GIT COMMANDS

1.git config:

Git config command is super helpful. Especially when you are using Git for the first time, or you have a new Git installation. This command will set up your identity - Name and Email address. And this information will be used with every commit.
Usage

$ git config --global user.name "Your name"
$ git config --global user.email "Your email"

==========

2. git version

As its name implies, it's just to check which version of Git you are using. At the moment, writing this guide, the latest version of Git for Windows is 2.31.1. It was released on 27th March 2021.
Usage

$ git version

==========

3. git init

This is probably the first command you use to start a new project in Git. This command will create a blank new repository, and then you can store your source code inside this repo.
Usage

$ git init

Or you can use the repository name with your git init command.

$ git init <your repository name>

==========

4. git clone

The git clone command will use an existing repository to copy. There is one main difference between the git init and git clone.
You will use the Git clone when you need to make a copy on an existing repository. The git clone command internally uses the git init command first and then checks out all its contents.
Usage

$ git clone <your project URL>

==========

5. git add

The Git add command will add all the new code files or modified files into your repository. This command offers different options to add files and folders.
Here is the usage of the Git add command.
Usage

$ git add your_file_name (it will add a single file to your staging area)

$ git add * (this option will add all the modified and new files to the staging area)

==========

6. git commit

This Git command is essential. Your project quality may drop if you will not use this command appropriately. There is another article about how to use Git commands property, and you can read that here.
In simple words, the Git commit will add your changes to your local repository.
Usage

$ git commit -m “your useful commit message”

==========

7. git status

This Git command is convenient to see how many files are there which need your attention. You can run this command at any time.
You can use it in between Git add, and Git commits to see the status.
Usage

$ git status

==========

8. git branch

Most of the time, you have multiple branches in your Git repository. In simple terms, the branch is an independent line of code development.
With the Git branch command, you can manage your branches effectively. There are many different options and switches of the Git branch.
To make it simple, here I will highlight how you can create and delete a Git branch (in case you need more depth, you can refer to - Git Branching and Merging section of this article).
Usage
(i) To list all branches:

$ git branch

(ii) To create a new branch:

$ git branch <branch_name>

(iii) To delete a branch:

$ git branch -d <branch_name>

==========

9. git checkout

This Git command is used to switch between branches. This is one of the powerful git commands and can use used as a swiss knife,
In simple words, here is the syntax to switch to another branch.
Usage

$ git checkout <branch_name>

Also, you can create and checkout to a branch in a single like, here is the usage for that

$ git checkout -b <your_new_branch_name>

==========

10. git remote

Git remote command acts like a border, and If you need to connect with the outside world, you have to use the Git remote command. This command will connect your local repository to the remote.

Usage

```
$ git remote add <shortname> <url>
```

Example

```
$ git remote add origin https://dev.azure.com/aCompiler/_git/DemoProject
```

==========

11. git fetch

When you need to download other team members' changes, you have to use git fetch.

This command will download all information for commits, refs, etc., so you can review it before applying those changes in your local repository.

Usage

```
$ git fetch
```

==========

12. git push

Once you are connected with the remote repository (with the help of the git remote command), it's time to push your changes to that repository.

Usage

```
$ git push -u <short_name> <your_branch_name>
```

Example

```
$ git push -u origin feature_branch
```

You should have origin and upstream set up before you use Git push. And here is the command to set up upstream.

Usage

```
$ git push --set-upstream <short_name> <branch_name>
```

Example

```
$ git push --set-upstream origin feature_branch
```

==========

13. git pull

The Git pull command downloads the content (and not the metadata) and immediately updates your local repository with the latest content.

Usage

```
$ git pull <remote_url>
```

==========

14. git stash

This Git command temporarily stores your modified files. You can work in stashed with the following Git command.

Usage

```
$ git stash
```

And you can view all of your stashes with the following command

```
$ git stash list
```

And if you need a apply a stash to a branch, simply use apply

```
$ git stash apply
```

==========

15. git log

With the help of the Git log, you can see all the previous commits with the most recent commit appear first.

Usage

```
$ git log
```

By default, it will show you all the commits of the currently checked out branch, but you can force it to see all the commits of all the branches with all options.

```
$ git log --all
```

==========

16. git shortlog

The shortlog command shows you a summary from the Git log command. This command is helpful if you are just interested in the short summary.

This command is helpful to see who worked on what as it group author with their commits.

Usage

```
$ git shortlog
```

==========

17. git show

Compared to the Git log, this command git show will show you details about a specific commit.

Usage

```
$ git show <your_commit_hash>
```

==========

18. git rm

Sometimes you need to delete files from your codebase, and in that case, you can use the Git rm command.

It can delete tracked files from the index and the working directory.

Usage

```
$ git rm <your_file_name>
```

==========

19. git merge

Git merge helps you to integrate changes from two branches into a single branch.
Usage
$ git merge <branch_name>
This command will merge the <branch_name> into your current selected branch.
==========

20. git rebase

Git rebase similar to the git merge command. It integrates two branches into a single branch with one exception. A git rebase command rewrites the commit history. You should use the Git rebase command when you have multiple private branches to consolidate into a single branch. And it will make the commit history linear.
Usage
$ git rebase <base>
==========

21. git bisect

The Git bisect command helps you to find bad commits.
Usage
i) To start the git bisect
$ git bisect start

ii) let git bisect know about a good commit
$ git bisect good a123

iii) And let git bisect know about a bad commit
$ git bisect bad z123
With Git bisect you can narrow down the broken code within a few minutes.
==========

22. git cherry-pick

Git cherry-pick is a helpful command. It's a robust command and allows you to pick any commit from any branch and apply it to any other branch.
Usage
$ git cherry-pick <commit-hash>
Git cherry-pick doesn’t modify the history of a repository; instead, it adds to the history.
==========

23. git archive

Git archive command will combine multiple files into a single file. It's like a zip utility, so it means you can extract the archive files to get individual files.

Usage

$ git archive --format zip HEAD > archive-HEAD.zip

It will create a zip archive of the current revision.

==========

24. git pull --rebase

Most of the time, you need to do rebase (and no merge) when you use Git pull. In that case, you can use the option

Usage

$ git pull --rebase

It will help you to keep the history clean. Also, you can avoid multiple merges.

==========

25. git blame

If you need to examine the content of any file line by line, you need to use git blame. It helps you to determine who made the changes to a file.

Usage

$ git blame <your_file_name>

==========

26. git tag

In Git, tags are helpful, and you can use them to manage the release. You can think of a Git tag like a branch that will not change. It is significantly more important if you are making a public release.

Usage

$ git tag -a v1.0.0

==========

27. git verify-commit

The git verify-commit command will check the gpg signature. GPG or “GNU Privacy Guard” is the tool used in sign files and contains their signatures.

Usage

$ git verify-commit <commit>

==========

28. git verify-tag

In the same way, you can confirm a tag.

Usage

$ git verify-tag <tag>

==========

29. git diff

Most of the time, you need to compare two git files or branches before you commit or push. Here is a handy command to do that.

Usage
i) to compare the working directory with the local repo:
$ git diff HEAD <filename>
ii) to compare two branches:
$ git diff <source branch> <target branch>
==========

30. git citool

Git citool is a graphics alternative of the Git commit.
Usage
$ git citool
==========

31. git mv

To rename a git file. It will accept two arguments, source and target file name.

Usage

$ git mv <old-file-name> <new-file-name>
==========

32. git clean

You can deal with untracked files by using the Git clean command. You can remove all the untracked files from your working directory by using this command. In case you want to deal with tracked files you need to use the Git reset command.
Usage
$ git clean
==========

33. git help

There are many commands in Git, and if you need more help with any command, you can use git help at any time from the terminal.
Usage
$ git help <git_command>
==========

34. git whatchanged

This command does the same thing as git log but in a raw form. And it’s in the git because of historical reasons.
Usage
$ git whatchanged
==========