Staging

Allows you to continue making changes to the working directory, and when you decide you want to interact with version control, it allows you to commit (record changes) to the repository you are working on.

```
git add index.html
```

Adds “index.html” file to the stage.

```
git add “*.html”
```

Adds all html files in the directory.

```
git add -A
```

Adds all files including deleted ones.

```
git add .
```

Stages all the files in the current directory.

Commit

Create a snapshot of the staged changes along a timeline of a git projects history and record changes to the repository.

```
git commit –m “appropriate message”
```

Create a new commit from changes added to the staging area.

```
git commit --amend
```

Instead of creating a new commit, staged changes will be added to the previous commit.

Log

Shows the commit logs.

```
git log
```

Shows all commit logs.

```
git log -n 2
```

Displays only two last commits.

```
git log “index.html”
```

It shows the commits that are related to the "index.html" file. Makes it easier to see file’s history.

```
git log --since='Apr 1 2021' --until='Apr 4 2021'
```

Displays log of all commits between specified dates.

Show

Display expanded details on Git objects such as blobs, trees, tags, and commits. It has specific behavior per object type.

```
git show
```

show any object in Git in readable format.

Diff

Shows what is changed in a file.

```
git diff
```

Shows what is changed in file but not staged.

```
git diff --staged
```

Shows difference of what is staged file but yet not committed.

Reset

Move the repository back to previous commit, discarding any changes made after that commit.

This command updates branch and changes the commit history.

```
git reset <file>
```

Unstage a file while retaining the changes in working directory.

```
git reset commithash
```

Reset our repository back to the specific commit. (commithash being the first 7 characters of the commit hash we found in the log).

Restore

Helps us to unstage or discard uncommitted local changes. It can be used for restoring files in the working tree from the index or another commit. This command does not update our branch.

```
git restore
```

Restore working tree files.

Rm

Remove file from working directory and staging area.

```
git rm file.txt
```

Removes the file from both git repository and filesystem.

```
git rm file.txt --cached
```

Removes the file only from git repository.

```
git rm *.txt --dry-run
```

No files are actually removed. You will only see an output of the files that Git would remove.

Branches

lets you create, list, rename, and delete branches. It doesn't let you switch between branches tho.

```
git branch
```

Displays branches list. A ‘*’ will appear next to the currently active branch.

```
git branch branch-name
```
Creates a branch with this name.

```
git branch –d branch-name
```
Deletes the [branch-name] branch.

Merge

This is used for merging the created branch with master.

```
git merge branch-name
```
Join [branch-name] branch into current branch (the one you are on currently).

Stash

Temporarily shelves (or stashes) changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on.

```
git stash
```
Put current changes in your working directory into stash for later use.

```
git stash pop
```
Apply stored stash content into working directory, and clear stash.

```
git stash drop
```
Delete a specific stash from all your previous stashes.

```
git stash list
```
Display list of stashes.

```
git stash show
```
Display a summary of stash.

Checkout

Lets you navigate between the branches created by ” git branch”.

```
git checkout branch-name
```
Switches to this branch.

```
git checkout --  file.txt
```
Discards changes in the file and it reverts to the last change we made (last commit).

Rebase

The process of moving or combining a sequence of commits to a new base commit.

```
git rebase branch-name
```
Apply any commits of current branch ahead of specified one.

```
git rebase base-branch topic-branch
```
Checks out the topic branch for you and replays it onto the base branch.

Patch files

They are used in order to store differences that need to be applied to a file or a group of files on your system.

```
git format-patch master
```
Creates patch files from commits coming from the master branch.

```
git format-patch master -o patches
```
Creates Git patch files in a directory named “patches”.

Remote repositories

Lets you create, view, and delete connections to other repositories.

```
git remote
```
List the remote connections you have to other repositories.

```
git remote add remote-name remote-URL
```
This associate the name remote-name with remote-URL.

```
git remote rename old-name new-name
```
Renames an existing remote.

```
git remote set-url remote-name new-URL
```
Changes an existing remote repository URL (It also can be used for switching remote URLs from SSH to HTTPS).

Push

Updates the remote branch with local commits. You can also think of it as update or publish (Git can’t push empty directories).

```
git push remote branch
```
Transmit local branch commits to the remote repository branch (It pushes the files from master to the origin).

```
git push <remote> --all
```
Push all of your local branches to the specified remote.

```
git push <remote> --tags
```
Tags are not automatically pushed. The --tags flag sends all of your local tags to the remote repository.

```
git push --all
```
Push all branches.

Pull

It is used to fetch and download content from a remote repository and immediately update the local repository to match that content. Git pull is a combination of two commands, Git fetch followed by Git merge.

git pull <remote>	Fetch the specified remote's copy of the current branch and immediately merge it into the local copy.
git pull remote-name branch-name	Fetch and merge any commits from the tracking remote to local branch.
git pull --rebase rep-URL ref	Use a rebase merging strategy instead of a merge commit.

Fetch

Downloads commits, files, and refs from a remote repository into your local repo. Fetching is what you do when you want to see what everybody else has been working on.

git fetch remote-name	Fetch all of the branches from the repository.
git fetch remote-name branch-name	Fetch the specified branch from the repository.

Handling conflicts

A merge conflict is an event that takes place when Git is unable to automatically resolve differences in code between two commits. Git can merge the changes automatically only if the commits are on different lines or branches.

Conflicts generally arise when two people have changed the same lines in a file, or if one developer deleted a file while another developer was modifying it. In these cases, Git cannot automatically determine what is correct. In this cases, it is the developers' responsibility to resolve the conflict.

Some git commands for resolving the conflicts:

git log --merge	Helps to produce the list of commits that are causing the conflict.
git diff	Helps to identify the differences between the states repositories or files.
git checkout	It is used to undo the changes made to the file, or for changing branches.
git rest --mixed	It is used to undo changes to the working directory and staging area.
git merge --abort	Helps in exiting the merge process and returning back to the state before the merging began.
git reset	It is used at the time of merge conflict to reset the conflicted files to their original state.

Tags

Git tags are used to capture the specific point in the history that is further used to point to a released version. A tag does not change like a branch.

git tag tag-name	Creates a tag with specified name.
git tag -a tag-name –m "message"	Executing this command will create a new annotated tag identified with specified name and message.
git tag	Displays tags list.

.gitignore

It tells Git which files or folders to ignore in a project when committing your project in the git repository.

.file-name	This will ignore all files with this name.
.directory-path/	Ignores entire directory
*.log !file-name.log	This will ignore all files with "log" extenstion except file-name.log
git status –ignored	Displays all ignored files.

.gitkeep

It is not allowed in Git to store empty directories in branches. So if we need to track an empty directory to push, we can accomplish it by creating a .gitkeep file in a folder.

Go to the empty folder: touch .gitkeep	With this commands, you can add, commit and push your folder to the repository.

Clone

It is used to target an existing repository and create a clone, or copy of the target repository. By cloning with

Git, you get the entire repository - all files, all branches, and all commits.

git clone repo-URL

Initializes a new Git repository in the my-project folder on your local machine and populates it with the contents of the central repository.

git clone --mirror

Clone a repository but without the ability to edit any of the files. This includes the refs, or branches.

Pull request

It is an event that takes place in software development when a contributor/developer is ready to begin the process of merging new code changes with the main project repository. During a pull request, the repository maintainer reviews new code updates from a developer to determine whether or not it is ready to be released. They review the set of changes, discuss potential modifications, and even push follow-up commits if necessary.

Dotfiles

These files are configuration files for various programs, and they help those programs manage their functionality.

.gitignore

It's mentioned in the document. Refer to it if needed.

.gitkeep

It's mentioned in the document. Refer to it if needed.

.gitattribute

Allows you to specify the files and paths attributes that should be used by git when performing git actions, such as git commit, etc. In other words git automatically saves the file according to the attributes specified, every time a file is created or saved. The file must be encoded in UTF-8.

.gitmodules

Git submodules allow you to keep a git repository as a subdirectory of another git repository. Git submodules are simply a reference to another repository at a particular snapshot in time.
A git submodule is a record within a host git repository that points to a specific commit in another external repository. Submodules are very static and only track specific commits. They do not track git refs or branches and are not automatically updated when the host repository is updated. When adding a submodule to a repository a new .gitmodules file will be created. The .gitmodules file contains meta data about the mapping between the submodule project's URL and local directory.
A submodule can be located anywhere in a parent Git repository's working directory and is configured via a .gitmodules file located at the root of the parent repository. This file contains which paths are submodules and what URL should be used when cloning and fetching for that submodule.

For example:

```
[submodule "libfoo"]
        path = include/foo
        url = git://foo.com/git/lib.git

[submodule "libbar"]
        path = include/bar
        url = git://bar.com/git/lib.git
```