**Devops Tools**

**Git (Version Control System)**

**Version Control System**

Version control systems, often shortened to VCS, are specialized software whose primary goal is to manage changes to codebases over time, a process called version control. The main aim of such a system is to be able to recall specific versions later.

While it is possible to version control any file, the most versatile file types to be version controlled are plain text files such as software source code and documentation.

Some of the most popular version control systems are [Git](https://git-scm.com/), [Mercurial](https://www.mercurial-scm.org/), and [Subversion](https://subversion.apache.org/).

**Advantages of using Version Control System**

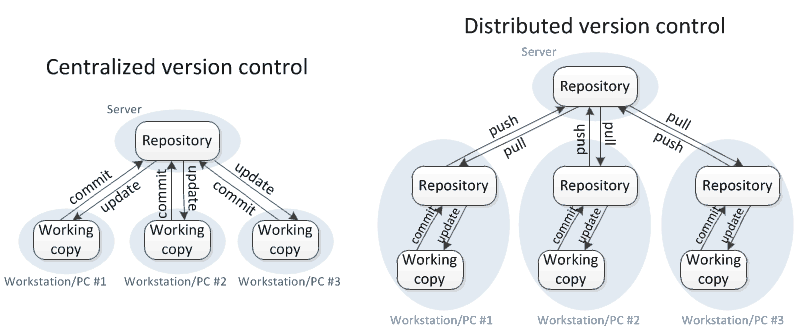
* In teams, using a version control system allows many contributors to work on the same codebase simultaneously without overwriting or conflicting with each others’ work.
* Version control systems also enable developers to quickly revert the software to the last known good state in case of a failure or a bug.
* By storing the entire history of every change across every file, VCS makes it easy to identify the change that introduced bugs and the developer who made the change.
* Distributed version control systems, like Git, also make it possible to restore the entire code along with the history from any developer’s local copy in case the central repository is lost.

**Types of Version Control systems**

There are two general varieties of version control: *centralized* and *distributed*. Distributed version control is more modern, runs faster, is less prone to errors, has more features, and is somewhat more complex to understand. You will need to decide whether the extra complexity is worthwhile for you.

Some popular version control systems are Git (distributed), Mercurial (distributed), and Subversion (centralized).

The main difference between centralized and distributed version control is the number of repositories. In centralized version control, there is just one repository, and in distributed version control, there are multiple repositories. Here are pictures of the typical arrangements:



In **centralized version control**, each user gets his or her own working copy, but there is just one central repository. As soon as you commit, it is possible for your co-workers to update and to see your changes. For others to see your changes, 2 things must happen:

* You commit
* They update

In **distributed version control**, each user gets his or her own repository *and* working copy. After you commit, others have no access to your changes until you push your changes to the central repository. When you update, you do not get others' changes unless you have first pulled those changes into your repository. For others to see your changes, 4 things must happen:

* You commit
* You push
* They pull
* They update

**What is a Repository?**

Ans : A repository is a database of all the edits to, and/or historical versions (snapshots) of, your project.

There are two types of repositories.

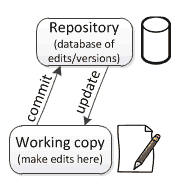
1. Remote repository
2. Local repository

Remote repository : The code base available remotely(In another server), to access it we need to connect to that machine and get the code(Clone).

Local repository : Its available in our system, we can say each and every developer machine will have this. The only way to communicate with Remote repository is through Local repository.

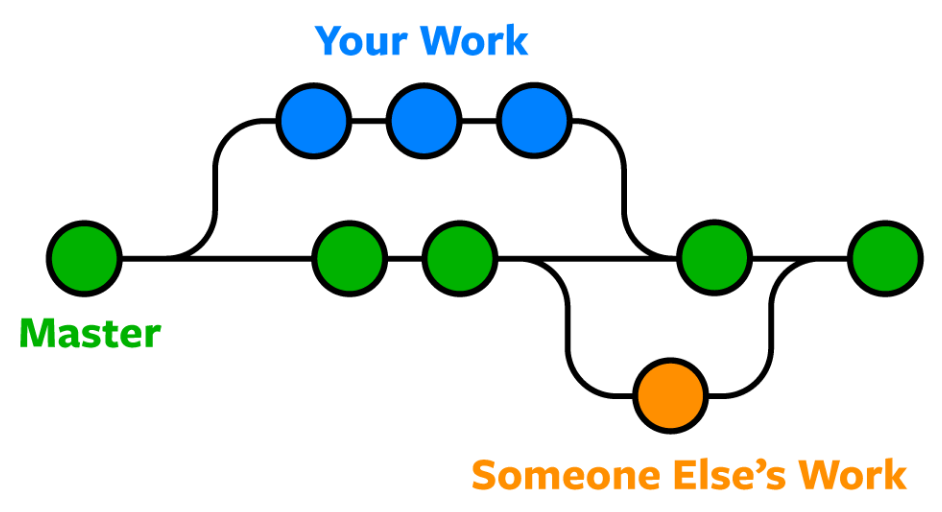
**What is a Working Copy?**

Your *working copy* (sometimes called a *checkout*) is your personal copy of all the files in the project. You make arbitrary edits to this copy, without affecting your teammates. When you are happy with your edits, you commit your changes to a *repository*.



**What is Git?**

Git is the most commonly used version control system. Git tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions should you ever need to. Git also makes collaboration easier, allowing changes by multiple people to all be merged into one source.



Git is software that runs locally. Your files and their history are stored on your computer. You can also use online hosts (such as [GitHub](https://github.com/) or [Bitbucket](https://bitbucket.org/)) to store a copy of the files and their revision history. Having a centrally located place where you can upload your changes and download changes from others, enable you to collaborate more easily with other developers. Git can automatically merge the changes, so two people can even work on different parts of the same file and later merge those changes without losing each other’s work!

**Git Repositories**

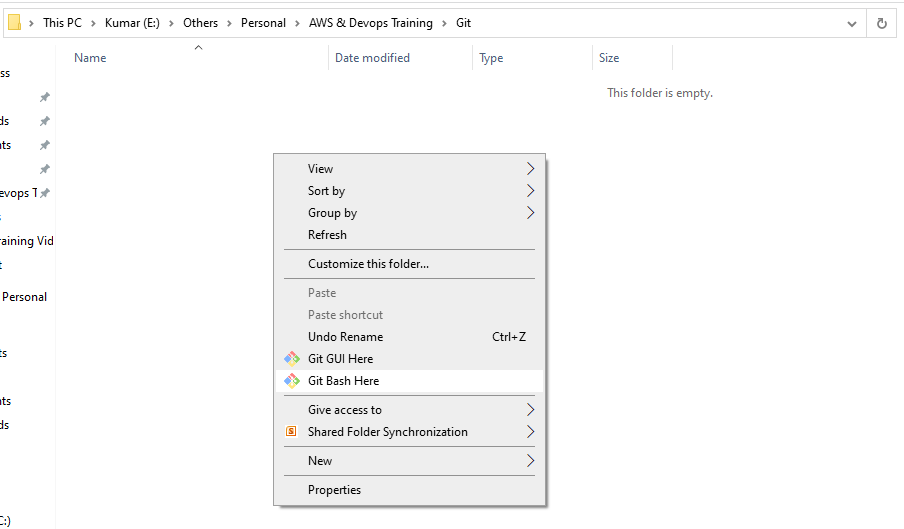
A Git **repository** (or **repo** for short) contains all of the project files and the entire revision history. You’ll take an ordinary folder of files (such as a website’s root folder), and tell Git to make it a repository. This creates a **.git** subfolder, which contains all of the Git metadata for tracking changes.

Git created **init** and **clone**. Both of which give you a repository in which you can manage your source code. The difference being of course that you would use the init command to create a repository from scratch, whereas you’d use clone to literally clone, or copy an existing repository into the directory you ran the command from. Once this is done, we’re able to start our workflow!

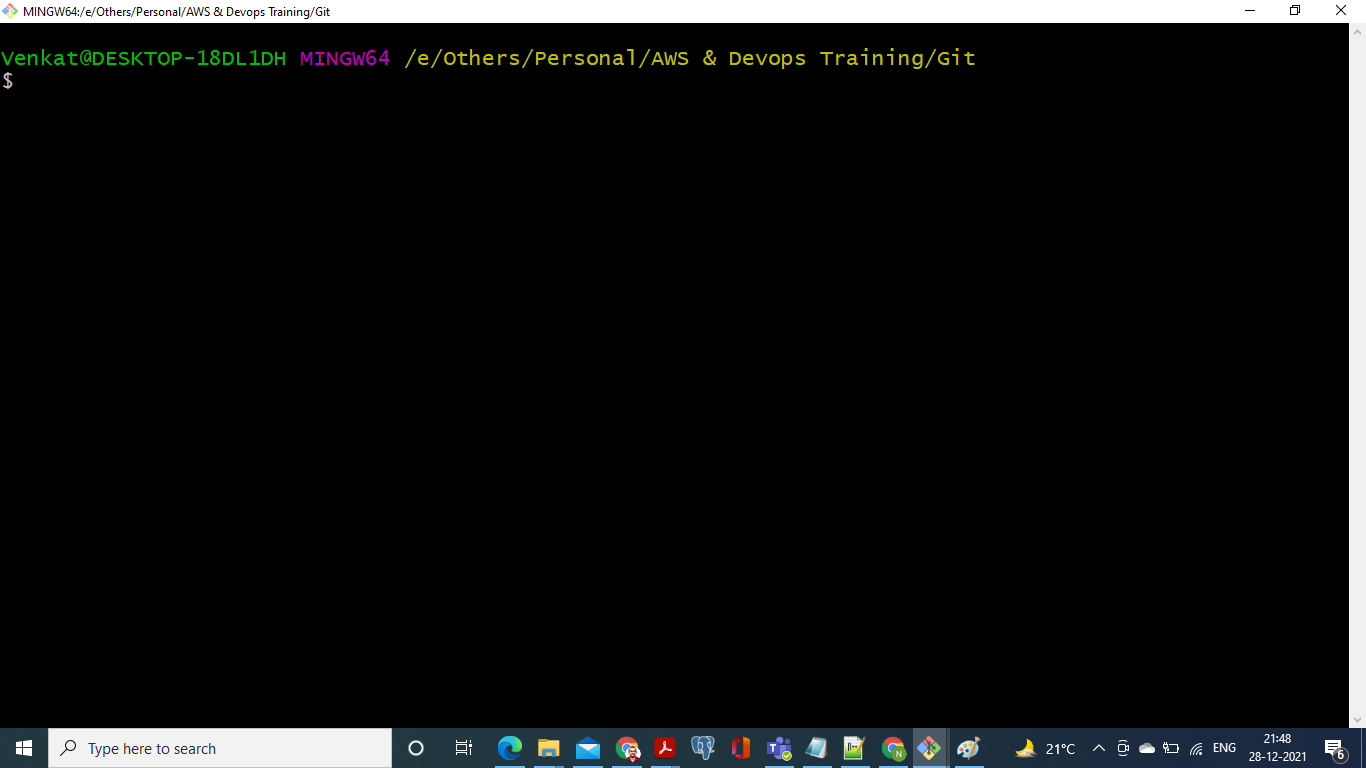
**Steps to create a local repository**

**git init  : it enables your directory as a working directory.**

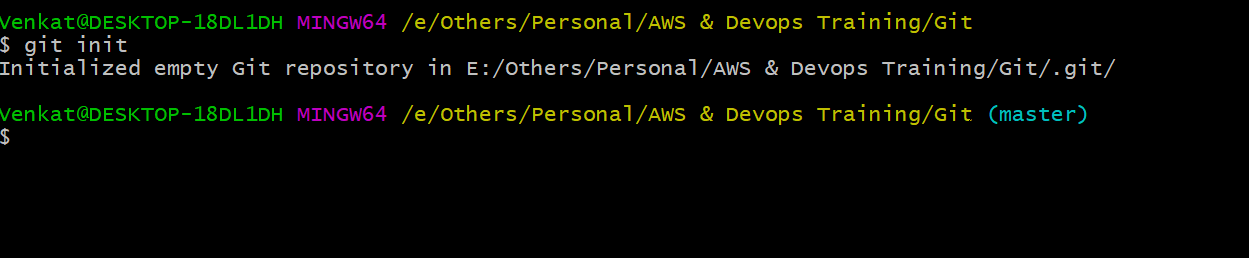
1. Create an empty folder and open Git Bash here.

****

1. Initially Git Bash terminal looks like below.



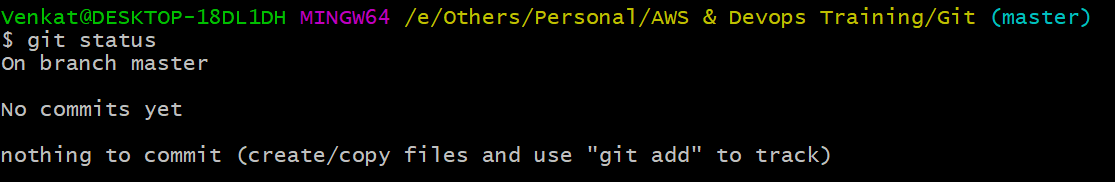
1. Now execute command **git init**

****

You can see the difference(master), now a working directory has been created.

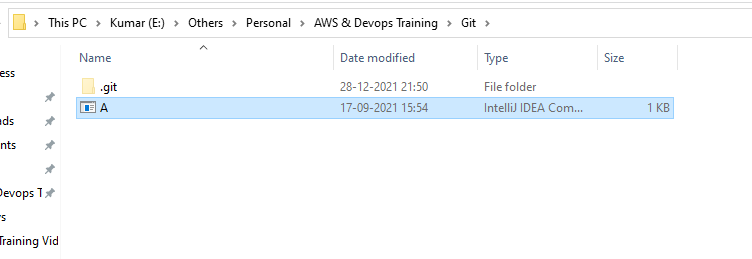
Now you are able to execute all git commands, which was not possible before.

1. **git status :** To check the status of the working directory.



Now there are no changes in working copy.

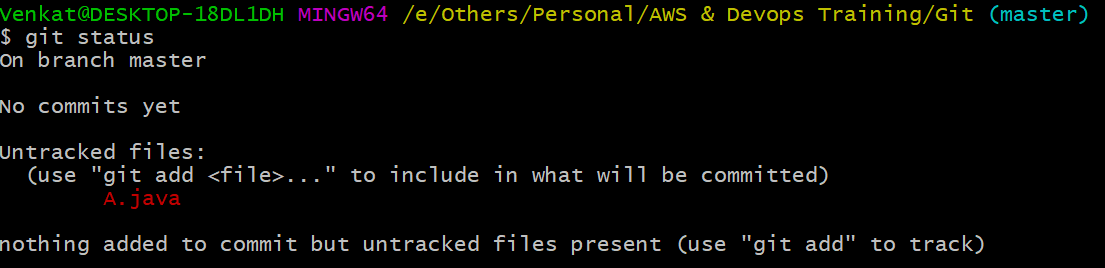
1. Add some files to your working copy.



.git represents a local repository.

Now the A.java file has been copied to the working directory.

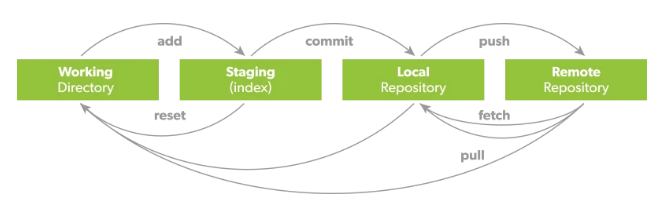
Check the status again.



Here A.java is a new file added, and its considered as untracked file, to make the file tracked use the command **git add file-name**

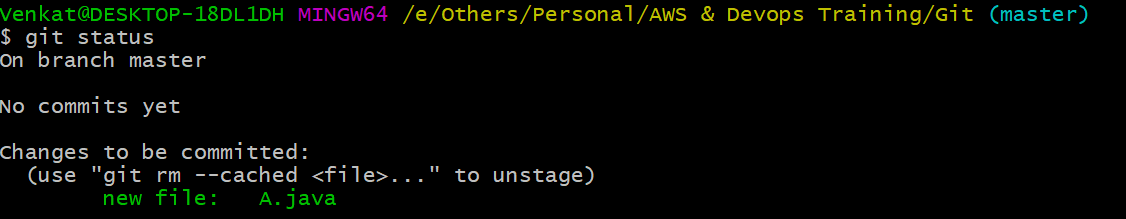
**https://lh5.googleusercontent.com/riP0lJ9zETH4u6uEH-8o3IXNsYkScZL4IQ45FattaGo-77nC0ljPbxZuZb2w-Yjs5Inmzpvj6j7d1mE7yBjoA1Ql5vgGonYnFbIRjYsFPgLWsAORQP5ubO_W0ed9sOI6hffCidw**

**Now we should know a little bit of workflow.**



Created/Updated/Deleted files will be treat as a untracked files, by using **add** command we can send them to Staging area(Tracked files)

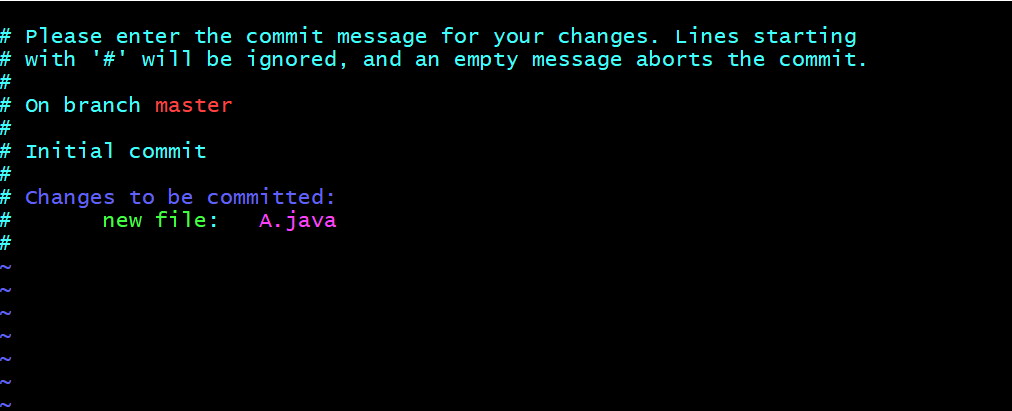
From staging we should send those files to the Local Repository by using the commit command.



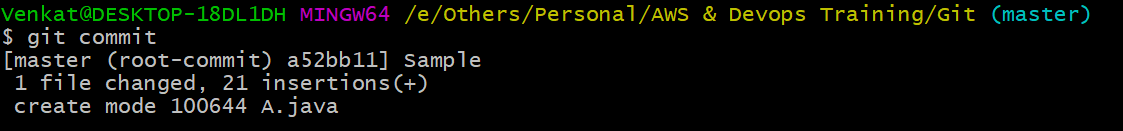
The file shown in green color represents a tracked file.

https://lh5.googleusercontent.com/ARlaRW_BV_4pHhkgmbSpHr_WmUQzOqpkrOjR5MCbt8UOgJW2c0GLiYW2bqOMKFuT13Mqa_i4XQ5n2T79gcgff1yj0LKGAyU6ct41WokXt1vdKZa752JJe95f3Jbu3oDDTAV49ig

Above command popup a new window to provide commit message.



Provide a message and proceed.



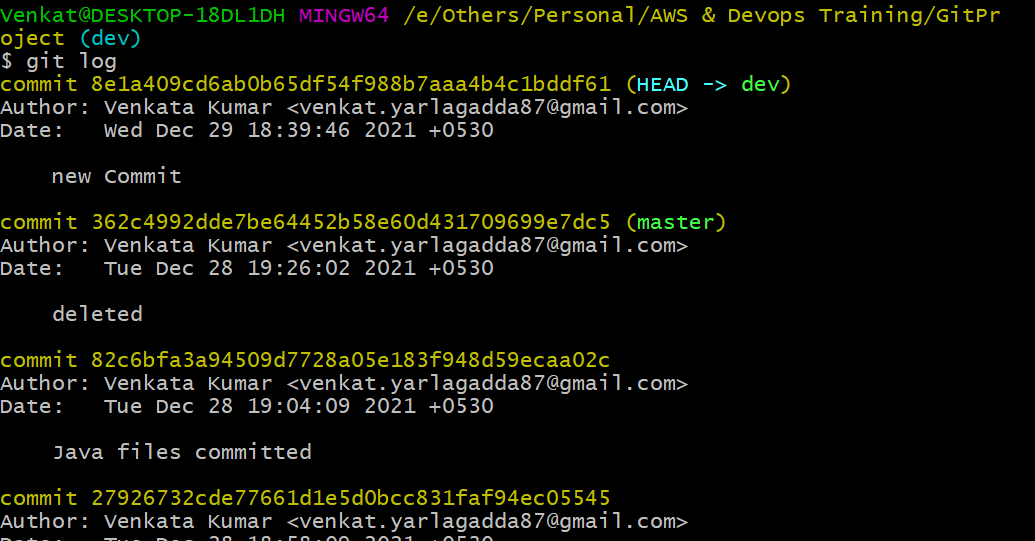
Now the file has been moved to the Local Repository.

What is **Git Commit**?

A Git commit records a snapshot of your changes, marks your name, email address and commit message giving you a commit id.

**git log command**

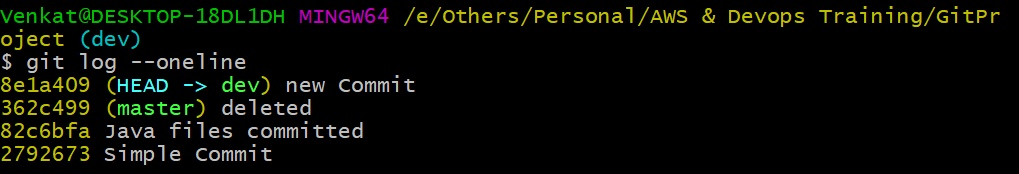
This command shows you the history of commits you did till now.



Who committed, date, to which branch everything will be tracked by this.

Same command we can use in different ways like below.

**git log –oneline**



All commits show you in short format.

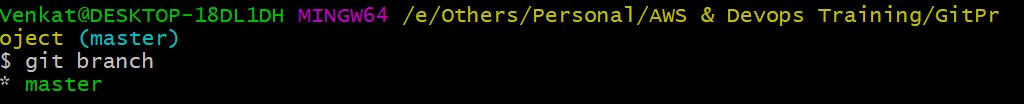
**Git Branches**

Git lets you branch out from the original code base. This lets you more easily work with other developers, and gives you a lot of flexibility in your workflow.

Here’s an example of how Git branches are useful. Let’s say you need to work on a new feature for a website. You create a new branch and start working. You haven’t finished your new feature, but you get a request to make a rush change that needs to go live on the site today. You switch back to the master branch, make the change, and push it live. Then you can switch back to your new feature branch and finish your work. When you’re done, you merge the new feature branch into the master branch and both the new feature and rush change are kept!

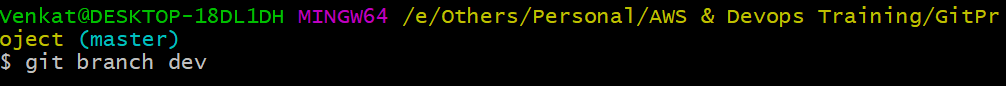
Let us create a branch like the one below.

**git branch  :** This command shows you no of branches available with the current code base.

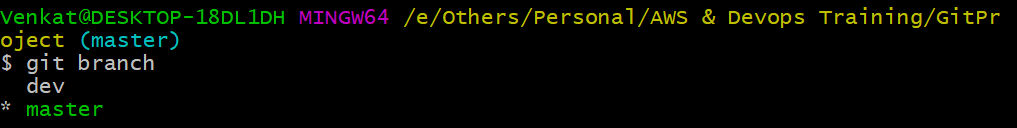


Initially there is only one branch available that is master(default).

**git branch branch-name**

****

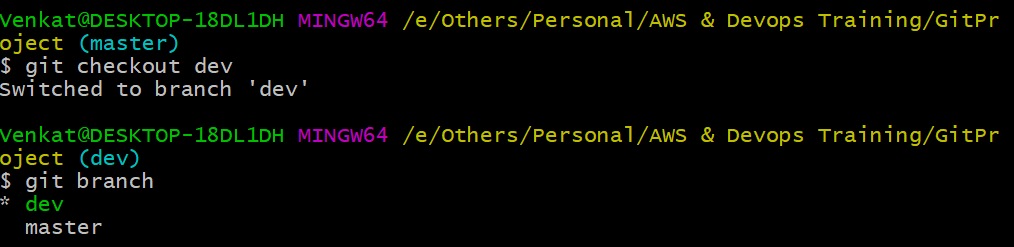
Above command creates the branch called dev. To check the branches use **git branch**

****

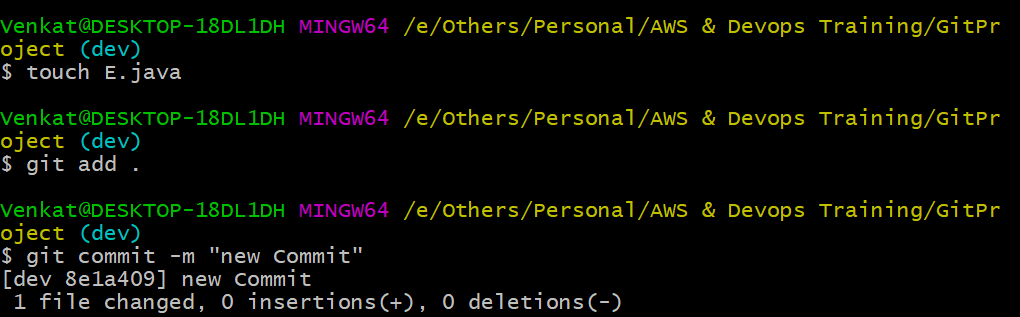
**\* mark and green colored text indicates current branch.**

**How to switch to another branch?**

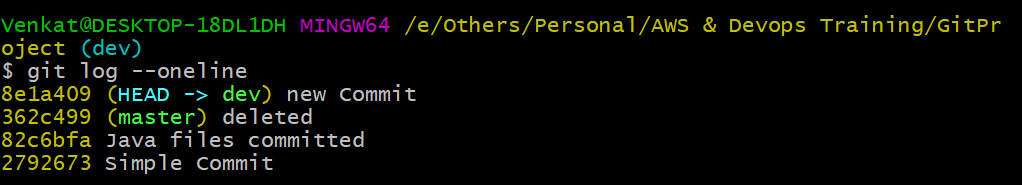
**git checkout/switch branch-name**

****

The changes done in one branch won’t affect other branch code.Each and every commit will be preserved with the specific branch.



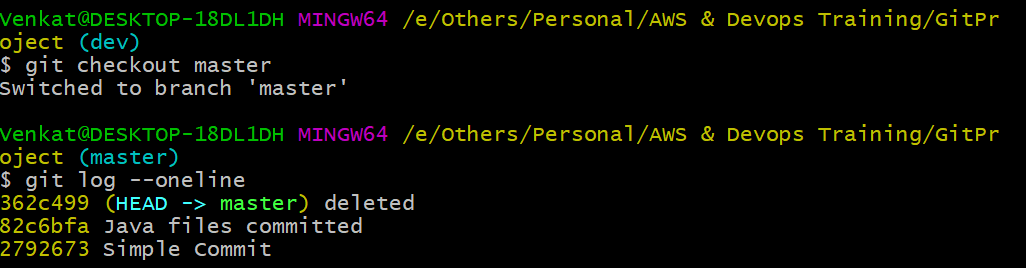
Add one new file in the new branch(dev) and commit it to the local repository. Now check the log once.



Here HEAD represents the current branch.

There are commits available which are related to the master branch and the latest commits of the current branch.

To verify them in the master branch, first switch to it.



So above status not showing dev branch commits. To avail them we need to merge dev with master.

**git merge dev**

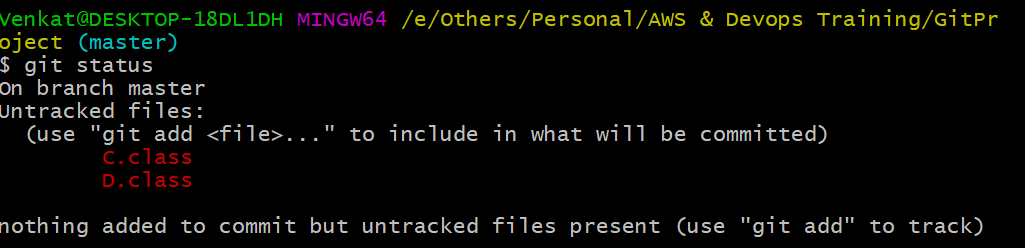
****

**\* To do this we need to switch back from dev to master.**

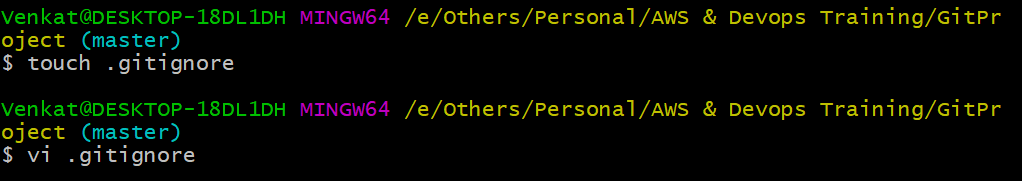
Even merge also treats a separate commit as this is important for developers to revert the merge incase of anything wrong.

**.gitignore :** To avoid some set of files from commit we can use the file called .gitignore. If you mention the kind of files to stop in this file then those files will not be considered as untracked files in the Working Directory.

**Before creation of .gitignore, I have added some .class files.**

****

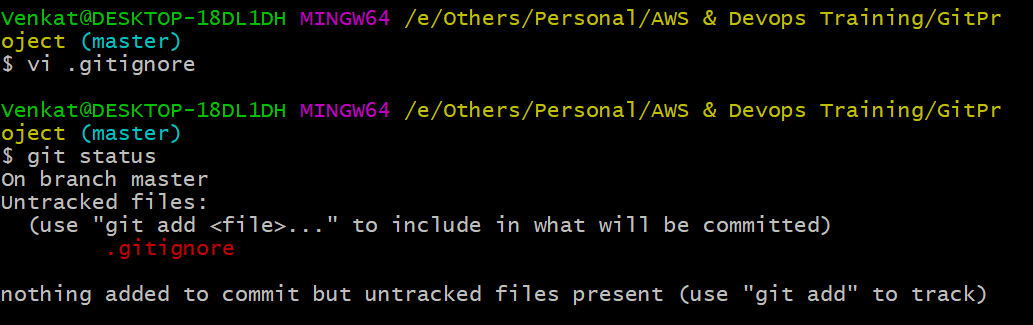
Then create a file like below.



Above command opens vi editor like below.



I want to stop .class files, so mentioned in the above format and saved.



Now you can observe there are no .class files available.

**Pick selective commits**

Cherry picking in Git means to choose a commit from one branch and apply it onto another.

This is in contrast with other ways such as merge and rebase which normally apply many commits onto another branch.

1. Make sure you are on the branch you want to apply the commit to.  
    **git switch master**
2. Execute the following:  
    **git cherry-pick <commit-hash> (commit id)**