DOWNLOAD & INSTALL

Git Install for Linux, Mac and Windows

https://git-scm.com/downloads

CONFIGURE

Git config levels and files

--local ***(Repository's .git directory: .git/config)***

--global ***(User's home directory. ~/.gitconfig on unix systems & C:\Users\<username>\.gitconfig on windows)***

--system ***(System root path: ROOT/etc/gitconfig on unix systems and C:\ProgramData\Git\config on Windows)***

Configure Microsoft VS Code Editor

```
$ git config --global core.editor
'"<Path>/Code.exe" -w'
```

Configure user info

```
$ git config --global user.name "XDK
$ git config --global user.email
"XDK@XDK.com"
```

Configure CLI console settings

```
$ git config --global color.ui auto
```

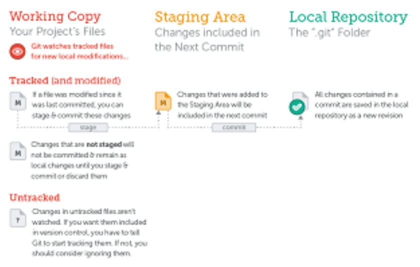
Show config list

```
$ git config –l
```

To Create Alias, Add Alias at GIT Global Config

```
$ git config –global alias.<short_cmnd>
"long Command"
```

LOCAL GIT STRUCTURE



Working Copy
Your Project's Files
Git watches tracked files for new local modifications...
Staging Area
Changes included in the Next Commit
Local Repository
The ".git" Folder
Tracked (and modified)
If a file was modified since it was last committed, you can stage & commit these changes
stage
Changes that were added to the Staging Area will be included in the next commit
commit
All changes contained in a commit are saved in the local repository as a new revision
Changes that are not staged will not be committed & remain as local changes until you stage & commit or discard them
Untracked
Changes in untracked files aren't watched. If you want them included in version control, you have to tell Git to start tracking them. If not, you should consider ignoring them.

CREATE REPOSITORY

Initialize the current directory as a working directory for Git

```
$ git init
```

List which files are staged, unstaged, and untracked.

```
$ git status
```

Clone a repository.

```
$ git clone <repository>
```

Cloning to a specific folder.

```
$ git clone <repository> <directory>
```

Clone a specific tag.

```
$ git clone --branch <tag> <repo>
```

Shallow clone.

```
$ git clone -depth=1 <repository>
```

STAGE FILES & FOLDERS

Stage all changes in <directory> for the next commit

```
$ git add .
$ git add -A
$ git add <file1> <file1> <file1>
```

Rename Files

```
$ git mv current_name new_name
```

Move file

```
$ git mv file_name directory_name
```

COMMIT

Commit by including files & folders by specific msg

```
$ git commit –a –m  "COMMIT MESSAGE"
```

Change the last commit files or messages

```
$ git commit - -amend –m "Commit msg"
```

COMMIT HISTORY

Command to verify last Commits

```
$ git log
```

Get the changes over time for a specific file

```
$ git log -p <file>
```

Get GIT abbrev Commit Hash

```
$ git log - - abbrev-log
```

Get Git Online Commit

```
$ git log - -all - -online - -graph - -decorate
```

Date Base Search

```
$ git log - -since="2 days ago"
```

Get Details of any commit

```
$ git show <commit_id>
```

Show changes over time for a specific file

```
$ git log -p <file>
```

Who changed what and when in <file>

```
$ git blame <file>
```

Show a log of changes to the local repository's HEAD

```
$ git reflog - -all
```

COMPARISON IN GIT

Compare Working Directory & Stage Area

```
$ git diff
```

Compare Work Directory and GIT Repo

```
$ git diff HEAD
```

Compare Stage Area & GIT Repo

```
$ git diff - -cache
$ git diff - -staged HEAD <file name>
```

Compare Commits

```
$ git diff <Commit ID> <Commit ID>
```

Compare Tags

```
$ git diff <Tag Name> <Tag Name>
```

BRANCHES

List all existing branches

```
$ git branch -av
```

Create a new branch (CurrentHEAD)

```
$ git branch <new-branch>
```

Swtich branch

```
$ git checkout <branch>
```

Rename branch

```
$ git branch -m <old name> <new name>
```

Delete a local branch

```
$ git branch -d <branch>
```

Mark the current commit with a tag

```
$ git tag <tag-name>
```

MERGES

Merge current branch to target branch

```
$ git checkout <target branch>
$ git merge <current branch>
```

REBASE

Rebase current branch to target branch

```
$ git checkout <target branch>
$ git rebase <current branch>
```

Abort a rebase

```
$ git rebase --abort
```

Continue a rebase after resolving conflicts

```
$ git rebase --continue
```

TAGGING

Lightweight tags

```
$ git tag <tag name> -m "tag description"
```

Annotated tags to store extra metadata

```
$ git tag -a <tag name> -m "tag
description"
```

Editing tags

```
$ git tag -a -f <tag name> <commit id>
```

Sorting tags

```
$ git tag - -sort=<type>
```

Sorts in a lexicographic order

```
$ git tag -l - -sort=-version:refname "v*"
```

Deleting tags

```
$ git tag -d <tag name>
```

STASH

List stash

```
$ git stash list
```

Save stash

```
$ git stash save "comments"
```

Apply stash

```
$ git stash apply
```

Apply specific version

```
$ git stash apply stash@{1}
```

Deleting stash

```
$ git stash drop stash@{0}
```

PUBLISH

List all currently configured remotes

```
$ git remote -v
```

Show information about a remote

```
$ git remote show <remote>
```

Add new repository

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

Download all changes from <remote>, but don't integrate into HEAD

```
$ git fetch <remote>
```

Download changes and directly merge/integrate into HEAD

```
$ git pull <remote> <branch>
```

Publish local changes on a remote

```
$ git push <remote> <branch>
```

Delete a branch on the remote

```
$ git branch -dr <remote/branch>
```

Publish your tags

```
$ git push --tags
```