How to get started with Git on Linux

DEK: If you've begun your journey as a developer, chances are pretty good you'll need to know how to use git. Here's an introductory guide that walks you through the basics on Linux.

Git is the most widely-used version control system on the market. It's huge and used by developers around the globe. Of course, git's not just used as a version control system, but also a means to clone repositories to your desktop or server, so you can then install whatever application is found in that repository.

Although you might not need to know about git during your first steps with Linux (or as a developer), at some point you will. Let me walk you through those first steps.

Requirements

I'm going to demonstrate git on the Ubuntu-based Pop!\_OS Linux. You don't have to be using that particular distribution but if you're using a version of Linux that's not based on either Ubuntu or Debian, you'll need to alter the installation command from using apt to the package manager of choice for your distribution (such as dnf for RHEL-based distributions). You'll also need a user with sudo privileges.

That's it. Let's take our first steps with git.

Installing git

1: Open the terminal window

Click to open your desktop menu and search for the terminal application. Once you've found it, launch the applications.

2: Install git

From the terminal window, install git with the command:

sudo apt install git -y

Create a local repository

1: Create a folder

With git installed, we now have to create a local repository. To do that, we must create a folder to house the repository. For this, issue the following command:

mkdir ~/myproject

2: Initialize the repository

Change into the new directory with the command cd ~/myproject. To initialize the repository, run the command:

git init

That's all there is to initializing your first repository.

Adding files to your repository

1: Create a README file

From the project directory, create a README.txt file with the command:

nano README.txt

In that file, you can add any information you want to give other developers or users the necessary information you want to impart. Once you've finished, save the file with the [Ctrl]+[X] keyboard shortcut.

2: Make git aware of the file

After adding the file to the repository, git will automatically notice the file but can't do anything with it. In order to allow git to use the file, you must add what is called a commit. A commit is an operation that sends the latest changes made to the source code to the repository. In other words, if you don't make a commit, git will not be aware of any changes. In our case, git won't be aware that the newly added file contains any information.

To do this, we'll use the git add command like so:

git add README.txt

If you've created more than one file in the repository, you can add them all at once with the command:

git add .

3: Check the status

You can check the status of the repository with the command:

git status

The output to the above command will look something like this:

Changes to be committed:

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

modified: README.txt

We now have changes that must be committed.

4: Creating a commit

The next step is to create a commit for our newly added README.txt file. When you create a commit, you add information to it so that anyone else working on the project knows what has been done. For example, we just added the README.txt file, so we'll want to create a commit to indicate that very thing. A git commit looks something like this:

git commit -m "Added README.txt"

The output of the above command will look something like this:

[master f2dd2d8] Added README.txt

1 file changed, 2 insertions(+), 1 deletion(-)

If you issue the git status command, you should see output similar to this:

On branch master

Your branch is ahead of 'origin/master' by 1 commit.

(use "git push" to publish your local commits)

nothing to commit, working tree clean

5: Create a branch

Pushing your commits

This next step in the process requires that you have a GitHub account. You will also have to have created an access token, which is done in GitHub Settings > Developer Settings > Personal Access Tokens. Once you've created a Personal Access Token, make sure to copy it, as you cannot view it again without regenerating it.

To finally make those changes available, we push them with the command:

git push

You'll be asked for your GitHub username and password (which is the Personal Access Token you just created). Once you successfully authenticate to your GitHub account, you should see something like this in the output:

Enumerating objects: 10, done.

Counting objects: 100% (10/10), done.

Delta compression using up to 16 threads

Compressing objects: 100% (6/6), done.

Writing objects: 100% (6/6), 770 bytes | 770.00 KiB/s, done.

Total 6 (delta 1), reused 0 (delta 0), pack-reused 0

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To https://github.com/USERNAME/newproject.git

6fab6c3..c024f0d master -> master

Where USERNAME is your GitHub username.

Also, if you log into your GitHub account, you'll see that a new repository has been created that includes all of the files in your local repository. At this point you can add new files or edit existing files from GitHub in your web browser. If you do that, you can then pull down those changes to your local repository, with the command:

git pull

Any file that you added to the GitHub repository will be pulled down and available to your local repository for editing.

And that's the gist of the basic git workflow. Although it might seem a bit cumbersome at first, once you get the hang of it, you'll find it to be considerably easier than you think.