Git config

The git config command is a convenience function that is used to set Git configuration values on a global or local project level. These configuration levels correspond to .gitconfig text files. Executing git config will modify a configuration text file.

<--local>

By default, git config will write to a local level if no configuration option is passed. Local level configuration is applied to the context repository git config gets invoked in.

< --global>

Global level configuration is user-specific, meaning it is applied to an operating system user.

< --system>

System-level configuration is applied across an entire machine. This covers all users on an operating system and all repos.

Listing configuration options

$ git config --list
$git config -l

Configuration structure

<section>.<key>

Get /set value

```
git config --global user.name
git config --global user.email
git config --global user. name "YourName"
```

Delete a configuration

$ git config --global --unset user.name

Note : The first time using installing git you should set your config user.name user.email

Creating a new Repository

The git init command creates a new Git repository. It can be used to convert an existing, unversioned project to a Git repository or initialize a new, empty repository. Most other Git commands are not available outside of an initialized repository, so this is usually the first command you'll run in a new project.

Executing git init creates a .git subdirectory in the current working directory, which contains all of the necessary Git metadata for the new repository. This metadata includes subdirectories for objects, refs, and template files. A HEAD file is also created which points to the currently checked out commit.

$ mkdir myRepo
$ cd myRepo
$ git init

Git add

The `git add` command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, `git add` doesn't really affect the repository in any significant way—changes are not actually recorded until you run git commit.

Git commit

Developing a project revolves around the basic edit/stage/commit pattern. First, you edit your files in the working directory. When you're ready to save a copy of the current state of the project, you stage changes with `git add`. After you're happy with the staged snapshot, you commit it to the project history with `git commit`. The git reset command is used to undo a commit or staged snapshot.

$echo "BATMAN">>REDME.md
$ git add REDME.md
$ git commit -m "meassage"

Git status

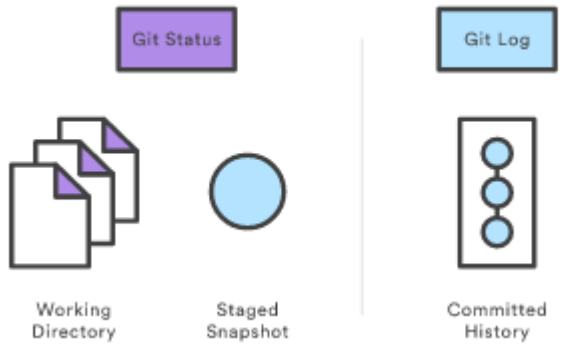
The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git. Status output does *not* show you any information regarding the committed project history. For this, you need to use git log.

$git status

Git log

The `git log` command displays committed snapshots. It lets you list the project history, filter it, and search for specific changes. While `git status` lets you inspect the working directory and the staging area, `git log` only operates on the committed history.

$git log



A few useful options to consider. The --graph flag that will draw a text based graph of the commits on the left hand side of the commit messages. --decorate adds the names of branches or tags of the commits that are shown. --oneline shows the commit information on a single line making it easier to browse through commits at-a-glance.

$git log --graph --oneline