GIT/ GITHUB

Basic Commands:

To get present working directory **Pwd**

list of folders **ls**

Creating a working directory **mkdir name**

To go into the folder **cd folder_name**

To go back to the previous directory **cd _ _**

To get the hidden files **ls -a**

creating a local repository, cloning it with a remote, and performing basic Git operations:

- **Create a Directory for the Local Repository**
  mkdir my-local-repo

  to open
  cd my-local-repo

- **Initialize a Local Git Repository**
  git init

- **Generate an SSH Key**
  ssh-keygen

- **Set Up a Remote Repository**
  Create a new repository. Copy the SSH URL of the repository

- **Clone the Remote Repository**
  git clone sshurl

- **Create a File and Add Content**
  echo "This is my first file" > file.txt

- **Check the status of the repository**
  git status

  The file file.txt will appear as untracked.

- **Stage the File**

  git add file.txt

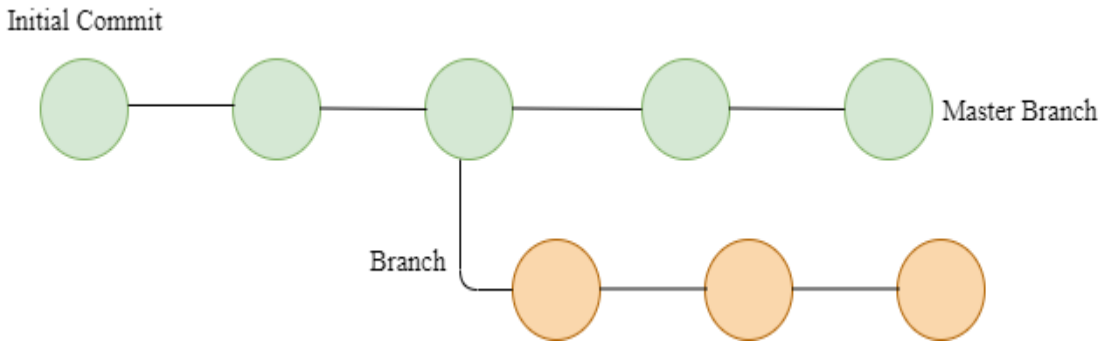
- **Commit Changes**
  git commit -m "Added file.txt with initial content"
  0r
  git commit –amend
  or
  git author

- **Push Changes to the Remote Repository**
  git push origin main

- **Check the Git Difference**
  git diff

Branch in Git:

A branch in Git is essentially a lightweight, movable pointer to a commit. It allows developers to create separate lines of development, work on new features, fixes, or experiments without affecting the main codebase. Branching makes it easier to work on multiple aspects of a project simultaneously.



View All Branches:

git branch

Create a Branch:

git branch branch-name
or
git checkout -b branch-name

Switch to a Branch

Git checkout branch-name

Delete a Branch:

Git branch -d branch-name

If not merged

Git branch -D branch-name

What is a Merge in Git

Merging in Git is the process of integrating changes from one branch into another. It combines the work done in parallel on different branches and brings them into a unified codebase.

Fast-Forward Merge:

Occurs when there are no new commits on the target branch since branching off.
Git simply moves the pointer of the target branch to the latest commit in the source branch.

git checkout main
git merge feature-branch

Merge without conflit:

Used when both the source and target branches have diverged with new commits.
Git uses a common ancestor commit to combine changes and creates a new merge commit.

git checkout main
git merge feature-branch

Merge with conflit:

A merge conflict occurs when Git is unable to automatically combine changes from two branches because the same part of a file was modified differently in both branches. Git pauses the merge process and asks you to resolve the conflict manually.

Creating a merge conflict

To show a simple example of how a merge conflict can happen, we can manually trigger a merge conflict from the following set of commands in any UNIX terminal / GIT bash

Step 1: Create a new directory using the **mkdir** command, and **cd** into it.

Step 2: initialize it as a new Git repository using the **git init** command and create a new text file using the **touch** command.

Step 3: Open the text file and add some content in it, then **add** the text file to the repo and **commit** it.

Step 4: Now, its time to create a new branch to use it as the conflicting merge. Use **git checkout** to create and checkout the new branch.

Step 5: Now, overwrite some conflicting changes to the text file from this new branch.

Step 6: Add the changes to git and **commit** it from the new branch. With this new branch: new_branch_for_merge_conflict we have created a commit that overrides the content of test_file.txt

Step 7: Again **checkout** the master branch, and this time **append** some text to the test_file.txt from the master branch.

Step 8: add these new changes to the staging area and **commit** them.

Step 9: Now for the last part, try **merging** the new branch to the master branch and you will encounter the second type of merge conflict.

Handling the Merge Conflict

As we have experienced from the proceeding example, Git will produce some descriptive output letting us know that a CONFLICT has occurred. We can gain further insight by running the **git status** command.

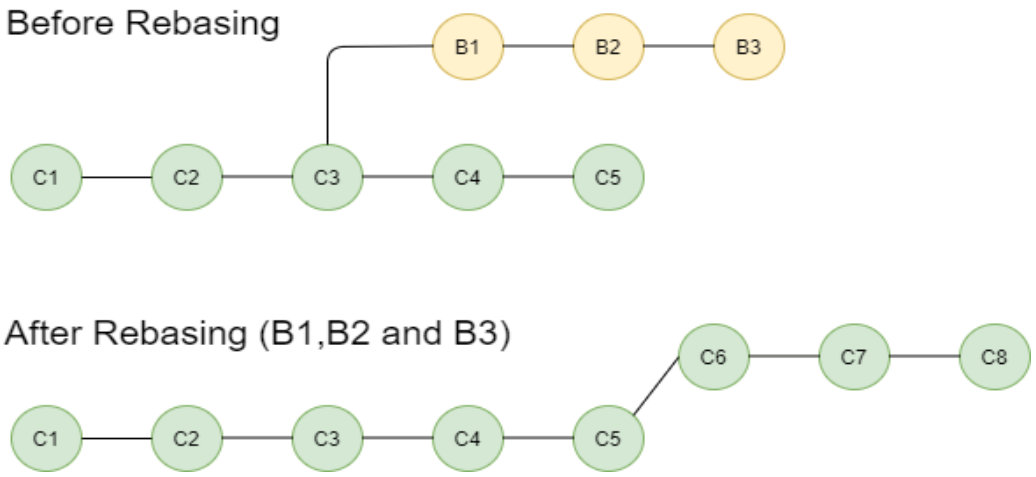
On opening the test_file.txt we see some "conflict dividers".

The ======= line is the "center" of the conflict. All the content between the center and the <<<<<<< HEAD line is content that exists in the current branch master which the HEAD ref is pointing to. Alternatively, all content between the center and >>>>>>> new_branch_for_merge_conflict is content that is present in our merging branch.

To resolve our merge conflict, we can manually remove the unnecessary part from any one of the branches, and only consider the content of the branch that is important for further use, along with removing the "conflict dividers" from our file. Once the conflict has been resolved we can use the git add command to move the new changes to the staging area, and then git commit to commit the changes.

Git Rebase:

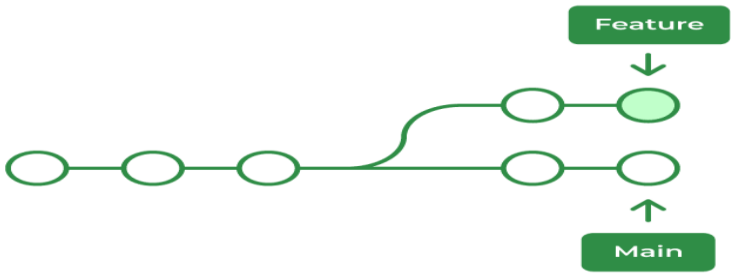
Rebasing in Git is a process of integrating a series of commits on top of another base tip. It takes all the commits of a branch and appends them to the commits of a new branch.



Uses of Git Rebase :

The main aim of rebasing is to maintain a progressively straight and cleaner project history. Rebasing gives rise to a perfectly linear project history that can follow the end commit of the feature all the way to the beginning of the project without even forking. This makes it easier to navigate your project.

You can integrate the feature branch into the main branch in two ways. the first one is by merging directly into a main branch or first rebasing and then merging. The below diagram shows If you rebase the feature branch first it will facilitate a fast-forward merge. Integrating upstream updates into your local repository is frequently done by rebasing. Git merge causes an unnecessary merging to commit each time you want to see how the project has advanced when you pull in upstream modifications



Stashing in Git:

Stashing in Git is a way to temporarily save uncommitted changes (both staged and unstaged) without committing them to the repository. It allows you to work on something else or switch branches without losing your current work.

Why Use Stashing

- To save work in progress without committing.
- To switch branches or update the working directory without conflicts.
- To quickly shelve changes for later use.

Commands for Stashing:

1. To stash your current changes, use the command git stash.
2. To add a message and identify the stash, use git stash save "Message describing the changes".
3. To view all stashed changes, use git stash list.
4. The stash list will display entries like stash@{0}: WIP on main: 123abc Initial commit or stash@{1}: On main: Added new feature.
5. To reapply the most recent stash, use git stash apply.
6. To apply a specific stash, such as stash@{1}, use git stash apply stash@{1}.
7. To remove a specific stash, such as stash@{1}, use git stash drop stash@{1}.
8. To clear all stashes, use git stash clear.
9. To apply the most recent stash and remove it from the stash list, use git stash pop.

Reverting:

Reverting in Git is a way to undo changes by creating a new commit that undoes the changes introduced by a previous commit. Unlike git reset, it preserves the commit history and is safer for collaborative workflows.

Why Use Revert?

Safe Undo: Revert does not rewrite history, making it ideal for shared branches.

Preserve History: It keeps all previous commits intact, even the ones being undone.

Collaborative Use: It avoids issues caused by rewriting history in a team environment.

STEPS:

1. To revert a single commit, use the command git revert <commit-hash>.
2. This command creates a new commit that undoes the changes introduced by commit .
3. Git will open an editor with a default commit message describing the revert.
4. Save and close the editor to complete the revert process.
5. If you want to skip the editor and apply the revert with the default message, use git revert --no-edit <commit-hash>.
6. After the revert is successful, use git push origin <branch-name> to push the changes to the remote repository.

GIT RESET

git reset is a command used to undo changes in your working directory and staging area. It allows you to reset the state of your repository to a previous commit, modifying the commit history or the staging area. It is commonly used when you want to discard changes or move your branch pointer to a previous state.

Types of Reset in Git

Soft Reset (git reset --soft)
Mixed Reset (git reset --mixed) (default)
Hard Reset (git reset --hard)

Soft Reset

A **soft reset** moves the HEAD pointer (current commit) to a previous commit, but it **keeps the changes** staged for commit (in the staging area).

Mixed Reset (Default)

A **mixed reset** moves the HEAD pointer to a previous commit and **unstages** the changes (removes changes from the staging area but keeps them in the working directory).

Hard Reset

A **hard reset** moves the HEAD pointer to a previous commit and **discards all changes**, both staged and unstaged, in the working directory and staging area.

To Track a Remote branch:

git checkout –track origin/branch-name

git push origin -delete B1 ---to delete branch from repo

git branch -D b1 ---delete branch from local

Git – LFS (Large File Storage):

Git is a powerful version control system that tracks changes in your codebase. However, it struggles with large files, such as high-resolution images, videos, and large datasets. This is where Git LFS (Large File Storage) comes into play. Git LFS is an extension that improves Git's handling of large files, making it an important tool for developers and teams working with large media files or binaries.

What is Git LFS?

Git LFS is designed to manage large files efficiently in Git repositories. Instead of storing the actual large files in the repository, Git LFS stores pointers to these files, while the files themselves are stored on a remote server. This approach reduces the load on your repository and ensures faster cloning and fetching operations.

Key Features of Git LFS

Efficient Storage: Git LFS keeps your repository size manageable by storing large files separately.

Seamless Integration: It integrates smoothly with your existing Git workflow.

Bandwidth Optimization: Only the required versions of large files are downloaded, saving bandwidth and reducing clone times.

Compatibility: Works with existing Git commands, requiring minimal changes to your workflow.

How Does Git LFS Work

Git LFS replaces large files in your repository with tiny pointer files. These pointer files contain metadata about the original large files. The actual large files are stored in a dedicated LFS server. When you check out a branch, Git LFS automatically replaces the pointer files with the corresponding large files from the LFS server.

In order to use Git LFS, first, we need to install it on our system. We can use the following command to do so.

sudo apt-get install git-lfs

Step 1: Run the command on the repository.

git lfs install

It will initialize the LFS in the repository and will update the Git Hooks.

Step 2:

git lfs track "*.jpg"

This will tell git lfs to handle the jpg files, in case you want any other file, then you can specify that particular extension.

Step 3: As the configuration that is done on Step 2, is stored in the .gitattributes file, so we will add it for commit using the following command
git add .gitattributes

Step 4: After we have done this, we will create a new branch and add all the large files there and push the new branch into the remote repository.

Step 5: Now clone the repository from the remote.

Step 6: After cloning if we look at the size of the cloned repository and the original repository we can see that there is a big difference between their sizes.

Step 7: Now if we check out to that branch where the large files are present, then at that moment only the actual files will be downloaded from the remote servers.

signed commit:

A **signed commit** in Git is a commit that is cryptographically signed using a GPG (GNU Privacy Guard) or SSH key to verify the identity of the author. This provides an additional layer of security and ensures that the commit was made by the claimed author.
Signed commits are commonly used in open-source projects and organizations that require verification of who made a commit, enhancing trust in the integrity of the code.

Step 1: Set Up GPG Key for Signing

Generate a GPG key if you don't already have one. This key will be used to sign your commits. You will need to follow the steps to create the key, including selecting the key type and providing your name and email address.

List the generated GPG keys on your system to identify the one you want to use for signing commits.

Configure Git to use your GPG key for signing. You need to tell Git which key to use by setting it in the global configuration.

Step 2: Configure Git to Sign Commits

Once the GPG key is set up, you can configure Git to automatically sign all of your commits. This ensures that every commit you make is signed without needing to specify it each time.

Alternatively, you can choose to manually sign individual commits by specifying the signing option during commit creation.

Step 3: Make Signed Commits

When making a commit, Git will use your configured GPG key to sign the commit automatically if you've set it up for all commits.

If you want to sign a specific commit, you can do so manually by using the signing option while making the commit.

Step 4: Viewing Signed Commits

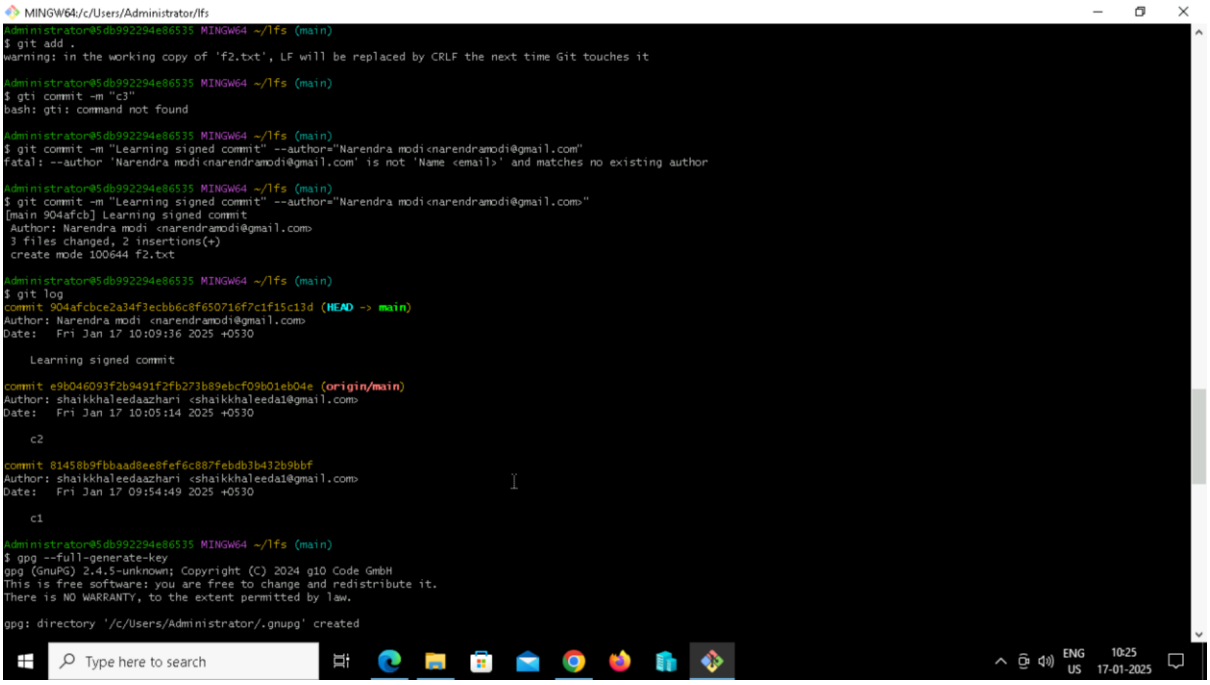
To verify that a commit is signed, you can view the commit details. Signed commits will show a "Good signature" message along with the name and email of the signer.

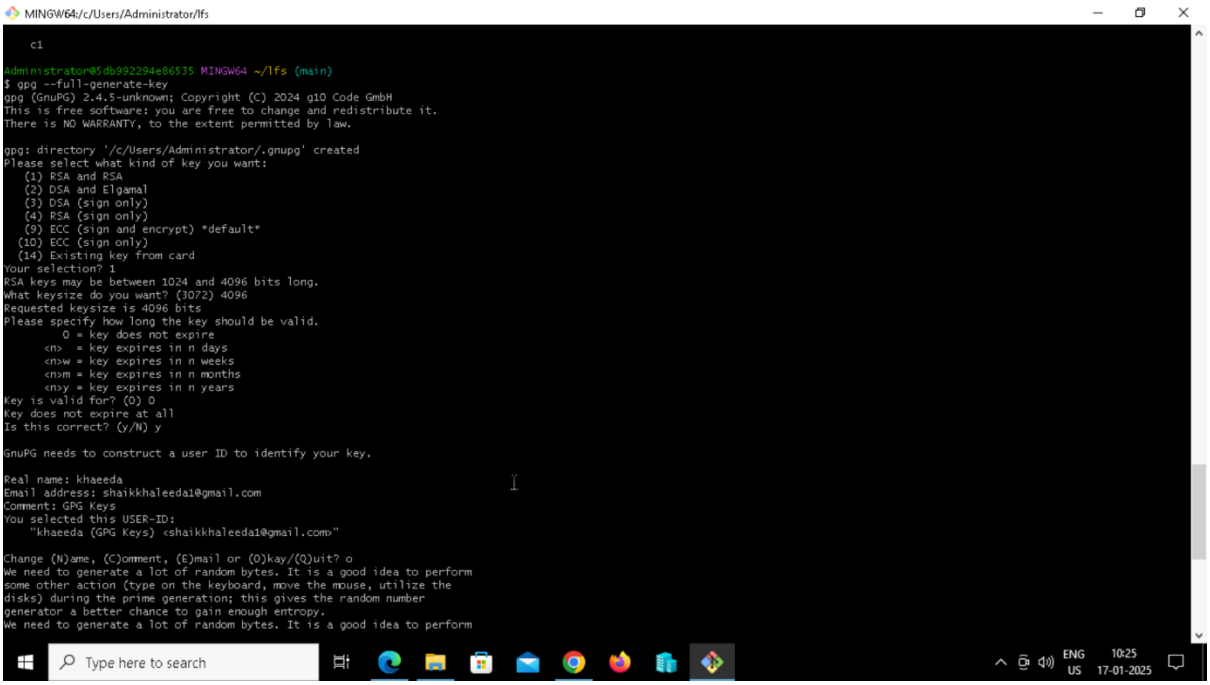
Git will display whether the signature is valid or if there is an issue with the commit signature.

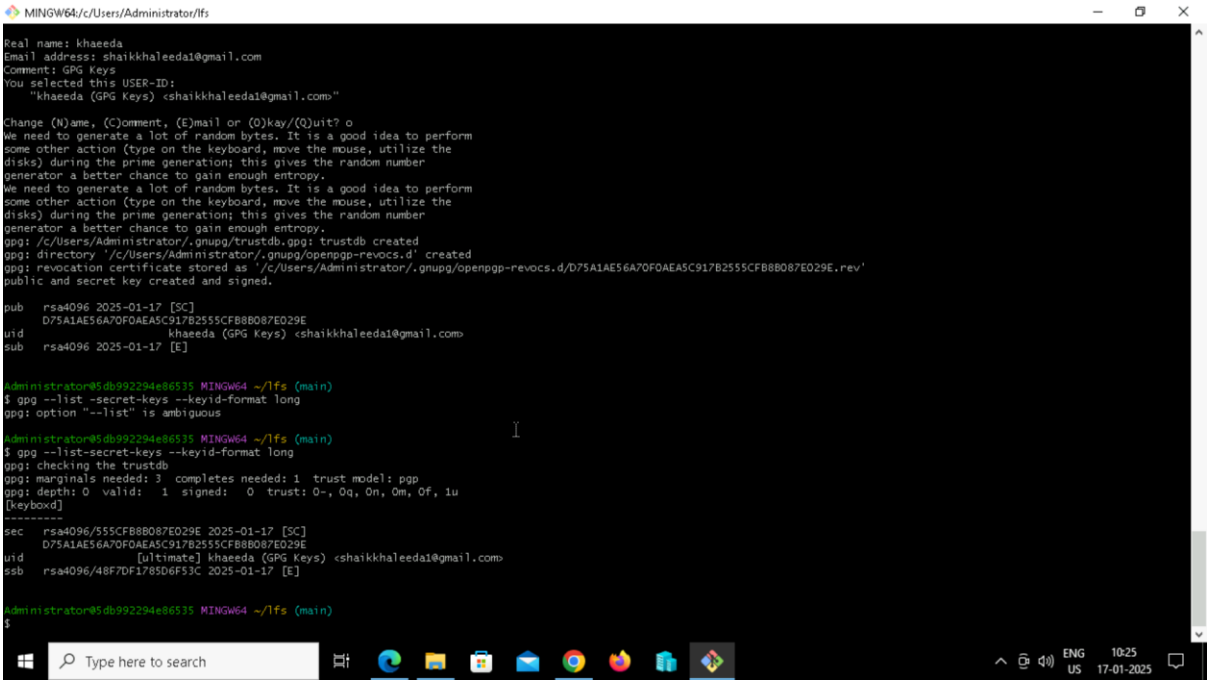
Step 5: Verifying Signed Commits

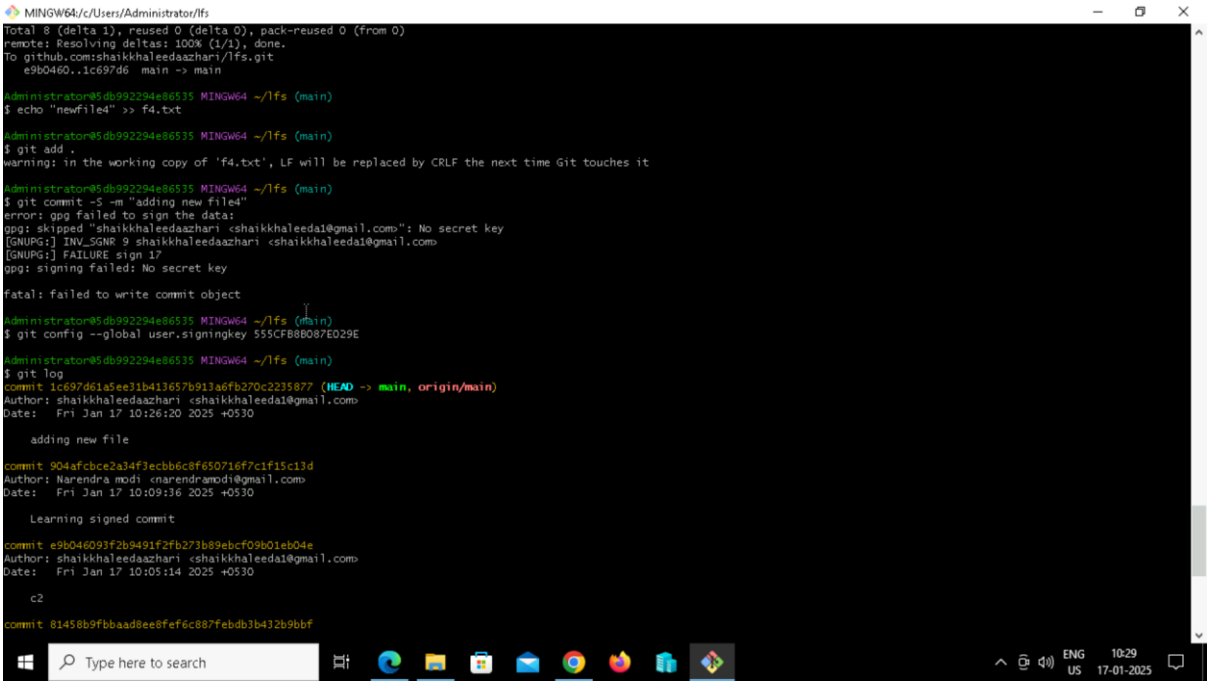
To verify that the commit signature is valid, you can use Git's verification feature. It will check if the commit was signed correctly with the specified GPG or SSH key.

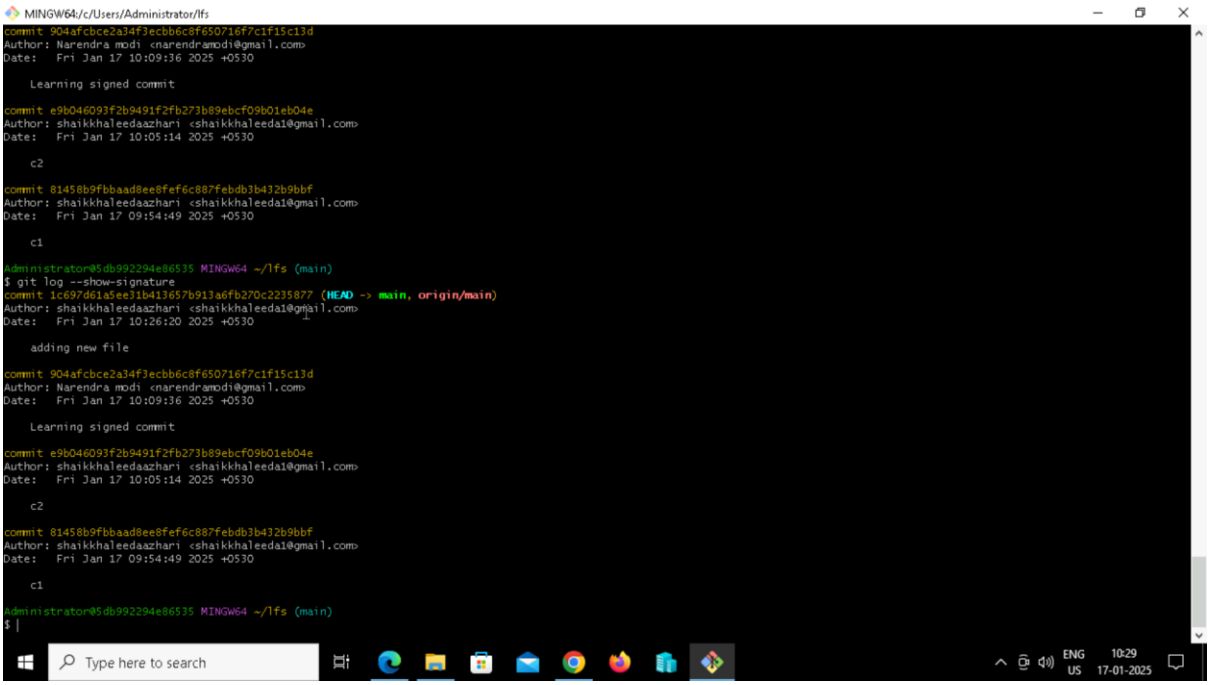
This step helps ensure that the commit was made by the person it claims to be from, providing an extra layer of authenticity to the codebase.

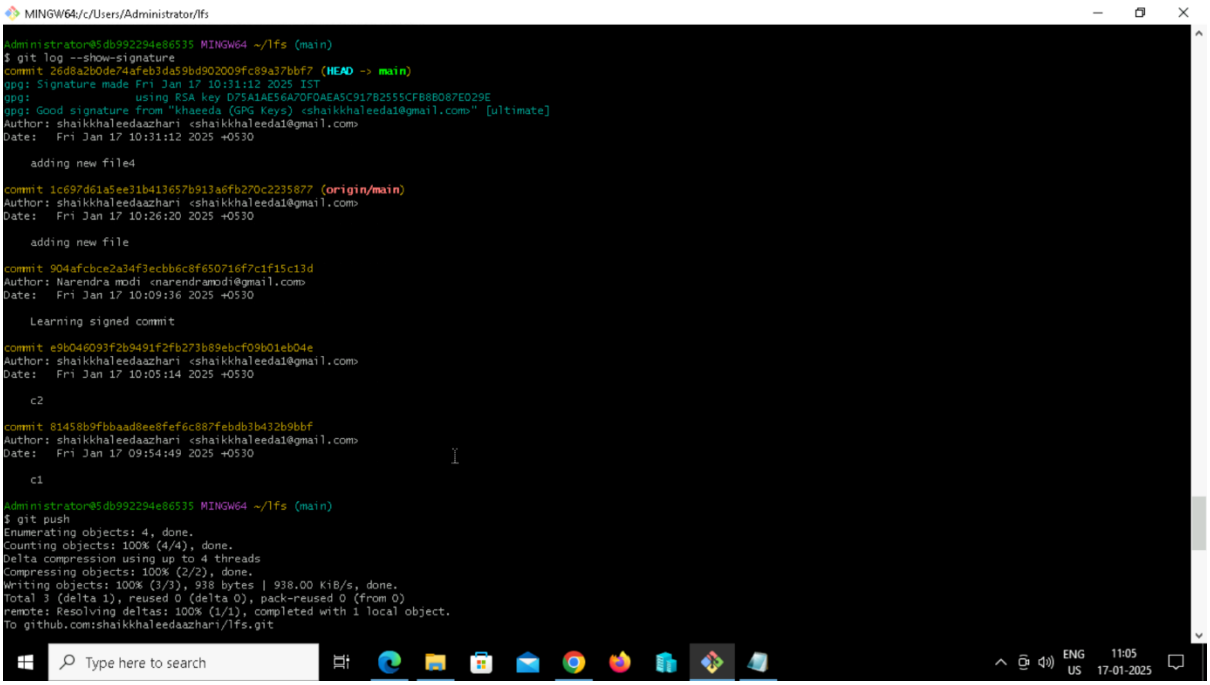


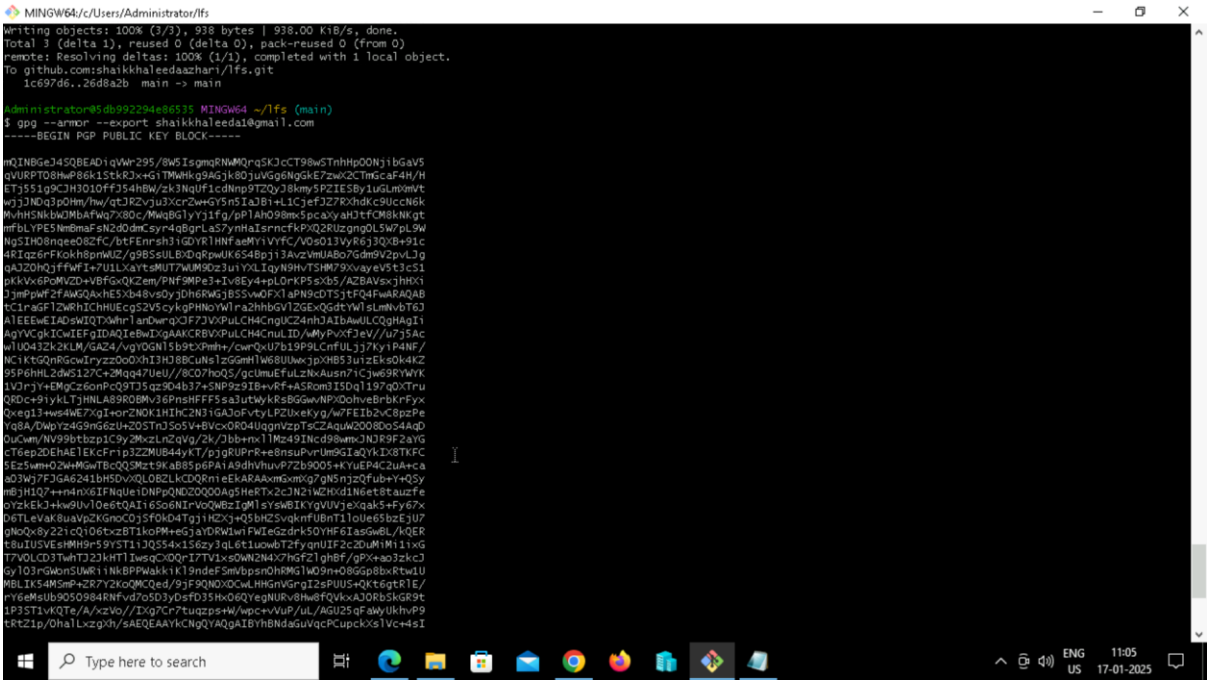


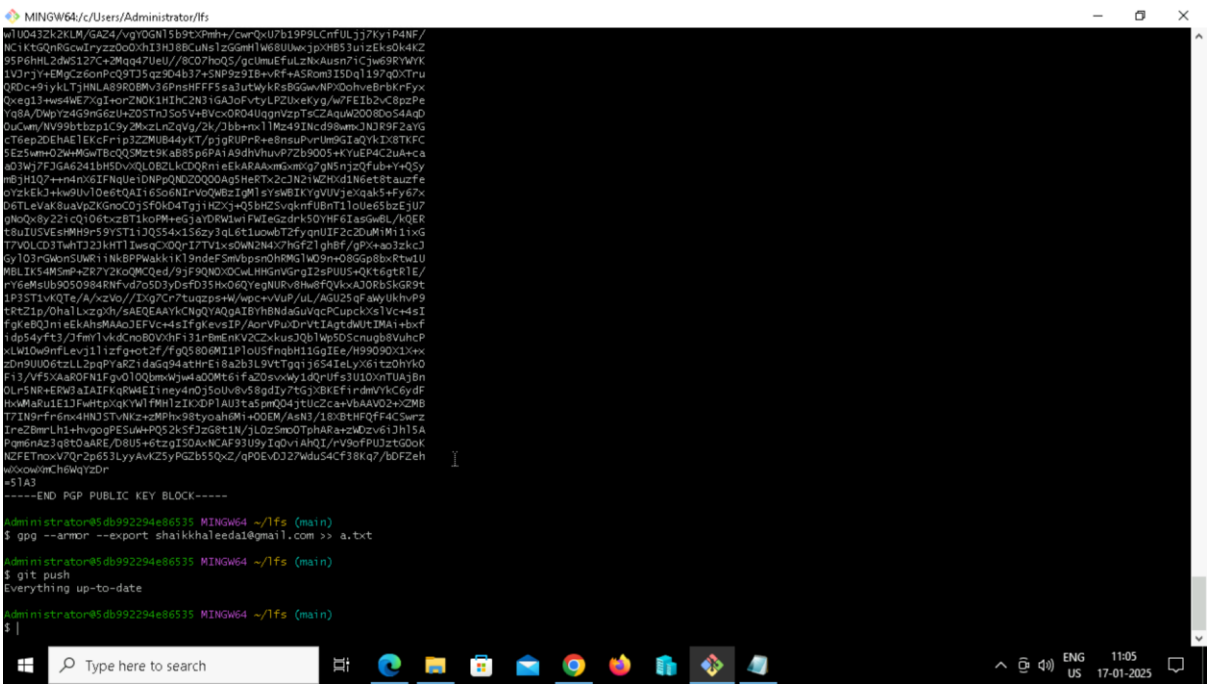

MINGW64:/c/Users/Administrator/lfs
Real name: khaeeda
Email address: shaikkhaleeda1@gmail.com
Comment: GPG Keys
You selected this USER-ID:
"khaeeda (GPG Keys) <shaikkhaleeda1@gmail.com>"
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /c/Users/Administrator/.gnupg/trustdb.gpg: trustdb created
gpg: directory '/c/Users/Administrator/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/c/Users/Administrator/.gnupg/openpgp-revocs.d/D75A1AE56A7OFOAEA5C917B2555CFB8BO87EO29E.rev'
public and secret key created and signed.
pub rsa4096 2025-01-17 [SC]
D75A1AE56A7OFOAEA5C917B2555CFB8BO87EO29E
uid khaeeda (GPG Keys) <shaikkhaleeda1@gmail.com>
sub rsa4096 2025-01-17 [E]
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ gpg --list -secret-keys --keyid-format long
gpg: option "--list" is ambiguous
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ gpg --list-secret-keys --keyid-format long
gpg: checking the trustdb
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: depth: O valid: 1 signed: O trust: O-, Oq, On, Om, Of, 1u
[keyboxd]
---
sec rsa4096/555CFB8B087EO29E 2025-01-17 [SC]
D75A1AE56A7OFOAEA5C917B2555CFB8BO87EO29E
uid [ultimate] khaeeda (GPG Keys) <shaikkhaleeda1@gmail.com>
ssb rsa4096/48F7DF1785D6F53C 2025-01-17 [E]
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$
Type here to search
ENG
US
10:25
17-01-2025


MINGW64:/c/Users/Administrator/lfs
Total 8 (delta 1), reused O (delta O), pack-reused O (from O)
remote: Resolving deltas: 100% (1/1), done.
To github.com:shaikkhaleedaazhari/lfs.git
e9bO46O..1c697d6 main -> main
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ echo "newfile4" >> f4.txt
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ git add .
warning: in the working copy of 'f4.txt', LF will be replaced by CRLF the next time Git touches it
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ git commit -S -m "adding new file4"
error: gpg failed to sign the data:
gpg: skipped "shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>": No secret key
[GNUPG:] INV_SGNR 9 shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>
[GNUPG:] FAILURE sign 17
gpg: signing failed: No secret key
fatal: failed to write commit object
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ git config --global user.signingkey 555CFB8B087EO29E
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ git log
commit 1c697d61a5ee31b413657b913a6fb27Oc2235877 (HEAD -> main, origin/main)
Author: shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>
Date: Fri Jan 17 10:26:20 2025 +0530
adding new file
commit 9O4afcbce2a34f3ecbb6c8f650716f7c1f15c13d
Author: Narendra modi <narendramodi@gmail.com>
Date: Fri Jan 17 10:09:36 2025 +0530
Learning signed commit
commit e9bO46093f2b9491f2fb273b89ebcfO9bO1eb04e
Author: shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>
Date: Fri Jan 17 10:05:14 2025 +0530
c2
commit 81458b9fbbaad8ee8fef6c887febdb3b432b9bbf
Type here to search
ENG
US
10:29
17-01-2025


MINGW64:/c/Users/Administrator/lfs
commit 904afcbce2a34f3ecbb6c8f650716f7c1f15c13d
Author: Narendra modi <narendramodi@gmail.com>
Date: Fri Jan 17 10:09:36 2025 +0530
Learning signed commit
commit e9b046093f2b9491f2fb273b89ebcf09b01eb04e
Author: shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>
Date: Fri Jan 17 10:05:14 2025 +0530
c2
commit 81458b9fbbaad8ee8fef6c887febdb3b432b9bbf
Author: shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>
Date: Fri Jan 17 09:54:49 2025 +0530
c1
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ git log --show-signature
commit 1c697d61a5ee31b413657b913a6fb270c2235877 (HEAD -> main, origin/main)
Author: shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>
Date: Fri Jan 17 10:26:20 2025 +0530
adding new file
commit 904afcbce2a34f3ecbb6c8f650716f7c1f15c13d
Author: Narendra modi <narendramodi@gmail.com>
Date: Fri Jan 17 10:09:36 2025 +0530
Learning signed commit
commit e9b046093f2b9491f2fb273b89ebcf09b01eb04e
Author: shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>
Date: Fri Jan 17 10:05:14 2025 +0530
c2
commit 81458b9fbbaad8ee8fef6c887febdb3b432b9bbf
Author: shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>
Date: Fri Jan 17 09:54:49 2025 +0530
c1
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$
Type here to search
ENG
US
10:29
17-01-2025


MINGW64:/c/Users/Administrator/lfs
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ git log --show-signature
commit 26d8a2b0de74afeb3da59bd902009fc89a37bbf7 (HEAD -> main)
gpg: Signature made Fri Jan 17 10:31:12 2025 IST
gpg: using RSA key D75A1AE56A70FDAEA5C917B2555CFB8B087ED29E
gpg: Good signature from "khaeeda (GPG Keys) <shaikkhaleeda1@gmail.com>" [ultimate]
Author: shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>
Date: Fri Jan 17 10:31:12 2025 +0530
adding new file4
commit 1c697d61a5ee31b413657b913a6fb270c2235877 (origin/main)
Author: shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>
Date: Fri Jan 17 10:26:20 2025 +0530
adding new file
commit 904afcbce2a34f3ecbb6c8f650716f7c1f15c13d
Author: Narendra modi <narendramodi@gmail.com>
Date: Fri Jan 17 10:09:36 2025 +0530
Learning signed commit
commit e9b046093f2b9491f2fb273b89ebcf09b01eb04e
Author: shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>
Date: Fri Jan 17 10:05:14 2025 +0530
c2
commit 81458b9fbbaad8ee8fef6c887febdb3b432b9bbf
Author: shaikkhaleedaazhari <shaikkhaleeda1@gmail.com>
Date: Fri Jan 17 09:54:49 2025 +0530
c1
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 938 bytes | 938.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:shaikkhaleedaazhari/lfs.git
Type here to search
ENG
US
11:05
17-01-2025


MINGW64:/c/Users/Administrator/lfs
Writing objects: 100% (3/3), 938 bytes | 938.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:shaikkhaleedaazhari/lfs.git
1c697d6..26d8a2b main -> main
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ gpg --armor --export shaikkhaleeda1@gmail.com
-----BEGIN PGP PUBLIC KEY BLOCK-----
mQINBGeJ4SQBEADiqVWr295/8W5IsgmqRNWMQrqSKJcCT98wSTnhHpOONjibGaV5
qVURPTO8HwP86k1StkR3x+GiTMWHkg9AGjk8OjuVGg6NgGkE7zwX2CTmGcaF4H/H
tTy551g9C3H3O1OFf354HBW/zk3NqUfidNnp9TZQy38kmy5PZIE5By1uGLmYmWt
wjjJNDq3pOHm/hw/qtJRZvju3XcrZwHGY5n5IaJBi+L1CjefJZ7RXhdKc9UccN6k
MvhHSNkbWJMbAfWq7X8Oc/MWqBGlyYj1fg/pPlAhO98mx5pcaXyaHJtfCM8kNKgt
mfbLYPE5NmBmaFsN2dOdmCsyr4qBgrLaS7ynHaIsrncfkPXQ2RUzgngOL5W7pL9W
NgSIHO8nqeeO8ZfC/btFEnrsh3iGDYRlHNfaeMYiVYfC/VOsO13VyR6j3QXB+91c
4RIqz6rFKokh8pnWUZ/g9BSsULBXDqRpwUK6S4Bpji3AvzVmUABo7Gdm9V2pvLJg
qAJZOhQjfFWfI+7U1LXaYtsMUT7WUM9Dz3uiYXLIqyN9HvTSHM79XvayeV5t3cS1
pkkVx6PoMVZD+VBfGxQkZem/PNf9MPe3+Iv8Ey4+pLOrKP5sXb5/AZBAVsxjhHXi
JjmPpWf2fAWGQAxhE5Xb48vsOyjDh6RWGjBSSvwOFXlaPN9cDTSjtFQ4FwARAQAB
tC1raGFlZWRhIChHUEcgS2V5cykgPHNoYWlra2hhbGVlZGExQGdtYWlsLmNvbT6J
AlEEEwEIADsWIQTXWhrlanDwrqXJF7JVXPuLCH4CngUCZ4nhJAIbAwULCQgHAgIi
AgYVCgkICwIEFgIDAQIeBwIXgAAKCRBVXPuLCH4CnuLID/wMyPvXfJeV//u7j5Ac
wlUO43Zk2KLM/GAZ4/vgYDGN15b9tXPmh+/cwrQxU7b19P9LCnfULjj7KyiP4NF/
NCiKtGQnRGcwIryzzOoOXhI3HJ8BCuNslzGGmHlW68UwxjpXHB53uizEksOk4KZ
95P6hHL2dWS127C+2Mqq47UeU//8CO7hoQS/gcUmuEfuLzNxAusn7iCjw69RYWYK
1VJrjY+EMgCz6onPcQ9TJ5qz9D4b37+SNP9z9IB+vRf+ASRom3I5Dql97qOXTru
QRDc+9iykLTjHNLA89RO8Mv36PnsHFFF5sa3utWykRsBGGwvNPXOohveBrbKrFyx
Qxeg13+ws4WE7XgI+orZNOK1HIhC2N3iGAJoFvtyLPZUxeKyg/w7FEIb2vC8pzPe
Yq8A/DWpYz4G9nG6zU+ZOSTnJSo5V+BVcxORO4UqgnVzpTsCZAquW2OO8DoS4AqD
OuCwm/NV99btbzp1C9y2MxzLnZqVg/2k/Jbb+nxllMz49INcd98wmxJNJR9F2aYG
cT6ep2DEhAElEKcFrip3ZZMUB44ykT/pjgRUPrR+e8nsuPvrUm9GIaQYkIX8TKFC
5Ez5wm+O2W+MGwTBcQQSMzt9kAB85p6PAiA9dhVhuvP7Zb9OO5+KYuEP4C2uA+ca
aO3Wj7FJGA6241bH5DvXQLOBZLkCDQRnieEkARAAxmGxmXg7gN5njzQfub+Y+QSy
mBjH1Q7++n4nX6IFNqUeiDNPpQNDZOQOOAg5HeRTx2cJN2iWZHXdiN6et8tauzfe
oYzkEkJ+kw9UvlOe6tQAIi6So6NIrVoQWBzIgMlsYsWBIKYgVUVjeXqak5+Fy67x
D6TLeVaK8uaVpZKGnoCOjSfOkD4TgjiHZXj+Q5bHZSvqknfUBnTl1oUe65bzEjU7
gNoQx8y22icQiO6txzBT1koPM+eGjaYDRW1wiFWIeGzdrk5OYHF6IasGwBL/kQER
t8uIUSVEsHMH9r59YST1iJQS54x1S6zy3qL6t1uowbT2fyqnUIF2c2DuMiMi1ixG
T7VOLCD3TwhTJ2JkHTlIwsqCxOQrI7TV1xsOWN2N4X7hGfZlghBf/gPX+ao3zkcJ
GylO3rGWonSUWRiiNkBPPWakkiK19ndeFSmVbpsnOhRMGlWO9n+O8GGp8bxRtw1U
MBLIK54MSmP+ZR7Y2koQMCQed/9jF9QNOXDCwLHHGnVGrgI2sPUUS+QKt6gtRlE/
rY6eMsUb9O5O984RNfvd7o5D3yDsfD35Hx06QYegNURv8Hw8fQVkxAJORbSkGR9t
1P3ST1vKQTe/A/xzVo//IXg7Cr7tuqzps+W/wpc+vVuP/uL/AGU25qFaWyUkhvP9
tRtZ1p/OhalLxzgXh/sAEQEAAYkCNgQYAQgAIBYhBNdaGuVqcPCupckXslVc+4sI
fgKeBQJnieEkAhsMAAoJEFVc+4sIfgKevsIP/AorVPuXDrVtIAgtdWUtIMAi+bxf
idp54yft3/JfmYlvkdCnoBOVXhFi31rBmEnKV2CZxkusJQblWp5DScnugb8VuhcP
xLW1Ow9nfLevj1izfg+ot2f/fgQ58O6MI1PloUSfnqbH11GgIEe/H99O9O1X1+x
zDn9UUO6tzLL2pqPYaRZidaGq94atHrEi8a2b3L9VtTqqij6S4IeLyX6itzOhYkO
Fi3/Vf5XAaROFN1FgvOlOQbmxWjw4aOOMt6ifaZOsvxWy1dQrUfs3U1OXnTUAjBn
OLr5NR+ERW3aIAIFKqRW4EIiney4nOj5oUv8v58gdIy7tGjXBkEfirdmVYkC6ydF
HxWMaRu1E1JFwHtpXqKYWlfMHlzIKXDPlAU3ta5pmQO4jtUcZca+VbAAVO2+XZMB
T7IN9rfr6nx4HNJSTvNKz+zMPhx98tyoah6Mi+OOEM/AsN3/18XBtHFQfF4CSwrz
IreZBmrLh1+hvgogPESuW+PQ52kSfJzG8t1N/jLOzSmoOTphARa+zWDzv6iJhl5A
Pqm6nAz3q8tOaARE/D8U5+6tzgIS0AxNCAF93U9yIqOviAhQI/rV9ofPUJztGOoK
NZFETnoxV7Qr2p653LyyAvkZ5yPGZb55QxZ/qPOEvDJ27WduS4Cf38Kq7/bDFZeh
wXowWmCh6WqYzDr
=51A3
-----END PGP PUBLIC KEY BLOCK-----
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ gpg --armor --export shaikkhaleeda1@gmail.com >> a.txt
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ git push
Everything up-to-date
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$
Type here to search
ENG
US
11:05
17-01-2025


MINGW64:/c/Users/Administrator/lfs
wlUO43Zk2KLM/GAZ4/vgYDGN15b9tXPmh+/cwrQxU7b19P9LCnfULjj7KyiP4NF/
NCiKtGQnRGcwIryzzOoOXhI3HJ8BCuNslzGGmHlW68UwxjpXHB53uizEksOk4KZ
95P6hHL2dWS127C+2Mqq47UeU//8CO7hoQS/gcUmuEfuLzNxAusn7iCjw69RYWYK
1VJrjY+EMgCz6onPcQ9TJ5qz9D4b37+SNP9z9IB+vRf+ASRom3I5Dql97qOXTru
QRDc+9iykLTjHNLA89RO8Mv36PnsHFFF5sa3utWykRsBGGwvNPXOohveBrbKrFyx
Qxeg13+ws4WE7XgI+orZNOK1HIhC2N3iGAJoFvtyLPZUxeKyg/w7FEIb2vC8pzPe
Yq8A/DWpYz4G9nG6zU+ZOSTnJSo5V+BVcxORO4UqgnVzpTsCZAquW2OO8DoS4AqD
OuCwm/NV99btbzp1C9y2MxzLnZqVg/2k/Jbb+nxllMz49INcd98wmxJNJR9F2aYG
cT6ep2DEhAElEKcFrip3ZZMUB44ykT/pjgRUPrR+e8nsuPvrUm9GIaQYkIX8TKFC
5Ez5wm+O2W+MGwTBcQQSMzt9kAB85p6PAiA9dhVhuvP7Zb9OO5+KYuEP4C2uA+ca
aO3Wj7FJGA6241bH5DvXQLOBZLkCDQRnieEkARAAxmGxmXg7gN5njzQfub+Y+QSy
mBjH1Q7++n4nX6IFNqUeiDNPpQNDZOQOOAg5HeRTx2cJN2iWZHXdiN6et8tauzfe
oYzkEkJ+kw9UvlOe6tQAIi6So6NIrVoQWBzIgMlsYsWBIKYgVUVjeXqak5+Fy67x
D6TLeVaK8uaVpZKGnoCOjSfOkD4TgjiHZXj+Q5bHZSvqknfUBnTl1oUe65bzEjU7
gNoQx8y22icQiO6txzBT1koPM+eGjaYDRW1wiFWIeGzdrk5OYHF6IasGwBL/kQER
t8uIUSVEsHMH9r59YST1iJQS54x1S6zy3qL6t1uowbT2fyqnUIF2c2DuMiMi1ixG
T7VOLCD3TwhTJ2JkHTlIwsqCxOQrI7TV1xsOWN2N4X7hGfZlghBf/gPX+ao3zkcJ
GylO3rGWonSUWRiiNkBPPWakkiK19ndeFSmVbpsnOhRMGlWO9n+O8GGp8bxRtw1U
MBLIK54MSmP+ZR7Y2koQMCQed/9jF9QNOXDCwLHHGnVGrgI2sPUUS+QKt6gtRlE/
rY6eMsUb9O5O984RNfvd7o5D3yDsfD35Hx06QYegNURv8Hw8fQVkxAJORbSkGR9t
1P3ST1vKQTe/A/xzVo//IXg7Cr7tuqzps+W/wpc+vVuP/uL/AGU25qFaWyUkhvP9
tRtZ1p/OhalLxzgXh/sAEQEAAYkCNgQYAQgAIBYhBNdaGuVqcPCupckXslVc+4sI
fgKeBQJnieEkAhsMAAoJEFVc+4sIfgKevsIP/AorVPuXDrVtIAgtdWUtIMAi+bxf
idp54yft3/JfmYlvkdCnoBOVXhFi31rBmEnKV2CZxkusJQblWp5DScnugb8VuhcP
xLW1Ow9nfLevj1izfg+ot2f/fgQ58O6MI1PloUSfnqbH11GgIEe/H99O9O1X1+x
zDn9UUO6tzLL2pqPYaRZidaGq94atHrEi8a2b3L9VtTqqij6S4IeLyX6itzOhYkO
Fi3/Vf5XAaROFN1FgvOlOQbmxWjw4aOOMt6ifaZOsvxWy1dQrUfs3U1OXnTUAjBn
OLr5NR+ERW3aIAIFKqRW4EIiney4nOj5oUv8v58gdIy7tGjXBkEfirdmVYkC6ydF
HxWMaRu1E1JFwHtpXqKYWlfMHlzIKXDPlAU3ta5pmQO4jtUcZca+VbAAVO2+XZMB
T7IN9rfr6nx4HNJSTvNKz+zMPhx98tyoah6Mi+OOEM/AsN3/18XBtHFQfF4CSwrz
IreZBmrLh1+hvgogPESuW+PQ52kSfJzG8t1N/jLOzSmoOTphARa+zWDzv6iJhl5A
Pqm6nAz3q8tOaARE/D8U5+6tzgIS0AxNCAF93U9yIqOviAhQI/rV9ofPUJztGOoK
NZFETnoxV7Qr2p653LyyAvkZ5yPGZb55QxZ/qPOEvDJ27WduS4Cf38Kq7/bDFZeh
wXowWmCh6WqYzDr
=51A3
-----END PGP PUBLIC KEY BLOCK-----
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ gpg --armor --export shaikkhaleeda1@gmail.com >> a.txt
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$ git push
Everything up-to-date
Administrator@5db992294e86535 MINGW64 ~/lfs (main)
$
Type here to search
ENG
US
11:05
17-01-2025

Why Use Signed Commits?

Verify Author Identity:
A signed commit ensures the commit's author is who they claim to be, helping to prevent impersonation.

Security and Trust:
It helps maintain the security and authenticity of the codebase, especially when collaborating with external contributors or in a team environment.

Compliance and Auditing:
In some organizations, signed commits are a requirement for auditing purposes to ensure that the correct person made the changes.

Prevent Tampering:
Signed commits can help ensure that commits are not tampered with after being pushed to the repository.