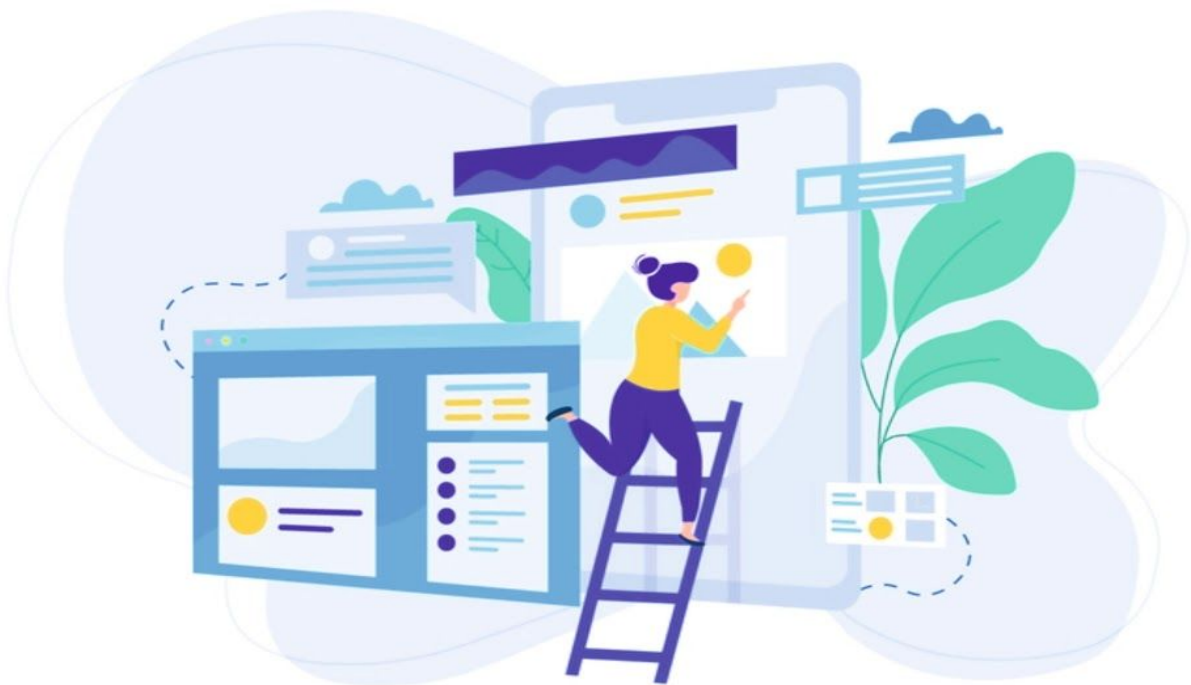


TEACHMEBRO.COM

GIT TUTORIAL FOR BEGINNERS



BY ALTAF SHAIKH



What is git?

Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Git is easy to learn and has a tiny footprint with lightning-fast performance. Git was initially designed and developed by Linus Torvalds for Linux kernel development. This tutorial explains how to use Git for project version control in a distributed environment.

So let's see what do we mean by version control system.

Version Control System

Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

Listed below are the functions of a VCS –

- It allows developers to work simultaneously.
- Maintains a history of every version.
- It does not allow overwriting each other's changes.

Following are the types of VCS –

- Centralized version control system (CVCS).
- Distributed/Decentralized version control system (DVCS).

Centralized Version Control

Centralized version control systems are based on the idea that there is a single “central” copy of your project somewhere (probably on a server), and programmers will “commit” their changes to this central copy. There are quite a lot of **tools** available to meet this need, including Perforce, Mercurial, SVN are examples of centralized version control.

Distributed Version Control System

A **centralized version control system (CVCS)** uses a central server to store all files and enables team collaboration. But the major drawback of CVCS is its single point of failure, i.e., failure of the central server. Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all. And even in a worst-case, if the disk of the central server gets corrupted and proper

backup has not been taken, then you will lose the entire history of the project. Here, a distributed version control system (DVCS) comes into the picture.

DVCS clients not only check out the latest snapshot of the directory but they also fully mirror the repository. If the server goes down, then the repository from any client can be copied back to the server to restore it. Every checkout is a full backup of the repository. **Git** does not rely on the central server and that is why you can perform many operations when you are offline. You can commit changes, create branches, view logs, and perform other operations when you are offline. You require a network connection only to publish your changes and take the latest changes.

Advantages:

- **Distributed model:** This means your work is your own. You can let others see only what is necessary. Not everything has to be public. There are other advantages to the distributed model, such as the speed (since most everything is local) and possibility of working offline
- **Branching and merging are easy:** Branching is a walk in the park. It feels like a natural part of the workflow. They are cheap (fast and consume very little space) so that you can branch whenever you want. This means you can sandbox your features and ideas till they are ready for the mainstream.
- **Workflow is flexible:** Compared to Centralized VCS, git has the qualities that allow you to choose your own workflow.
- **Data integrity is assured:** Because git uses SHA1 trees, data corruption due to external reasons can be easily detected.
- **Security:** Git uses a common cryptographic hash function called secure hash function (SHA1), to name and identify objects within its database. Every file and commit is check-summed and retrieved by its checksum at the time of checkout. It implies that it is impossible to change file, date, and commit message and any other data from the Git database without knowing Git.
- **Icing on the cake:**
 - **Fast:** Git is very fast, even when compared to other DVCS, for local as well as network operations
 - **Staging area:** Make sure your commits have logically grouped changes and not everything else you are working on.

- **Free:** I am sure you don't want to spend 450\$ for your personal project. Your manager will appreciate it if you save him N x 450\$

Disadvantages:

- **Steep learning curve:** Many commands with many options, some commands are non-intuitive and need a level of understanding the internals of git, commands and arguments are inconsistent to some degree
- **Binary files are a big no:** If your project has non-text files that are updated frequently (images for websites or MS Office documents), then git becomes bloated and slow. (I believe it still does better than most systems)

That's it!! In the next tutorial we will see how to install and configure git on your system.

Getting Started - Installing Git

In the previous tutorial we have learned and understand git. In this tutorial we will see how to install git on Linux, Mac OS and Windows operating system.

Installing Git

Before you start using Git, you have to make it available on your computer. Even if it's already installed, it's probably a good idea to update to the latest version. You can either install it as a package or via another installer, or download the source code and compile it yourself.

Installing on Linux

Git is already installed in all Linux flavours but if your system doesn't have git installed then follow along the tutorial. If you want to install the basic Git tools on Linux via a binary installer, you can generally do so through the package management tool that comes with your distribution. If you're on Fedora (or any closely-related RPM-based distribution, such as RHEL or CentOS), you can use `dnf`:

```
$ sudo dnf install git-all
```

If you're on a **Debian-based** distribution, such as Ubuntu, kali linux try `apt`:

```
$ sudo apt install git-all
```

For more options, there are instructions for installing on several different Unix distributions on the Git website, at <https://git-scm.com/download/linux>.

Installing on macOS

There are several ways to install Git on a Mac. The easiest is probably to install the Xcode Command Line Tools. On Mavericks (10.9) or above you can do this simply by trying to run **git** from the Terminal the very first time.

```
$ git --version
```

If you don't have it installed already, it will prompt you to install it.

If you want a more up to date version, you can also install it via a binary installer. A **macOS** Git installer is maintained and available for download at the Git website, at <https://git-scm.com/download/mac>.

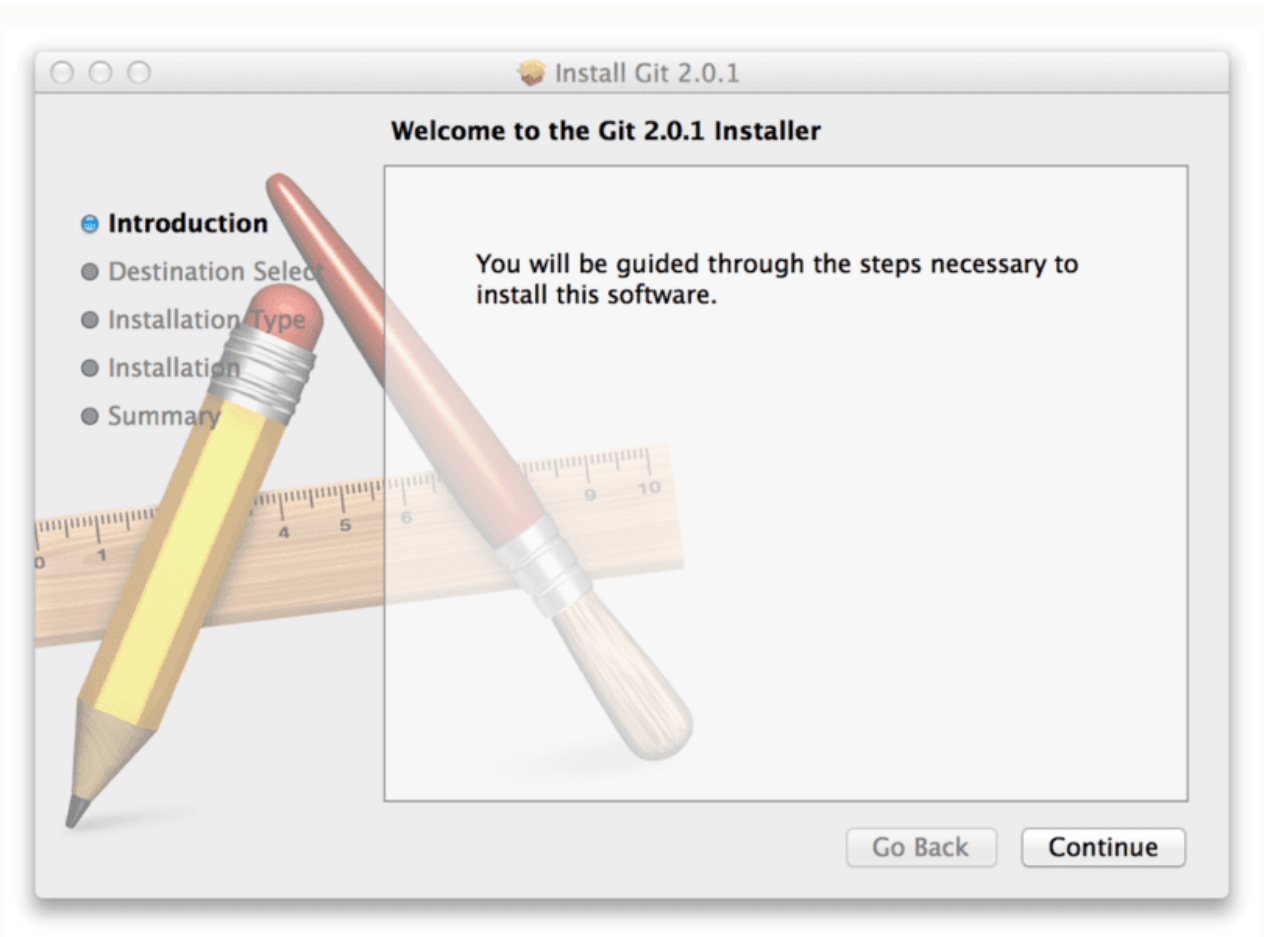
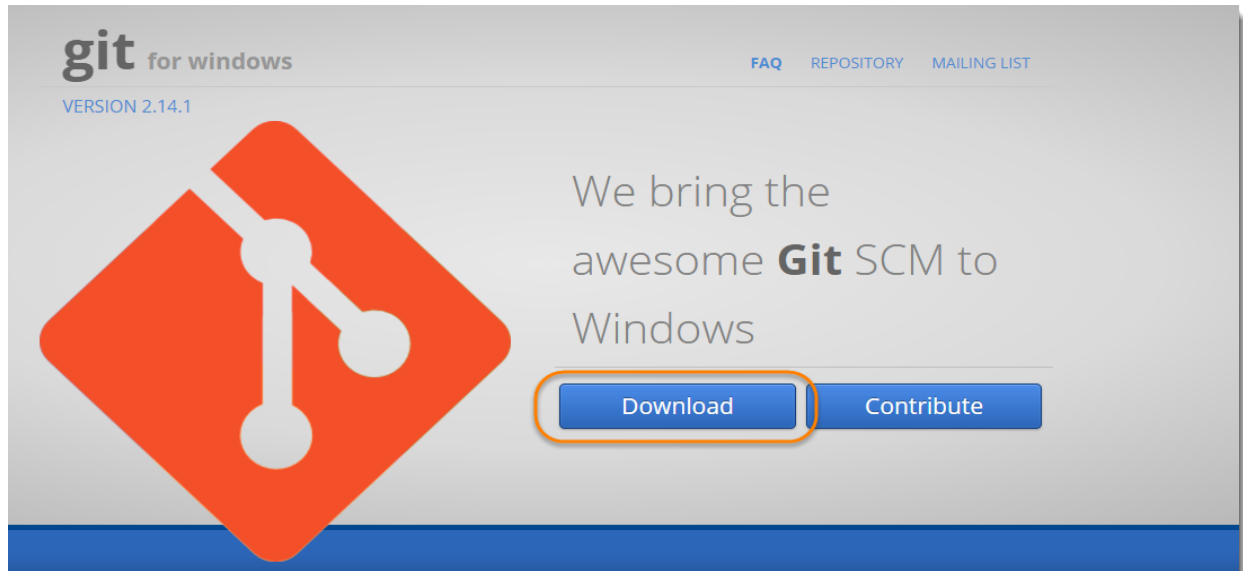


Figure 7. Git macOS Installer.

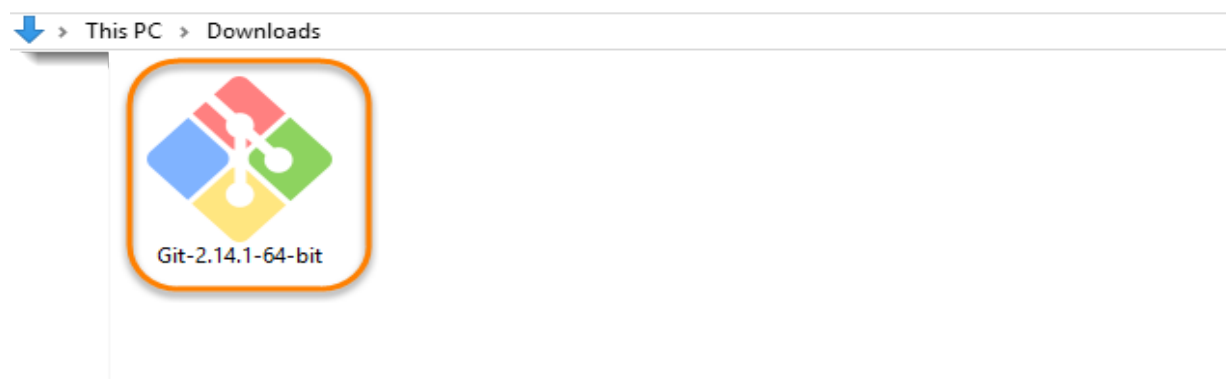
You can also install it as part of the GitHub for macOS install. Their GUI Git tool has an option to install command line tools as well. You can download that tool from the GitHub for macOS website, at <https://desktop.github.com>.

Steps to Install Git on Windows

1) Download the latest [Git for Windows](#).



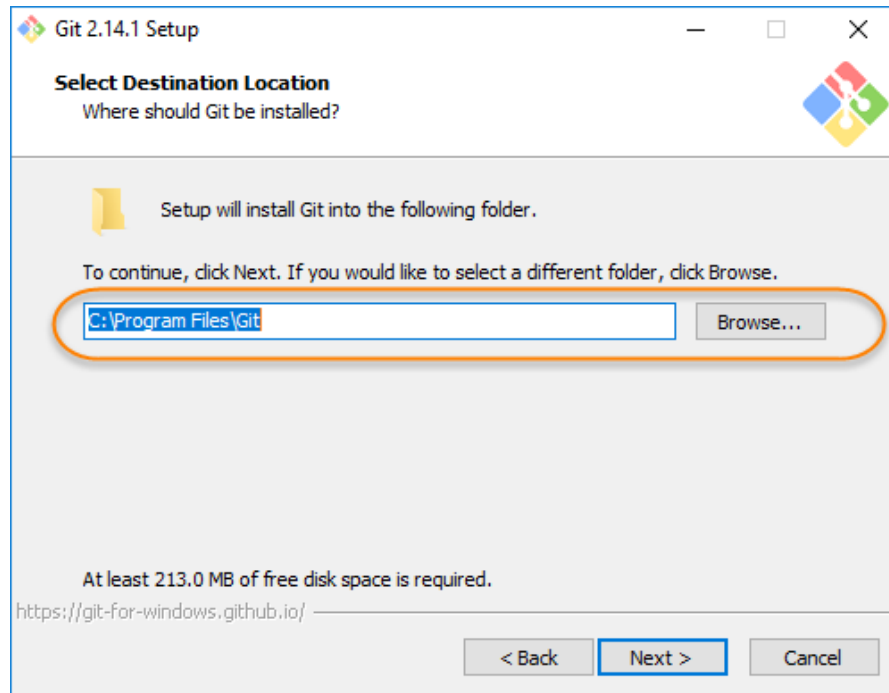
2) Go to the folder where new downloads gets store, at my machine by default folder is **Download** folder. **Double click** on the installer. The installer gets save on the machine as per the Windows OS configuration. My machine is **64 bits**.



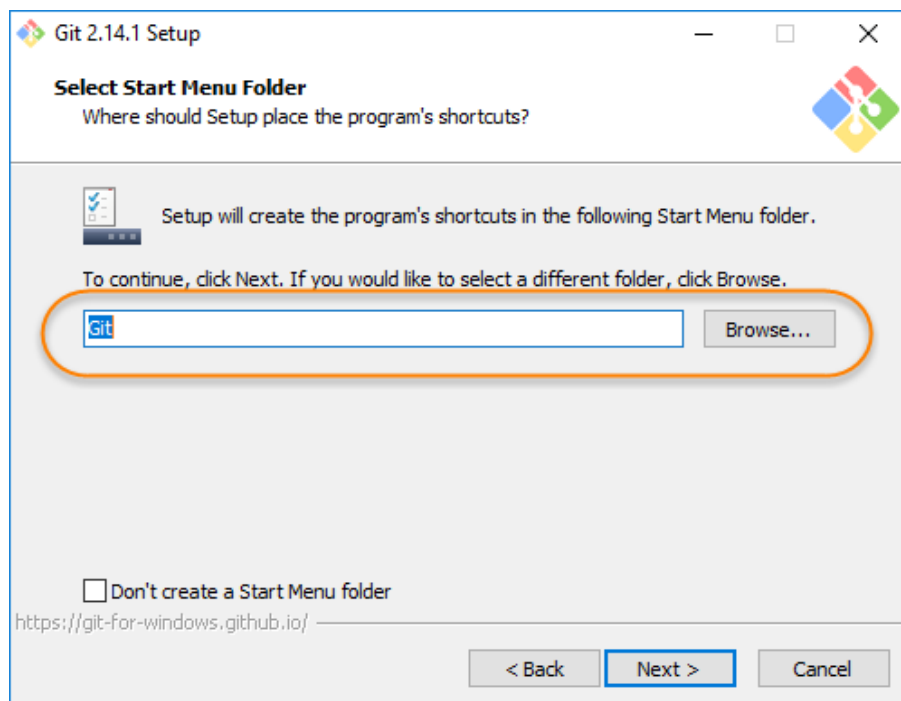
Note:

When you've successfully started the installer, you should see the Git Setup wizard screen. Follow the Next and Finish prompts to complete the installation. The default options are pretty sensible for most users.

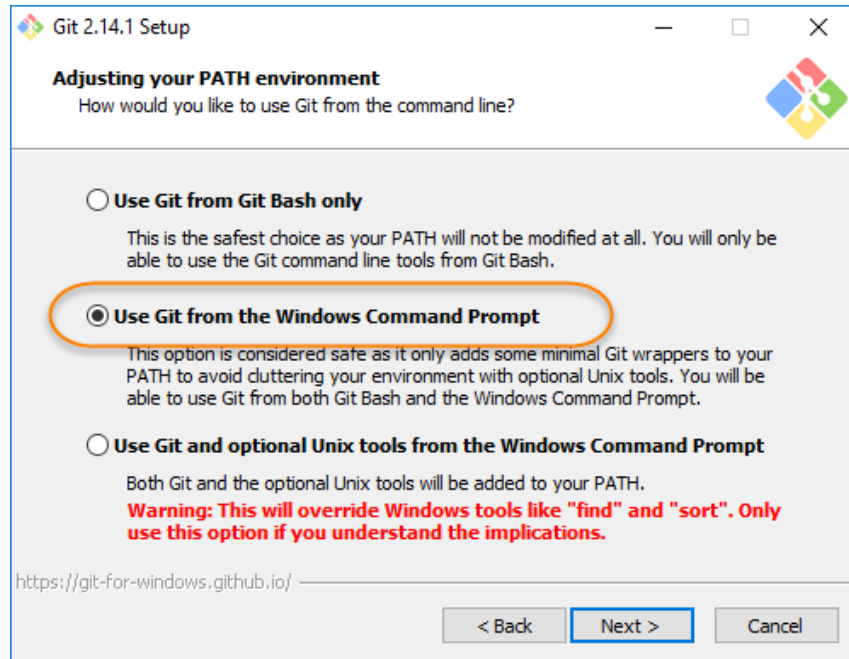
3) You may like to keep the installation to another folder, so here is the chance to do so. I just want to keep it in the suggested default folder in my **Program Files\Git**.



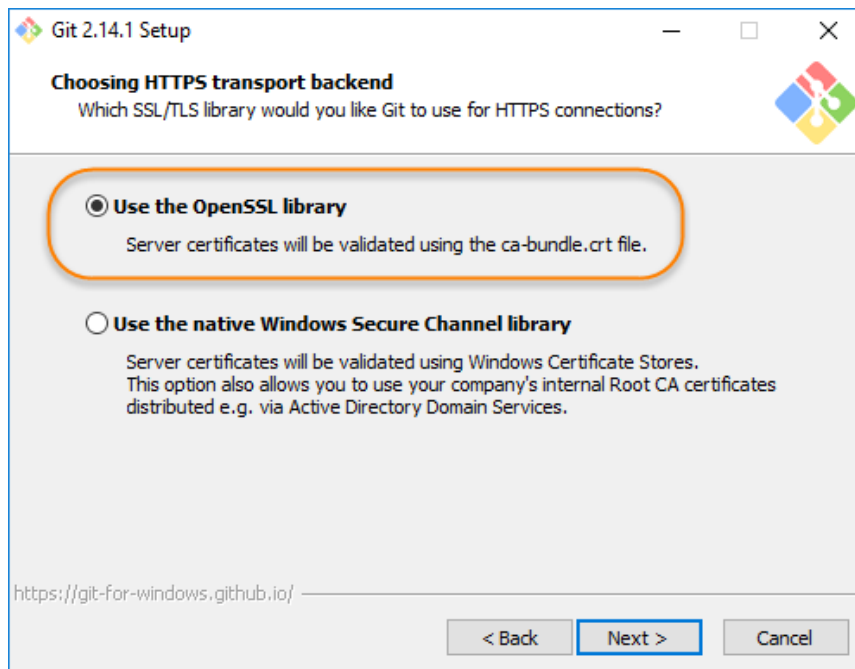
4) This is the option to store the *shortcut of the Git* under the **Program Menu**.



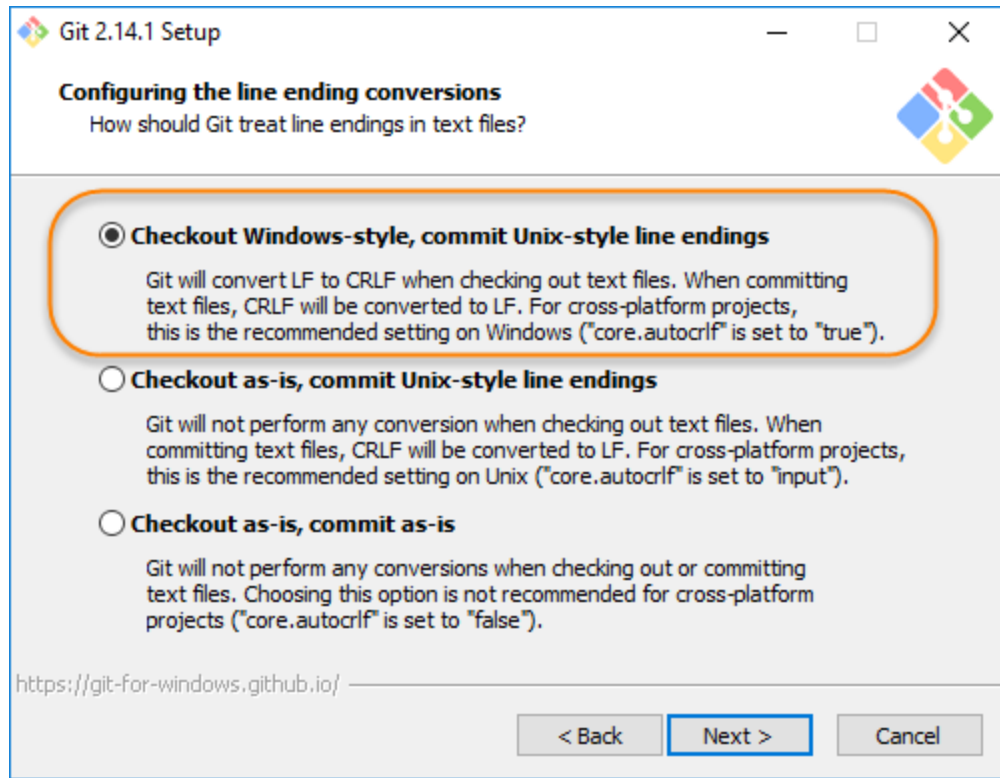
5) This is asking your choice, whether you like to Git from the *Windows Command Prompt* or you like to use some other program like *Git Bash*. As of now just select the *Windows Cmd* for simplicity of the tutorial, later we will cover *Git Bash* and other as well.



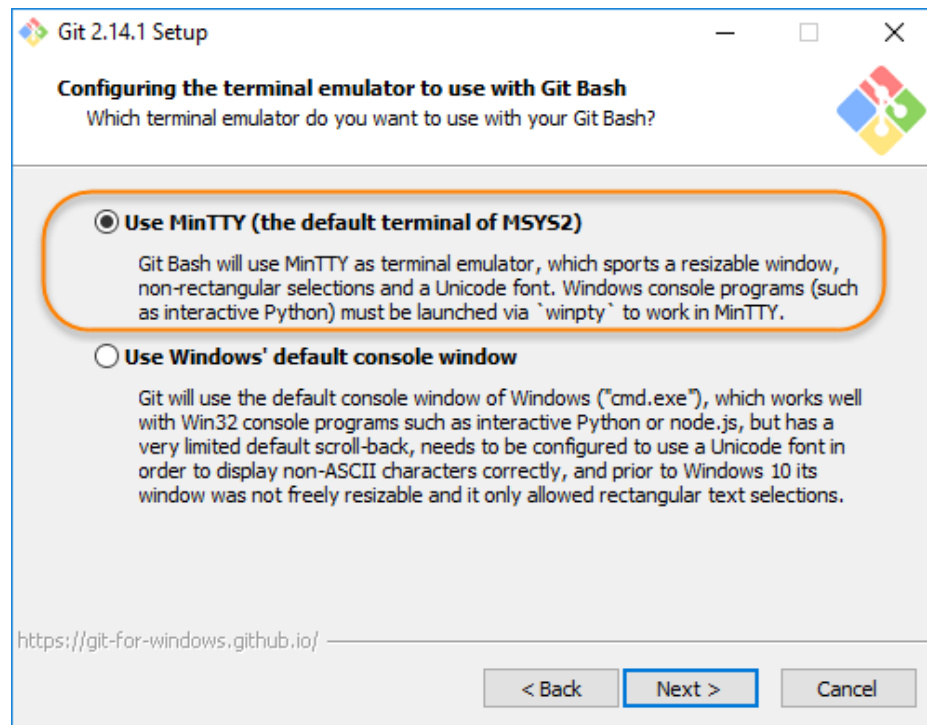
6) If you have *PuTTY/TortoiseSVN* installed, you may see this screen, otherwise just ignore this. Regardless, use *OpenSSL* to make things easy.



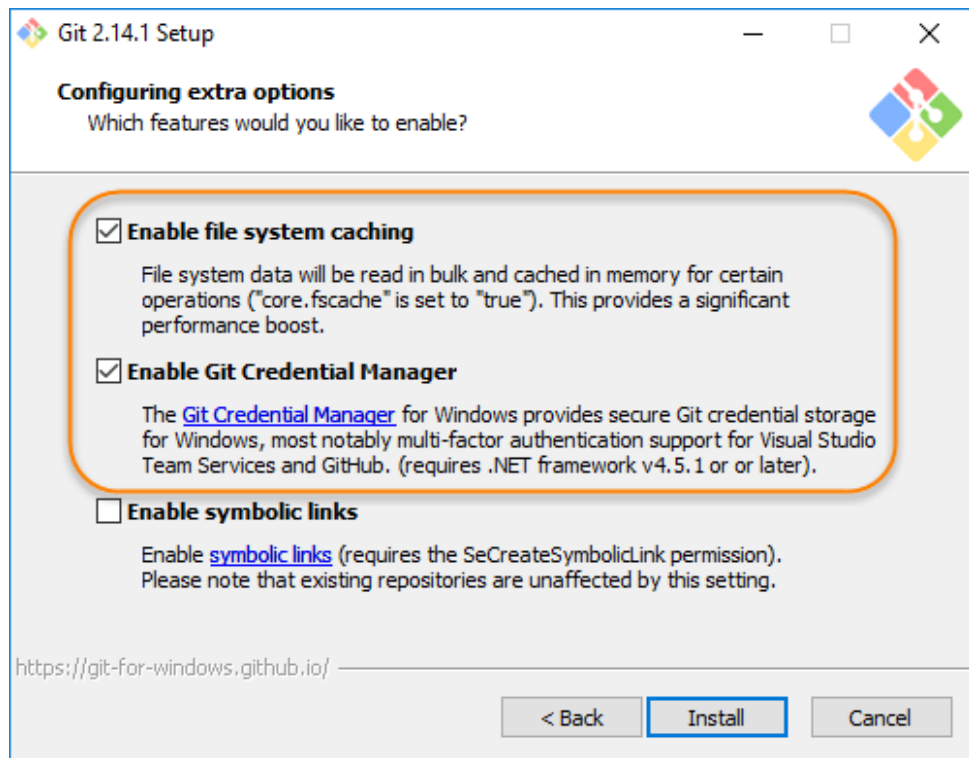
7) Here, we recommend to choose the option of *Checkout Windows-style, commit Unix-style line endings*. Select next once you have done this.



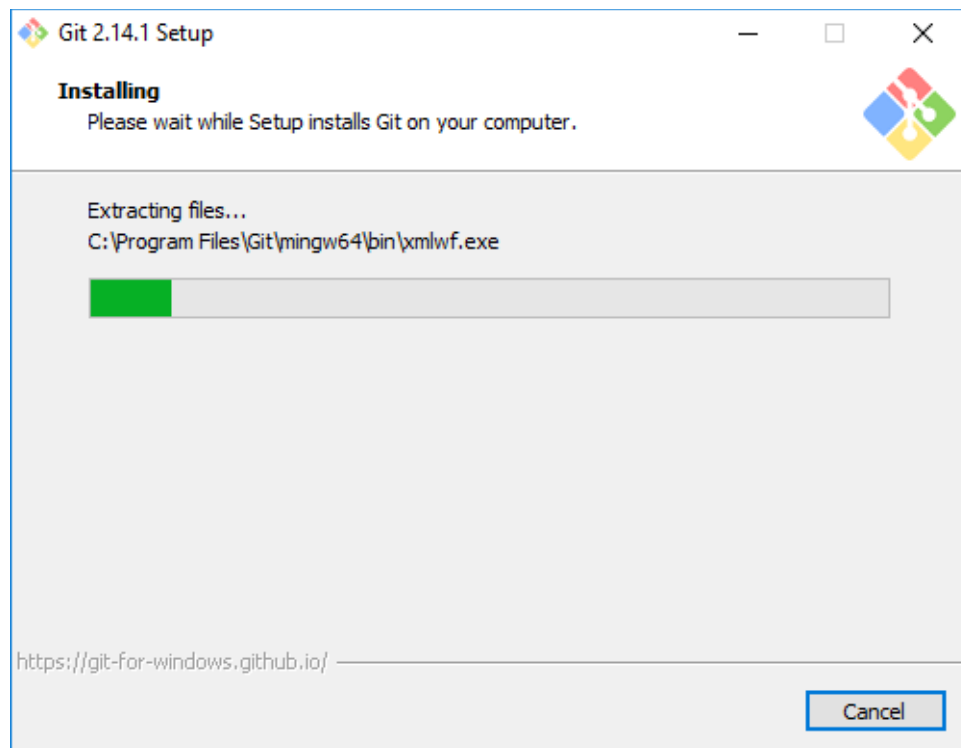
8) Again, just go with default selection and move forward.



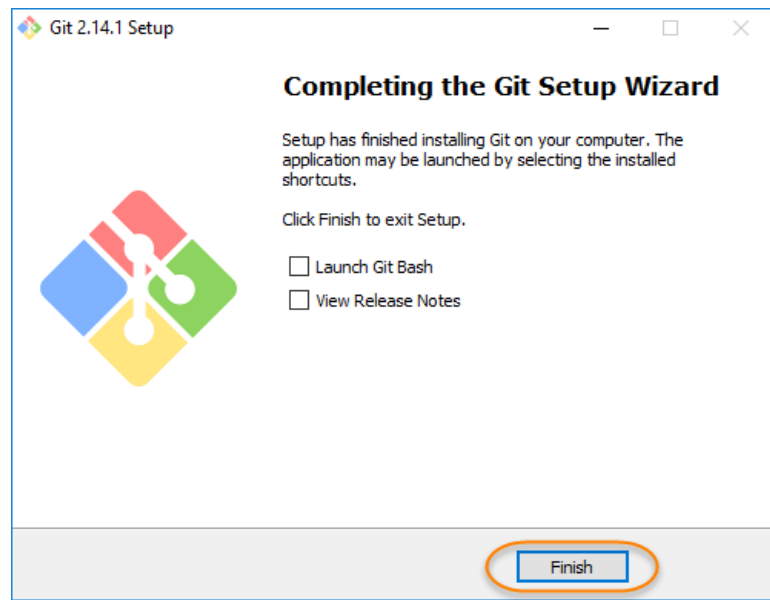
9) Just go with default selections, as we will cover the details in later advance chapter.



10) Now, it's all done. This will just take a few minutes to complete the installation as per your machine speed.



11) Once done, just click on Finish button



12) Let's just verify if the installation went well for Git. Go to *cmd* and type *git* and press *enter*. you should get the following output on the screen.

```
Command Prompt
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>git
usage: git [--version] [--help] [-C <path>] [-c name=value]
       [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
       [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
       [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
       <command> [<args>]

These are common Git commands used in various situations:


start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one


work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index


examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status


grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG


collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.

C:\Users\Lenovo>
```

Configure GitHub Credentials with Git:

Configure your local Git installation to use your GitHub credentials by entering the following:

```
git config --global user.name "github_username"
```

```
git config --global user.email "email_address"
```

Note: Replace github_username and email_address with your GitHub credentials.

Example:

```
git config --global user.name "altaf99"
```

```
git config --global user.email "iamaltafshaikh07@gmail.com"
```

Checking Your Settings

If you want to check your configuration settings, you can use the **git config --list** command to list all the settings Git can find at that point:

```
$ git config --list
```

Output:

```
user.email=iamaltafshaikh07@gmail.com
```

```
user.name=altaf99
```

Now we are ready to use the git.

Basic Git Commands:

1) git init

Utility: To initialize a git repository for a new or existing project.

How to: *git init* in the root of your project directory.

2) git clone

Utility: To copy a git repository from a remote source, it also sets the remote to original source so that you can pull again.

How to: *git clone <:git url:>*

Example:

git clone <https://github.com/altaf99/Friends-The-Social-Network.git>

3) git status

Utility: To check the status of files you've changed in your working directory, i.e, what all has changed since your last commit.

How to: *git status* in your working directory. lists out all the files that have been changed.

4) git add

Utility: adds changes to stage/index in your working directory.

How to: *git add* .

5) git commit

Utility: To save changes to the local repository safely. Commit takes a snapshot of the project and treats it as a version.

How to: *git commit -m "sweet little commit message"*

6) git push

Utility: It is used to transfer or push the commit, which is made on a local branch in your computer to a remote repository

How to: *git push <:remote:> <:branch:>*

Example: *git push origin master*

7) git pull

Utility: It is used to fetch and download content from a remote repository and immediately update the local repository to match that content.

How to: *git pull <:remote:> <:branch:>*

Example: *git pull origin master*

8) git --version

Utility: To check the version if the git is installed

How to: *git --version*

9) git remote

Utility: To check what remote/source you have or add a new remote.

How to: *git remote* to check and list. And *git remote add <:remote_url:>* to add a remote.

Example:

git remote add <https://github.com/altaf99/Friends-The-Social-Network.git>

So after learning the basic commands of git let's see Git in action. If you have a project directory to track your files then you can skip this section and if not then continue.

Initializing a Repository from Non-existing Directory:

So if you don't have a project folder to track with git then we will create one from scratch. Follow the below steps:

1. Create a folder with a name '**test**' on Desktop or you can create it anywhere on your system.
2. Then open your terminal and change your directory to '**test**' folder in my case it will be `cd /home/altaf/Desktop/Tutorial/Git/test` in your case `cd 'path of your test directory'`
3. Now type `git init` to initialise a git repository.

```
altaf@kali:~/Desktop/Tutorial/Git/test$ git init
```

```
Initialized empty Git repository in /home/altaf/Desktop/Tutorial/Git/test/.git/
```

This creates a new subdirectory named `.git` that contains all of your necessary repository files — a Git repository skeleton.

4. Now create two files in the test directory named them **test1.txt** and **test2.txt**, you can use the below command to create them directly from the terminal.

```
$touch test1.txt test2.txt
```

And also add some random data inside both the files.

So now we have a project folder with some files. Now move ahead to start working in the project folder with git and now you can skip the next section, you have already created a project folder.

Initializing a Repository in an Existing Directory

If you have a project directory that is currently not under version control and you want to start controlling it with Git, you first need to go to that project's directory. If you've never done this, it looks a little different depending on which system you're running:

for Linux:

```
$ cd /home/user/my_project
```

for macOS:

```
$ cd /Users/user/my_project
```

for Windows:

```
$ cd C:/Users/user/my_project
```

and type:

```
$ git init
```

Example:

```
altaf@kali:~/Desktop/Tutorial/Git/test$ git init
```

Output:

Initialized empty Git repository in /home/altaf/Desktop/Tutorial/Git/test/.git/

This creates a new subdirectory named `.git` that contains all of your necessary repository files — a Git repository skeleton.

So now we have a project folder which is now under version control and now we can start controlling it with Git.

Git in Action:

Now let's start controlling our project with git,

Run the **git status** command to check the status of your local repository,

```
altaf@kali:~/Desktop/Tutorial/Git/test$ git status
```

Output:

```
On branch master
```

```
No commits yet
```

```
Untracked files:
```

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

```
test1.txt
```

```
test2.txt
```

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

It says you have untracked files **test1.txt** and **test2.txt**, use **git add .** command to track all the files present in the directory or use can use `git add test1.txt` and `git add test2.txt` to track a file using the filename.

Now run the following commands:

```
altaf@kali:~/Desktop/Tutorial/Git/test$ git add .
```

```
altaf@kali:~/Desktop/Tutorial/Git/test$ git status
```

```
On branch master
```

```
No commits yet
```

```
Changes to be committed:
```

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

```
new file: test1.txt
```

```
new file: test2.txt
```

Now we have added the files into the staging area (where we save the file so that git is able to track any changes made to those files) and This stages them for the first commit.
we are ready to commit the changes.

Now run the following commands:

```
$ git commit -m "First commit"
```

```
[master (root-commit) 8db38b6] First commit
```

```
2 files changed, 0 insertions(+), 0 deletions(-)
```

```
create mode 100644 test1.txt
```

```
create mode 100644 test2.txt
```

This will Commit the files that you've staged in our local repository.

So before pushing our code into the Remote repository i.e on Github (our online repository) let's check one more command.

Git Log:

The Git Log tool allows you to view information about previous commits that have occurred in a project.

```
altaf@kali:~/Desktop/Tutorial/Git/test$ git log
```

```
commit 8db38b60919a67b9f912d1e9e782a5ccb7588eb4 (HEAD -> master)
```

```
Author: altaf99 <iamaltafshaikh07@gmail.com>
```

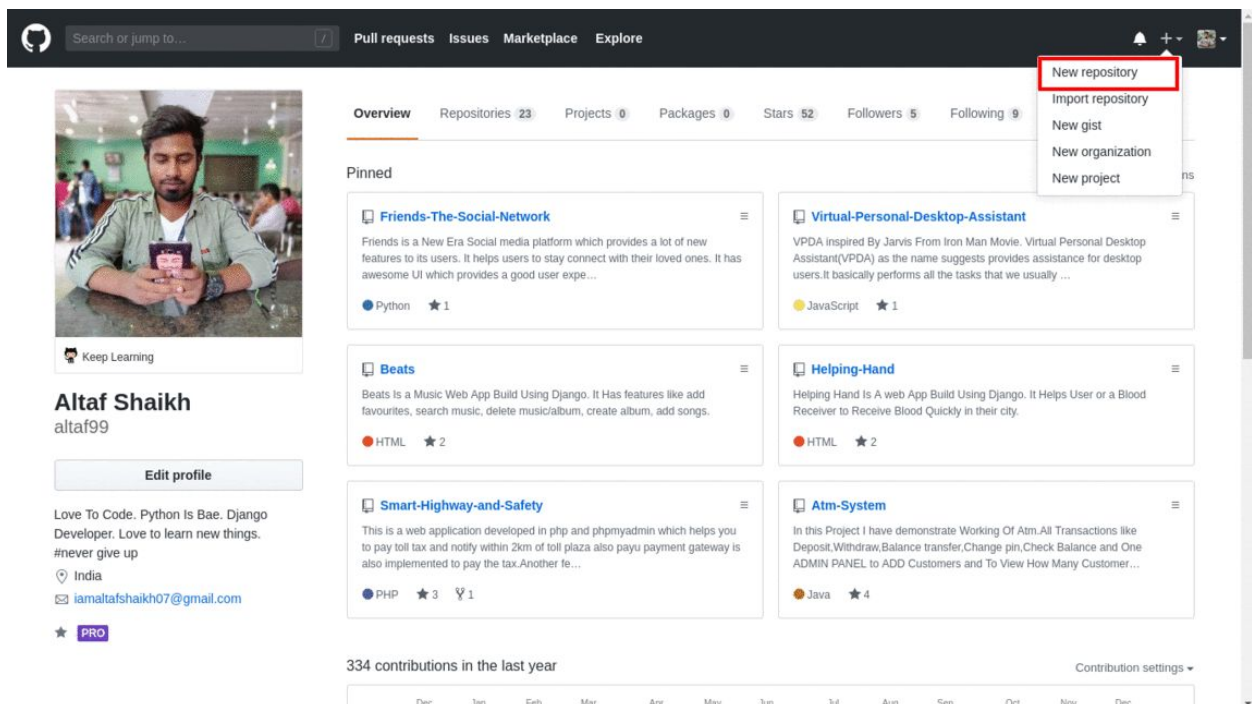
```
Date: Wed Apr 1 22:37:02 2020 +0530
```

```
First commit
```

Pushing our code on Remote Repository (Github):

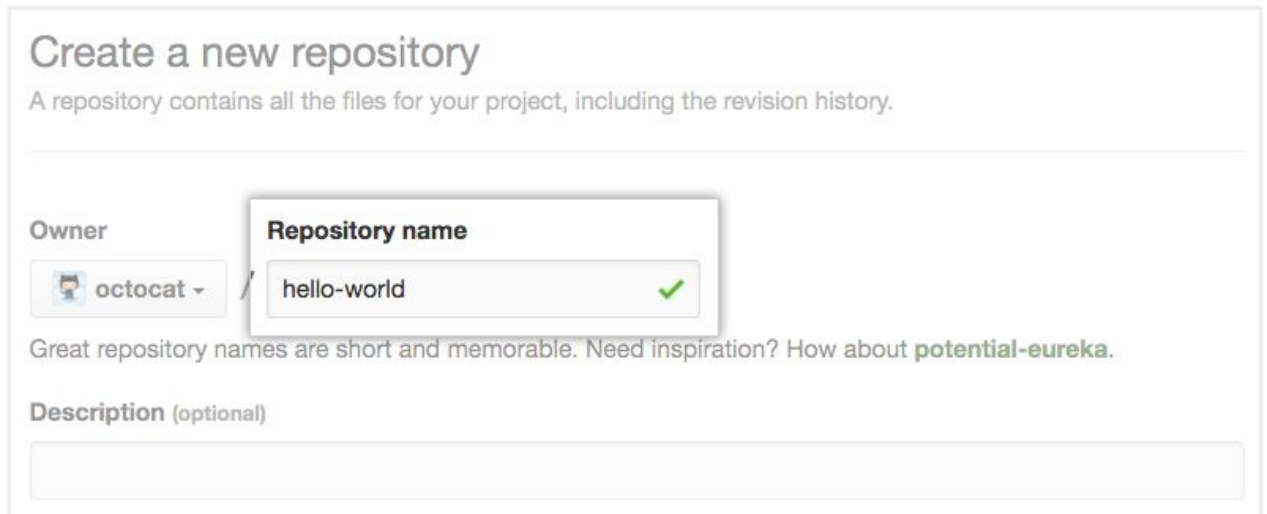
Before pushing our code we need a repository to share/upload our code, so let's create one.

1. Go to <https://github.com/> and login into your account.
2. In the upper-right corner of any page, click on + , and then click New repository.



The screenshot shows a GitHub profile page for a user named 'Altaf Shaikh' (username: altaf99). The profile includes a bio, location (India), email, and a 'PRO' badge. The 'Pinned' section displays six repositories: 'Friends-The-Social-Network' (Python, 1 star), 'Virtual-Personal-Desktop-Assistant' (JavaScript, 1 star), 'Beats' (HTML, 2 stars), 'Helping-Hand' (HTML, 2 stars), 'Smart-Highway-and-Safety' (PHP, 3 stars, 1 fork), and 'Atm-System' (Java, 4 stars). The 'New repository' dropdown menu is open in the top right corner, showing options: 'New repository', 'Import repository', 'New gist', 'New organization', and 'New project'. The 'New repository' option is highlighted with a red box. At the bottom, a bar chart shows '334 contributions in the last year'.

3. Type **“hello-world”** as a name for your repository, and an optional description.



Create a new repository
A repository contains all the files for your project, including the revision history.

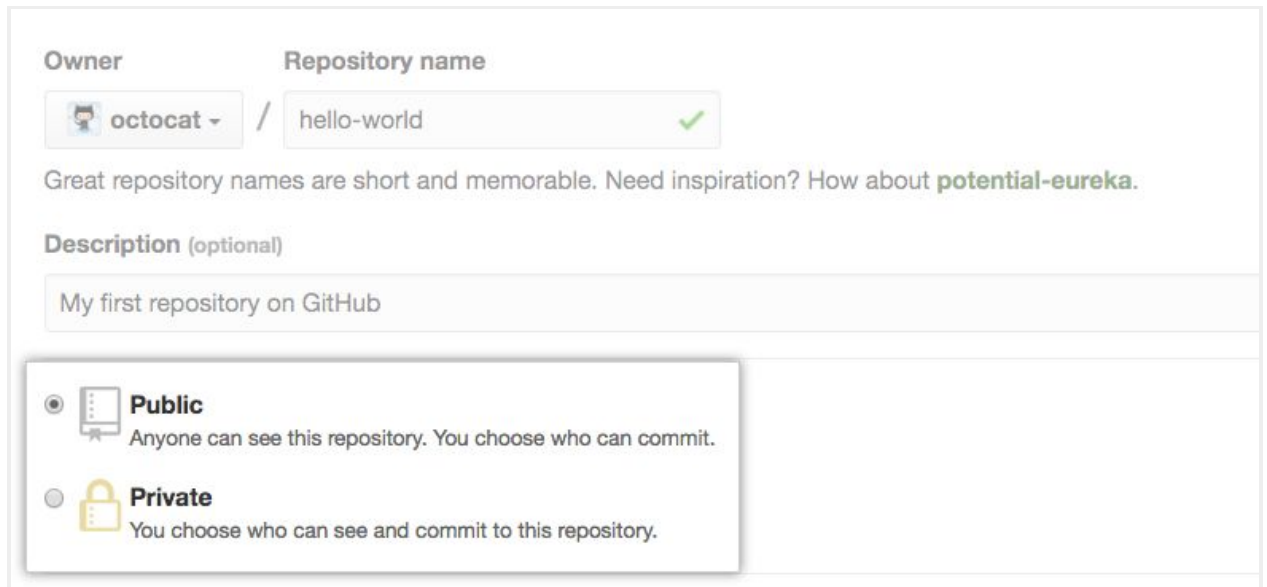
Owner
octocat

Repository name
hello-world ✓

Great repository names are short and memorable. Need inspiration? How about **potential-eureka**.

Description (optional)

4. Choose to make the repository either public or private. Public repositories are visible to the public, while private repositories are only accessible to you, and people you share them with.



Owner
octocat

Repository name
hello-world ✓

Great repository names are short and memorable. Need inspiration? How about **potential-eureka**.

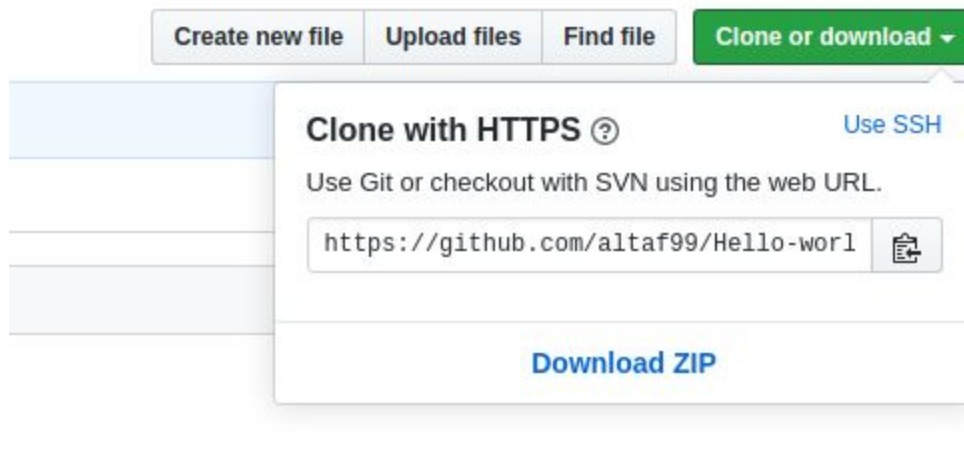
Description (optional)
My first repository on GitHub

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

5. When you're finished, click Create repository.

6. At the top of your GitHub repository's Quick Setup page, click to copy the remote repository URL.



Now to connect Remote repository “hellow-world” with our local repository will use git remote add origin command which we have previously learned.

In Terminal, add the URL for the remote repository where your local repository will be pushed.

```
$ git remote add origin https://github.com/altaf99/Hello-world.git
```

Now Verifies the new remote URL

```
$ git remote -v
```

```
altaf@kali:~/Desktop/Tutorial/Git/test$ git remote -v
```

```
origin https://github.com/altaf99/Hello-world.git (fetch)
```

```
origin https://github.com/altaf99/Hello-world.git (push)
```

Now let's Push the changes in your local repository to GitHub.

```
$ git push origin master
```

Pushes the changes in your local repository up to the remote repository you specified.

Kudos!! We have Pushes the changes in your local repository up to the remote repository we have specified.

Congratulations!! We have successfully pushed the code into our remote repository and also we have learned and understand the basics of Git and we are also successfully able to implement the Git commands. Hope you enjoyed the tutorial, please share this tutorial with your friends if you find this tutorial helpful.

Thank You!!

We will see the advanced concept of Git in our upcoming tutorials.

For more such tutorials visit www.teachmebro.com