

GIT VCS

# What is GIT.

- **Git** is an **open-source distributed version control system**. It is designed to handle minor to major projects with high speed and efficiency. It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.
- Git was created by **Linus Torvalds** in **2005** to develop Linux Kernel. It is also used as an important distributed version-control tool for **the DevOps**.

# Features of GIT

- **Open Source**
- **Scalable**
- **Distributed**

One of Git's great features is that it is **distributed**. Distributed means that instead of switching the project to another machine, we can create a "clone" of the entire repository.

- **Security**
- **Speed**

Git is very **fast**, so it can complete all the tasks in a while.

# Benefits of GIT.

- **Saves Time**

Git is lightning fast technology. Each command takes only a few seconds to execute so we can save a lot of time as compared to login to a GitHub account.

- **Offline Working**

One of the most important benefits of Git is that it supports **offline working**. If we are facing internet connectivity issues, it will not affect our work. In Git, we can do almost everything locally.

- **Undo Mistakes**

One additional benefit of Git is we can **Undo** mistakes.

- **Track the Changes**

Git facilitates with some exciting features such as **Diff**, **Log**, and **Status**, which allows us to track changes so we can **check the status, compare** our files or branches.

# What is GitHub.

- GitHub is a Git repository hosting service.
- It offers both **distributed version control and source code management (SCM)** functionality of Git.

# What is GIT Version control system.

- A version control system is a software that tracks changes to a file or set of files over time so that you can recall specific versions later. It also allows you to work together with other programmers.
- The version control system is a collection of software tools that help a team to manage changes in a source code. It uses a special kind of database to keep track of every modification to the code.
- Developers can compare earlier versions of the code with an older version to fix the mistakes.

# Benefits of the Version Control System

- Complete change history of the file
- Simultaneously working
- Branching and merging
- Traceability

# Types of version control system.

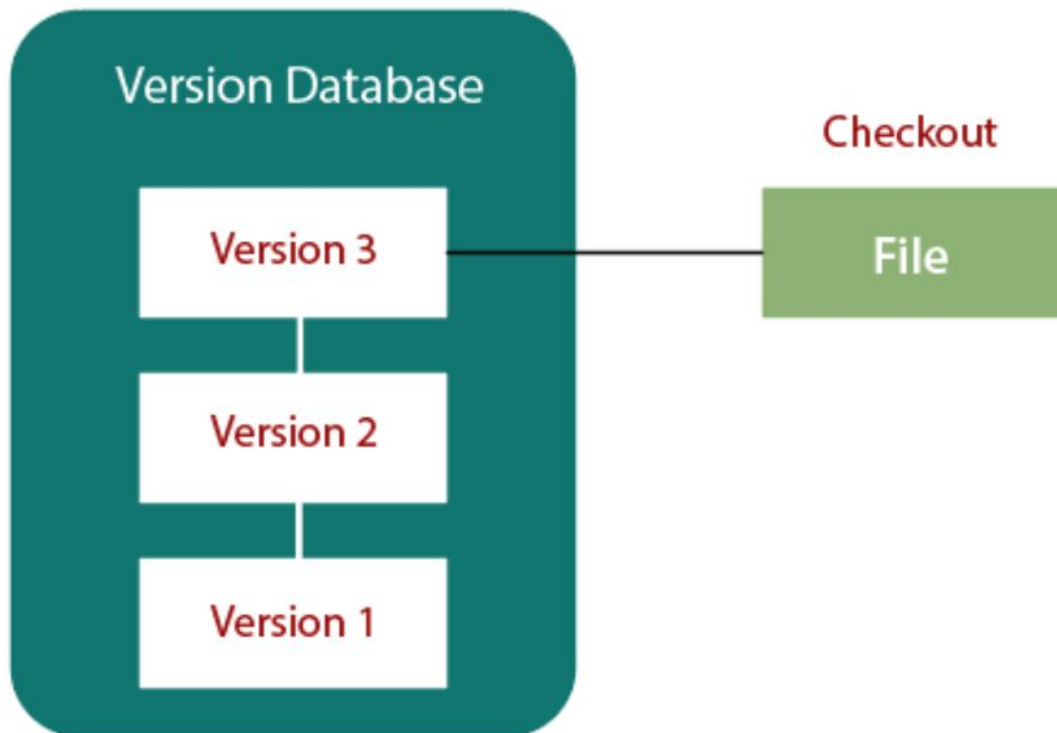
- Localized version Control System
- Centralized version control systems
- Distributed version control systems



# Localized Version Control Systems

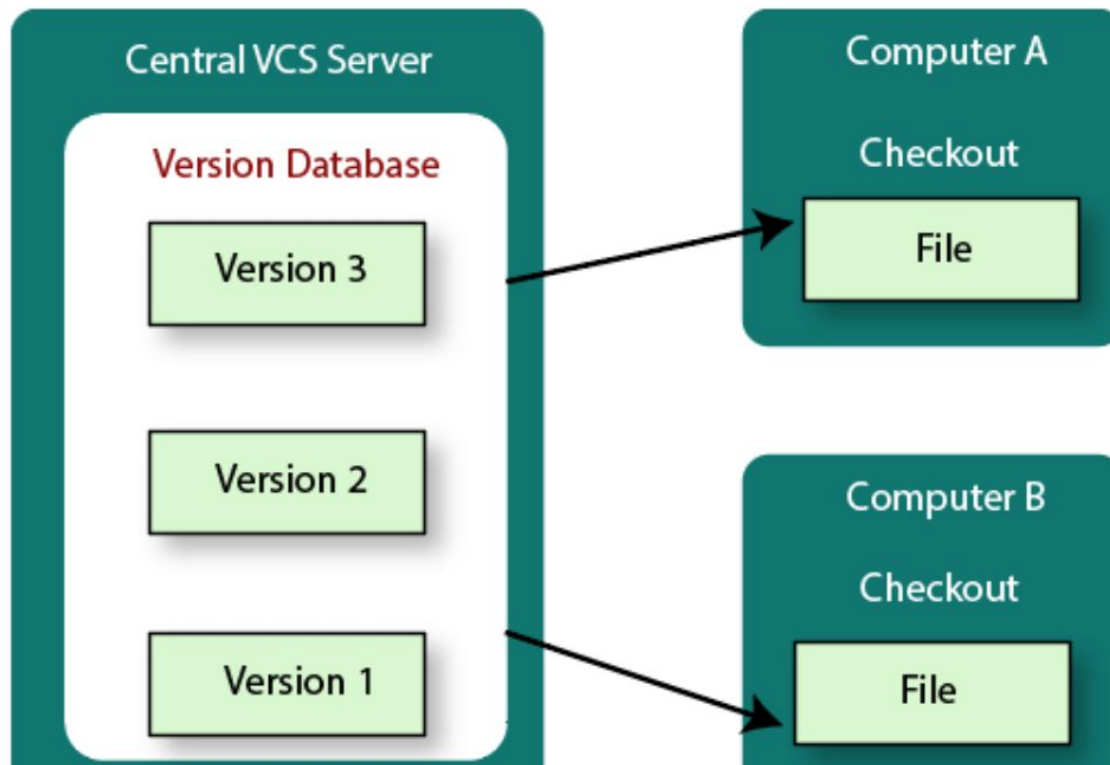
The localized version control method is a common approach because of its simplicity. But this approach leads to a higher chance of error. Also cant integrate other programmers changes in the system.

Local Computer



# Centralized Version Control System

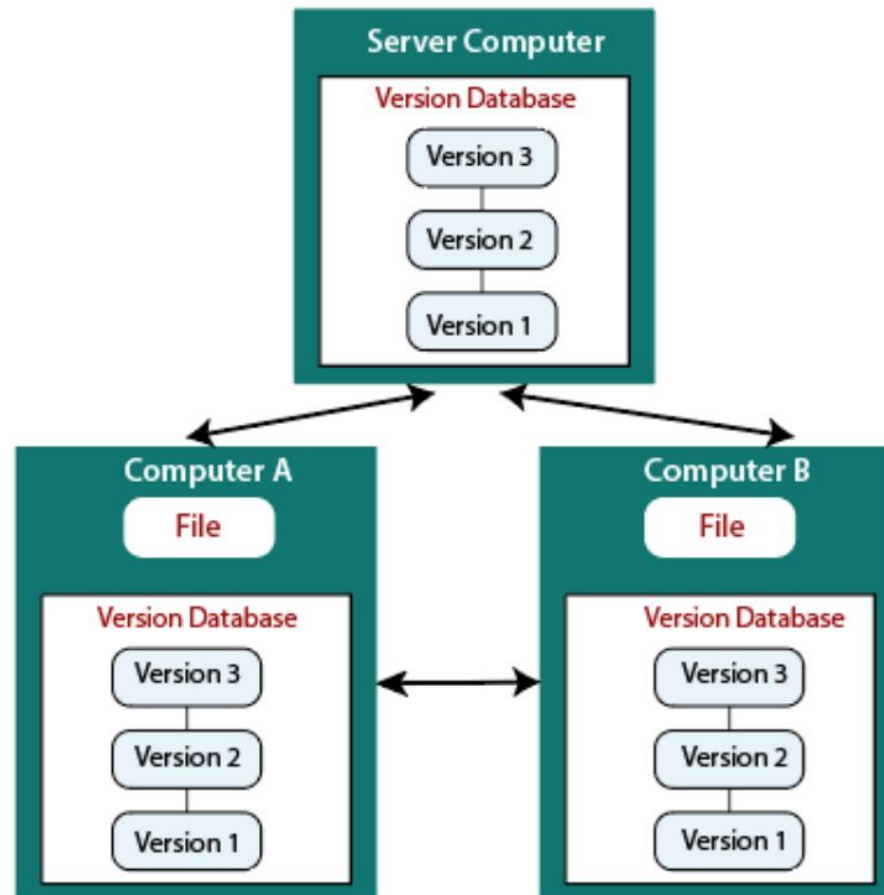
The developers needed to collaborate with other developers on other systems. The localized version control system failed in this case. To deal with this problem, Centralized Version Control Systems were developed.



# Distributed Version Control System

- Centralized Version Control System uses a central server to store all the database and team collaboration. But due to single point failure, which means the failure of the central server, developers do not prefer it. Next, the Distributed Version Control System is developed.
- In a Distributed Version Control System (such as Git, Mercurial, Bazaar or Darcs), the user has a local copy of a repository. The local repository contains all the files and metadata present in the main repository.

# Distributed Version Control System

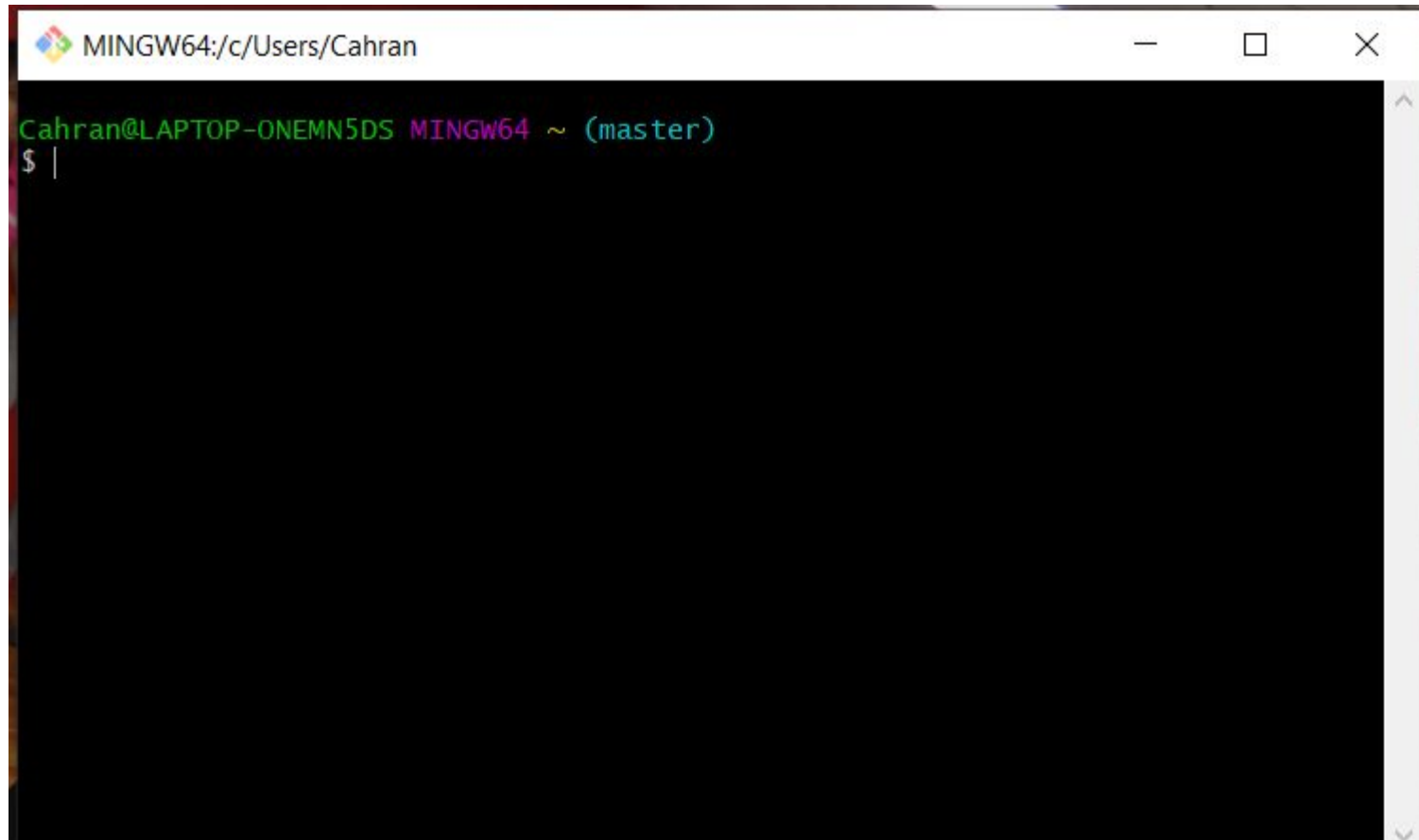


# Install git.

- Install git from the following location.
- <https://git-scm.com/downloads>.

# Git

- After the installation, open the application and it opens given below:



```
MINGW64:/c/Users/Cahran  
Cahran@LAPTOP-ONEMN5DS MINGW64 ~ (master)  
$ |
```

The image shows a screenshot of a MINGW64 terminal window. The title bar at the top reads "MINGW64:/c/Users/Cahran" and includes standard window control buttons (minimize, maximize, close). The terminal content shows the user "Cahran" at the prompt "Cahran@LAPTOP-ONEMN5DS" in a green font. The environment is "MINGW64" in pink, the home directory is "~" in green, and the current branch is "(master)" in blue. The prompt character "\$" is followed by a vertical bar "|", indicating the terminal is ready for input.

# Setting the environment.

- Setting **user.name** and **user.email** are the necessary configuration options as your name and email will show up in your commit messages.
- `$ git config --global user.name "Charan Kumar"`
- `$ git config --global user.email "kcharankumar@gmail.com"`

# Git commands.

- The git init command is the first command that you will run on Git. The git init command is used to create a new blank repository. It is used to make an existing project as a Git project.
- \$ Git init
- Create a new file and add it to the repository and commit it and push it to master.



# Add files and commit.

- Create file by name my\_first\_file.txt from explorer.
- `$git add "my_first_file.txt"`
- `$git commit -m "This is the first file to commit"`

# Cloning

- We can clone hithub repository as a local copy.
- Go to github.com, place few files. Commit the changes.
- Click on code under repository. Select the URL.
- In the command line of gitbash create a new directory, move to that directory and then give the following command
- \$git clone <URL from the github>
- All the files are cloned from github to local git.

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test2/clone_data (master)
$ git clone http://github.com/kcharankumar/Test.git
Cloning into 'Test'...
warning: redirecting to https://github.com/kcharankumar/Test.git/
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), done.

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test2/clone_data (master)
$ |
```

# GIT Repository.

- A quick recap, In Git, the repository is like a data structure used by VCS to store metadata for a set of files and directories. It contains the collection of the files as well as the history of changes made to those files.

There are two ways to obtain a repository. They are as follows:

- Create a local repository and make it as Git repository.
- Clone a remote repository (already exists on a server).

# Repository.

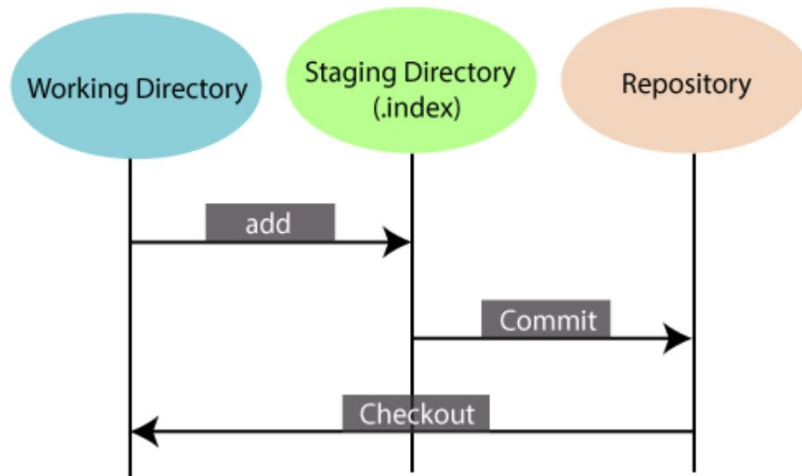
Local repository

- \$git init
- \$git add “file”
- \$git commit –m “Comments”

Cloning a remote repository.

- \$git clone “Repository URL”

# GIT Index.



**Working Directory:** When you worked on your project and made some changes, you are dealing with your project's working directory.

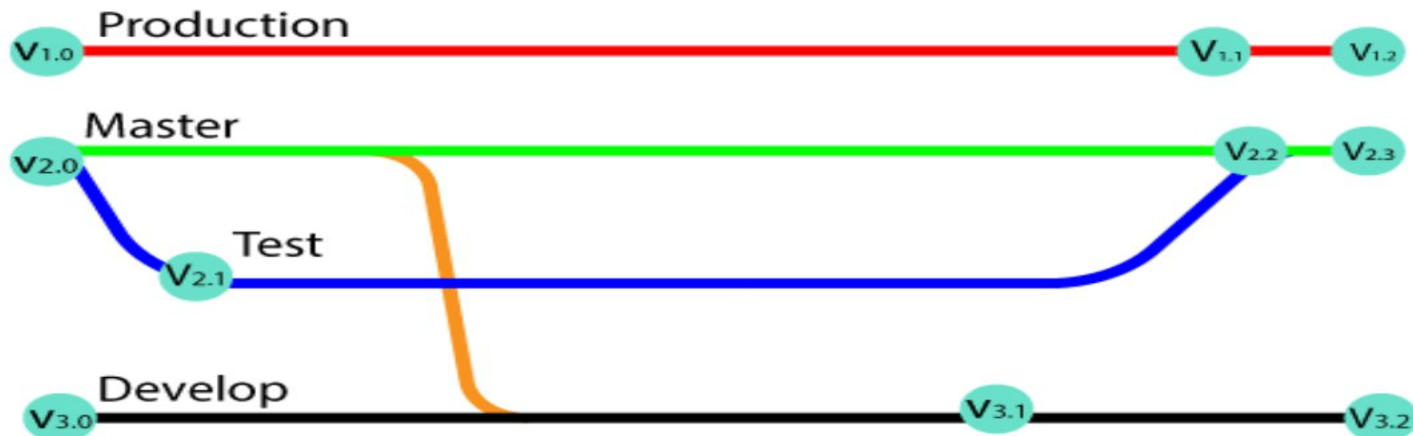
**Staging Area :** The staging area can be described as a preview of your next commit. When you create a git commit, Git takes changes that are in the staging area and make them as a new commit.

**Repository :** In Git, Repository is like a data structure used by Git to store metadata for a set of files and directories. It contains the collection of the files as well as the history of changes made to those files.

Index is the one which stores information about the file like Time of last update, name of the file, version of the file.

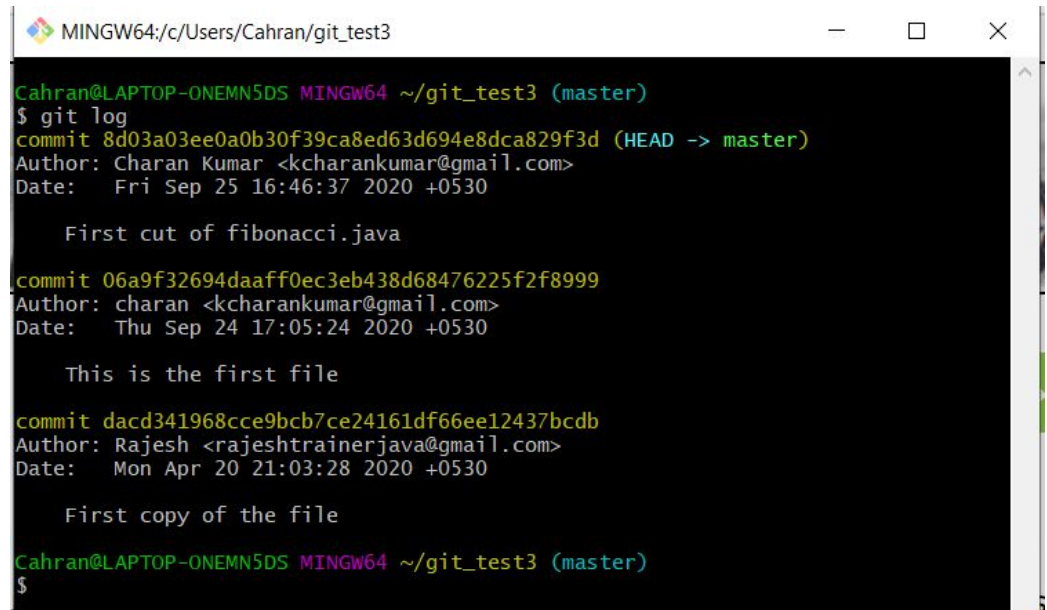
# GIT Tag

- Tags make a point as a specific point in Git history. Tags are used to mark a commit stage as relevant. We can tag a commit for future reference. Primarily, it is used to mark a project's initial point like v1.1.
- Tags are much like branches, and they do not change once initiated. We can have any number of tags on a branch or different branches. The below figure demonstrates the tags on various branches.



# Git log

- Git log is a utility tool to review and read a history of everything that happens to a repository.
- Generally, the git log is a record of commits.



```
MINGW64:/c/Users/Cahran/git_test3

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git log
commit 8d03a03ee0a0b30f39ca8ed63d694e8dca829f3d (HEAD -> master)
Author: Charan Kumar <kcharankumar@gmail.com>
Date:   Fri Sep 25 16:46:37 2020 +0530

    First cut of fibonacci.java

commit 06a9f32694daaff0ec3eb438d68476225f2f8999
Author: charan <kcharankumar@gmail.com>
Date:   Thu Sep 24 17:05:24 2020 +0530

    This is the first file

commit dacd341968cce9bcb7ce24161df66ee12437bcd
Author: Rajesh <rajeshttrainerjava@gmail.com>
Date:   Mon Apr 20 21:03:28 2020 +0530

    First copy of the file

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$
```

# Git diff.

- Diff will specify the difference of changes done from repository to the local copy.
- For this create a file, add a file to the staging area and commit it repository and then modify the working copy of the same file and give the git diff command.
- \$git diff fibonacci.java

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git diff fibonacci.java
diff --git a/git_test3/fibonacci.java b/git_test3/fibonacci.java
index b55df72..17f0683 100644
--- a/git_test3/fibonacci.java
+++ b/git_test3/fibonacci.java
@@ -6,4 +6,6 @@ import java.io.*;

    class fibonacci
    {
+   public static void main (String args[])
+   {}
    }
\ No newline at end of file

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ |
```



# Git status

- The git status command is used to display the state of the repository and staging area. It allows us to see the tracked, untracked files and changes. This command will not show any commit records or information.
- Mostly, it is used to display the state between **Git Add** and **Git commit** command. We can check whether the changes and files are tracked or not.

# Git status

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git status
warning: could not open directory 'Application Data/': Permission denied
warning: could not open directory 'Cookies/': Permission denied
warning: could not open directory 'Local Settings/': Permission denied
warning: could not open directory 'My Documents/': Permission denied
warning: could not open directory 'NetHood/': Permission denied
warning: could not open directory 'PrintHood/': Permission denied
warning: could not open directory 'Recent/': Permission denied
warning: could not open directory 'SendTo/': Permission denied
warning: could not open directory 'Start Menu/': Permission denied
warning: could not open directory 'Templates/': Permission denied
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   fibonacci.java

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   fibonacci.java
        modified:   fifth.txt
```

Here the files are modified in the local copy but not staged and committed.

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git status
warning: could not open directory 'Application Data/': Permission denied
warning: could not open directory 'Cookies/': Permission denied
warning: could not open directory 'Local Settings/': Permission denied
warning: could not open directory 'My Documents/': Permission denied
warning: could not open directory 'NetHood/': Permission denied
warning: could not open directory 'PrintHood/': Permission denied
warning: could not open directory 'Recent/': Permission denied
warning: could not open directory 'SendTo/': Permission denied
warning: could not open directory 'Start Menu/': Permission denied
warning: could not open directory 'Templates/': Permission denied
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   fibonacci.java
        modified:   fifth.txt
```

After adding these files in the staging area, git status is specifying these two files are not committed in the repository.

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git commit fibonacci.java -m "main() is added"
[master f1765e7] main() is added
1 file changed, 3 insertions(+), 1 deletion(-)

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git commit fifth.txt -m "json information is added"
[master 1445154] json information is added
1 file changed, 3 insertions(+), 1 deletion(-)

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git status
warning: could not open directory 'Application Data/': Permission denied
warning: could not open directory 'Cookies/': Permission denied
warning: could not open directory 'Local Settings/': Permission denied
warning: could not open directory 'My Documents/': Permission denied
warning: could not open directory 'NetHood/': Permission denied
warning: could not open directory 'PrintHood/': Permission denied
warning: could not open directory 'Recent/': Permission denied
warning: could not open directory 'SendTo/': Permission denied
warning: could not open directory 'Start Menu/': Permission denied
warning: could not open directory 'Templates/': Permission denied
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

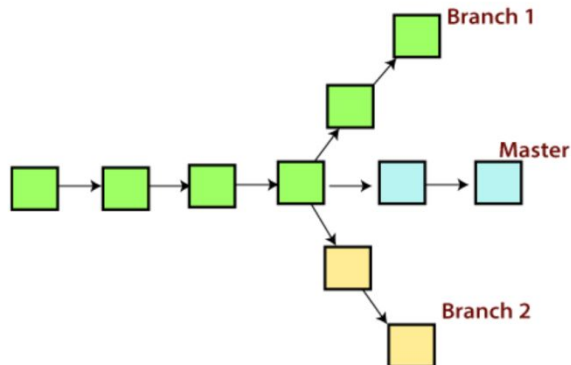
After committing the changes, status shows that there are no modified or to be committed.

# Git mv

- Git mv is used to rename a file.
- Create a file as OldFile.txt and commit it.
- `$git mv -f OldFile.txt NewFile.txt`
- Check whether the file name is modified or not. Open the content of the file and verify it.

# GIT Branching

- A branch is a version of the repository that diverges from the main working project. It is a feature available in most modern version control systems. A Git project can have more than one branch. These branches are a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug, you spawn a new branch to summarize your changes. So, it is complex to merge the unstable code with the main code base and also facilitates you to clean up your future history before merging with the main branch.



# GIT Branch

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git branch MyBranch

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git branch --list
  MyBranch
* master

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git checkout MyBranch
Switched to branch 'MyBranch'

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (MyBranch)
$ git add BranchFile.txt
fatal: pathspec 'BranchFile.txt' did not match any files

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (MyBranch)
$ git add BranchFile.txt

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (MyBranch)
$
```

## Git checkout

switches from branch to branch. A new file is added in the branch.

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git checkout MyBranch
Switched to branch 'MyBranch'

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (MyBranch)
$ git merge master
Updating 41388e2..6289b48
Fast-forward
 git_test3/factorial.txt | 4 ++++
 git_test3/fibonacci.java | 4 +++-
 2 files changed, 7 insertions(+), 1 deletion(-)
 create mode 100644 git_test3/factorial.txt

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (MyBranch)
$ git checkout master
Switched to branch 'master'
```

Create a file in master and create a branch. Switch to that branch. Merge from master to newly created branch.

Git merge merges from one branch to another branch.

After modifying the file in the branch, it has to be staged by git add and commit it and merge it in the master branch.

# Git merge

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (MyBranch)
$ git add factorial.txt

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (MyBranch)
$ git commit -m "sop is added"
[MyBranch d82959a] sop is added
1 file changed, 3 insertions(+)

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (MyBranch)
$ git checkout master
Switched to branch 'master'

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git merge
fatal: No remote for the current branch.

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test3 (master)
$ git merge MyBranch
Updating c5b6e10..d82959a
Fast-forward
 git_test3/factorial.txt | 3 +++
1 file changed, 3 insertions(+)
```

# Git ignore

- Create a .gitignore file  
\$touch .gitignore  
in gitignore file, add MyFile.txt
- Add .gitignore file and commit it.
- Even if MyFile.txt is there in the directory, it will not show in git status to add the file.  
When this file to be added in the staging area, it says this file is in the ignored list.

# Git ignore

- If the file is added then it will push into the github.

```
MINGW64:/c/Users/Cahran/git_test5/test/test
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (master)
$ touch .gitignore
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (master)
$ cat .gitignore
MyFile.txt
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (master)
$
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (master)
$
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (master)
$ git add MyFile.txt
The following paths are ignored by one of your .gitignore files:
MyFile.txt
hint: Use -f if you really want to add them.
hint: Turn this message off by running
hint: "git config advice.addIgnoredFile false"
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (master)
$ |
```



# Git stash

MINGW64:/c/Users/Cahran/git\_test5/test/test

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (master)
$ git branch
* master
  test3

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (master)
$ git checkout test3
Switched to branch 'test3'

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (test3)
$ git status
On branch test3
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    MyFile2.txt

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

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (test3)
$ git status
On branch test3
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    MyFile2.txt
    testBranchFile1.txt

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

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (test3)
$ git add testBranchFile1.txt

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (test3)
$ git status
On branch test3
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   testBranchFile1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    MyFile2.txt
    testBranchFile2.txt
```

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (master)
$ git branch
* master
  test3

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (master)
$ git checkout test3
Switched to branch 'test3'

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (test3)
$

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (test3)
$ git status
On branch test3
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    MyFile2.txt
    testBranchFile2.txt

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

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (test3)
$ git stash apply
On branch test3
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   testBranchFile1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    MyFile2.txt
    testBranchFile2.txt

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (test3)
$ git status
On branch test3
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   testBranchFile1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    MyFile2.txt
    testBranchFile2.txt
```

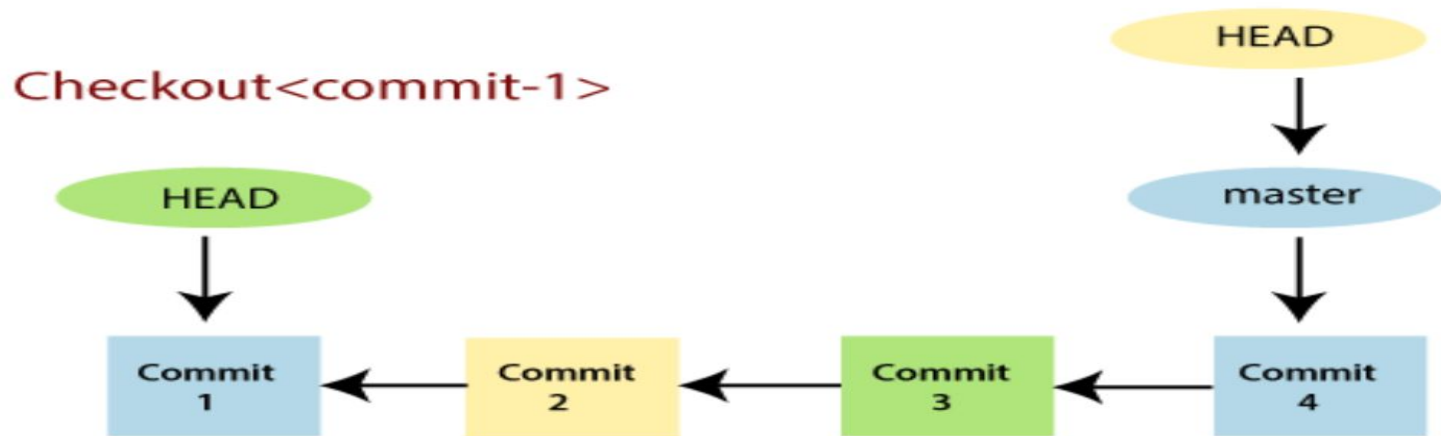
# Git stash

- After the changes are in staging area in a branch and if you want to switch to another branch without committing the changes. Then you can save the current state and then come back later and can revert from stash by using stash apply.
- Created a branch test, two files are created. One file is added to staging area. Applied git stash on it. Switch to another branch using git checkout. Come back to same branch. Apply git stash and check which files are in staging area and which files are not.

# Git head.

- The **HEAD** points out the last commit in the current checkout branch. It is like a pointer to any reference. The HEAD can be understood as the "**current branch.**" When you switch branches with 'checkout,' the HEAD is transferred to the new branch.

# Git head



- The above fig shows the HEAD referencing commit-1 because of a 'checkout' was done at commit-1. When you make a new commit, it shifts to the newer commit.

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test (master)
$ git show head
commit 71e7bd2f2eba70d4f86831795666153257270d26 (HEAD -> master)
Author: Charan Kumar <kcharankumar@gmail.com>
Date: Tue Sep 29 18:26:31 2020 +0530

New file created:
```

# Git show head

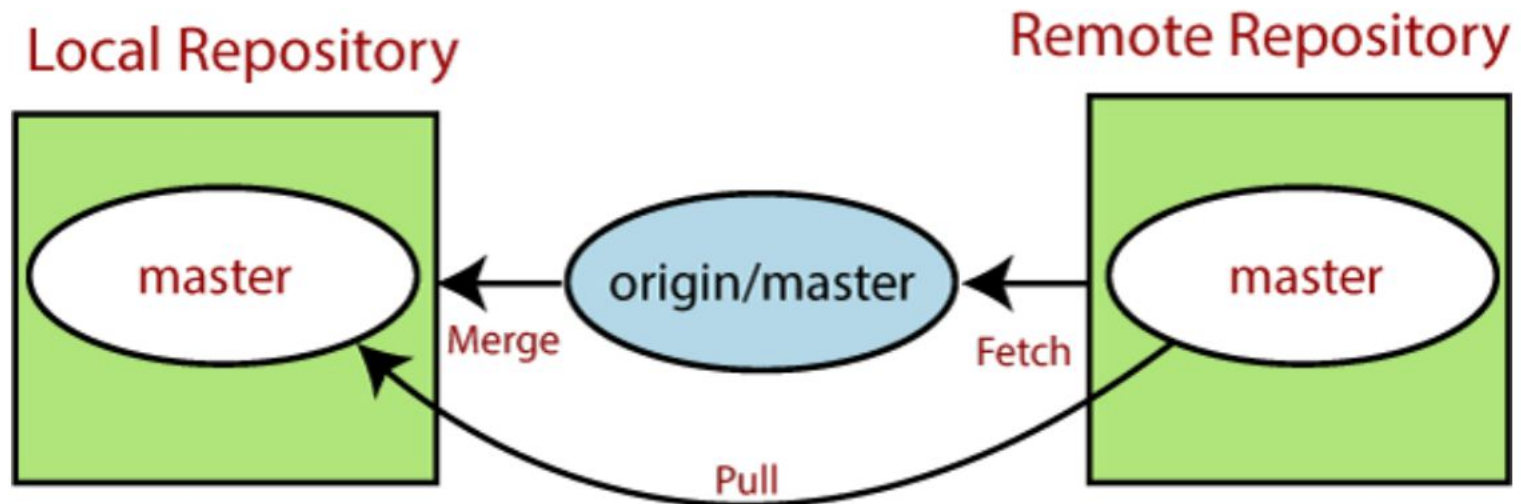
- Git show head shows the head location of the committed file. You can also see the same in git log.

# Git remote

- <https://docs.github.com/en/get-started/getting-started-with-git/managing-remote-repositories> --> For git remote add.
- `$git remote -v`  
Shows the remote server location.
- `$git remote rm origin`  
This will remove the origin or you can specify the remote server added.
- `$git remote add origin https://github.com/kcharankumar/Test.git`  
This will add the remote server. RemoteSer is the name given to the remote server.
- `$git remote -v`
- Create a new file in github.
- `$git pull origin master`  
This file is pulled into the local repository.
- `$git remote rm RemoteSer` □ This removes the RemoteSer connection with the remote server.
- `$git remote -v`
- If git push and pull is not working, first clone it with the remote git as
- `$git clone https://github.com/kcharankumar/Test.git` --> This will get the files from remote repository to the local repository.

# Git pull

- Git pull pulls the data from the remote repository to the local repository.



# Git revert

- Git revert will revert the previous commit.
- Create a file and commit it.
- Make the changes to the same file and commit it again.
- Check the log and get the head id.
- `$git revert <head id from log file>`

This will prompt a message to edit for removing the committed changes.

- Cat file name.



# Git revert

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test/test (master)
$ cat RevertFile.txt
This is the first line...
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test/test (master)
$ cat RevertFile.txt
This is the first line...
New line is added
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test/test (master)
$ git add RevertFile.txt

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test/test (master)
$ git commit -m "New line is added"
[master 74b774f] New line is added
1 file changed, 2 insertions(+), 1 deletion(-)

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test/test (master)
$ git log
commit 74b774f9332e71bf0700927005603013a303667b (HEAD -> master)
Author: Charan Kumar <kcharankumar@gmail.com>
Date: Tue Sep 29 19:58:12 2020 +0530

    New line is added
```

```
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test/test (master)
$ git revert 74b774f9332e71bf0700927005603013a303667b
[master 4cc9641] Revert "New line is added"
1 file changed, 1 insertion(+), 2 deletions(-)

Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test/test (master)
$ cat RevertFile.txt
This is the first line...
Cahran@LAPTOP-ONEMN5DS MINGW64 ~/git_test5/test/test/test (master)
$ |
```

# Git tag

- Create 3 files named file1, file2, file3.
- Add some content into it.
- Commit the changes.
- Create a tag for version1.0 for these files as

```
$git tag version1.0
```

```
$git tag □ Shows all the tags.
```

```
$git show version1.0 □ Displays the files of version1.0
```

- Modify these files file1, file2 and file3.
- Commit the changes.
- Create another tag version2.0 as

```
$git tag version2.0
```

```
$git tag
```

```
$git show version2.0
```

```
$git checkout version2.0 □ See the file content of version2.0
```

```
$git checkout version1.0 □ See the file content of version1.0
```

Thank you