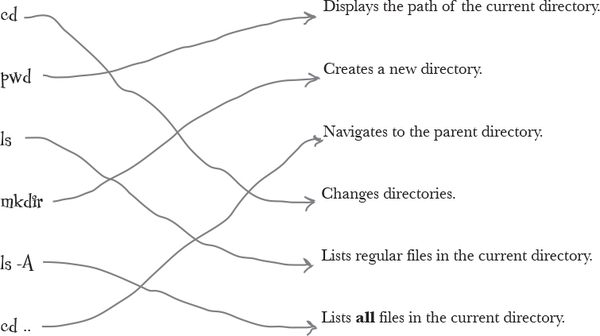
So, anytime you have whitespace in an argument and you wish to treat it as one argument, you need to use quotes.



The command line does not really care if you use double quotes or single quotes.



Anytime you need a space, simply use a hyphen or an underscore. This helps you avoid using quotes (of any kind) when supplying arguments.



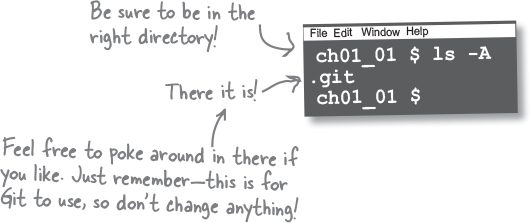
let’s create our first Git repository. To do this, we simply run git init inside our newly created folder

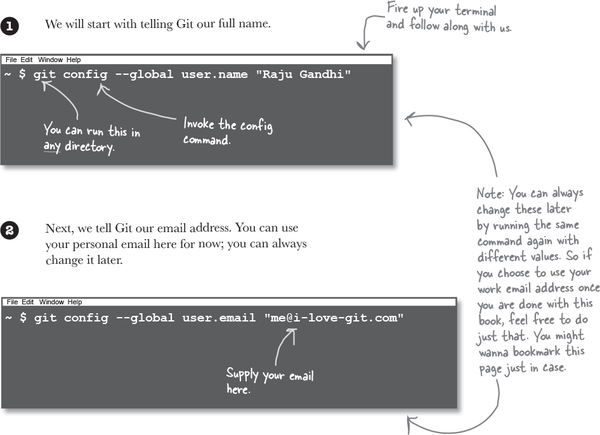


git init, where init is short for initialize. Git realizes we are asking it to create a repository at this location, and it responds by creating a hidden folder called .git and stuffs it with some configuration files and a subfolder where it will store our snapshots when we ask it to.

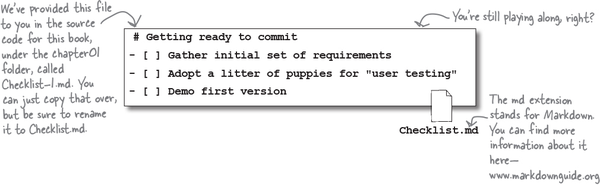


One way to confirm this happened is by listing all the files using our terminal, like so.



Git expects you to give it a few details about yourself. This way, when you do create a “snapshot,” Git knows who created it. 

Well, asking Git to “save your progress” involves “committing” your work to Git. Essentially, this means that Git stores a revision of your work. Once you do that, you can continue working away merrily till you feel it’s time to store another revision, and the cycle continues.



Save the file as Checklist.md in the HawtDawg directory.

Now we are ready to commit our work. This involves two Git commands, namely git add and git commit.



Notice that the git add command takes as its argument the name of the file you wish to add to Git. And the git commit command has a flag, -m, followed by the commit message. The -m stands for “message” and is a mechanism for you to provide a meaningful reminder as to why you made this change.

You have completed a whirlwind tour of Git. You installed Git, initialized a Git repository, and committed a file to Git’s memory.

We learned that committing to Git is a two-step process. You first add the files and then commit

Checklist.md and README.md, but you only added Checklist.md. When you create a commit, Git will only store the changes made to Checklist.md

Git creates a commit object that it stores inside the .git folder

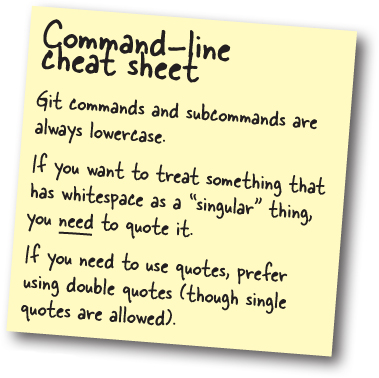
The Git repository itself is divided into two parts—the first part is called the “index,” and the second part is what we will refer to as the “object database.”

When we run git add <filename>, Git makes a copy of the file and puts it in the index. We can think of the index as the “staging area,” wherein we can put things till we are sure we want to commit to them.

Now when we run the git commit command, it takes the contents of the staging area and stores those in the object database, also known as Git’s memory bank

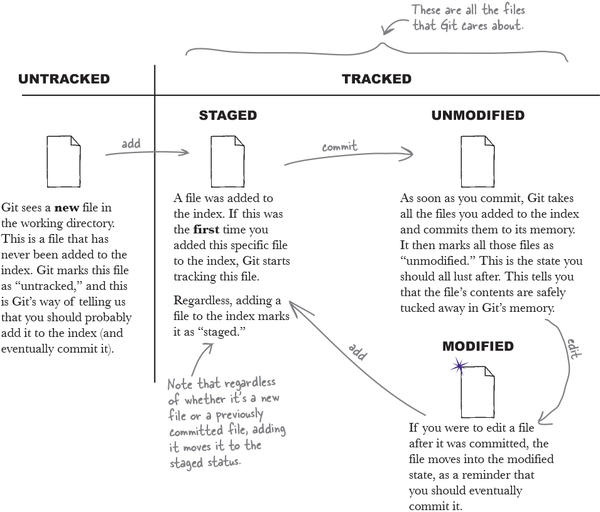
A pointer to the location inside the .git folder where Git has stored your changes, called a tree.



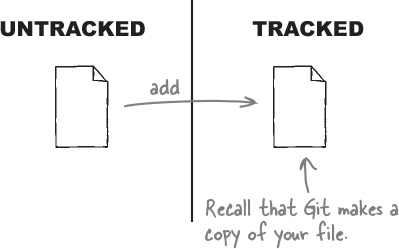


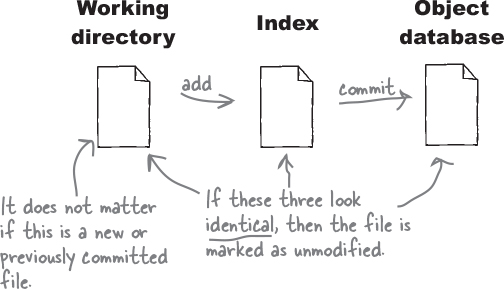
Q: What if I edited several files? Is there a way to add multiple files to the index?

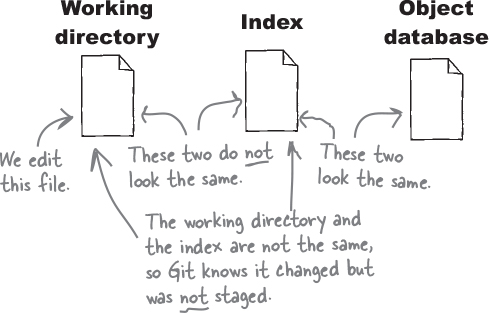
A: You can supply multiple filenames separated by whitespace to the git add command, like so: git add file1 file2



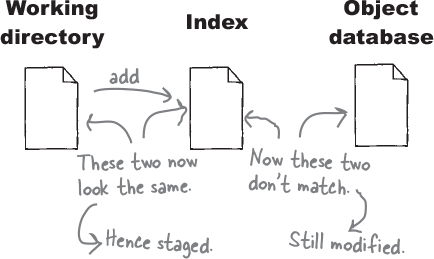
A file that Git has never seen before (that is, a file that has never been added to the index) is marked as “untracked.







And we complete the circle—if we commit, the contents of the index will be committed, and the file will be marked as “unmodified.”



Recall that any file in your working directory is either untracked or tracked. Also, a tracked file can be either staged, unmodified, or modified.

You create a new file in the repository called Hello.txt.

Image

You add Hello.txt to the index (using git add).

Image

You commit all the changes that you staged (using git commit).

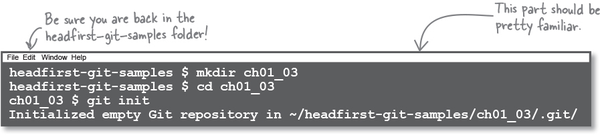
Image

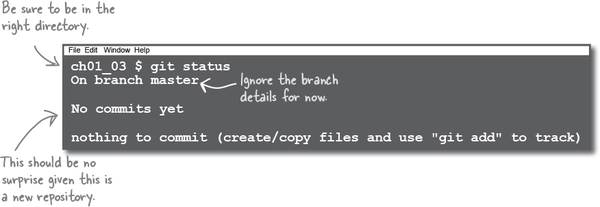
You edit Hello.txt with some new content.

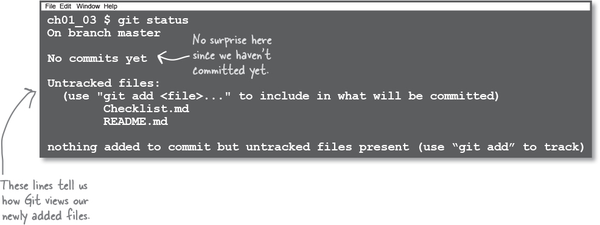
Image

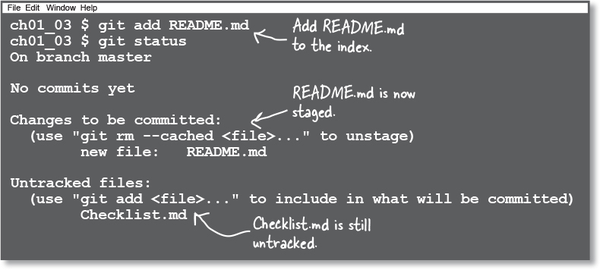
Now if you add the file again, Git overwrites the index with the latest changes reflected in that file. In other words, the index is a temporary scratch pad—one you can use to stuff edits into till you are sure you want to commit.

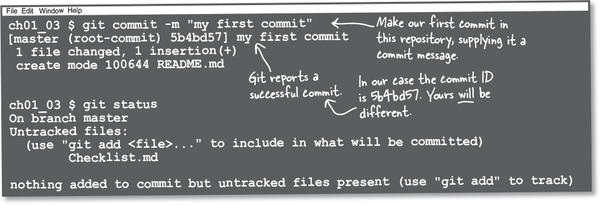
As you continue to work with Git, it’s often useful to check the status of the files in your working directory. One of the most useful commands in your Git arsenal is the git status command.

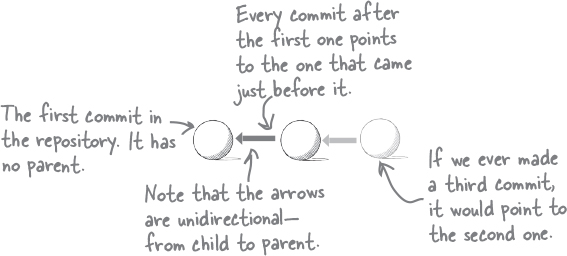












BULLET POINTS

A version control system like Git allows you to store snapshots of your work.

Git is much more than a tool that allows you to record snapshots. Git allows us to confidently collaborate with other team members.

Using Git effectively requires you to be comfortable with the command line.

The command line offers a slew of other capabilities, including creating and navigating directories and listing files.

Git is available as an executable, which you install, and it makes Git available to use in the command line with the name git.

Once you install Git, you need to tell Git your full name and your email address. Git will use this whenever you use Git to take a snapshot of your work.

If you want Git to manage the files for any project, we have to initialize a Git repository at the root level of the project.

To initialize Git you use the init command, like so: git init

The result of initializing a new Git repository is that Git will create a hidden folder called .git in the directory where you ran the git init command. This hidden folder is used by Git to store your snapshots, as well as some configuration for Git itself.

Any directory that is managed by Git is referred to as the working directory.

Git, by design, has an index, which acts as a “staging area.” To add files to the index, you use the git add <filename> command.

Committing in Git translates to taking a snapshop of the changes that were stored in the index. The command to create a commit is git commit, which requires that you supply it with a commit message to describe the changes you are commiting, using the -m (or --message) flag:

git commit -m “some message”

Every file in the working directory is assigned one or more states.

A brand new file added to the working directory is marked as “untracked,” which suggests that Git does not know about this file.

Adding a new file to Git’s index does two things—it marks the file as being “tracked” and creates a copy of that file into the index.

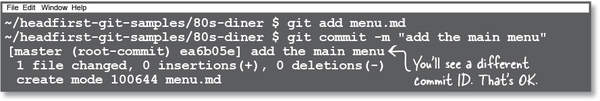
When you make a commit, Git creates a copy of the files in the index and stores them in the object database. It also creates a commit object that records metadata about the commit, including a pointer to the files that were just stored, the author name and email, and the time the commit was made, as well as the commit message.

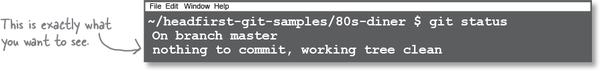
Every commit in Git is identified by a unique identifier, refererred to as the commit ID.

At any time you can ask Git for the status of the files in the working directory and the Git repository, using the git status command.

Every commit except the initial commit in Git stores the commit ID of the commit that appeared just before it, thus creating a string of commits, like leaves on a branch.

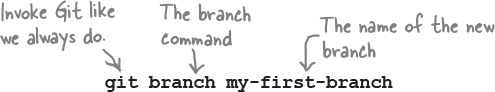
This string of commits is referred to as the commit history.



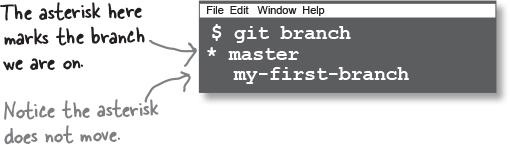


It turns out that when you initialize a new Git repository and make your first commit, you are already working with branches! Git by default uses a branch called master, which explains why git status reported that you were on that branch.

Let’s start by creating a new branch. You can use the branch command, giving it the name of the branch you wish to create as an argument.

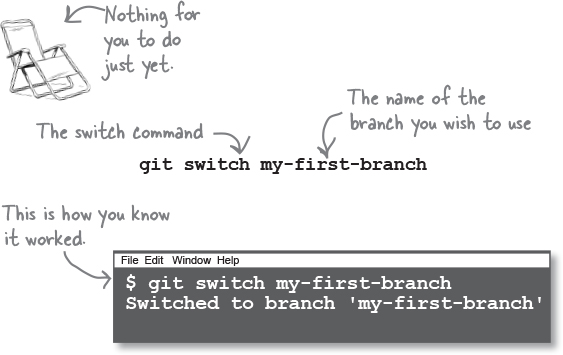


Git does not report success or failure, but you can list all your branches by using the same branch command, except with no arguments.



Switching tracks

you also just learned that creating a new branch does not mean you can start using it. To switch to another branch,



You can use git branch to list all the branches again:



Older versions of Git used the git checkout command to switch branches. While that still works, we prefer to show you the latest (and now correct) way to do things.

You can invoke the git switch command with the -c (or --create) flag, giving it the name of the branch you wish to create, like so:

git switch -c my-first-branch

This will prompt Git to create the branch called my-first-branch and switch to it immediately.

