Terminology

### Branch

A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process discussed in Git Basics, the first module of this series. You can think of them as a way to request a brand new working directory, staging area, and project history. New commits are recorded in the history for the current branch, which results in a fork in the history of the project.

### HEAD

Git’s way of referring to the current snapshot. Internally, the git checkout command simply updates the HEAD to point to either the specified branch or commit. When it points to a branch, Git doesn't complain, but when you check out a commit, it switches into a “detached HEAD” state.

### Hook

A script that runs automatically every time a particular event occurs in a Git repository. Hooks let you customize Git’s internal behavior and trigger customizable actions at key points in the development life cycle.

### Master

The default development branch. Whenever you create a git repository, a branch named "master" is created, and becomes the active branch.

### Tag

A reference typically used to mark a particular point in the commit chain. In contrast to a head, a tag is not updated by the commit command.

### Version Control

A system that records changes to a file or set of files over time so that you can recall specific versions later.

### Working Tree

The tree of actual checked out files, normally containing the contents of the HEAD commit's tree and any local changes you've made but haven't yet committed.

# Git commands

### git add

Moves changes from the working directory to the staging area. This gives you the opportunity to prepare a snapshot before committing it to the official history.

### git branch

This command is your general-purpose branch administration tool. It lets you create isolated development environments within a single repository.

### git checkout

In addition to checking out old commits and old file revisions, git checkout is also the means to navigate existing branches. Combined with the basic Git commands, it’s a way to work on a particular line of development.

### git clean

Removes untracked files from the working directory. This is the logical counterpart to git reset, which (typically) only operates on tracked files.

### git clone

Creates a copy of an existing Git repository. Cloning is the most common way for developers to obtain a working copy of a central repository.

### git commit

Takes the staged snapshot and commits it to the project history. Combined with git add, this defines the basic workflow for all Git users.

### git commit --amend

Passing the --amend flag to git commit lets you amend the most recent commit. This is very useful when you forget to stage a file or omit important information from the commit message.

### git config

A convenient way to set configuration options for your Git installation. You’ll typically only need to use this immediately after installing Git on a new development machine.

### git fetch

Fetching downloads a branch from another repository, along with all of its associated commits and files. But, it doesn't try to integrate anything into your local repository. This gives you a chance to inspect changes before merging them with your project.

### git init

Initializes a new Git repository. If you want to place a project under revision control, this is the first command you need to learn.

### git log

Lets you explore the previous revisions of a project. It provides several formatting options for displaying committed snapshots.

### git merge

A powerful way to integrate changes from divergent branches. After forking the project history with git branch, git merge lets you put it back together again.

### git pull

Pulling is the automated version of git fetch. It downloads a branch from a remote repository, then immediately merges it into the current branch. This is the Git equivalent of svn update.

### git push

Pushing is the opposite of fetching (with a few caveats). It lets you move a local branch to another repository, which serves as a convenient way to publish contributions. This is like svn commit, but it sends a series of commits instead of a single changeset.

### git rebase

Rebasing lets you move branches around, which helps you avoid unnecessary merge commits. The resulting linear history is often much easier to understand and explore.

### git rebase -i

The -i flag is used to begin an interactive rebasing session. This provides all the benefits of a normal rebase, but gives you the opportunity to add, edit, or delete commits along the way

### git reflog

Git keeps track of updates to the tip of branches using a mechanism called reflog. This allows you to go back to changesets even though they are not referenced by any branch or tag.

### git remote

A convenient tool for administering remote connections. Instead of passing the full URL to the fetch, pull, and push commands, it lets you use a more meaningful shortcut.

### git reset

Undoes changes to files in the working directory. Resetting lets you clean up or completely remove changes that have not been pushed to a public repository.

### git revert

Undoes a committed snapshot. When you discover a faulty commit, reverting is a safe and easy way to completely remove it from the code base.

### git status

Displays the state of the working directory and the staged snapshot. You’ll want to run this in conjunction with git add and git commit to see exactly what’s being included in the next snapshot.