# Versioning Commits

In Git, files can have the following statuses:

**Untracked**: This a file that exists in the working tree whose changes are not being monitored by Git and aren't listed in the **gitignore** file.

**Unstaged**: This a file whose changes are being tracked by Git; the file has been changed since the last commit and has yet to be moved to the **index**.

**Staged**: This is a file whose changes are being tracked by Git; the file has been changed since the last commit and has been moved to the index. This file is a file that is ready for the Git commit. Staged files are the files in the index that are different from their corresponding files in the **HEAD**, that is, the version of the file in the most recent commit.

**git status** is a utility that is used by Git. It's used to retrieve the details of files that are untracked, unstaged, or staged. **git status** lists files in order of their statuses.

The **git status** output is lengthy in nature. To view a brief list and status, use the **-s** or **--short** option with the **git status** command.

you should be able to use **git status** to view untracked, unstaged, and staged files.

To ensure flexibility, atomic commits are preferred and emphasized in version control. The term atomic commits refers to units of change that can be treated as a single unit. This normally implies a single file or a small set of files. This enables the addition and removal of changes without affecting a large set of files.

The **git diff** command is used to compare one snapshot of changes to another. As the name suggests, this utility supports evaluating the differences between two snapshots of a repository.

### Comparing the Working Tree to the Index

To compare the entire working tree to the index, run the git diff command without specifying a path:

$ git diff

This command supports examining the differences of a specific file or directory by accepting a path:

git diff -- [path\_to\_a\_file\_or\_directory]

1. git diff -- src/lib/
2. git diff -- src/lib/compute.py

(1) and (2) compare the version in the working directory of the specified paths to the version that's present in the index.

### Comparing the Working Tree to an Arbitrary Commit or Branch

A comparison can be made between the working tree and a specific commit on the same branch, or even the tip of a given branch. To compare the working tree to a given commit, use the following syntax:

$ git diff [commit\_hash] -- [path\_to\_a\_file\_or\_directory]

1. $ git diff HEAD -- src/
2. $ git diff f4e4e8d5b292dc94468b6f88223cac4f55c03713 -- src/lib/
3. $ git diff master

(1) Compares the version in the working directory of the **src** directory, to the version of the most recent commit on the current branch.

(2) Compares the version in the working directory of the src directory to the version in the snapshot represented by the hash f4e4e8d5b292dc94468b6f88223cac4f55c03713.

(3) Compares the version in the working directory of the **src** directory to the version at the tip of the branch **master**.

### Comparing the Index to an Arbitrary Commit

To compare the files in the index to a specific commit, (for example, the most recent commit, also referred to as the tip of the branch), you can use the **--staged** or **--cached** option with the **git diff** command. A commit hash is required for this scenario. The **git diff** command defaults to **HEAD** in the absence of a specific commit hash:

 **git diff --cached [commit\_hash]** or

**git diff --cached [commit\_hash] -- [path\_to\_a\_file\_or\_directory]**

1 **git diff --cached HEAD -- /src/lib/compute.py**

(1) compares the version in the index of the **compute.py** file to the version at the tip of the current branch.

### Comparing Commits and Branches

**git diff** provides a variant of the command's usage that supports comparing commits and branches.

To compare two commits or the tips of two branches, use the following syntax:

**git diff [commit\_hash or branch\_name] [commit\_hash or branch\_name]** or

**git diff [commit\_hash or branch\_name] [commit\_hash or branch\_name] -- [path\_to\_a\_file\_or\_directory]** or

**git diff [commit\_hash or branch\_name]..[commit\_hash or branch\_name]**

1. **git diff ft-add-encapsulating-class master**
2. **git diff ft-add-encapsulating-class..master**
3. **git diff da39a3ee5e6b4b0d3255bfef95601890afd80709 6f7e437faa5a7fce15d1ddcb9eaeaea377667b**

(1) and (2) compare the differences between the tips of the specified branches.

(3) compares the differences between the files at the point referenced by the specified hashes

Using the **...** notation, **git diff** is capable of comparing the changes that have been made on branch A to branch B. This occurs from the point where the two branches share an ancestor to the most recent commit of branch B.

To achieve this, use the following syntax:

**git diff [branch\_A]...[branch\_B] -- [path\_to\_a\_file\_or\_directory]**

1. **git diff ft-add-encapsulating-class...master**

(1) lists the changes that have occurred in the branch **master**, since the **ft-add-encapsulating-class** branch was created from the branch **master**.

Note

When .. or … are used with the **git diff** command, it implies the comparison of two points in history, and not a range.

**git diff** defaults to HEAD when a commit hash or branch name is not specified.

The **git add** command uses the following syntax:

**git add [options] [path\_to\_files]**

The **[options]** used with **git add** include the following:

**-n or --dry-run**

This option simulates the behavior of **git add** for the specified file:

**-f or --force**

This option adds ignored files to the index:

**-i or --interactive**

This option creates an interactive prompt that can be used to add files from the working tree to the index:

**-p or --patch**

The **--patch** option caters to adding portions of a file to the index

The **git commit** command saves the files in the index. This commit operation stores a message along with the commit. The message describes the additions or alterations associated with the created snapshot.

The syntax of this command is as follows: **git commit [options].**

The **git commit** command requires that a message be provided for each commit operation.

The options supported by this command include **-m [text] or --message [text].**

This message is used to associate the index file with the commit action:

**-a or -all**

This option instructs the **git commit** utility to stage tracked files that are unstaged, that is, the tracked files have been added to the index if the files are yet to be staged. Untracked files are not added to the index.

**-p or --patch**

This launches the interactive patch tool. The options are akin to those that are available through the **git add** command. See the preceding table for more information on this.

**-C [commit hash]** or **--reuse-message=[commit hash]**

This instructs **git commit** to reuse a commit message and the author information of the specified commit hash.

**-F [file]** or **--file=[file]**

This command specifies a file from which a commit message should be obtained.

**-t [file]** or **--template [file]**

This command specifies the commit message template file.

**-e** or **--edit**

This command edits the provided commit message. This refers to the message provided by the **-F**, **-t**, and **-m** options.

**--no-edit**

This command uses the specified message as is. Do not launch an editor to edit the message.

**--author=[author]**

This command overrides the details of a commit author, and takes the following form:

**git commit --author="Kifeh Polyswarm <kifeh@poly-swarm.com>"**

**--date=[date]**

As you can see, this overrides the date details used in a commit.

**-q** or **--quiet**

This command suppresses the summary message that's returned after running the **git commit** command

**git commit -C 474b5caaf480f7a367c1c456a53868c7fe32b9df --no-edit**

### *****git rm*****

The **git rm** command performs two roles. These roles are used to remove files from the working directory and the index.

**-r**

This option is applicable when using the **git rm** command in a directory. It removes the directory's contents recursively. This means that the directory and its contents are removed.

**--cached**

This option removes the specified files from the index only.

**-f** or **--force**

**git rm** checks the files marked for removal for matches, with the files in HEAD, at the tip of the current branch. This check is conducted before the file(s) are removed. The **-f** option overrides this check.

The ***rm*** command removes the specified file from the working tree only. The ***git rm*** command, on the other hand, removes the file from the index and the working tree. This provides a shorter process for deleting files, since with ***rm***, you need to run ***git add*** to impact the deletion process in the index.

### **git mv**

In the event that you need to update the index for both, old and new paths automatically, the **git mv** command serves that purpose.

This command has two forms of implementation:

1. git mv [options] [source] [destination]
2. git mv [options] [source] … [destination]

(1) is used to rename a file.

(2) is used to move a file.

## History and Logs

The **git log** command lists the history of a branch and the repository, by extension. It uses options and a range to define the duration for which the logs should be retrieved:

**git log [options] [version range] [path\_to\_file\_or\_directory] [version range]**

The **git log** command can display the history of a branch when given a range of version hashes:

**git log [hash\_1]..[hash\_2]**

### **Options**

### The **--follow** command retrieves and displays the history of a file, beyond rename events:

The **--decorate[=short** or **full** or **no]** command displays the **ref** name of the listed commits as seen in the following screenshots:

**git log --decorate=short**

### *The short option omits the* ***ref/heads/,ref/remotes/,*** *and* ***ref/tags/*** *prefixes from the* ***ref*** *name that is displayed.*

### *The full option displays the full ref name. It includes the* ***ref/heads/, ref/remotes/,*** *and* ***ref/tags/*** *prefixes in the ref name that is displayed.*

The **-L [start],[end]:[path\_to\_file]** command views the changes that have been made to a section of a file, from line number X to line number Y:

**git log -L 6,12:./src/lib/compute.py**

The **--[number]** , -**n [number]** ,and **--max-count=[number]** the specified number of commits only

### **git log -3 or git log -n 3**

The **--skip=[number]** command skips the specified commits and displays the rest:

**git log --skip=4**

The **--since=[date]** or **--after=[date]** commits that have been created after a given date:

**git log --after=25/08/2018**

The **--until=[date]** or **--before=[date]** commits that precede a given date:

**git log --before=24/08/2018**

The **--pretty=[format]** command displays the history of a branch using a prescribed format:

**git log --pretty=oneline**

### ***git log --pretty=medium***

**git log --pretty=format:[format string]**

The format is a string that's in the form of **%placeholder1 %placeholder2 %placeholderN**.

The supported placeholders include the following and are shown in the following screenshots:

* **%H**: The commit hash
* **%h**: The abbreviated commit hash
* **%T**: A tree hash
* **%t**: An abbreviated tree hash
* **%P**: The parent hash
* **%p**: The abbreviated parent hash
* **%an**: Author name
* **%ae**: Author email
* **%ad**: Author date
* **%ar**: A relative author date
* **%at**: The Unix timestamp version of the author date
* **%s**: The subject of the commit
* **%b**: The body of the commit
* **%n**: A newline

**git log --pretty=format:"%H %an"**

### **git log --pretty=format:"%H %an %ae"**

### **git log --pretty=format:"%H %an %ae %n %s %n %b"**

### Amending a Single Most Recent Commit

The most recent commit can be edited by using **--amend** in the **git commit** command.

**Amending Multiple Commits**

The **git rebase** command provides the **reword** and **edit** options to edit the commits. The **reword** option allows you to edit a message, while the **edit** option supports editing a commit message, as well as the contents of a commit.

## Fetching and Delivering Code

the **git remote** utility, to explore how we can manage the connection between the local and upstream repositories.

The **git remote** utility includes commands that help with managing the remote/upstream repositories that are associated with a local repository.

add the remote tracked repository by using the **git remote add origin git@github. com:kifeh-polyswarm/remote-demo.git**

View the remote configuration by using the **git remote -v command**

Rename the remote configuration from **origin** to **source-truth** by using the **git remote rename origin source-truth command**

View the specifics of the **source-truth remote** by using the **git remote show source-truth** command:

*The* ***prune*** *command removes local branches that correspond to branches that have been deleted from the remote repository:*

**$ git remote prune source-truth**

The configured remote can be removed by using the **remove** command. For example, to remove the **source-truth** remote, run **git remote remove source-truth:**

### Fetching, Pushing, and Pulling Changes

To enable collaboration in a distributed version control system, Git provides the means to retrieve and publish your contributions to the shared repository. To demonstrate this, we will explore **git fetch, git push,** and **git pull**.

**git fetch**

To navigate changes that were made to a repository, you need to utilize references to the changes made to the repository, and consequently a branch. This command allows you to explore the changes before integrating them into your work.

The **git fetch** command downloads remote-tracking branches and tags from the remote repository. These branches and tags indicate changes that have been made to the remote repository.

The content retrieved by this command is isolated from the content in the local repository, and, when you do this, the work that is being undertaken locally isn't affected by the downloaded content.

*Remote-tracking branches keep track of changes occurring on the branches in the remote repository. Remote-tracking branches can be viewed by using the* ***git branch -r*** *command. The* ***refs*** *for remote-tracking branches are stored in* ***/.git/refs/remotes/[remote\_name]****; for example,* ***/.git/refs/remotes/origin.***

you can retrieve or download all of the branches of the repository specified by **[remote]** with **git fetch [remote]** e.g. **git fetch origin:**

**git fetch [remote] [branch]**

You can fetch the **[branch]** from the upstream repository specified by **[remote]** with **git fetch origin master:**

**git fetch --all**

You can fetch branches from all of the remote connections defined for a repository. For example, if you have a remote **origin** and **upstream**, as you will see in the Chapter 4: Branches, this command will fetch branches from the two repositories identified by **origin** and **upstream**:

**git fetch --prune**

You can remove remote-tracking references that have ceased to exist in the remote repository, and then proceed to fetch the branches and their corresponding commits, files, and **refs** with: **+refs/heads/\*:refs/remotes/origin/\***.

#### Note

The **git fetch** command uses the **refspec** defined in the repository-level config. This is defined in the **remote.[remote\_name].fetch** config value. The **refspec** can be retrieved by running the **git config --local --list** command.

The **refspec** dictates that **refs** stored in the remote repository in **refs/heads/** are tracked locally, under **refs/remotes/origin/.**

The **+** indicates that references should be updated, including in scenarios where the commit is not a **fast-forward.**