

Table of Contents

Version Control Systems:	2
Git and GitHub	2
Why Git is popular?.....	2
Popular GUI tools for GIT	2
Recommended Basic Configuration.....	2
Basic Commands	3
Staging Area	4
Entering detailed description while commit.....	4
How to ignore files from staging or commit	4
How to remove files only from staging.....	5
Git Diff Tools	5
Global Settings to setup VS Code as default diff tool	5
Branching and Merging.....	5
Rename / Delete the branch:.....	6
Merging, Conflicts and abort merging	6
How to update the last commit without doing new commit	6
Rollback to specified commit.....	6
How to discard local changes / how to overwrite files from higher level	7
How to do get repo from specific commit	7
GIT HUB.....	7
Git Clone.....	8
Git remote.....	8
Git Push	8
Git Fetch.....	8
Git Pull.....	8
What are Pull Requests?	8
Tags	9
Steps to setup SSH in your local to push changes to Git Hub	9
References:	10

Version Control Systems:

VCS can be classified into 2 types

1. Centralized → single point of failure. If the server goes offline, you can't work. Ex: TFS, VSS, SVN
2. Distributed → The every user has their own local copy of the repository. They can even work in offline. Ex: Git

Git and GitHub

- Git is a distributed version control system for tracking changes in source code during software development.
- GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.

Other popular web based repository is Bitbucket, Gitlab, Gerrit etc.

Why Git is popular?

- Free
- Open Source
- Fast
- Scalable

Popular GUI tools for GIT

- Gitkraken
- Source Tree

Recommended Basic Configuration

- Name
- Email
- Default Editor
- Line Ending

Configurations can be done at 3 levels

1. System: Applicable for all users at system level
2. Global: All repositories of current user

3. Local: The current repository

```
$ git config --global user.name "theertha"
$ git config --global user.email theerthaks@gmail.com
$ git config --global core.editor "code --wait" → setting for vs-code
$ git config --global -e → this commands opens up the config in the default editor
$ git config --global core.autocrlf true → this is very crucial to configure end of line indicator
```

Basic Commands

Initializing a repository

```
git init
```

Staging files

```
git add file1.js          # Stages a single file
git add file1.js file2.js # Stages multiple files
git add *.js              # Stages with a pattern
git add .                 # Stages the current directory and all its content
```

Viewing the status

```
git status                # Full status
git status -s             # Short status
```

Committing the staged files

```
git commit -m "Message" # Commits with a one-line message
git commit               # Opens the default editor to type a long message
```

Skipping the staging area

```
git commit -am "Message"
```

Removing files

```
git rm file1.js          # Removes from working directory and staging area
git rm --cached file1.js # Removes from staging area only
```

Renaming or moving files

```
git mv file1.js file1.txt
```

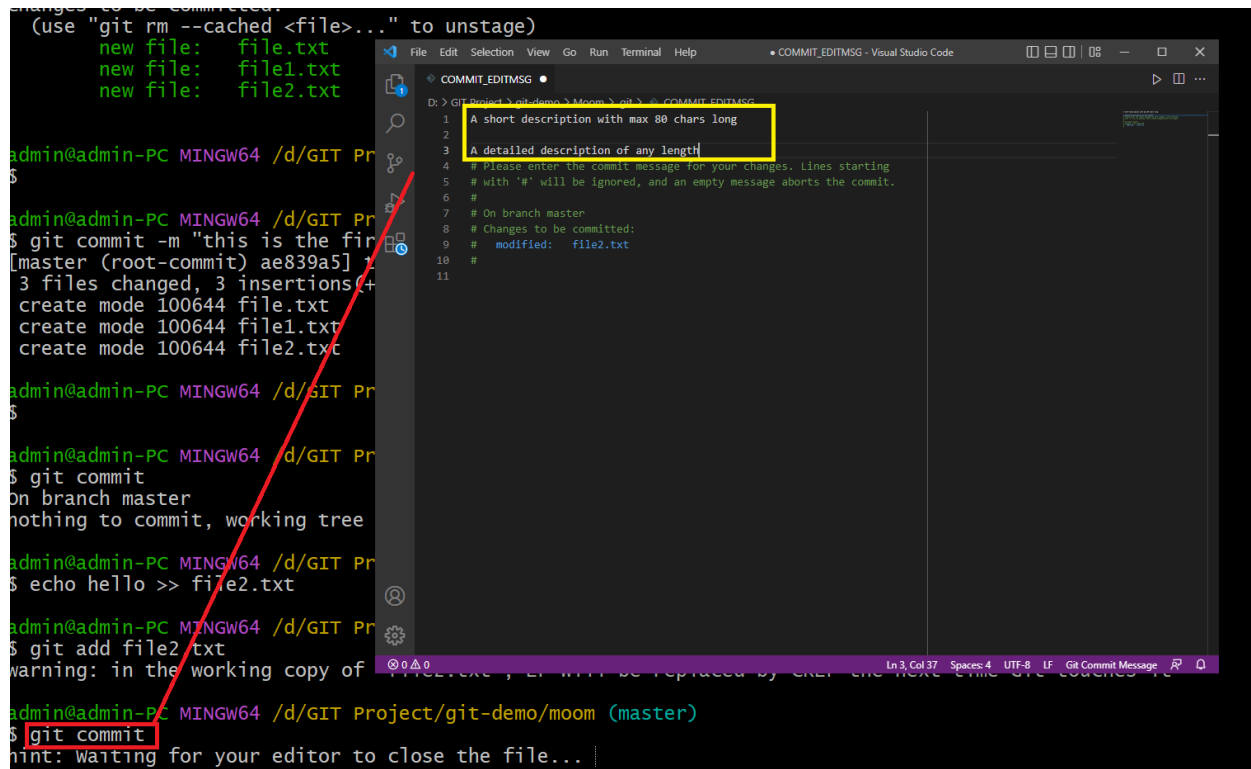
Staging Area

The staging area is intermittent storage where work can be saved before pushing to repo. Staging is optional, if you are confident, then directly can push to Repo.

Once the commit is done from staging to repo, the files still remain in staging.

If you remove the file from work folder as if that is not required, the same should be removed from staging.

Entering detailed description while commit



The screenshot shows a terminal window on the left and a Visual Studio Code editor on the right. The terminal displays the following commands and output:

```
admin@admin-PC MINGW64 /d/GIT Project/git-demo/moom (master)
$ git commit -m "this is the first commit"
[master (root-commit) ae839a5] 3 files changed, 3 insertions(+), 0 deletions(-)
create mode 100644 file.txt
create mode 100644 file1.txt
create mode 100644 file2.txt

admin@admin-PC MINGW64 /d/GIT Project/git-demo/moom (master)
$ git commit
On branch master
nothing to commit, working tree clean

admin@admin-PC MINGW64 /d/GIT Project/git-demo/moom (master)
$ echo hello >> file2.txt

admin@admin-PC MINGW64 /d/GIT Project/git-demo/moom (master)
$ git add file2.txt
warning: in the working copy of file2.txt, LF will be replaced by CRLF. The next time you commit, the difference will be replaced by CRLF. Use 'git add -f' to avoid this.

admin@admin-PC MINGW64 /d/GIT Project/git-demo/moom (master)
$ git commit
```

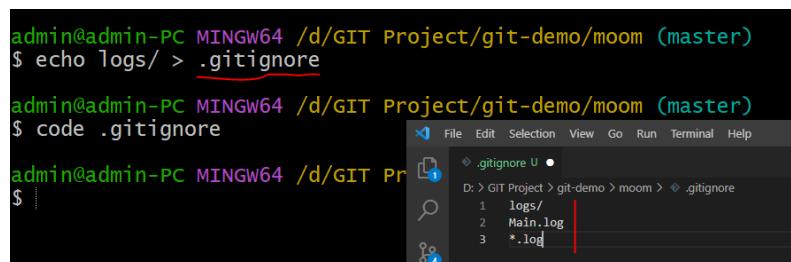
The VS Code editor shows the 'COMMIT_EDITMSG' file with the following content:

```
1 A short description with max 80 chars long
2
3 A detailed description of any length
4
5 Please enter the commit message for your changes. Lines starting
6 # with '#' will be ignored, and an empty message aborts the commit.
7
8 # On branch master
9 # Changes to be committed:
10 #   modified:   file2.txt
11
```

How to ignore files from staging or commit

.gitignore is the file to have a list of files to ignore while committing.

Note: Do not forget to commit the .gitignore file to repo



The screenshot shows a terminal window on the left and a Visual Studio Code editor on the right. The terminal displays the following commands and output:

```
admin@admin-PC MINGW64 /d/GIT Project/git-demo/moom (master)
$ echo logs/ > .gitignore

admin@admin-PC MINGW64 /d/GIT Project/git-demo/moom (master)
$ code .gitignore

admin@admin-PC MINGW64 /d/GIT Project/git-demo/moom (master)
$
```

The VS Code editor shows the '.gitignore' file with the following content:

```
1 logs/
2 Main.log
3 *.log
```

<https://github.com/github/gitignore> ➔ You can get language specific standard templates

How to remove files only from staging

```
$ git rm -h
usage: git rm [<options>] [--] <file>...

-n, --dry-run          dry run
-q, --quiet            do not list removed files
--cached              only remove from the index
-f, --force            override the up-to-date check
-r                   allow recursive removal
--ignore-unmatch       exit with a zero status even if nothing matched
--sparse              allow updating entries outside of the sparse-checkout cone
--pathspec-from-file <file>
                        read pathspec from file
--pathspec-file-nul    with --pathspec-from-file, pathspec elements are separated with NUL character
```

```
$ git rm --cached -r
```

Git Diff Tools

Most of the modern editor has inbuilt tools to compare the code versions. However there exist few popular tools, can use if required

- KDiff3
- P4Merge
- WinMerge
- VSCode

Global Settings to setup VS Code as default diff tool

```
$ git config --global diff.tool vscode
$ git config --global diff.tool.vscode.cmd "code --wait --diff $LOCAL $REMOTE"
```

Note: Sometime the above changes might not properly write to config. Hence open config file and ensure these entries are present

\$ git difftool ➔ opens up the VS Code with staged file and file in working directory

\$ git difftool --staged ➔ opens up the VS Code with staged file and file checked in during last commit

\$ git difftool head ➔ opens up the VS Code with local file and repo file

Branching and Merging

Branching is a way to take a copy from the master node and start working on that copy. When changes are done, **Merge** the changes to master node so that if next developer takes a new branch from master, he gets master node version which has changes done by previous developer.

1. \$ git branch JPNagar_Branch
2. \$ git checkout -b JayaNagar_Branch

Above are the 2 ways to create a branch from the branch you are presently checkout. Option 1 just creates a branch but still not checked out for edit, 2 is create and check out for edit.

Checkout is very important as it moves to HEAD to specified branch to track your changes.

\$ git branch ➔ will list down all the branch names and highlights the current one in green color

```
$ git branch
* NewCar
master
```

Rename / Delete the branch:

Rename: Checkout to the branch and execute `$ git branch -m New_Car`

Delete: Checkout to different branch and execute `$ git branch -d New_Car`

Merging, Conflicts and abort merging

Once you are done with changes in your branch, merge the changes to master or any other branch. In order to do so, first check out to TARGET branch and then issue merge command

```
$ git merge YourLocalBranch
```

Suppose if you have made some changes to TARGET Branch and changes to same files in SOURCE branch, then if you try to merge it, it will force you to resolve the conflicts by your own and merge it. You can check git status and **resolve the conflicts** in the error files

Just in case if you want to cancel merging in case of conflicts, then execute

```
$ git merge --abort
```

How to update the last commit without doing new commit

Just in case if you committed the changes to local repository and soon after that you realize something needs to be corrected. Now either you can perform new commit or just update the last commit. It is highly recommended to do this only in your local repo.

It refers to updating the changes to repo without updating the last commit message. That's it.

In order to do so, do your changes and push to staging and issue commit command without message but with option `--amend`

```
$ git commit --amend
```

Rollback to specified commit

`--hard` → remove associated files from repo till working folder

```
$ git reset ID_OF_COMMIT_TILL_WHERE_RESET_REQUIRED --hard
```

```

admin@admin-PC MINGW64 /d/GIT Project/Car (New_Car)
$ git log --oneline
6aeaa57 (HEAD -> New_Car) third commit
130570a second commit
e6953c2 (master) first commit

admin@admin-PC MINGW64 /d/GIT Project/Car (New_Car)
$ git reset e6953c2 --hard
HEAD is now at e6953c2 first commit

admin@admin-PC MINGW64 /d/GIT Project/Car (New_Car)
$ git log --oneline
e6953c2 (HEAD -> New_Car, master) first commit

```

How to discard local changes / how to overwrite files from higher level

- \$ git restore --staged file1.txt → Overwrite files from repo to staged
- \$ git restore file1.txt → Overwrite files from staged to local
- \$ git restore . → . refers to all content

How to do get repo from specific commit

It is same as moving to different branch using check out. 2 ways do that are:

```

$ git checkout HEAD~2
$ git checkout 977fd61

```

```

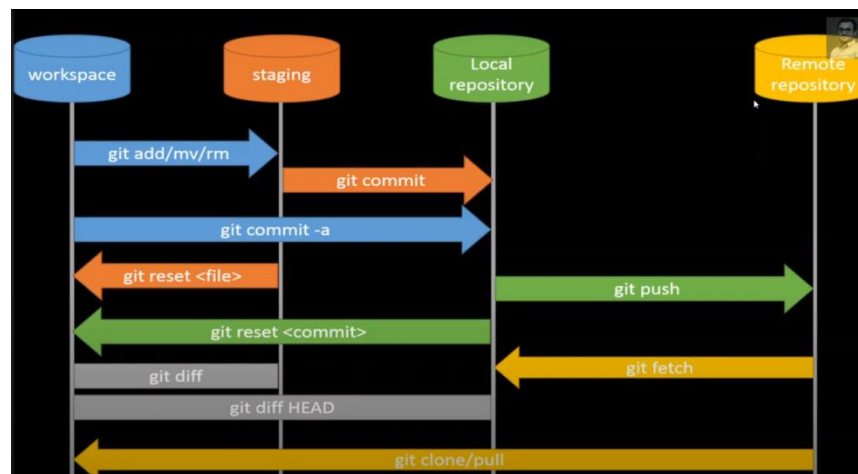
$ git log --oneline
37fbc23 (HEAD -> master) File3 added
977fd61 File2 added
37cec64 File1 added

admin@admin-PC MINGW64 /d/GIT Project/Car (master)
$ git checkout 977fd61
Note: switching to '977fd61'.

```

GIT HUB

Github allows us to store the git repository in the cloud / server. So that can be accessed from anywhere and shared between users



Git Clone

To work with remote repository, first we need to have that repo in our local. Git clone command gets entire repo and its history to your local machine to work with. It matches default branch with local master branch.

```
$ git clone <<REPO URL>>
```

Note: When you cloned, no need to add remotes as it already set to Origin by default

To see all remote branches, use

```
$ git branch -r
```

In order to map other remote branch to local branch, use

```
$ git switch RemoteBranch ➔ it will create a local branch with same name and map it
```

Git remote

Remotes are nothing but a remote repository hosted in git hub

```
$ git remote -v ➔ list down the repository to which your local repo is connected to.
```

In order to connect your local repo with remote repository, use:

```
$ git remote add origin <<URL>>
```

where origin is just an alias to that url. You can use different alias as you wish. But commonly people use it as origin. No special meaning behind it

```
$ git remote rename <<old name>> <<new name>>
```

```
$ git remote remove <<alias name>> ➔ detach your repo from remote repo
```

Git Push

```
$ git push Remote_Alias_Name Local_Branch_Name
```

Ex: *\$ git push origin master*

Note that you can push only one branch at a time to Git hub

Git Fetch

Bring the remote data to local repo. User can review it and merge it with local work if required

```
$ git fetch origin master
```

Git Pull

Overwrites the local working directory and repo from the remote repository

```
$ git pull origin master
```

What are Pull Requests?

When user has pushed the changes to Git hub, it awaits for admin to review and merge the changes to master branch.

After pushing the changes from local, user has to login to Git hub and create a pull request. After it has been reviewed and approved, admin user can merge the changes to repository

Tags

They are nothing but assigning a version name to last commit.

You can also push the repo to Git hub using tags.

`$ git push origin v1.0.0` → any specific tag

`$ git push origin --tags` → all tags from local

```
admin@admin-PC MINGW64 /d/GIT Project/Car/DemoRepo (main)
$ git commit -m "fist commit"
[main (root-commit) 9a0fe34] fist commit
1 file changed, 1 insertion(+)
create mode 100644 file.txt

admin@admin-PC MINGW64 /d/GIT Project/Car/DemoRepo (main)
$ git tag v1.0.0

admin@admin-PC MINGW64 /d/GIT Project/Car/DemoRepo (main)
$ git log --oneline
9a0fe34 (HEAD -> main, tag: v1.0.0) fist commit

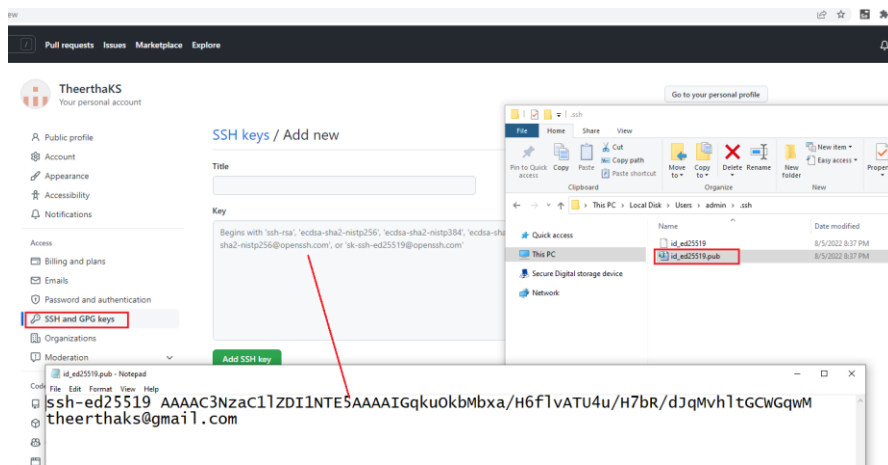
admin@admin-PC MINGW64 /d/GIT Project/Car/DemoRepo (main)
$ git tag
v1.0.0
```

Steps to setup SSH in your local to push changes to Git Hub

```
admin@admin-PC MINGW64 ~/Desktop/GIT
$ ssh-keygen -t ed25519 -C "theerthaks@gmail.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/admin/.ssh/id_ed25519):
Created directory '/c/Users/admin/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/admin/.ssh/id_ed25519
Your public key has been saved in /c/Users/admin/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:2SFIBAA2uj2l0FNKCHgN6MqirXdnwPcxozs9L0qSp7g theerthaks@gmail.com
The key's randomart image is:
+--[ED25519 256]--+
| .+.O+|.
|o. = ...o +
|+ o . .+.O.+
|O+. . . O+..
|=O.... S+.
|+B.o . + +
|O.+ . . o .
|.o oo o
| E..=
+-----[SHA256]-----+

admin@admin-PC MINGW64 ~/Desktop/GIT
$ eval "$(ssh-agent -s)"
Agent pid 1721

admin@admin-PC MINGW64 ~/Desktop/GIT
$ ssh-add ~/.ssh/id_ed25519
Identity added: /c/Users/admin/.ssh/id_ed25519 (theerthaks@gmail.com)
```



```
admin@admin-PC MINGW64 ~/Desktop/GIT
$ ssh -T git@github.com
The authenticity of host 'github.com
(20.207.73.82)' can't be established.
ED25519 key fingerprint is SHA256:+Di
Y3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UVC
OQU.
This key is not known by any other na
mes
Are you sure you want to continue con
necting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.co
m' (ED25519) to the list of known hos
ts.
Hi TheerthaKS! you've successfully au
thenticated, but Github does not prov
ide shell access.
```

References:

https://www.youtube.com/watch?v=vMdSqMf6BPY&list=PL_euSNU_eLbegnt7aR8I1gXfLhKZbxnYX