Git Intro

Introduction to Git:
Git is a powerful and widely-used distributed version control system that enables developers to efficiently manage and track changes in their codebase. Originally created by Linus Torvalds in 2005, Git has become an essential tool for software development and collaboration.

Key Points:

1. **Version Control**: Git allows developers to keep track of changes made to their code over time, providing a history of modifications, who made them, and why.
2. **Distributed System**: Unlike centralized version control systems, Git is distributed, meaning every developer has a complete copy of the project's history on their local machine. This fosters collaboration and enables offline work.
3. **Branching**: Git makes it easy to create branches, enabling developers to work on isolated features or bug fixes without affecting the main codebase. This promotes parallel development and experimentation.
4. **Collaboration**: Git facilitates collaborative coding by providing tools for merging and resolving conflicts when multiple developers work on the same project simultaneously.
5. **Snapshots**, Not Changes: Git doesn't store just the changes between versions; it stores entire snapshots of the project at each commit. This ensures the integrity and reliability of your codebase.
6. **Open Source**: Git is open source, with a thriving community and a wealth of resources available for users. Platforms like GitHub, GitLab, and Bitbucket offer hosting services for Git repositories.
7. **Command Line and GUI**: Git can be used via a command-line interface (CLI) for advanced users or through user-friendly graphical user interfaces (GUIs) for those who prefer a visual approach.

Installation
https://git-scm.com/

Configuration
After installation, you should configure your name and email in Git. This information is used to identify your commits.

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

Initializing a Repository
To start tracking changes in a project, navigate to the project's root directory and run:

```
git init
```

Cloning a Repository:
To clone an existing Git repository from a remote server (like GitHub or GitLab) to your local machine, use

```
git clone <repository_url>
```

Staging and Committing:
Git uses a two-step process to save changes:
Staging: You stage changes you want to include in the next commit using:

```
git add <file(s)>
```

Committing: Once changes are staged, you commit them with a message explaining the changes:

```
git commit -m "Your commit message"
```

Checking Status:

To see the status of your working directory (untracked files, modified files, etc.), use:

```
git status
```

Viewing Commit History:

To view the commit history of the repository, use:

```
git log
```

Branching:

Git allows you to work on different features or versions simultaneously using branches. To create a new branch, use:

```
git branch <branch_name>
```

To switch to a branch, use:

```
git checkout <branch_name>
```

To create and switch to a new branch in one command, use:

```
git checkout -b <branch_name>
```

Merging:

After working on a branch, you can merge it back into the main branch (e.g., "master") using:

```
git merge <branch_name>
```

Commonly used Git commands:

To track file using git :	# git init
To add all the files in staging area:	# git add --a (or) # git add .
To commit the code:	# git commit -m "Commit message"
To check status:	# git status
To list logs:	# git log
To unlink a folder from GIT:	# rm -rf .git

Git Command

1. git init:
- Description: Initializes a new Git repository in the current directory.
- Usage: Run this command in the root directory of your project to start version control.

2. git clone <repository_url>:
- Description: Creates a copy of a remote repository on your local machine.
- Usage: Replace <repository_url> with the URL of the remote repository you want to clone.

3. git add <file(s)>:
- Description: Stages changes for the next commit.
- Usage: You can specify specific files to stage or use git add . to stage all changes.

4. git commit -m "Your commit message":
- Description: Commits the staged changes with a descriptive message.
- Usage: Replace "Your commit message" with a brief explanation of the changes.

5. git status:
- Description: Displays the status of your working directory, including untracked files and modified files.
- Usage: Run this command to see the current state of your repository.

6. git log:
- Description: Shows a log of all commits in the current branch.
- Usage: Use this command to view the commit history, including commit messages, authors, and timestamps.

7. git branch <branch_name>:
- Description: Creates a new branch with the specified name.
- Usage: Replace <branch_name> with the desired name for your new branch.

8. git checkout <branch_name>:
- Description: Switches to the specified branch.
- Usage: Use this command to move between branches in your repository.

9. git checkout -b <branch_name>:
- Description: Creates a new branch and switches to it in one command.
- Usage: Replace <branch_name> with the desired name for your new branch.

10. git merge <branch_name>:
- Description: Combines the changes from one branch into another.
- Usage: Use this command while on the target branch to merge changes from <branch_name> into the current branch.

11. git remote add <remote_name> <remote_url>:
- Description: Adds a remote repository to your local Git configuration.
- Usage: Replace <remote_name> with a name for the remote (e.g., "origin") and <remote_url> with the URL of the remote repository.

12. git fetch <remote_name>:
- Description: Retrieves changes from a remote repository without merging them.
- Usage: Use this command to see what changes are available on the remote repository.

13. git pull <remote_name> <branch_name>:
- Description: Fetches changes from a remote repository and merges them into the current branch.
- Usage: Replace <remote_name> with the remote you want to pull from and <branch_name> with the branch to pull.

14. git push <remote_name> <branch_name>:
- Description: Pushes your local commits to a remote repository.
- Usage: Replace <remote_name> with the remote repository and <branch_name> with the branch to push.

Git Commands - step by step (working)

Checking GIT version:
git --version

Creating a directory
mkdir gitdemo2

Switching the dir:
cd gitdemo2/

Listing the current data:
ls

Creating a notepad file:
notepad demo.txt

Checking the current status:
git status

Initializing the GIT:
git init

Setting up the username:
git config --global user.name "Jeetu"

Setting up the email ID:
git config --global user.email "jitendrastomar5593@gmail.com"

Checking the default config: user.
git config --list

Initializing the GIT:
git init

Checking the status now:
git status

Adding the file(s) to the stagging
git add . //for multiple files
git add <file1> <file2>...

Checking the status again:
git status

Adding the remote GIT repo
git remote add origin https://github.com/jitendrastomar5593/gitdemo2.git

Setting up the main branch

git branch -M main

Committing the code
git commit -m "1st commit"

Pushing the data to git to main branch
git push -u origin main //OR
git push origin main

Checking the log:
git log

Creating a new file:
notepad 2ndfile.txt

Adding this file to stagging
git add 2ndfile.txt

Committing this code:
git commit -m "2nd commit"

Pushing the code GIT repo:
git push -u origin main

Checking the log again:
git log

Checking the remote:
git remote -v

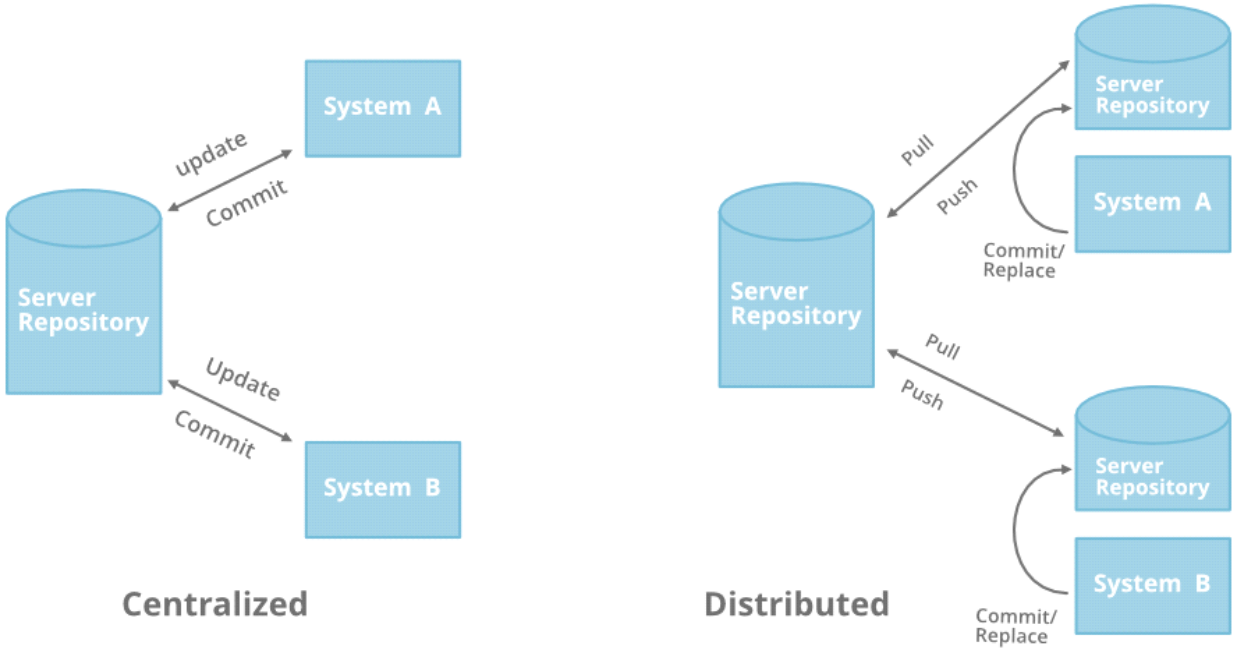
Changing the remote URL:
git remote set-url origin https://github.com/jitendrastomar5593/gitdemo2.git

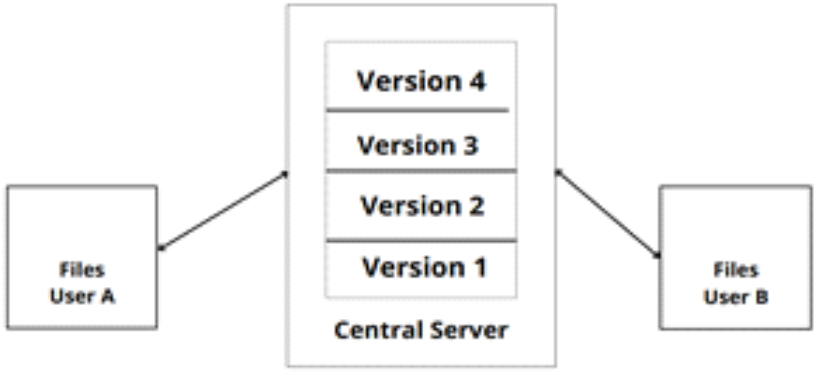
Checking the remote:
git remote -v

CVCS vs DVCS

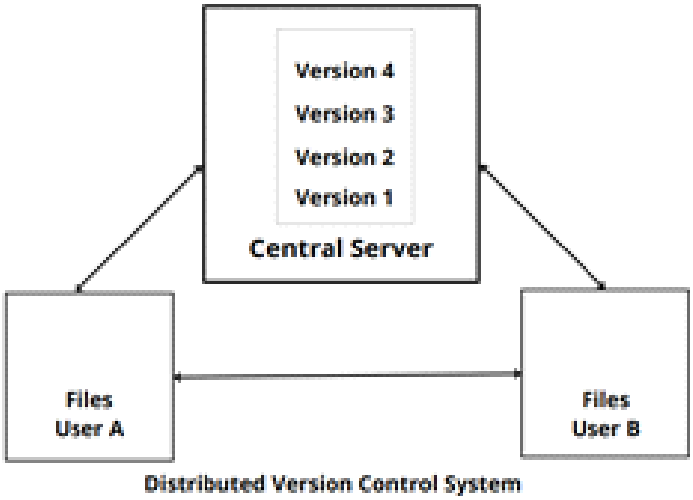
Centralized Version Control System	Distributed Version Control System
Repository Structure: • There is a central repository that stores the entire version history of the project. • Users typically have a working copy of the latest version of the files on their local machines. **Network Dependency:** • Requires a constant connection to the central server for most operations. • If the central server goes down or there are network issues, users may face difficulties in performing version control operations. **Branching and Merging:** • Branching and merging can be more challenging compared to DVCS. • Typically, branching involves creating a copy of the codebase on the central server, and merging often requires more careful coordination. **Speed:** • Speed of operations can be slower, especially when the central repository is large or when there are many users accessing it simultaneously. **History:** • The entire history of the project is stored centrally.	**Repository Structure:** • Each user has a complete copy of the repository, including the entire version history, on their local machine. • This allows users to work independently without requiring constant access to a central server. **Network Dependency:** • Users can work offline and commit changes to their local repository. • Synchronization with the central repository is required only when sharing changes with others. **Branching and Merging:** • Branching and merging are typically easier and more flexible in DVCS. • Each user can have their own branches locally, and merging is often less complex. **Speed:** • Many operations can be faster in a DVCS, as they are performed locally without relying on network access.

	History: • Each local copy contains the full history of the project, allowing users to access historical versions without needing a network connection.





Centralized Version Control System



Distributed Version Control System

Git as a Distributed Version Control System:
Git, specifically, is a widely used DVCS that offers several advantages:

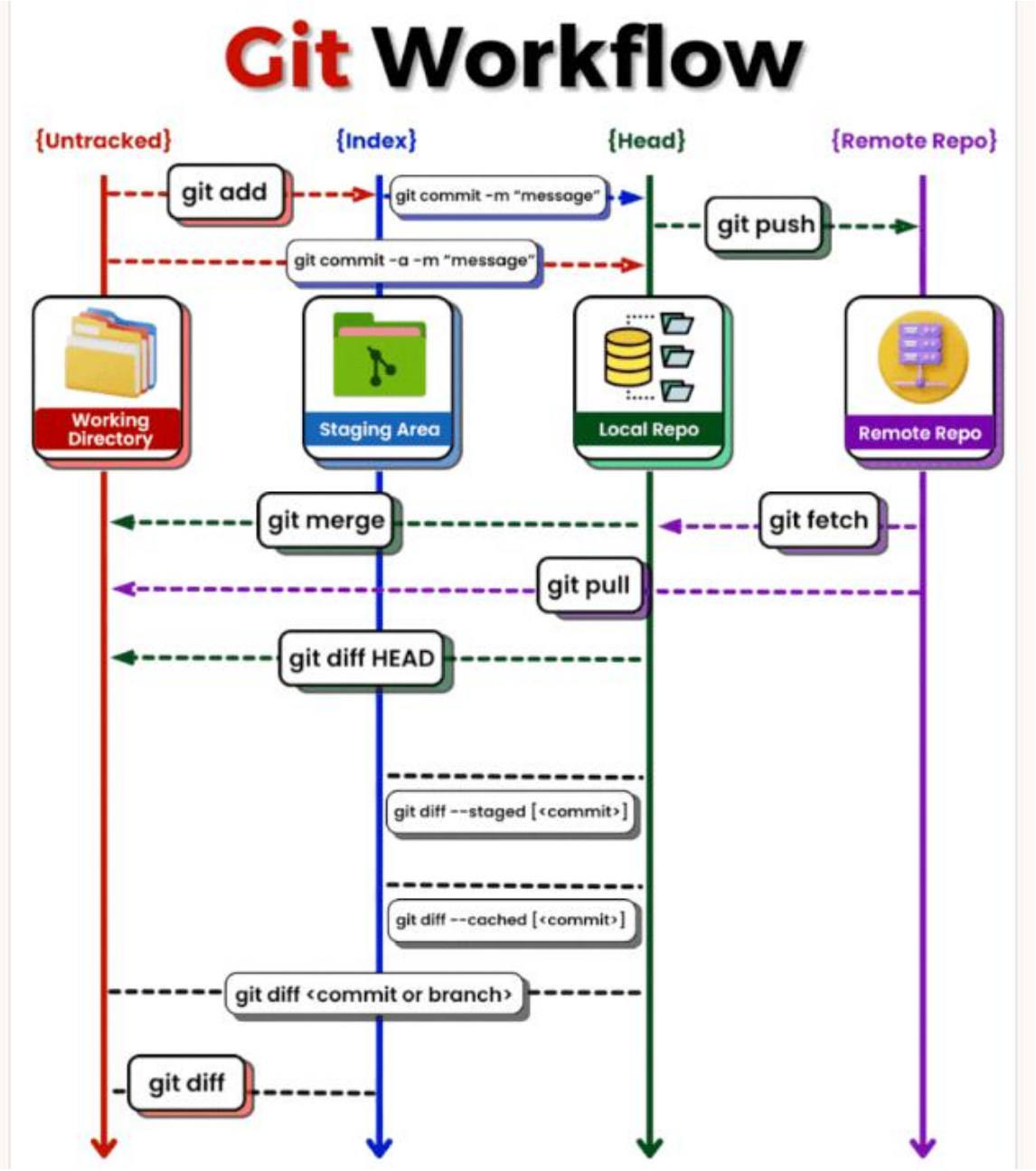
Branching and Merging:
- Git is known for its powerful and flexible branching and merging capabilities.
- Branches are lightweight, making it easy to create and switch between them.

Performance:
- Git is designed to be fast, and many operations can be performed locally without the need for a network connection.

Offline Work:
- Users can work offline, committing changes to their local repository, and then push those changes to a central repository when a network connection is available.

Full History:
- Each clone of a Git repository contains the complete history, making it robust and allowing users to work independently.

Working with Git Repositories



Repositories in Git are of two types:

- **Local Repository**:
  - Git allows the users to perform work on a project from all over the world because of its Distributive feature.
  - This can be done by cloning the content from the Central repository stored in the GitHub on the user's local machine.
  - This local copy is used to perform operations and test them on the local machine before adding them to the central repository.
- **Remote Repository**:
  - Git allows the users to sync their copy of the local repository to other repositories present over the internet.
  - This can be done to avoid performing a similar operation by multiple developers.
  - Each repository in Git can be addressed by a shortcut called remote.

Bare repositories:

- A bare repository is a remote repository that can interact with other repositories but

there is no operation performed on this repository.

- There is no Working Tree for this repository because of the same.
- A bare repository in Git can be created on the local machine of the user with the use of the following command:
    - # git init --bare
- A bare repository is always created with a .git extension.
- This is used to store all the changes, commits, refs, etc. that are being performed on the repository. It is usually a hidden directory.
- A Git repository can also be converted to a bare repository but that is more of a manual process. Git doesn't officially provide the support to do the same.

Renaming a Remote Repository:

git remote rename <old-repo-name> <new-repo-name>

Git VS GitHub

S.No.	Git	GitHub
1	Git is a software.	GitHub is a service.
2	Git is a command-line tool	GitHub is a graphical user interface
3	Git is installed locally on the system	GitHub is hosted on the web
4	Git is maintained by linux.	GitHub is maintained by Microsoft.
5	Git is focused on version control and code sharing.	GitHub is focused on centralized source code hosting.
6	Git is a version control system to manage source code history.	GitHub is a hosting service for Git repositories.
7	Git was first released in 2005.	GitHub was launched in 2008.
8	Git has no user management feature.	GitHub has a built-in user management feature.
9	Git is open-source licensed.	GitHub includes a free-tier and pay-for-use tier.
10	Git has minimal external tool configuration.	GitHub has an active marketplace for tool integration.
11	Git provides a Desktop interface named Git Gui.	GitHub provides a Desktop interface named GitHub Desktop.
12	Git competes with CVS, Azure DevOps Server, Subversion, Mercurial, etc.	GitHub competes with GitLab, Bit Bucket, AWS Code Commit, etc.

Start from: https://www.geeksforgeeks.org/working-on-git-bash/?ref=lbp

1. Introduction to Git:

- Git is a Version Control System (VCS).
- VCS meaning, it can contain several versions of a file.
- VCS helps in tracking of files.
- Recovery of files is easy.
- "Issues" can be introduced and also who introduced it, could be tracked.
- Rollback of code to previous state.
- It effectively tracks changes to source code, enabling effortless branching, merging, and versioning.

Starting with

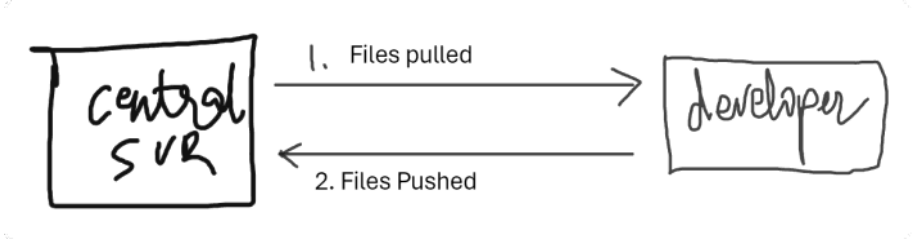
Local VCS:

- Pros: can track files & rollback.
- Cons: if HDD is crashed/lost/damaged, everything is LOST.

Centralized VCS:

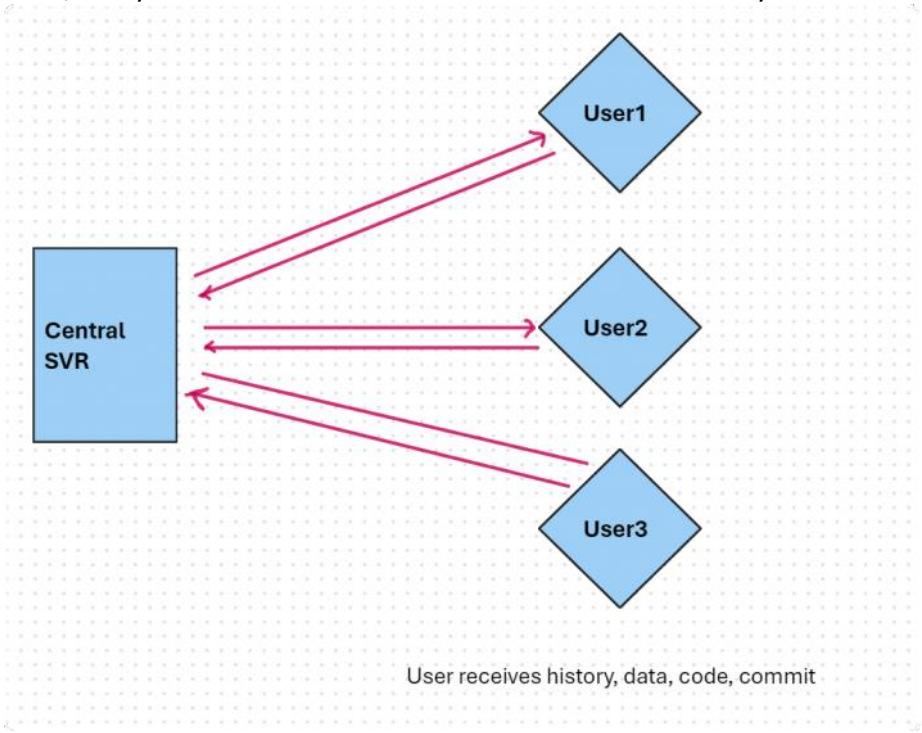
- Strong the code to a central location.
- Resolves issues of local VCS.

-



Distributed VCS:

- Here, every user will have his own full-set of data with history and commits.



Working with Git:

- When a folder is initialized with Git, it becomes a repository—a special location where Git logs all changes made to a hidden folder.

- In that folder, each time you change, add, or remove a file, Git takes note of the change and marks the file as “modified.”
- You can choose which modified files you want to save by staging them, so don’t worry.
- Consider staging as getting the changes ready for a particular snapshot that you want to keep.
- Once the staged changes are to your satisfaction, commit them, and Git will keep a permanent copy of those files in its history.
- Git is great because it maintains a complete record of each commit you make, allowing you to see.

What is GitHub?

- GitHub is a hosting service for Git repositories and if you have a project hosted on GitHub.
- GitHub allows you to store your repo on their platform.
- It is also comes with GitHub, ability to collaborate with other developers from any location.
- It's developed by Linus B. Torwalds.

Developers can review project history to find out:

- Which changes were made?
- Who made the changes?
- When were the changes made?
- Why were changes needed?

Characteristics of Git:

- Faster
- Reliable
- Captures snapshots
- Contains all history in a single file (.git)
- Almost every operation is locally executed.
- Git has integrity.
- Git generally adds the data.

Git Installation:

- Git Command Line Tool.
- Git bash (terminal program).

2. Git - Environment Setup & init commands

Installation:
Step 1. Go to git-scm.com/downloads.
Step 2. If you are using Linux distribution: sudo apt-get install git-core
Step 3. enter “git --version”.

Setting Git Environment:
Setting username:
git config --global user.name "Jitendra"

Setting email id:
git config --global user.email "jitendrastomar5593@gmail.com"

To set the editor:
git config --global core.editor vim

List of Git config:
git config --list

To list username:
git config user.name

To list email address:
git config user.email

Initializing a local repository:
git init

Checking status of the repository:
git status

Adding files to the repository:
git add

Committing changes:
git commit -m "Your commit message"

Accessing log of commits:
git log

Git clone command:
git clone "Remote_repo_URL"
git clone "https://github.com/jitendrastomar5593/PersonalDO.git"

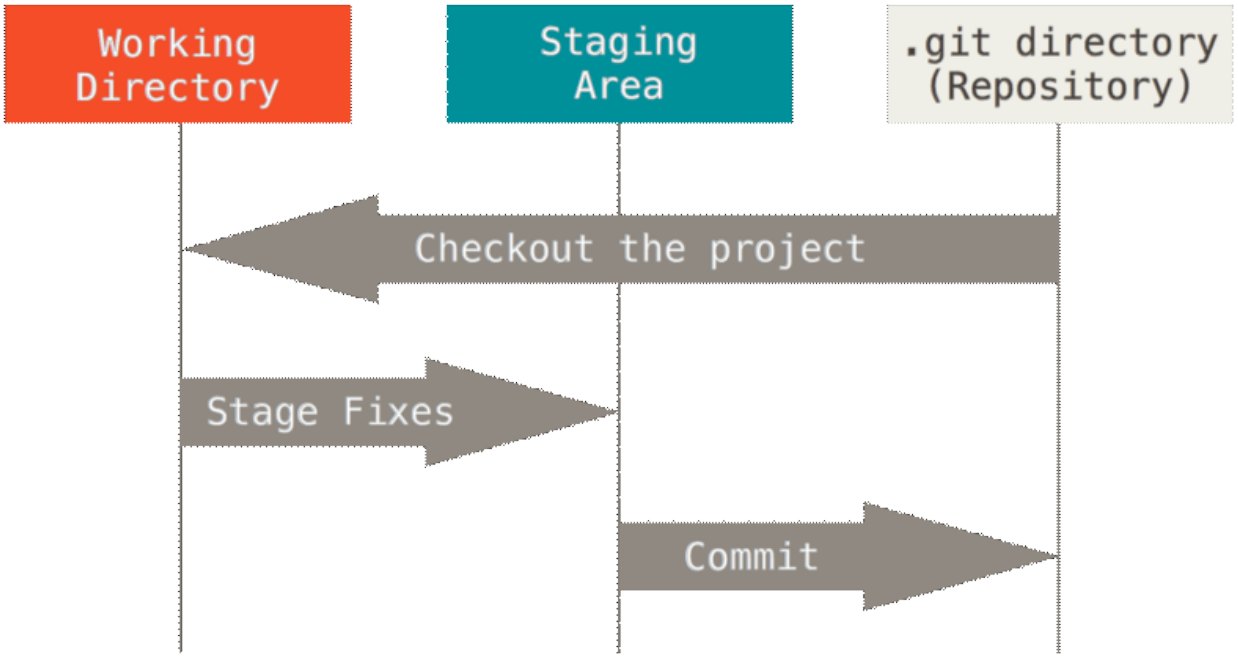
Parallel development commands
This command allows to create a branch (exact copy) for the project.
git branch branch_name

This command allows to switch from one branch to another.
git checkout branch_name

This command allows to merge a code of 2 branches in one branch.
git merge branch_name

3. Three stage architecture of Git

Sunday, November 26, 2023 9:33 PM



Working directory: your local computer working folder.
Staging area: here we keep files that are needed to be committed to final repo.
Git Directory: it's the actual repo. Container ".git" file in it.

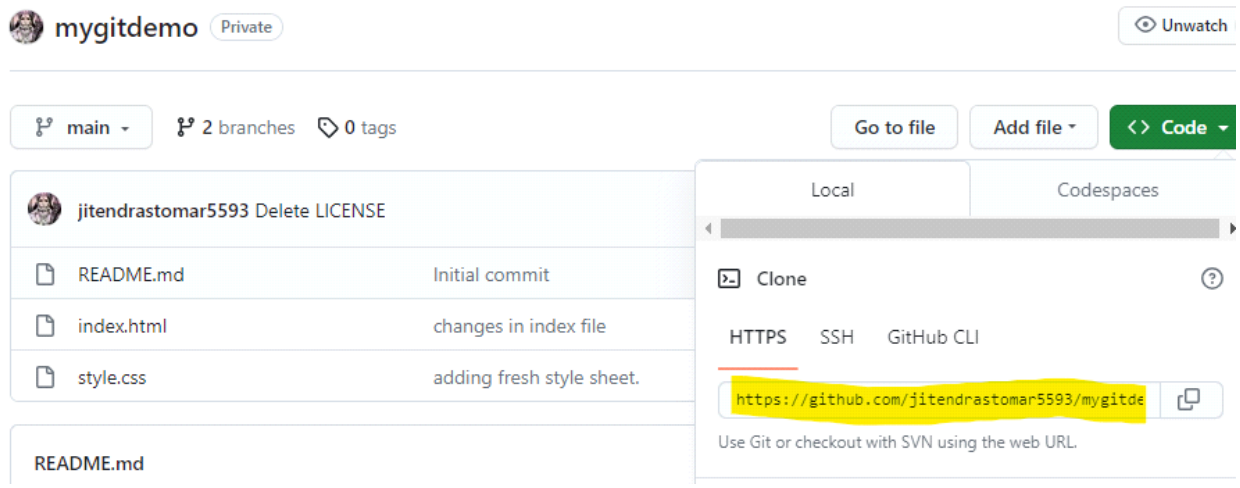
The basic Git workflow goes something like this:

1. You modify files in your working tree.
2. You selectively stage just those changes you want to be part of your next commit, which adds only those changes to the staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

4. Cloning a remote repo from GitHub:

Sunday, November 26, 2023 9:46 PM

Go to GitHub which you want to clone & copy the URL:

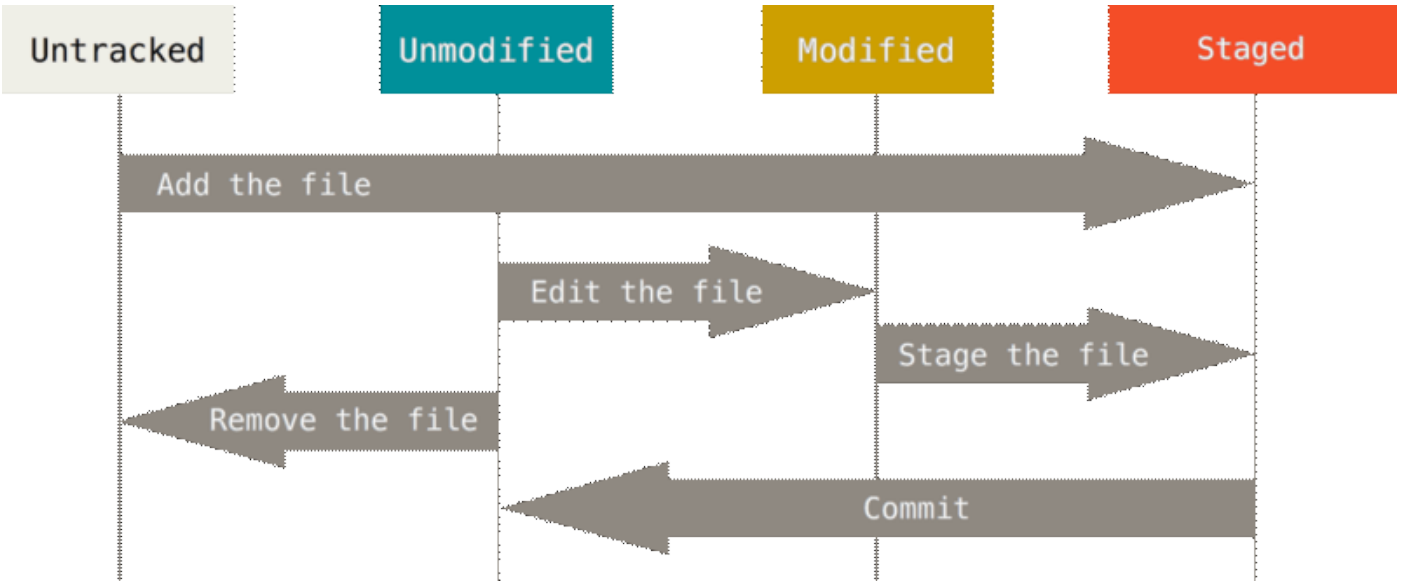


To create a clone:
git clone <URL>
git clone https://github.com/jitendrastomar5593/mygitdemo.git

To rename the repo name, while cloning:
git clone <URL> <new-name>
git clone https://github.com/jitendrastomar5593/mygitdemo.git mygit

```
admin@WorkPC MINGW64 ~/git-demo
$ ls

admin@WorkPC MINGW64 ~/git-demo
$ git clone https://github.com/jitendrastomar5593/mygitdemo.git mygit
Cloning into 'mygit'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (16/16), done.
Receiving objects: 100% (19/19), 15.79 KiB | 2.25 MiB/s, done.Receiving objects:  78% (15/19)

Resolving deltas: 100% (5/5), done.

admin@WorkPC MINGW64 ~/git-demo
$ ls
mygit/
```

5. File Status lifecycle:



- **Untracked**: in this stage, file aren't add for the staging yet. Could be checked using "git status" cmd.
- **Unmodified**: this stage could be achieved by using cmd "git add --a". This means that file are now added to the staging are with all modification required and ready to be committed.
- **Modified**: this stage come when there's a modification in a staged (by using cmd: "git add --a") file.
- **Staged**: this stage is achieved when all file are in unmodified state. If a file/directory is in this stage, it means that the file is ready to committed now.

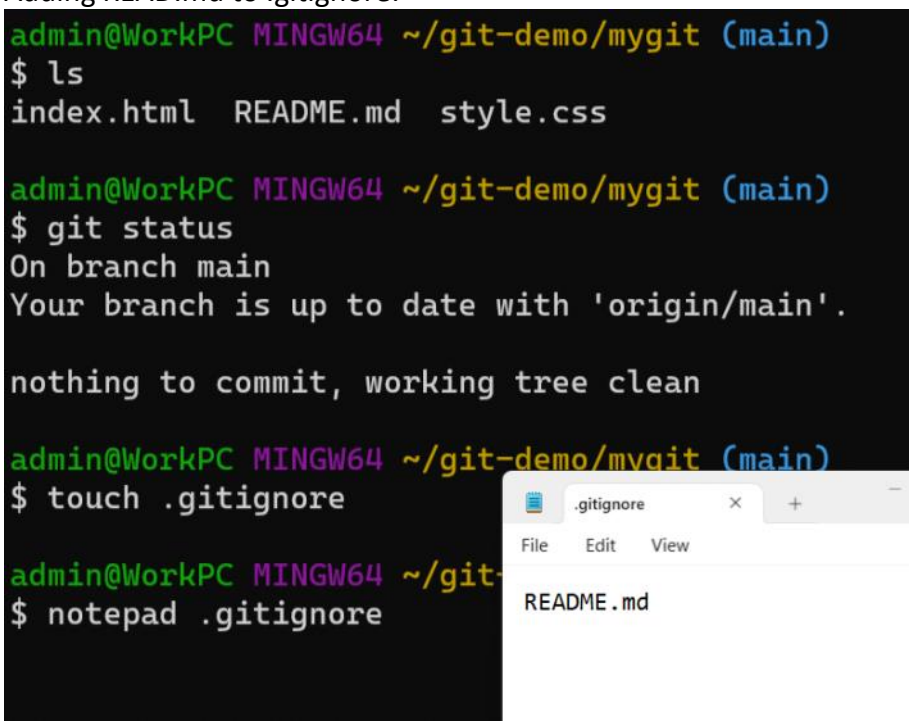
6. Git ignore file

- This file helps in ignoring all the other unwanted files in the code directory to be staged.
- This is achieved by creating a file named ".gitignore".
- All the files that must be left untracked & un-staged, must have their names this file entered manually.

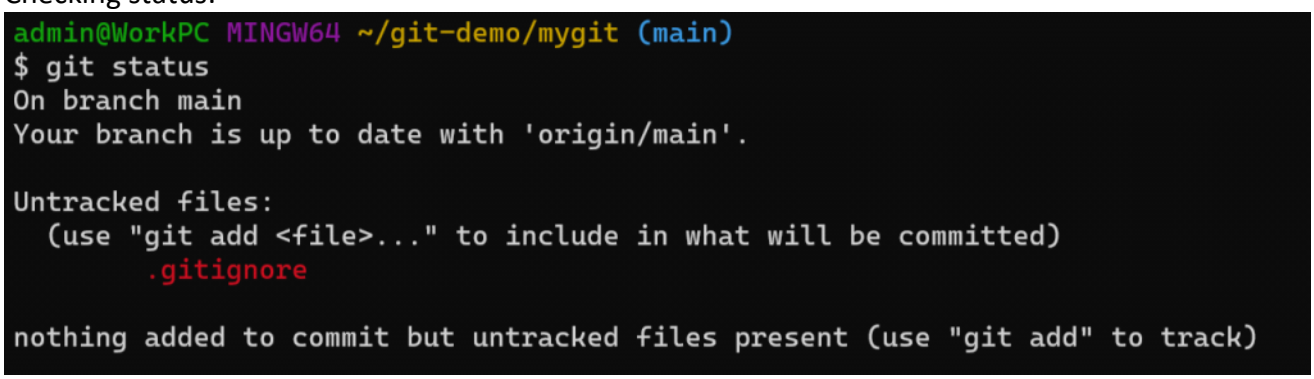
Basic commands related to .gitignore:
touch .gitignore
touch error.log // creating a file that must remain untracked.
git status // shows .gitignore & error.log as untracked.

Open .gitignore & add error.log name to it & check status again.
git status
git add .
git commit -m "ignore added"

Adding READ.md to .gitignore:



Checking status:



Adding file(s) to stage:

```
admin@WorkPC MINGW64 ~/git-demo/mygit (main)
$ git add --a

admin@WorkPC MINGW64 ~/git-demo/mygit (main)
$ git commit -m "added ignore file"
[main 356b02b] added ignore file
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore

admin@WorkPC MINGW64 ~/git-demo/mygit (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Pushing code to the repo:

```
admin@WorkPC MINGW64 ~/git-demo/mygit (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 357 bytes | 357.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/jitendrastomar5593/mygitdemo.git
   a8da5eb..356b02b  main -> main
```

7. Git diff

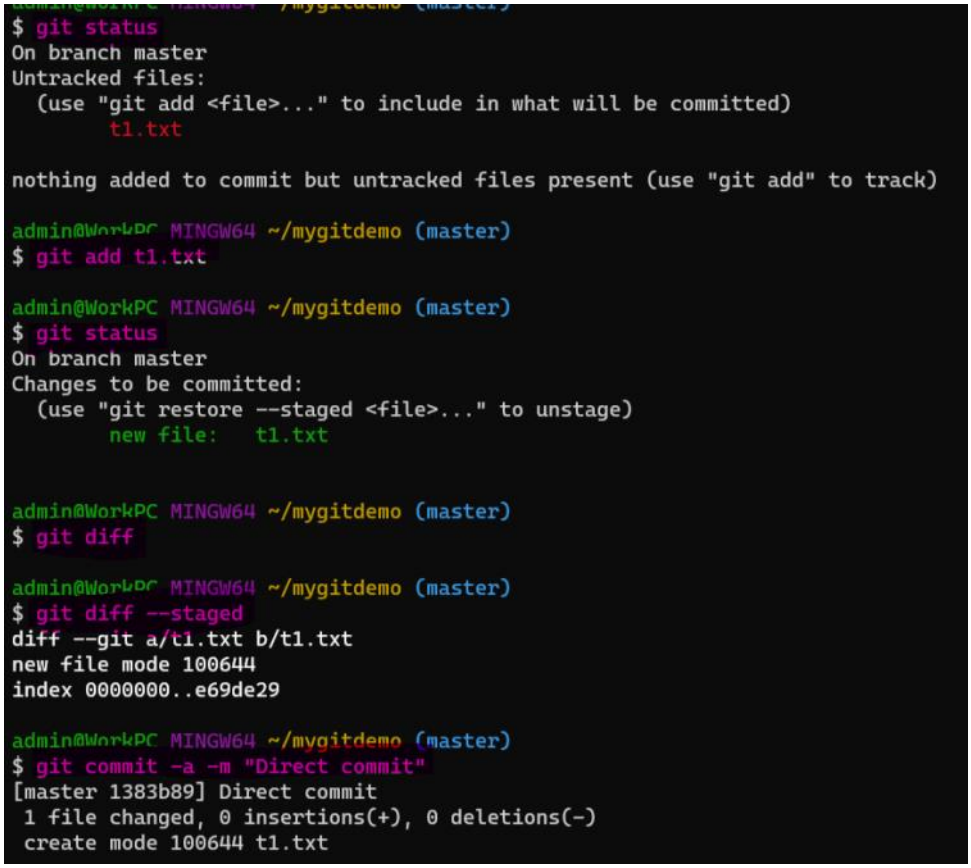
- It compares "working area" and "staging area".
- Shows, what was present earlier & what's present now.

Command:
git diff

If you want to see staging area vs old commit:
git diff --staged

Skipping the staging area:
git commit -a -m "Direct commit"

- This will commit all the 'tracked' & 'modified' files but 'will not' touch any 'untracked' files.



Renaming git files:
git mv <old-name> <new-name>
git mv t2.txt new_t2.txt //ensure that the file t2.txt is already getting tracked, else it will give fatal error.

Deleting tracked file:
rm <filename.txt>

To untrack all .gitignore files.

```
# git rm --cached <file-name>
# git status                //shows deleted file.
```

8. Git Log

To view logs
git log

```
$ git log
commit 07828e77a4e2be1fdd74647ae517ffa0e562a470 (HEAD -> master)
Author: Jeetu <jitendrastomar5593@gmail.com>
Date:   Mon Nov 27 13:28:29 2023 +0530

    added .gitignore

commit 20b81f7a5ff53412c274a9b9a16ec69a6e76c94e
Author: Jeetu <jitendrastomar5593@gmail.com>
Date:   Mon Nov 27 13:20:14 2023 +0530

    t2 renamed.

commit dff80019acda6fab288673a37233506acd5ce580
Author: Jeetu <jitendrastomar5593@gmail.com>
Date:   Mon Nov 27 13:18:23 2023 +0530

    t1 removed
```

To view log with actual values changes (line-by-line):
git log -p

```
$ git log -p
commit 07828e77a4e2be1fdd74647ae517ffa0e562a470 (HEAD -> master)
Author: Jeetu <jitendrastomar5593@gmail.com>
Date:   Mon Nov 27 13:28:29 2023 +0530

    added .gitignore

diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..cde045f
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1 @@
+new_t2.txt
diff --git a/new_t2.txt b/new_t2.txt
deleted file mode 100644
index e69de29..0000000
```

To view 2 recent logs:
git log -p -2

```
$ git log -p -2
commit 07828e77a4e2be1fdd74647ae517ffa0e562a470 (HEAD -> master)
Author: Jeetu <jitendrastomar5593@gmail.com>
Date:   Mon Nov 27 13:28:29 2023 +0530

    added .gitignore

diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..cde045f
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1 @@
+new_t2.txt
diff --git a/new_t2.txt b/new_t2.txt
deleted file mode 100644
index e69de29..0000000

commit 20b81f7a5ff53412c274a9b9a16ec69a6e76c94e
Author: Jeetu <jitendrastomar5593@gmail.com>
Date:   Mon Nov 27 13:20:14 2023 +0530

    t2 renamed.

diff --git a/new_t2.txt b/new_t2.txt
new file mode 100644
index 0000000..e69de29
```

To view precise logs:
git log --stat

```
$ git log --stat
commit 07828e77a4e2be1fdd74647ae517ffa0e562a470 (HEAD -> master)
Author: Jeetu <jitendrastomar5593@gmail.com>
Date:   Mon Nov 27 13:28:29 2023 +0530

    added .gitignore

 .gitignore | 1
 new_t2.txt | 0
 2 files changed, 1 insertion(+)
```

To view logs with hashes in one line:
git log --pretty=oneline

```
$ git log --pretty=oneline
07828e77a4e2be1fdd74647ae517ffa0e562a470 (HEAD -> master) added .gitignore
20b81f7a5ff53412c274a9b9a16ec69a6e76c94e t2 renamed.
dff80019acda6fab288673a37233506acd5ce580 t1 removed
1383b8928f2a6c9f394dfd46f3c3a25d39273e75 Direct commit
c4028500a12404586737b3653e8be4de6aa407d6 added context to README.md and removed Lic file.
ff5c3008bb8749ecca3e6cdba4ad8451a71298c8 (origin/master) changes in index file
325ff31d988c59978d711261f761600855ab38f4 Merge pull request #2 from jitendrastomar5593/master
a2d162053b98b8aec9ff6b739cf8e38fa2e744a9 adding fresh style sheet.
ea3693279f4a9e801eff4319d0fcc3bf37dbe418 Merge pull request #1 from jitendrastomar5593/master
0034840d74891baa12315445c8ad21f5ad90651a index file done
5c4306005572497a37e66cbe8f8892649dc71d3d Initial commit
```

To view logs in short:
git log --pretty=short

```
$ git log --pretty=short
commit 07828e77a4e2be1fdd74647ae517ffa0e562a470 (HEAD -> master)
Author: Jeetu <jitendrastomar5593@gmail.com>

    added .gitignore

commit 20b81f7a5ff53412c274a9b9a16ec69a6e76c94e
Author: Jeetu <jitendrastomar5593@gmail.com>

    t2 renamed.

commit dff80019acda6fab288673a37233506acd5ce580
Author: Jeetu <jitendrastomar5593@gmail.com>

    t1 removed
```

To view logs with full details:
git log --pretty=full

```
$ git log --pretty=full
commit 07828e77a4e2be1fdd74647ae517ffa0e562a470 (HEAD -> master)
Author: Jeetu <jitendrastomar5593@gmail.com>
Commit: Jeetu <jitendrastomar5593@gmail.com>

    added .gitignore
```

Author: person who created the file:
Commit: person who committed changed to that file.

To list logs since last 2 days:
git log --since=2.days

```
$ git log --since=2.days
commit 07828e77a4e2be1fdd74647ae517ffa0e562a470 (HEAD -> master)
Author: Jeetu <jitendrastomar5593@gmail.com>
Date:   Mon Nov 27 13:28:29 2023 +0530

    added .gitignore

commit 20b81f7a5ff53412c274a9b9a16ec69a6e76c94e
Author: Jeetu <jitendrastomar5593@gmail.com>
Date:   Mon Nov 27 13:20:14 2023 +0530

    t2 renamed.

commit dff80019acda6fab288673a37233506acd5ce580
Author: Jeetu <jitendrastomar5593@gmail.com>
Date:   Mon Nov 27 13:18:23 2023 +0530

    t1 removed
```

git log --since=2.weeks
git log --since=2.months
git log --since=2.years

To list hash, commit with author name:
git log --pretty=format:"%h -- %an"

```
admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git log --pretty=format:"%h -- %an"
07828e7 -- Jeetu
20b81f7 -- Jeetu
dff8001 -- Jeetu
1383b89 -- Jeetu
c402850 -- Jeetu
ff5c300 -- Jeetu
325ff31 -- Jitendra Singh Tomar
a2d1620 -- BOB
ea36932 -- Jitendra Singh Tomar
0034840 -- Alice
5c43060 -- Jitendra Singh Tomar

admin@WorkPC MINGW64 ~/mygitdemo (master)
$
```

To list hash, commit with author email:
git log --pretty=format:"%h -- %ae"

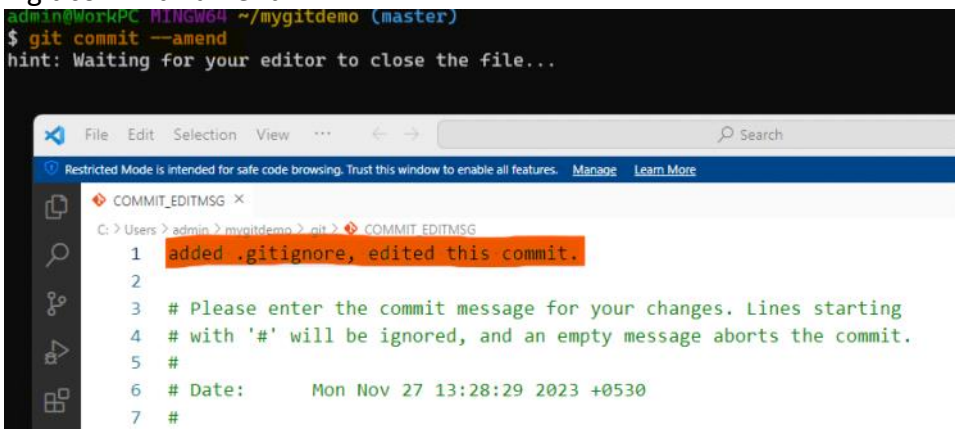
```
admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git log --pretty=format:"%h -- %ae"
07828e7 -- jitendrastomar5593@gmail.com
20b81f7 -- jitendrastomar5593@gmail.com
dff8001 -- jitendrastomar5593@gmail.com
1383b89 -- jitendrastomar5593@gmail.com
c402850 -- jitendrastomar5593@gmail.com
ff5c300 -- jitendrastomar5593@gmail.com
325ff31 -- jitendrastomar5593@gmail.com
a2d1620 -- bob@jeetusingh.in
ea36932 -- jitendrastomar5593@gmail.com
0034840 -- alice@jeetusingh.in
5c43060 -- jitendrastomar5593@gmail.com
```

Editing the commit for an existing (last) commit:
git log -p -1

```
$ git log -p -1
commit 07828e77a4e2be1fdd74647ae517ffa0e562a470 (HEAD -> master)
Author: Jeetu <jitendrastomar5593@gmail.com>
Date:   Mon Nov 27 13:28:29 2023 +0530

    added .gitignore

diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..cde045f
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1 @@
+new_t2.txt
diff --git a/new_t2.txt b/new_t2.txt
deleted file mode 100644
index e69de29..0000000
```

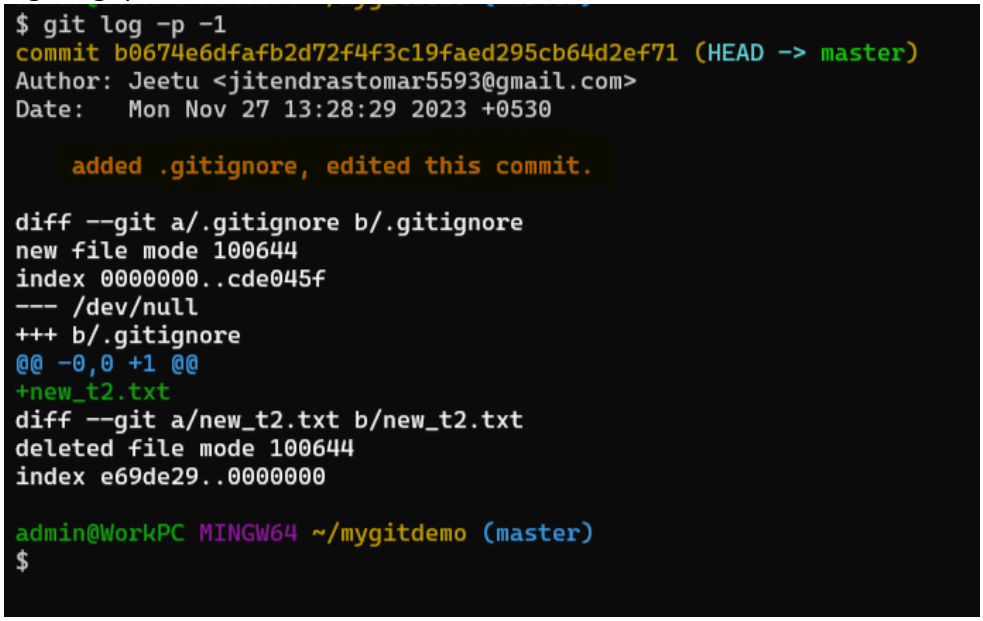
Editing commit

git commit --amend



Verifying the edited commit:

git log -p -1

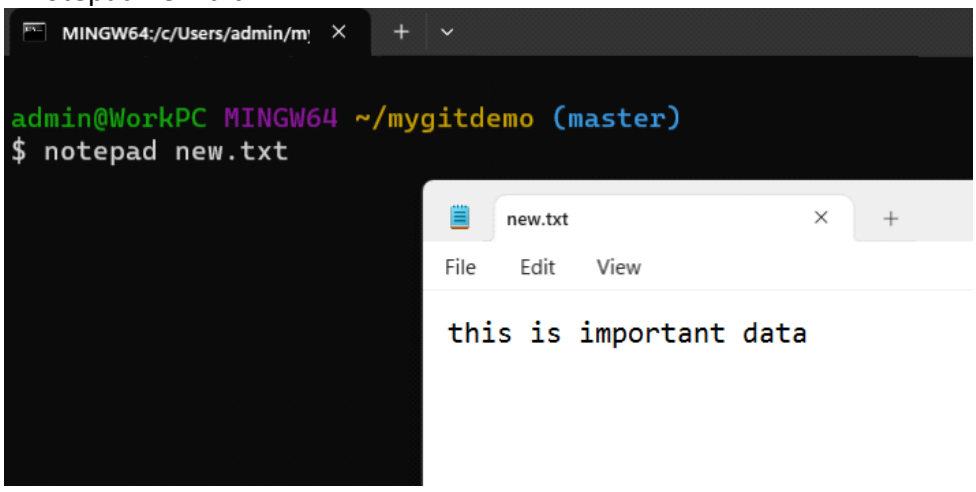


9. Un-staging & un-modifying in Git:

To un-stage a file (its only done on staged files, not on committed files):
git restore --staged <file-name.txt>

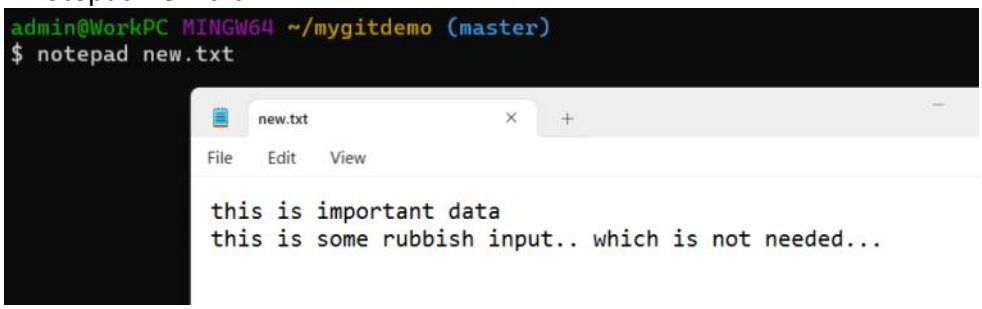
```
admin@WorkPC MINGW64 ~/mygitdemo (master)
$ touch t3.txt

admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git add t3.txt

admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   t3.txt


admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git restore --staged t3.txt

admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        t3.txt

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

admin@WorkPC MINGW64 ~/mygitdemo (master)
$
```

Restoring/Revert/Unmodified file:
notepad new.txt

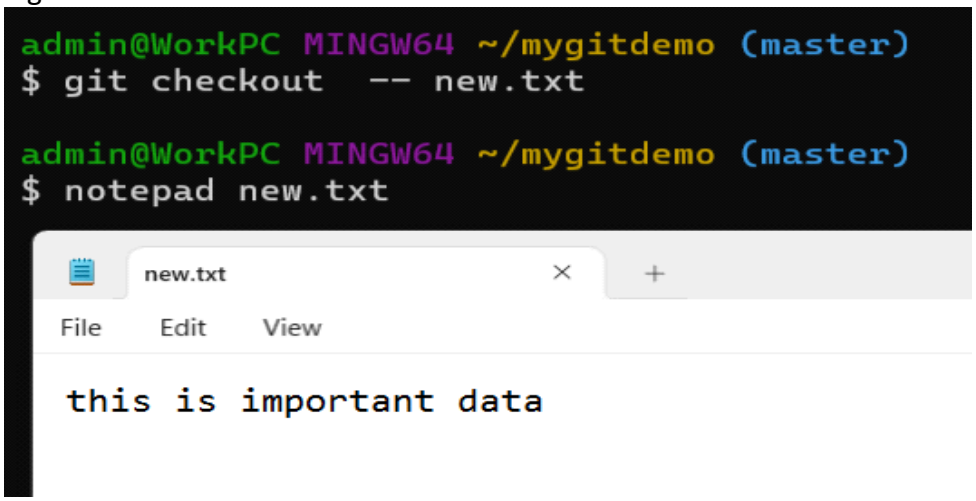


```
admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        new.txt

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

admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git add new.txt

admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git commit -m "performing revert"
[master 5babc48] performing revert
 1 file changed, 1 insertion(+)
 create mode 100644 new.txt

admin@WorkPC MINGW64 ~/mygitdemo (master)
$
```

Re-writing same new.txt file:
notepad new.txt



Reverting to last known commit:
git checkout -- <filename.txt>



To revert/checkout multiple files:
git checkout -f
git status

10. Remote repositories:

To add a remote URL as origin (origin is just a commonly used name, it could be different as well):
git remote add origin git@github.com:<URL>
git remote add origin git@github.com:jitendrastomar5593/mygitdemo.git
git remote
git remote -v

Pushing data/code to repo:
git push -u origin master

If pushing shows error, then need to generate SSH keys and attach it to GitHub.com account:
ssh-keygen -t rsa -b 4096 -c "jitendrastomar5593@gmail.com"

Adding SSH key to the agent:
ssh-add ~/.ssh/id_rsa

Fetch the public ssh key & add it to github:
GitHub.com --> settings --> SSH & GPG keys --> save the public key.

Now run the command:
git push -u origin master // now it should work.

11. Git Alias

Creating alias

git config --global alias.st status

```
admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git config --global alias.st status

admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git st
On branch master
nothing to commit, working tree clean

admin@WorkPC MINGW64 ~/mygitdemo (master)
$
```

git config --global alias.ci commit

```
admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git config --global alias.ci commit

admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git ci
On branch master
nothing to commit, working tree clean
```

git config --global alias.last 'log -p -1'

```
admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git config --global alias.last 'log -p -1'

admin@WorkPC MINGW64 ~/mygitdemo (master)
$ git last
commit 5babc484dac96fc672edc3328cba39d0c6821be8 (HEAD -> master)
Author: Jeetu <jitendrastomar5593@gmail.com>
Date:   Mon Nov 27 14:24:43 2023 +0530

    performing revert

diff --git a/new.txt b/new.txt
new file mode 100644
index 0000000..07a2f3a
--- /dev/null
+++ b/new.txt
@@ -0,0 +1 @@
+this is important data
\ No newline at end of file
```