# GIT

---

## DAY 1

### SCM (Source Code Management) and Git

### 1. SCM (Source Code Management):

- **What is SCM?**
  - SCM refers to tools and practices used to manage changes in source code, track versions, and handle code revisions.
  - SCM systems help manage the history of code changes and collaborate among multiple developers working on a project.

**Key Benefits of Using SCM:**

- **Collaboration:** Multiple developers can work on the same project simultaneously without overwriting each other's work.
- **Version Control:** Tracks all code changes, allowing developers to go back to previous versions if needed.
- **Code Review:** Changes can be reviewed before merging into the main codebase.
- **Risk Mitigation:** Prevents loss of code and ensures that any mistakes or bugs can be traced back and corrected.
- **Improved Productivity:** Developers can work in parallel, switch between versions, and merge changes easily.

---

### Git: A Distributed SCM Tool

### 1. Git Overview:

- **What is Git?**
  - Git is a **distributed version control system**. It helps track changes to code, manage versions, and collaborate on development.
  - Git is **incremental**, meaning it stores only the changes made (not the whole file each time).

### 2. Core Concepts in Git:

- **Commit:**
  - A commit is like a **snapshot** of your project at a specific point in time.
  - Each commit in Git records changes to the files and is assigned a unique ID (commit hash).
- **Incremental Versioning:**

    - Git tracks only changes (additions, deletions, modifications) rather than copying the entire file, making it more efficient.
    - **Example:**
      If you're traveling from **Tvm** (Thiruvananthapuram) to **Kazhakootam**, you only note the changes (Tvm to Kazhakootam). On another trip, you record all stops (Tvm → Chakka → Infosys → Tvm) but avoid repeating the same information.
- **Branching:**
    - Git allows developers to **branch** off the main codebase, make changes, and then merge them back. This lets developers work on different features simultaneously without interfering with each other's code.
    - **Feature Branching:** Each new feature is developed in a separate branch, keeping the main codebase stable.
- **Repository:**
    - A Git repository is where your project's **version history** is stored. It holds all the commits, branches, and information about your project.
- **Version History:**
    - Git keeps a detailed **history** of all changes made to the codebase. This helps you track progress, go back to previous states, and find bugs introduced in past commits.

---

**Benefits of Using Git:**

- **Collaboration:** Multiple developers can contribute to the same project without conflict.
- **Version Control:** Git keeps track of changes and allows reverting to previous versions of the code.
- **Code Review:** Changes can be reviewed before being merged into the main branch.
- **Risk Mitigation:** Mistakes are easier to identify and fix since all changes are recorded.
- **Improved Productivity:** Git allows for parallel development, easy switching between versions, and faster project management.

---

**Installing Git:**

**For Ubuntu (Debian-based systems):**

- To install Git:
- sudo apt install git -y

**For Fedora (Red Hat-based systems):**

- To install Git:
- sudo dnf install git -y

---

**Basic Git Commands:**

1. **git init**: Initializes a new Git repository in your project directory.
2. **git add .**: Stages all changes in your project for the next commit.
3. **git commit -m "message"**: Commits your staged changes with a message describing the changes.
4. **git push**: Pushes the local commits to the remote repository (e.g., GitHub).
5. **git pull**: Pulls the latest changes from the remote repository.
6. **git status**: Shows the current status of your files (modified, staged, untracked, etc.).
7. **git log --oneline**: Shows a simplified commit history with each commit's hash and message.
8. **git branch**: Lists the branches in your repository.
9. **git checkout branch-name**: Switches to the specified branch.
10. **git merge branch-name**: Merges changes from the specified branch into the current branch.

**Conclusion:**

- **Git** is an essential tool for **version control** and **collaboration** in software development.
- It tracks changes, helps developers work together, and allows efficient management of code with **branches**, **commits**, and **repositories**.
- **Installing Git** and getting started with the basic commands enables you to effectively manage code and keep a detailed history of all project changes.

---

# DAY 2

| Commands | Description |
|---|---|
| git add <file> | Adds a specific file to the staging area. |
| git add . or git add –all | Adds all modified and new files to the staging area. |
| git status | Shows the current state of your repository, including tracked and untracked files, modified files, and branch information. |
| git status –ignored | Displays ignored files in addition to the regular status output. |
| git diff | Shows the changes between the working directory and the staging area (index). |
| git diff <commit1> <commit2> | Displays the differences between two commits. |
| git diff –staged or git diff –cached | Displays the changes between the staging area (index) and the last commit. |
| git diff HEAD | Display the difference between the current directory and the last commit |
| git commit | Creates a new commit with the changes in the staging area and opens the default text editor for adding a commit message. |
| git commit -m "<message>" or git commit –message "<message>" | Creates a new commit with the changes in the staging area and specifies the commit message inline. |

**Getting & Creating Projects**

>git init - Initialize a local Git repository git clone

>ssh://git@github.com/[username]/[repository-name].git

Create a local copy of a remote repository

**Basic Snapshotting**

>git status - Check status

>git add [file-name.txt] - Add a file to the staging area

> git add -A - Add all new and changed files to the staging area

>git commit -m "[commit message]" - Commit changes

>git rm -r [file-name.txt] - Remove a file (or folder)

>git remote -v - View the remote repository of the currently working file or directory

**Branching & Merging**

>git branch - a List all branches (local and remote)

> git branch [branch name] - Create a new branch

>git branch -d [branch name] - Delete a branch

>git push origin --delete [branch name] - Delete a remote branch

>git checkout -b [branch name] - Create a new branch and switch to it

>git branch -m [old branch name] [new branch name] - Rename a local branch

>git checkout [branch name] - Switch to a branch

>git checkout - Switch to the branch last checked out

>git checkout --[file-name.txt] - Discard changes to a file

>git merge [branch name] - Merge a branch into the active branch

>git merge [source branch] [target branch] - Merge a branch into a target branch

>git stash  - Stash changes in a dirty working directory

>git stash clear - Remove all stashed entries

>git stash pop - Apply latest stash to working directory

**Sharing & Updating Projects**

>push origin [branch name]  - Push a branch to your remote repository

>git push -u origin [branch name] -  Push changes to remote repository

>git push origin --delete [branch name]  - Delete a remote branch

>git pull  - Update local repository to the newest commit

>git pull origin [branch name] -  Pull changes from remote repository

Inspection & Comparison

>git log - View changes

>git log --oneline  - View changes (briefly)

>git diff [source branch] [target branch]  - Preview changes before mergin

---

**DAY 3**

**Restoring**

>git restore  - for restoring from staging to working directory

>git restore  - workspace for restoring from local repository to working directory

>git restore  - staged for restoring from local repository to staging

# Pull request

```
Administrator@bf1f60d13ed8554 MINGW64 ~ (master)
$ mkdir myustt

Administrator@bf1f60d13ed8554 MINGW64 ~ (master)
$ cd myustt

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt (master)
$ git init
Initialized empty Git repository in C:/Users/Administrator/myustt/.git/

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt (master)
$ echo "HEllo world" >> hello.txt

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt (master)
$ ls
hello.txt

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt (master)
$ git add .
warning: in the working copy of 'hello.txt', LF will be replaced by CRLF the next time Git touches it

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt (master)
$ git commit -m "a1"
[master (root-commit) 8603daf] a1
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt (master)
$ git remote add origin git@github.com:SelmiNazeeb/myustt.git

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 222 bytes | 111.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:SelmiNazeeb/myustt.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt (master)
$ git checkout -b feature1
Switched to a new branch 'feature1'

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt (feature1)
```

```
dministrator@bf1f60d13ed8554 MINGW64 ~/myustt (feature1)
$ git branch
* feature1
  master

dministrator@bf1f60d13ed8554 MINGW64 ~/myustt (feature1)
$ echo "learning pull request" >> hello.txt

dministrator@bf1f60d13ed8554 MINGW64 ~/myustt (feature1)
$ cat hello.txt
HEllo world
learning pull request

dministrator@bf1f60d13ed8554 MINGW64 ~/myustt (feature1)
$ git add .
warning: in the working copy of 'hello.txt', LF will be replaced by CRLF the next time Git touches it

dministrator@bf1f60d13ed8554 MINGW64 ~/myustt (feature1)
$ git commit -m "learning pull request"
[feature1 b2d32d4] learning pull request
 1 file changed, 1 insertion(+)

dministrator@bf1f60d13ed8554 MINGW64 ~/myustt (feature1)
$ git push -u orgin feature1
fatal: 'orgin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.

dministrator@bf1f60d13ed8554 MINGW64 ~/myustt (feature1)
$ git push -u origin feature1
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 287 bytes | 143.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'feature1' on GitHub by visiting:
remote:      https://github.com/SelmiNazeeb/myustt/pull/new/feature1
remote:
To github.com:SelmiNazeeb/myustt.git
 * [new branch]      feature1 -> feature1
branch 'feature1' set up to track 'origin/feature1'.
```

# Git ignore

```
Administrator@bf1f60d13ed8554 MINGW64 ~/myustt (master)
$ mkdir newignore

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt (master)
$ cd newignore

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ git init
Initialized empty Git repository in C:/Users/Administrator/myustt/newignore/.git/

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ echo "hii hello" >> abc.class

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ ls
abc.class

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ git status
On branch master

No commits yet

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

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

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ git add .
warning: in the working copy of 'abc.class', LF will be replaced by CRLF the next time Git touches it

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ git commit -m "class file will not be ignored"
[master (root-commit) 4cf54f1] class file will not be ignored
 1 file changed, 1 insertion(+)
 create mode 100644 abc.class

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ ls
abc.class
```

```
Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ ls -a
./  ../  .git/  .gitignore  abc.class

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ cp abc.class newfile.class

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ ls
abc.class  newfile.class

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

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

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ echo "new line" >> abc.class

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ git status
On branch master
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:   abc.class

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

no changes added to commit (use "git add" and/or "git commit -a")

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ git add .
warning: in the working copy of 'abc.class', LF will be replaced by CRLF the next time Git touches it
```

```
Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
        modified:   abc.class


Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ git diff

Administrator@bf1f60d13ed8554 MINGW64 ~/myustt/newignore (master)
$ git diff --cached
diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..38105ec
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1,24 @@
+# Compiled class file
+*.class
+
+# Log file
+*.log
+
+# BlueJ files
+*.ctxt
+
+# Mobile Tools for Java (J2ME)
+.mtj.tmp/
+
+# Package Files #
+*.jar
+*.war
+*.nar
+*.ear
+*.zip
+*.tar.gz
+*.rar
+
+# virtual machine crash logs, see http://www.java.com/en/download/help/error_hotspot.xml
+hs_err_pid*
+replay_pid*
\ No newline at end of file
diff --git a/abc.class b/abc.class
index 8f763b7..9ff93b4 100644
--- a/abc.class
+++ b/abc.class
@@ -1 +1,2 @@
 hii hello
+new line
```

# Merge and Delete branch

```
Administrator@bf1f60d13ed8554 MINGW64 ~/testust (master)
$ git branch
* master

Administrator@bf1f60d13ed8554 MINGW64 ~/testust (master)
$ git checkout -b feature
Switched to a new branch 'feature'

Administrator@bf1f60d13ed8554 MINGW64 ~/testust (feature)
$ git branch
* feature
  master

Administrator@bf1f60d13ed8554 MINGW64 ~/testust (feature)
$ git checkout master
Switched to branch 'master'

Administrator@bf1f60d13ed8554 MINGW64 ~/testust (master)
$ git checkout feature
Switched to branch 'feature'

Administrator@bf1f60d13ed8554 MINGW64 ~/testust (feature)
$ echo "test" >> text.ust

Administrator@bf1f60d13ed8554 MINGW64 ~/testust (feature)
$ git add .
warning: in the working copy of 'text.ust', LF will be replaced by CRLF the next time Git touches it

Administrator@bf1f60d13ed8554 MINGW64 ~/testust (feature)
$ git commit -m "new test added"
[feature a009131] new test added
 1 file changed, 1 insertion(+)
 create mode 100644 text.ust

Administrator@bf1f60d13ed8554 MINGW64 ~/testust (feature)
$ git checkout master
Switched to branch 'master'

Administrator@bf1f60d13ed8554 MINGW64 ~/testust (master)
$ git branch -d feature
error: the branch 'feature' is not fully merged
hint: If you are sure you want to delete it, run 'git branch -D feature'
hint: Disable this message with "git config advice.forceDeleteBranch false"
```

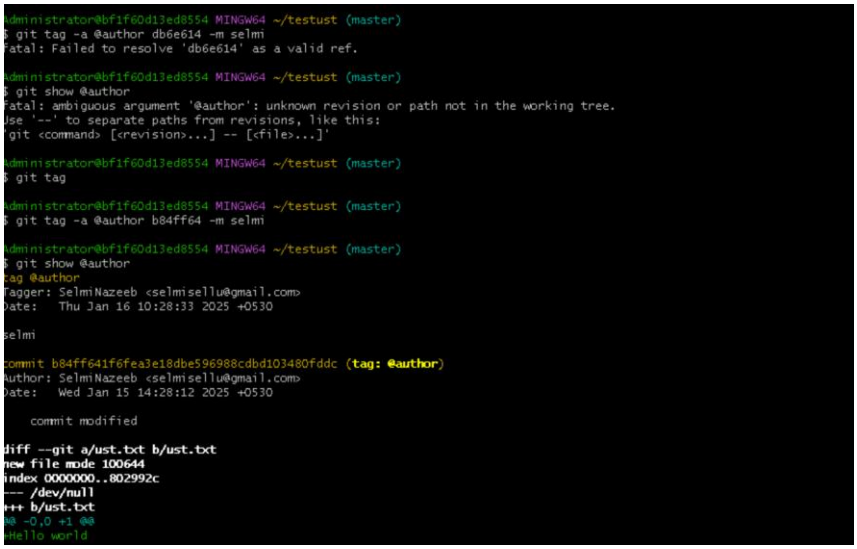
```
Administrator@bf1f60d13ed8554 MINGW64 ~/testust (master)
$ git merge feature
Updating fb2c5e2..a009131
Fast-forward
 text.ust | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 text.ust

Administrator@bf1f60d13ed8554 MINGW64 ~/testust (master)
$ git branch -d feature
Deleted branch feature (was a009131).
```

```
Administrator@bf1f60d13ed8554 MINGW64 /myustt (master)
$ git checkout --track origin/B1
branch 'B1' set up to track 'origin/B1'.
Switched to a new branch 'B1'

Administrator@bf1f60d13ed8554 MINGW64 /myustt (B1)
$ git branch
* B1
  master

Administrator@bf1f60d13ed8554 MINGW64 /myustt (B1)
$ ls
hello.txt  hello_world

Administrator@bf1f60d13ed8554 MINGW64 /myustt (B1)
$ git push origin --delete B1
To github.com:SelmiNazeeb/myustt.git
 - [deleted]         B1

Administrator@bf1f60d13ed8554 MINGW64 /myustt (B1)
$ git branch
* B1
  master

Administrator@bf1f60d13ed8554 MINGW64 /myustt (B1)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Administrator@bf1f60d13ed8554 MINGW64 /myustt (master)
$ git branch
  B1
* master

Administrator@bf1f60d13ed8554 MINGW64 /myustt (master)
$ git branch -D B1
Deleted branch B1 (was 4584c6e).

Administrator@bf1f60d13ed8554 MINGW64 /myustt (master)
$ git branch
* master
```

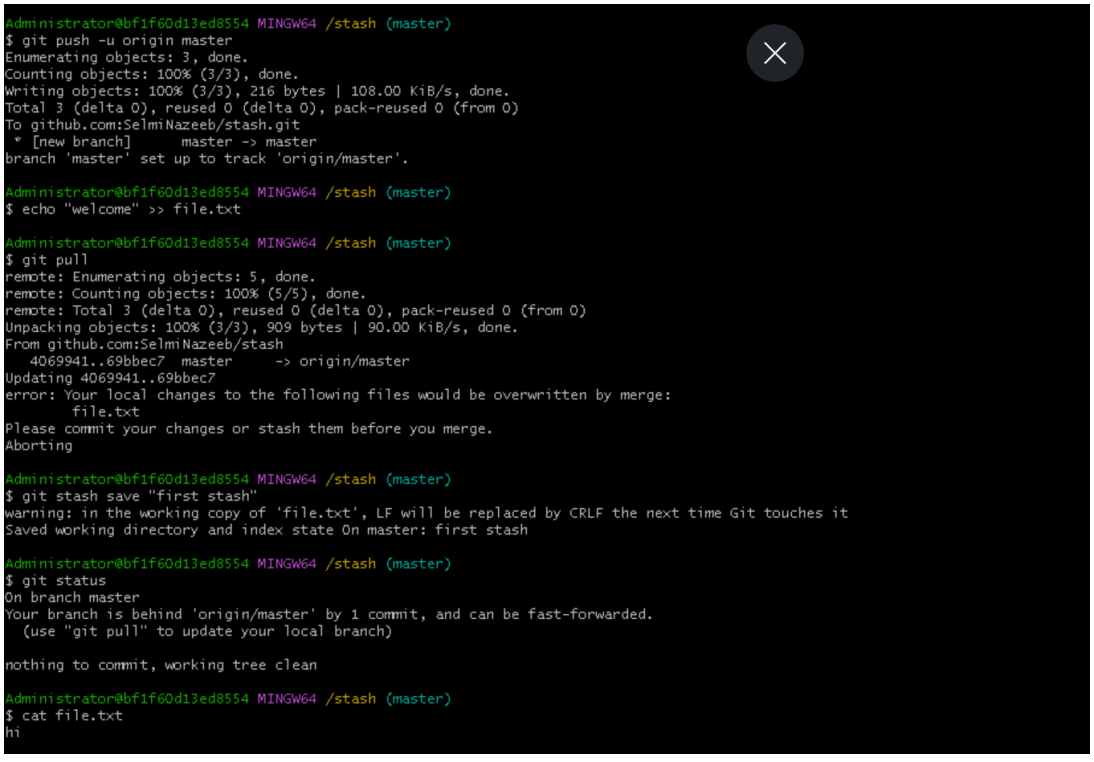
---

# DAY 4

## Tagging




## Stashing



```
Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git pull
Updating 4069941..69bbec7
Fast-forward
 file.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ cat file.txt
hi welcome to stash

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git stash list
stash@{0}: On master: first stash

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git stash apply
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
On branch master
Your branch is up to date with 'origin/master'.

Unmerged paths:
  (use "git restore --staged <file>..." to unstage)
  (use "git add <file>..." to mark resolution)
        both modified:   file.txt

no changes added to commit (use "git add" and/or "git commit -a")

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ cat file.txt
<<<<<<< Updated upstream
hi welcome to stash
=======
hi
welcome
>>>>>>> Stashed changes

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git add .

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git commit -m "v1"
[master ce3b2f9] v1
 1 file changed, 5 insertions(+)
```

```
Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 307 bytes | 307.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:SelmiNazeeb/stash.git
   69bbec7..ce3b2f9  master -> master

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git stash list
stash@{0}: On master: first stash

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git stash pop
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
On branch master
Your branch is up to date with 'origin/master'.

Unmerged paths:
  (use "git restore --staged <file>..." to unstage)
  (use "git add <file>..." to mark resolution)
        both modified:   file.txt

no changes added to commit (use "git add" and/or "git commit -a")
The stash entry is kept in case you need it again.

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git stash list
stash@{0}: On master: first stash
```

```
Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git stash drop stash@{0}
Dropped stash@{0} (43af7a9b465e71394dcf638592e8a8ce7ec7d762)

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git stash list

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ cat file.txt
<<<<<<< Updated upstream
hi welcome to stash
=======
hi
welcome
<<<<<<< Updated upstream
>>>>>>> Stashed changes
=======
>>>>>>> Stashed changes

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git stash save "2nd change"
file.txt: needs merge
error: could not write index

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git add .

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git stash save "2nd change"
Saved working directory and index state On master: 2nd change

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git stash list
stash@{0}: On master: 2nd change

Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git stash show stash@{0}
 file.txt | 3 +++
 1 file changed, 3 insertions(+)
```

```
Administrator@bf1f60d13ed8554 MINGW64 /stash (master)
$ git stash show -p stash@{0}
diff --git a/file.txt b/file.txt
index 90d2bc3..920343b 100644
--- a/file.txt
+++ b/file.txt
@@ -3,4 +3,7 @@ hi welcome to stash
 =======
 hi
 welcome
+<<<<<<< Updated upstream
+>>>>>>> Stashed changes
+=======
 >>>>>>> Stashed changes
```

# Revert

```
Administrator@bf1f60d13ed8554 MINGW64 /
$ mkdir testgit

Administrator@bf1f60d13ed8554 MINGW64 /
$ cd testgit/

Revert "m2"
```

```
[master 30886b4] Revert "m2"
 1 file changed, 1 deletion(-)
 delete mode 100644 file2.txt

Administrator@bf1f60d13ed8554 MINGW64 /testgit (master)
$ ls
file1.txt  file3.txt

Administrator@bf1f60d13ed8554 MINGW64 /testgit (master)
$ git log --oneline
30886b4 (HEAD -> master) Revert "m2"
80fe567 m3
edf02fc m2
7b432a4 m1
```

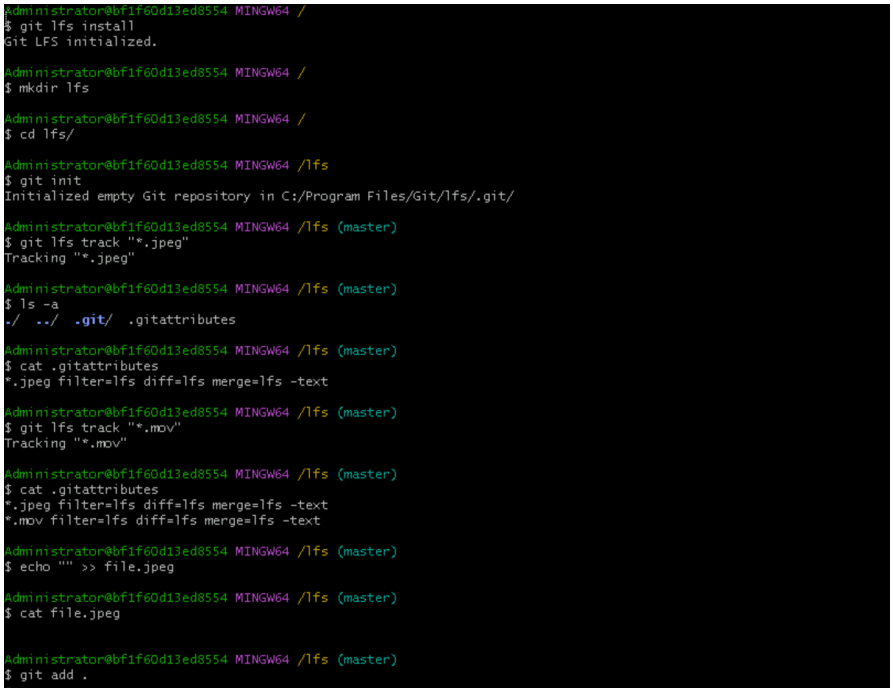
## Reset

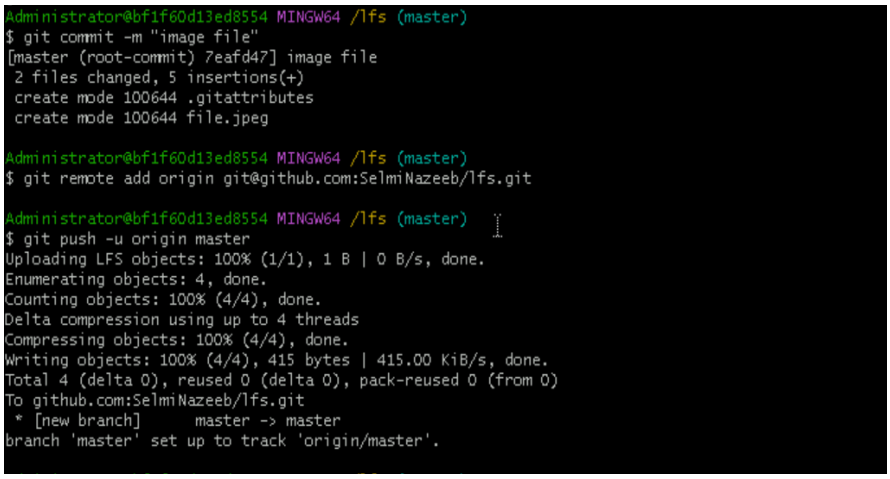
**>Soft** : moves the branch pointer (effects only in local repository) You will remove the last commit from the current branch, but the file changes will stay in working tree and in index

**>Hard**: it removes in all stages

**> Mixed**: moves the branch pointer and Default mode. You will still keep the changes in your working tree but not on inde

## Lfs



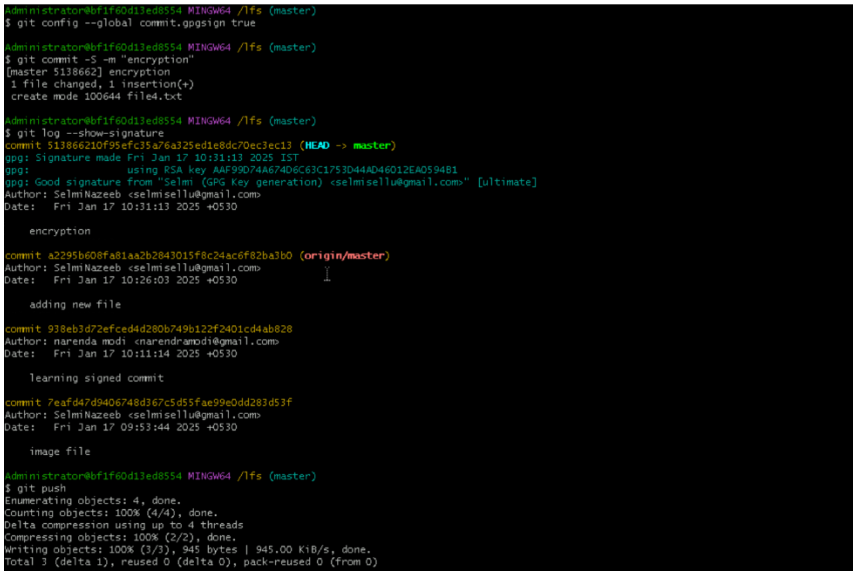



# Signed commit

```
Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ git commit -m "learning signed commit" --author="narenda modi<narendramodi@gmail.com>"
[master 938eb3d] learning signed commit
 Author: narenda modi <narendramodi@gmail.com>
 1 file changed, 1 insertion(+)
 create mode 100644 file2.txt

Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ git log
commit 938eb3d72efced4d280b749b122f2401cd4ab828 (HEAD -> master)
Author: narenda modi <narendramodi@gmail.com>
Date:   Fri Jan 17 10:11:14 2025 +0530

    learning signed commit

commit 7eafd47d9406748d367c5d55fae99e0dd283d53f (origin/master)
Author: SelmiNazeeb <selmisellu@gmail.com>
Date:   Fri Jan 17 09:53:44 2025 +0530

    image file
```

```
Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ gpg --full-generate-key
gpg (GnuPG) 2.4.5-unknown; Copyright (C) 2024 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory '/c/Users/Administrator/.gnupg' created
Please select what kind of key you want:
   (1) RSA and RSA
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
   (9) ECC (sign and encrypt) *default*
  (10) ECC (sign only)
  (14) Existing key from card
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 4096
Requested keysize is 4096 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Selmi
Email address: selmisellu@gmail.com
Comment: GPG Key generation
You selected this USER-ID:
    "Selmi (GPG Key generation) <selmisellu@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.
```

```
pub   rsa4096 2025-01-17 [SC]
      AAF99D74A674D6C63C1753D44AD46012EA0594B1
uid                      Selmi (GPG Key generation) <selmisellu@gmail.com>
sub   rsa4096 2025-01-17 [E]


Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ gpg --list-secret-keys --keyid-format long
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
[keyboxd]
---------
sec   rsa4096/4AD46012EA0594B1 2025-01-17 [SC]
      AAF99D74A674D6C63C1753D44AD46012EA0594B1
uid                 [ultimate] Selmi (GPG Key generation) <selmisellu@gmail.com>
ssb   rsa4096/D203830DA873EB71 2025-01-17 [E]


Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ echo "newfile" >> file3.txt

Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ git add .
warning: in the working copy of 'file3.txt', LF will be replaced by CRLF the next time Git touches it

Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ git commit -m "adding new file"
[master a2295b6] adding new file
 1 file changed, 1 insertion(+)
 create mode 100644 file3.txt

Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 571 bytes | 190.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To github.com:SelmiNazeeb/lfs.git
   7eafd47..a2295b6  master -> master

Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ echo "newfile again" >> file4.txt

Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ git add .
warning: in the working copy of 'file4.txt', LF will be replaced by CRLF the next time Git touches it

Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ git commit -S -m "encryption"
error: gpg failed to sign the data:
gpg: skipped "SelmiNazeeb <selmisellu@gmail.com>": No secret key
[GNUPG:] INV_SGNR 9 SelmiNazeeb <selmisellu@gmail.com>
[GNUPG:] FAILURE sign 17
gpg: signing failed: No secret key

fatal: failed to write commit object

Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ git config --global user.signingkey 4AD46012EA0594B1

Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ git log
commit a2295b608fa81aa2b2843015f8c24ac6f82ba3b0 (HEAD -> master, origin/master)
Author: SelmiNazeeb <selmisellu@gmail.com>
Date:   Fri Jan 17 10:26:03 2025 +0530

    adding new file

commit 938eb3d72efced4d280b749b122f2401cd4ab828
Author: narenda modi <narendramodi@gmail.com>
Date:   Fri Jan 17 10:11:14 2025 +0530

    learning signed commit

commit 7eafd47d94067483677c5d55fae99e0dd283d53f
Author: SelmiNazeeb <selmisellu@gmail.com>
Date:   Fri Jan 17 09:53:44 2025 +0530

    image file
```


Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ git config --global commit.gpgsign true
Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ git commit -S -m "encryption"
[master 5138662] encryption
1 file changed, 1 insertion(+)
create mode 100644 file4.txt
Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ git log --show-signature
commit 513866210f95efc35a76a325ed1e8dc70ec3ec13 (HEAD -> master)
gpg: Signature made Fri Jan 17 10:31:13 2025 IST
gpg: using RSA key AAF99D74A674D6C63C1753D44AD46012EA0594B1
gpg: Good signature from "Selmi (GPG Key generation) <selmisellu@gmail.com>" [ultimate]
Author: SelmiNazeeb <selmisellu@gmail.com>
Date: Fri Jan 17 10:31:13 2025 +0530
encryption
commit a2295b608fa81aa2b2843015f8c24ac6f82ba3b0 (origin/master)
Author: SelmiNazeeb <selmisellu@gmail.com>
Date: Fri Jan 17 10:26:03 2025 +0530
adding new file
commit 938eb3d72efced4d280b749b122f2401cd4ab828
Author: narenda modi <narendramodi@gmail.com>
Date: Fri Jan 17 10:11:14 2025 +0530
learning signed commit
commit 7eafd47d94067748d367c5d55fae99e0dd283d53f
Author: SelmiNazeeb <selmisellu@gmail.com>
Date: Fri Jan 17 09:53:44 2025 +0530
image file
Administrator@bf1f60d13ed8554 MINGW64 /lfs (master)
$ 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), 945 bytes | 945.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)