



Git



Managing Versions of a Document

We keep making many changes in while building a software or even writing anything.

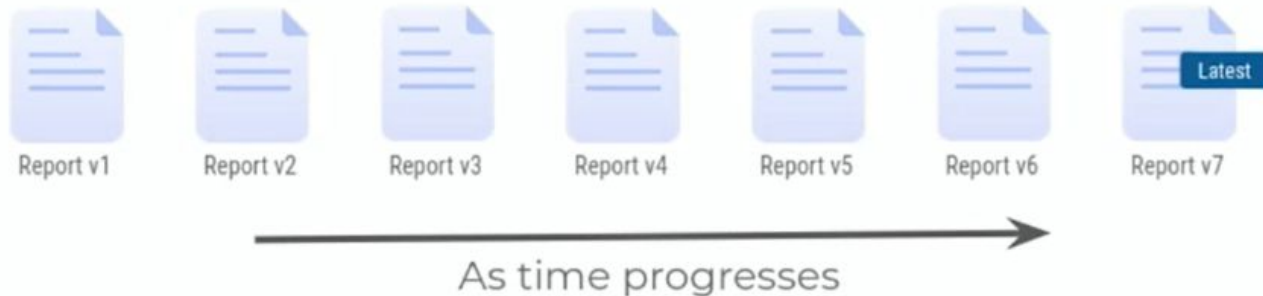


As time progresses →

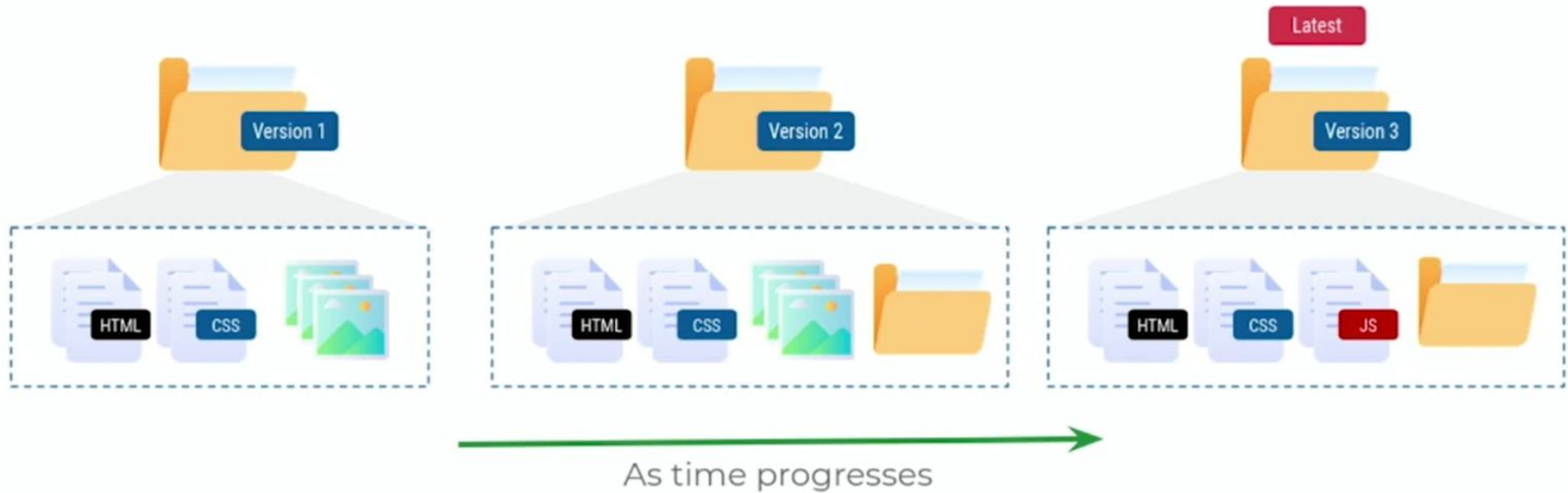
It is a ~~very~~ long established ~~point~~ fact that a ~~person~~ reader will be distracted by the readable content of a page when looking at its layout. ~~The reason why we use a placeholder text is~~ The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here,'

Advantages of Versioning

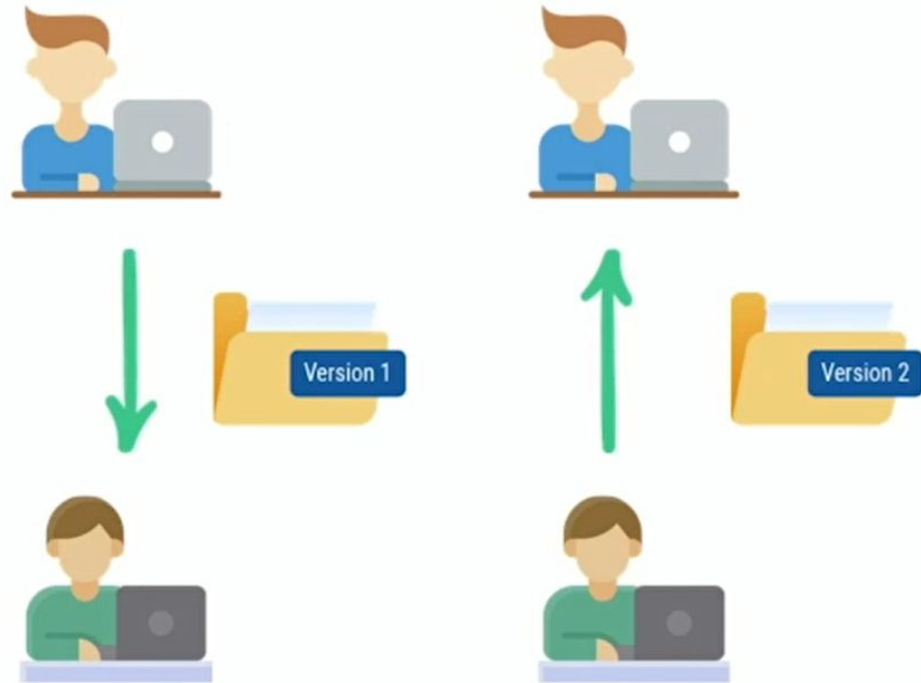
- We can quickly revert back to any of the older Versions, or pick up changes from older version.
- We can track how the files are modified over time.



Managing Versions of a project



Managing Versions of a project



Linux Kernel Project



997,300+ changes

71,400+ files



21 million
lines of code

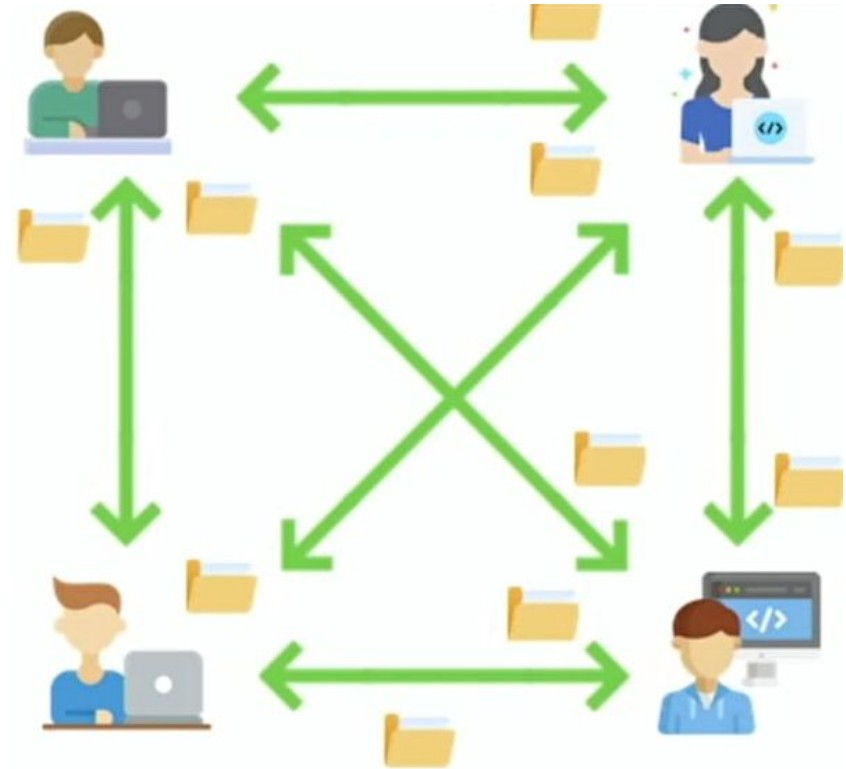


11,300+ developers

While working on a software Project, we keep making changes to existing files to add new features and fix existing bugs.

Collaboration

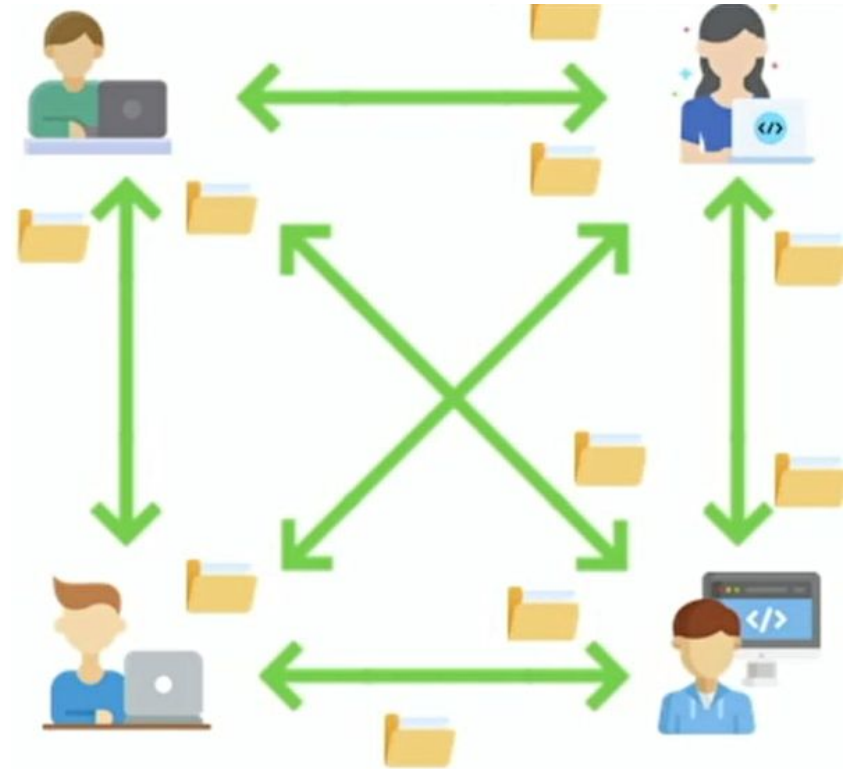
- While working on a huge Project with many files which being developed by multiple people it is hard to manually keep track of the changes.
- Any small change may completely crash the project.



Collaboration & Versioning

When working with multiple people
it is useful to know

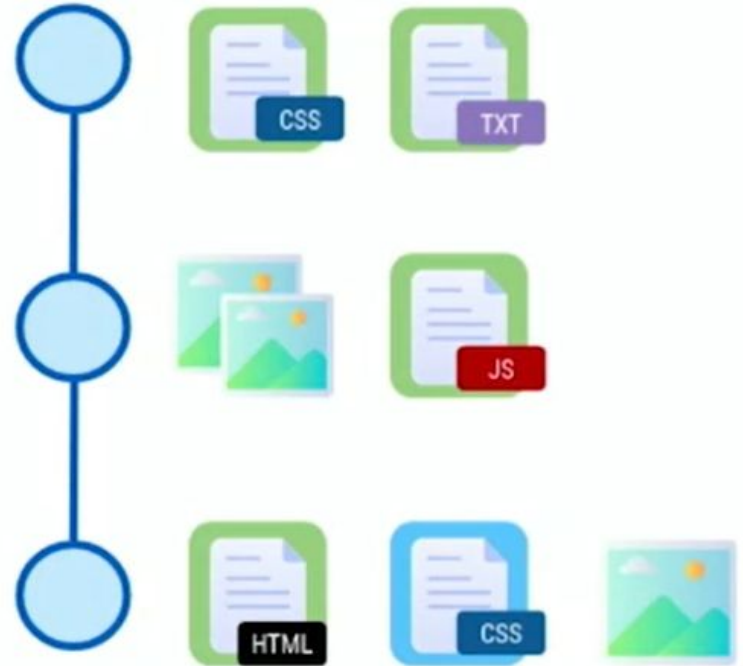
- Who made changes to given file?
- When are these changes made?



Source Code Management Version Control System

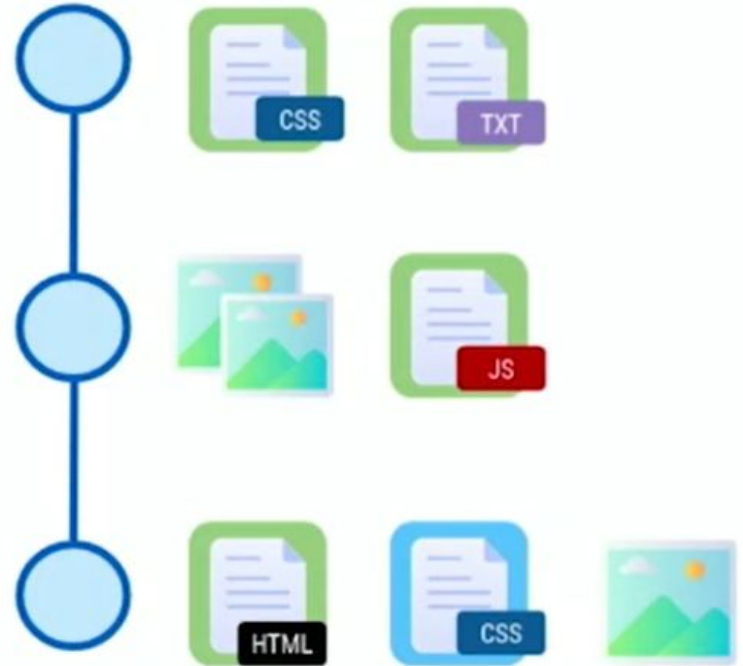
What is VCS?

A version control system records all changes made to a file or set of files, so a specific version may be called later if needed.



Source Code Management Version Control System

- ★ Each change to Project can be considered a new Version of the Project.
- ★ Version control system simplifies tracking changes to the project and allows us to switch back to any previous version.



Source Code Management

Benefits of Version Control System



- ★ Keep track of all modifications made in code.



- ★ Comparing earlier versions of code.

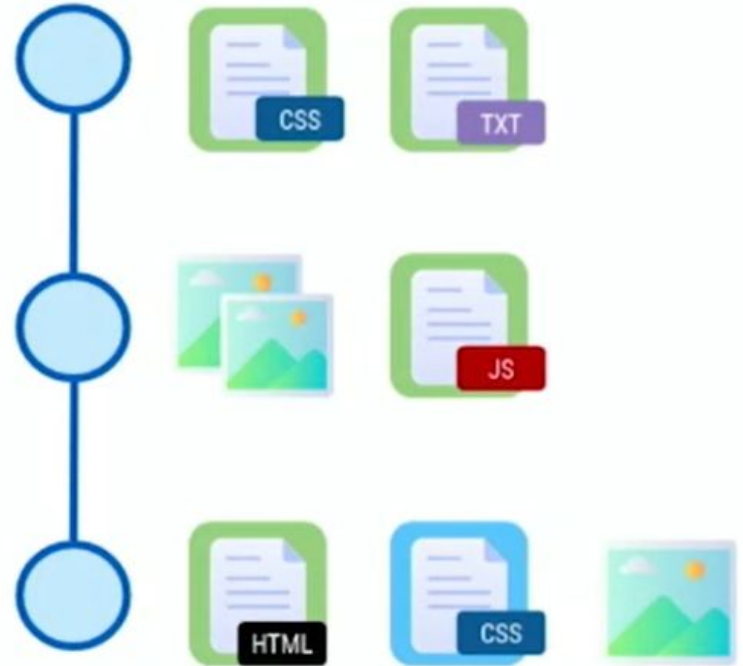


- ★ Managing and protecting Source Code.

Version Control System

There are several tools that helps us manage versions of the source code of software.

- Git
- Subversion
- Mercurial



Git

Git is a free, open-source and most widely used distributed version control system.

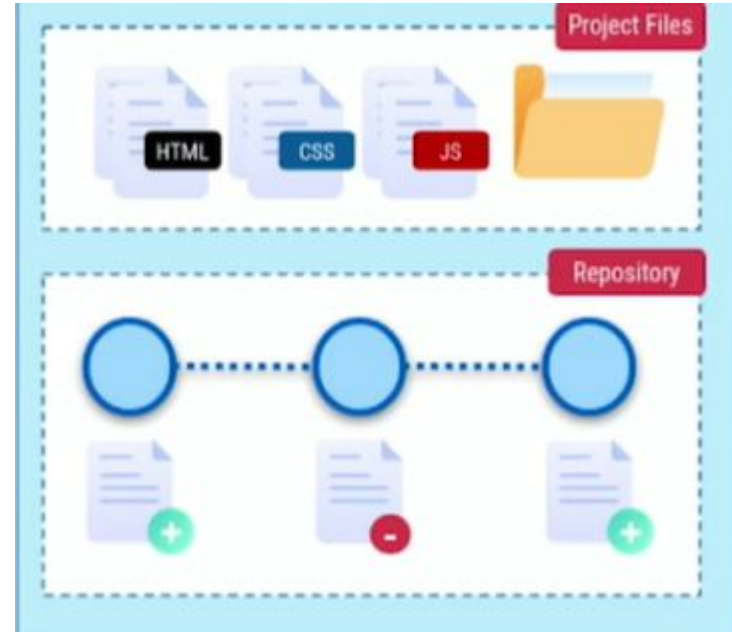
A software used for tracking changes in any set of files.



Git

Repository

A Git Repository (Repo) is like a database which maintains different versions of the project files.



Git

Snapshots

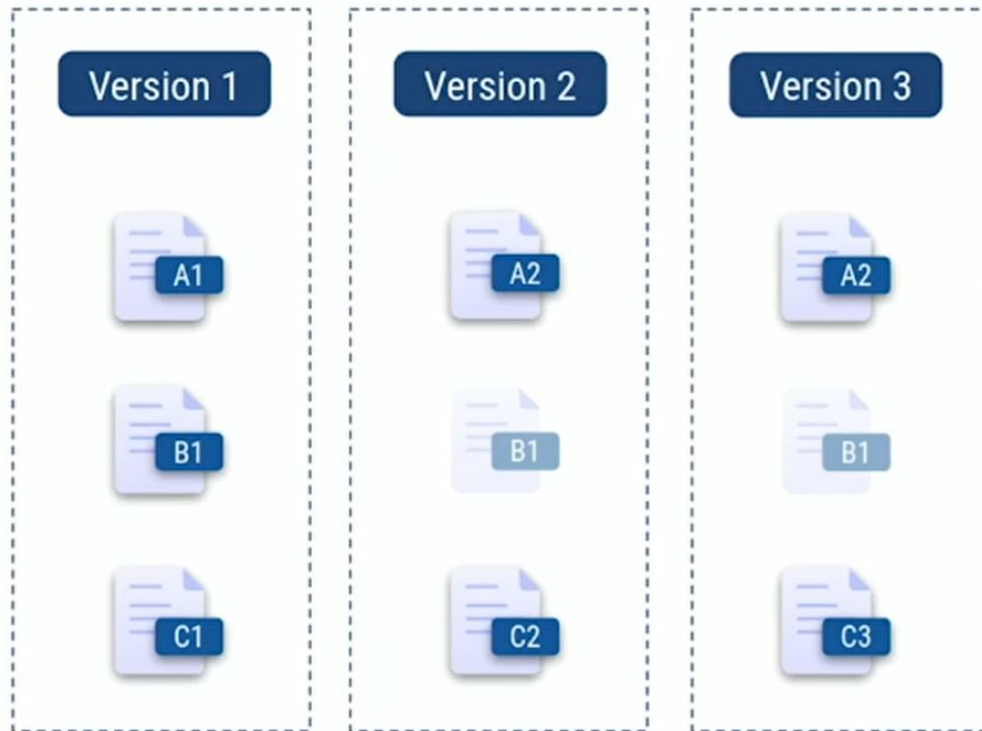


- Git allows us to take snapshots of our Project files to create versions of the Project.
- These versions or snapshots are referred to as Commits in Git terminology.

Git

Snapshots

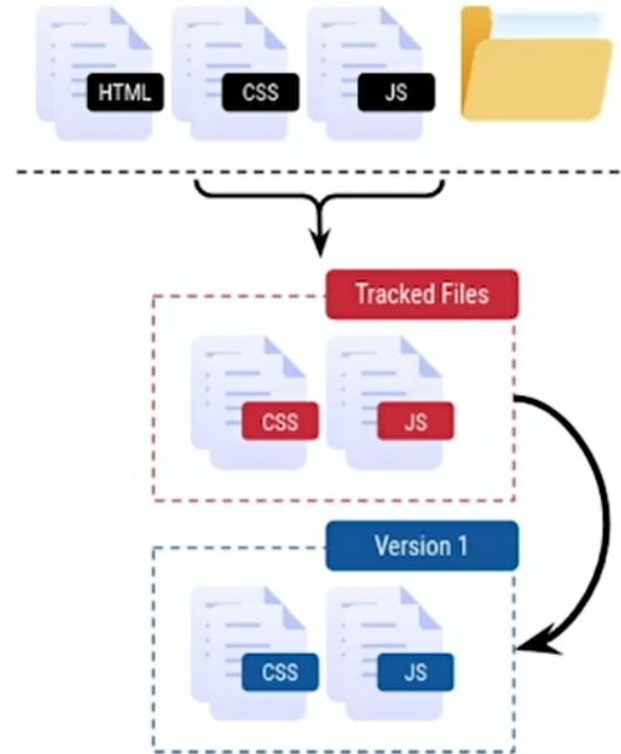
Example



Git

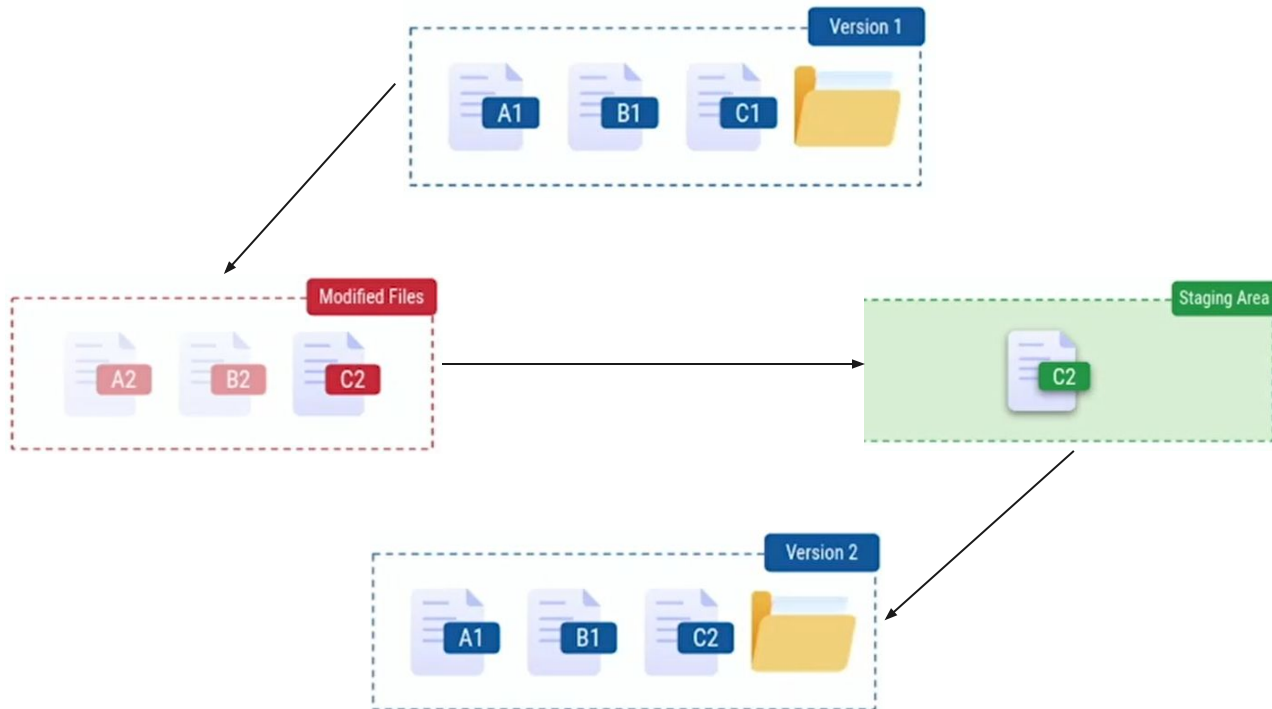
Tracking Files

- By default, Git doesn't track changes to a file and doesn't maintain versions automatically.
- We have to explicitly specify git to track file changes and save versions.



Git

Staging Area



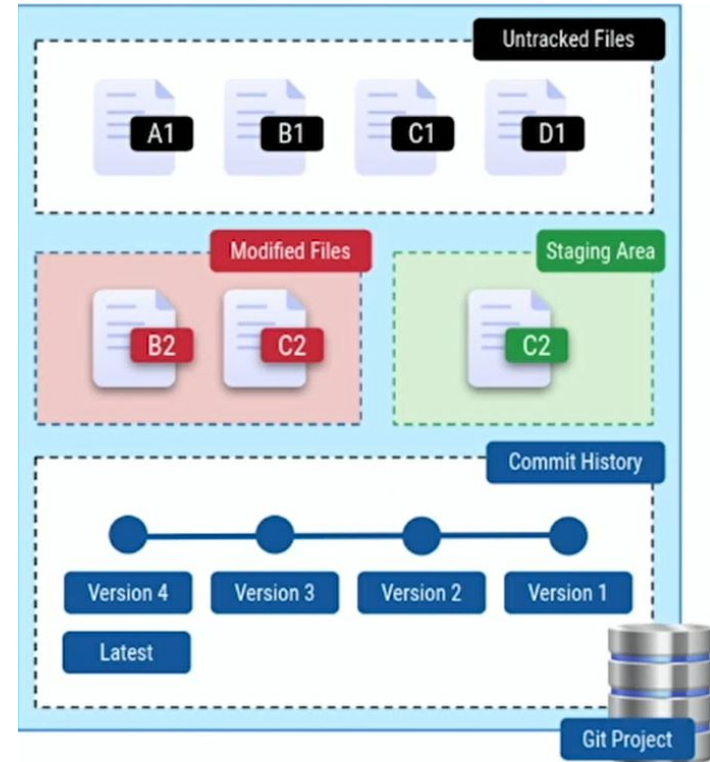
Git

Git's view of a Repository

Untracked Files : The set of files whose changes are not tracked by Git

Tracked Files : The set of Files which are watched by Git for any changes.

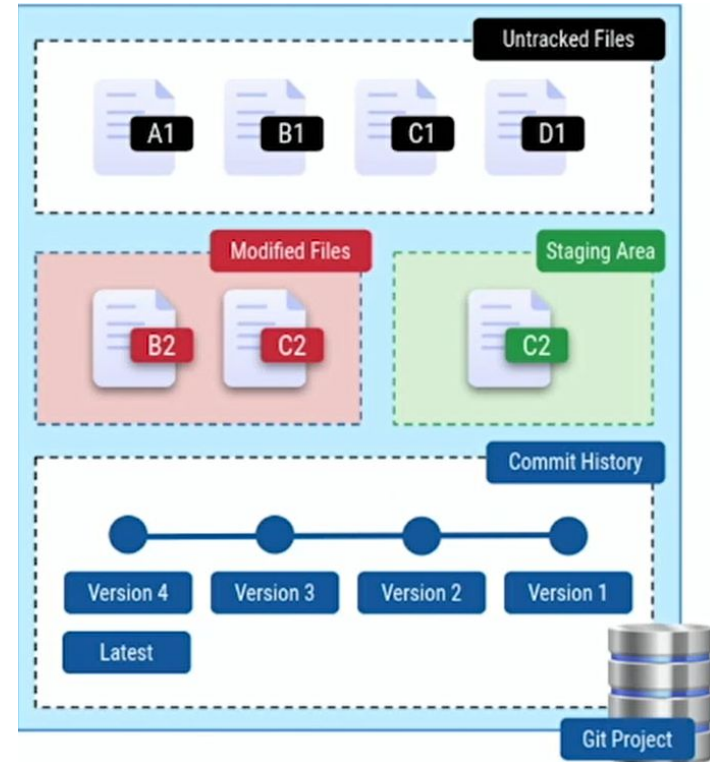
- Modified Files
- Staged Files
- Committed Files



Git

Git's view of a Repository

- **Modified Files** : These are the files which are modified after the latest snapshot.
- **Staged Files** : The set of files which are about to be committed to create new snapshot.
- **Committed Files** : These are the unmodified files which are same since last commit.



Git

Installing

On Linux

```
sudo apt install git
```

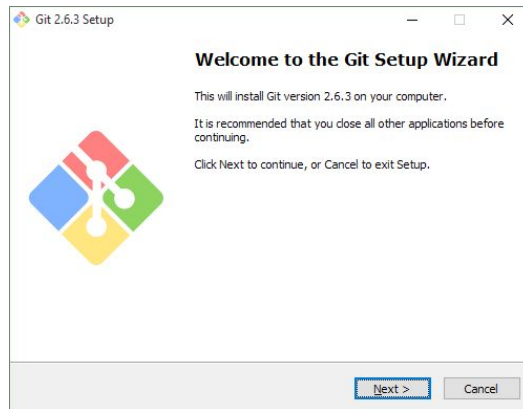
On Mac

```
sudo brew install git
```

On Windows

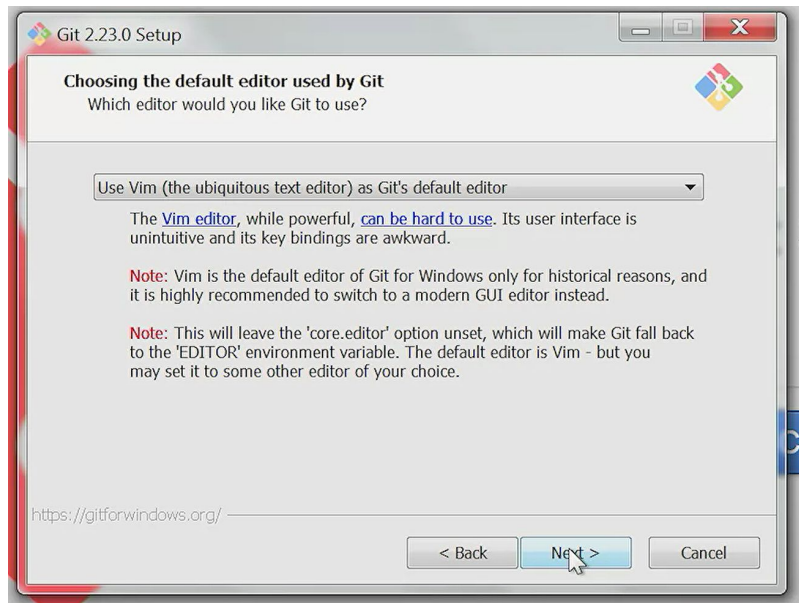
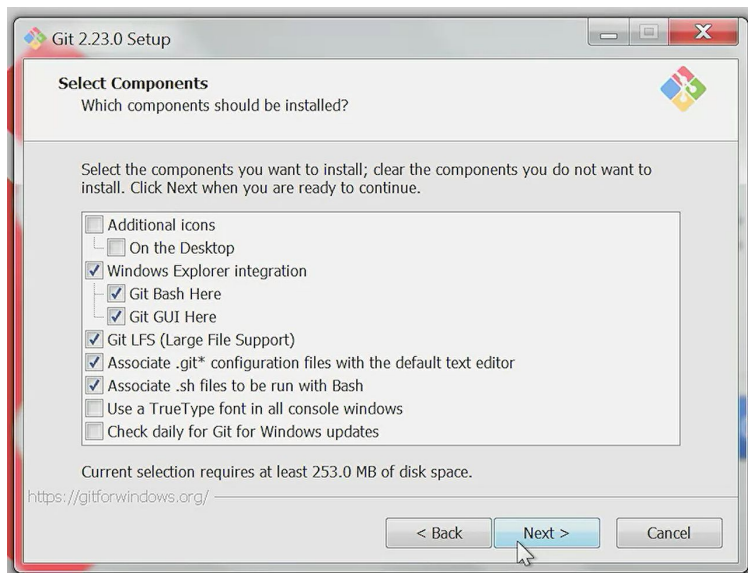


[Git - Downloading Package \(git-scm.com\)](https://git-scm.com)



Git

Installing on Windows



Git

Installing on Windows

☐ **Use Git from Git Bash only**

This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

☒ **Git from the command line and also from 3rd-party software**

(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.

☐ **Use Git and optional Unix tools from the Command Prompt**

Both Git and the optional Unix tools will be added to your PATH.

Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

<https://gitforwindows.org/>

< Back

Next >

Cancel

Git

Installing on Windows

Choosing HTTPS transport backend

Which SSL/TLS library would you like Git to use for HTTPS connections?



☒ **Use the OpenSSL library**

Server certificates will be validated using the ca-bundle.crt file.

☐ **Use the native Windows Secure Channel library**

Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.

<https://gitforwindows.org/>

Configuring the line ending conversions

How should Git treat line endings in text files?



☒ **Checkout Windows-style, commit Unix-style line endings**

Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

☐ **Checkout as-is, commit Unix-style line endings**

Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

☐ **Checkout as-is, commit as-is**

Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

<https://gitforwindows.org/>

< Back

Next >

Cancel

Git

Installing on Windows

Configuring the terminal emulator to use with Git Bash

Which terminal emulator do you want to use with your Git Bash?

☒ **Use MinTTY (the default terminal of MSYS2)**

Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `wingpty` to work in MinTTY.

☐ **Use Windows' default console window**

Git will use the default console window of Windows (`cmd.exe`), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

<https://gitforwindows.org/>

< Back

Next >

Cancel

Configuring extra options

Which features would you like to enable?

☒ **Enable file system caching**

File system data will be read in bulk and cached in memory for certain operations (`"core.fscache"` is set to `"true"`). This provides a significant performance boost.

☒ **Enable Git Credential Manager**

The [Git Credential Manager for Windows](#) provides secure Git credential storage for Windows, most notably multi-factor authentication support for Visual Studio Team Services and GitHub. (requires .NET framework v4.5.1 or later).

☐ **Enable symbolic links**

Enable [symbolic links](#) (requires the `SeCreateSymbolicLink` permission). Please note that existing repositories are unaffected by this setting.

<https://gitforwindows.org/>

< Back

Next >

Cancel

Git

The screenshot shows the Google Cloud Shell Editor web interface. At the top, there's a header with the 'Cloud Shell Editor' title and several utility icons. Below this is a menu bar with options: File, Edit, Selection, View, Go, Run, Terminal, and Help. The main content area is divided into three sections. On the left is a sidebar with icons for file operations and a 'Recent' list showing two entries: 'ravi_singh_19032 /home/ravi_singh_19032' and 'tutorial ~/tutorial'. The central part of the page has a 'Welcome to Cloud Shell' message, followed by a 'Cloud Shell Editor' heading and a brief description. Below this is a 'Start' section with links to 'Open Folder...' and 'Open Home Workspace'. To the right of the 'Start' section is a 'What's new in Version 1.14.0 (June 2021)' box containing three bullet points about new extensions and updates. Below that is a 'Learn' section with a link to 'Editor Overview'. At the bottom, a terminal window is open, displaying a welcome message and instructions on how to set the Cloud Platform project.

Cloud Shell Editor

File Edit Selection View Go Run Terminal Help

Welcome to Cloud Shell

Cloud Shell Editor

Create, build and deploy your cloud-native applications in an online editor. To get started, open your [home folder](#) or one of the options below to load your workspace.

Start

[Open Folder...](#)

[Open Home Workspace](#)

Recent

[ravi_singh_19032 /home/ravi_singh_19032](#)

[tutorial ~/tutorial](#)

What's new in Version 1.14.0 (June 2021)

- ☐ **Cloud Code extension v1.12.0** – Introducing Cloud Build support for Cloud Run and Kubernetes. Full Cloud Code release notes [here](#).
- ☐ **Golang extension v0.23.0** – Improved debugging workflows, access to Delve DAP. Full Golang release notes [here](#).
- ☐ **Cloud Shell Editor upgrade to Theia v1.14.0** – Improved plugin support, debugger updates, and multiple bug fixes. Review full Theia release notes [here](#).

[All Cloud Shell release notes](#)

Learn

[Editor Overview](#)

Learn about the Cloud Shell Editor, built using Eclipse Theia

```
Welcome to Cloud Shell! Type "help" to get started.
To set your Cloud Platform project in this session use "gcloud config set project [PROJECT_ID]"
ravi_singh_19032@cloudshell:~$ ~]
```

[Google Cloud Shell](#)

Working with Git

Initializing Repository

`git init` command is used to initialize a local repository

This command initializes an empty repository in working directory.

Syntax

```
$ git init
```

Working with Git

Working Directory

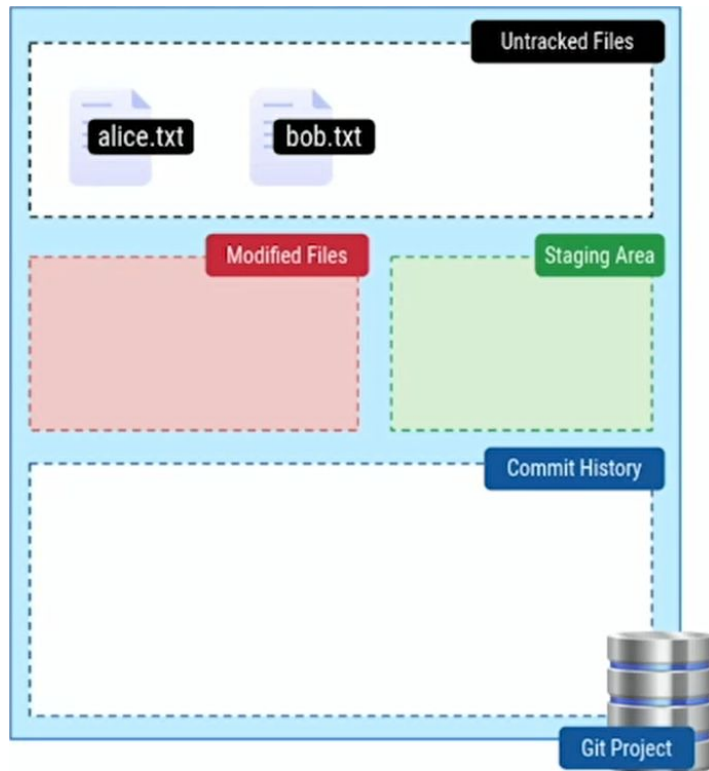
Consider creating two files **alice.txt** and **bob.txt** in tutorial folder

```
$ cd tutorial
$ touch alice.txt
$ touch bob.txt
$ ls
alice.txt
bob.txt
```

Bash

Working with Git

Git's view of Repository



Working with Git

Inspecting a Repository

Git track changes in the working directory.

`git status` command show changes in working directory.

```
$ git status
On branch master
No commits yet
Untracked files:
  alice.txt
  bob.txt
```

Bash

Working with Git

Commit

Git `commit` command takes a snapshot representing the staged changes.

```
git commit -m "<message>"
```

To skip staging area

```
git commit -a
```

Working with Git

Creating a commit

1. Add changes to staging area



2. Creating changes with commit in staging area



Working with Git

Creating a Commit

Adding changes to Staging Area

`git add` command adds the changes to the staging area.

Syntax

```
$ git add file_path
```

Bash

```
$ git add alice.txt
$ git status
Changes to be committed:
  new file:   alice.txt
Untracked files:
  bob.txt
```

Working with Git

Setting up Author Info

Configure who credits for the changes made from your device by setting the author details

```
1 git config --global user.name "Your Name"  
2  
3 git config --global user.email "youremailaddress@_Example_.com"
```

Working with Git

Committing Changes

Commit is a snapshot of the project's currently staged changes.

Here **message** provide useful information about what has changed and why.

```
ravi_singh_19032@cloudshell:~/tutorial$ git commit -m "adds alice file"

[master (root-commit) 303ab35] adds alice file
1 file changed, 3 insertions(+)
create mode 100644 alice.txt
```

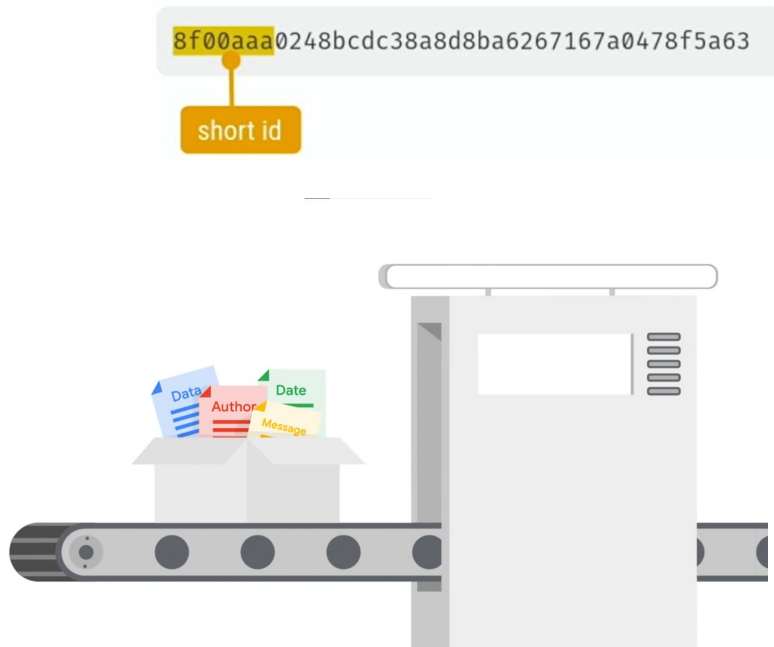
Working with Git

Commit ID

Commit IDs are unique strings(hashes) that are created whenever new commit is recorded.

Commit IDs are unique SHA-1 hashes.

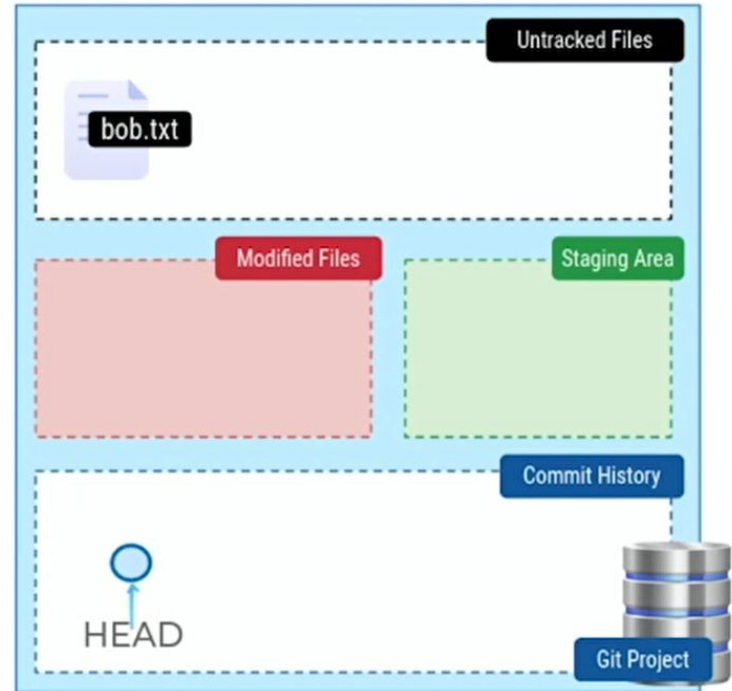
It is unique for every Commit.



Working with Git

Head

HEAD refers to the current commit



Working with Git

Commit Listing

`git log` list all the commits.

```
ssingh_raviprakash@cloudshell:~/tutorial$ git log
commit 0b6d3036b41a1efb3425f7d057742b20a83f69fd (HEAD -> master)
Author: Ravi Prakash Singh <ssingh.raviprakash@gmail.com>
Date: Sat Jun 12 10:53:08 2021 +0000

    Removed Alice File

commit 138ded3e0a8e063810481dd319d7f9bc8f68721c
Author: Ravi Prakash Singh <ssingh.raviprakash@gmail.com>
Date: Sat Jun 12 10:49:46 2021 +0000

    Adds Bob's file

commit f10af64eddc8ca0cb2aa786aeb2f545d48eca177
Author: Ravi Prakash Singh <ssingh.raviprakash@gmail.com>
Date: Sat Jun 12 04:56:54 2021 +0000

    Adds Alice file
ssingh_raviprakash@cloudshell:~/tutorial$
```

Working with Git

log Command

`git log --oneline` displays as one commit per line.

```
ssingh_raviprakash@cloudshell:~/tutorial$ git log --stat
commit 0b6d3036b41a1efb3425f7d057742b20a83f69fd (HEAD -> master)
Author: Ravi Prakash Singh <ssingh.raviprakash@gmail.com>
Date: Sat Jun 12 10:53:08 2021 +0000

    Removed Alice File

Alice.txt | 3 ---
1 file changed, 3 deletions(-)

commit 138ded3e0a8e063810481dd319d7f9bc8f68721c
Author: Ravi Prakash Singh <ssingh.raviprakash@gmail.com>
Date: Sat Jun 12 10:49:46 2021 +0000

    Adds Bob's file

Bob.txt | 2 ++
1 file changed, 2 insertions(+)
```

`git log --patch` displays files that have been modified, with location of modification.

```
ssingh_raviprakash@cloudshell:~/tutorial$ git log --oneline
0b6d303 (HEAD -> master) Removed Alice File
138ded3 Adds Bob's file
f10af64 Adds Alice file
ssingh_raviprakash@cloudshell:~/tutorial$
```

`git log --stat` displays files that have been modified. It shows summary of no of lines modified.

```
ssingh_raviprakash@cloudshell:~/tutorial$ git log -p
commit 0b6d3036b41a1efb3425f7d057742b20a83f69fd (HEAD -> master)
Author: Ravi Prakash Singh <ssingh.raviprakash@gmail.com>
Date: Sat Jun 12 10:53:08 2021 +0000

    Removed Alice File

diff --git a/Alice.txt b/Alice.txt
deleted file mode 100644
index 7037bf6..0000000
--- a/Alice.txt
+++ /dev/null
@@ -1,3 +0,0 @@
-Hi
-What about your learning
-Is it going good?
\ No newline at end of file
```

Working with Git

diff Command

`git diff` shows changes between commit, commits and working tree.

```
ssingh_raviprakash@cloudshell:~/tutorial$ git diff
diff --git a/Bob.txt b/Bob.txt
index a321560..cae6ca8 100644
--- a/Bob.txt
+++ b/Bob.txt
@@ -1,2 +1,3 @@
  Hi
-What's your name?
\ No newline at end of file
+What's your name?
+This is new line added.
\ No newline at end of file
```


Working with Git

.gitignore files

`.gitignore` files are used to tell the git tool to intentionally ignore some files in a given Git repository.

A few common examples of file patterns to exclude can be found [here](#).

Working with Git

Renaming and Deleting Files

This command is similar to Linux `rm` and `mv` commands, except the changes are also staged.

`git mv <filename>`

`git rm <filename>`

```
ssingh_raviprakash@cloudshell:~/tutorial$ git rm Alice.txt
rm 'Alice.txt'
ssingh_raviprakash@cloudshell:~/tutorial$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    Alice.txt

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

        .gitignore

ssingh_raviprakash@cloudshell:~/tutorial$
```

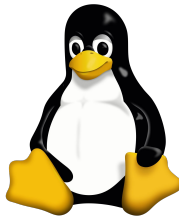
Cheat Sheet

git commit -a	<u>Stages files automatically</u>
git log -p	<u>Produces patch text</u>
git show	<u>Shows various objects</u>
git diff	<u>Is similar to the Linux `diff` command, and can show the differences in various commits</u>
git diff --staged	<u>An alias to --cached, this will show all staged files compared to the named commit</u>
git add -p	<u>Allows a user to interactively review patches to add to the current commit</u>
git mv	<u>Similar to the Linux `mv` command, this moves a file</u>
git rm	<u>Similar to the Linux `rm` command, this deletes, or removes a file</u>

Linux

Basic Commands

- pwd
- ls
- cd
- mkdir & rmdir
- rm
- touch
- cp
- mv
- echo
- cat
- nano



Linux

Basic Commands

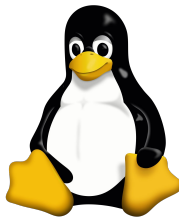
pwd- This command lets you know the current working directory

```
leomajor@ROG-Strix:~$ pwd  
/home/leomajor  
leomajor@ROG-Strix:~$ |
```

ls - list all the files in current directory

```
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents$ ls  
'My Music' 'My Pictures' 'My Videos' Untitled.ipynb
```

You can use -a flag to list hidden files and -l flag to get more information about files.



Linux

Basic Commands

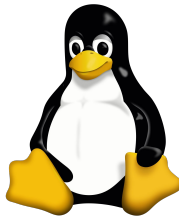
cd - This command is used to change directory

```
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents$ cd My\ Music
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/My Music$
```

mkdir - Used to create a folder

rmdir - Delete a Directory

```
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom$ mkdir Git
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom$ ls
Git
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom$ rmdir Git
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom$ ls
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom$
```



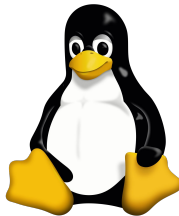
Linux

Basic Commands

rm - Used to delete files and directories.

You can use -r flag to delete a directory

```
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom/Git$ ls
a.txt b.txt c.txt
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom/Git$ rm a.txt
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom/Git$ ls
b.txt c.txt
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom/Git$ cd ..
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom$ rm -r Git
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom$ ls
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom$ |
```



Linux

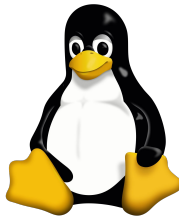
Basic Commands

touch - Used to create a file.

```
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom$ touch Hello.txt  
leomajor@ROG-Strix:/mnt/c/Users/ssing/Documents/Classroom$ ls  
Hello.txt
```

cp - Used to copy files

```
$ cp Hello.txt "New Folder"
```



Linux

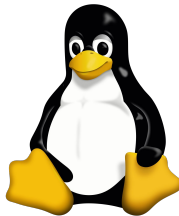
Basic Commands

echo - Used to move some data to text files or simply display on terminal.

```
$ echo "Hello World"  
Hello World
```

```
$ echo "This line will be entered in file" >> new.txt  
$ cat new.txt  
This line will be entered in file
```

cat - Used to display content of a file



Key Takeaways

- ❑ What is Version Control System?
- ❑ Git
- ❑ Installing Git
- ❑ Using Git
 - ❑ First Step with Git
 - ❑ Basic Workflow
 - ❑ Tracking Files and Committing
- ❑ Git Interaction
 - ❑ Using Git Locally
 - ❑ Staging Area
 - ❑ Info about changes (**log** and **diff** command)

