**Set Up and Initialization**

Check your Git version with the following command, which will also confirm that Git is installed:

1. git --version

Git allows you to configure a number of settings that will apply to all the repositories on your local machine. For instance, configure a username that Git will use to credit you with any changes you make to a local repository:

1. git config --global user.name “firstname lastname”

Configure an email address to be associated with each history marker:

1. git config --global user.email “valid-email”

Configure your preferred text editor as well:

1. git config --global core.editor “nano”

You can initialize your current working directory as a Git repository with init:

1. git init

To an existing Git repository hosted remotely, you’ll use git clone with the repo’s URL or server location (in the latter case you will use ssh):

1. git clone https://www.github.com/username/repo-name

Show your current Git directory’s remote repository:

1. git remote

For more verbose output, use the -v flag:

1. git remote -v

Add the Git upstream, which can be a URL or can be hosted on a server (in the latter case, connect with ssh):

1. git remote add upstream https://www.github.com/username/repo-name

**Staging**

When you’ve modified a file and have marked it to go in your next commit, it is considered to be a staged file.

Check the status of your Git repository, including files added that are not staged, and files that are staged:

1. git status

To stage modified files, use the add command, which you can run multiple times before a commit. If you make subsequent changes that you want to include in the next commit, you must run add again.

You can specify the specific file with add:

1. git add my\_script.py

With . you can add all files in the current directory, including files that begin with a .:

1. git add .

If you would like to add all files in a current directory as well as files in subdirectories, you can use the -all or -A flag:

1. git add -A

You can remove a file from staging while retaining changes within your working directory with reset:

1. git reset my\_script.py

**Committing**

Once you have staged your updates, you are ready to commit them, which will record changes you have made to the repository.

To commit staged files, you’ll run the commit command with your meaningful commit message so that you can track commits:

1. git commit -m "Commit message"

You can condense staging all tracked files by committing them in one step:

1. git commit -am "Commit message"

If you need to modify your commit message, you can do so with the --amend flag:

1. git commit --amend -m "New commit message"

**Branches**

A branch in Git is a movable pointer to one of the commits in the repository, it allows you to isolate work and manage feature development and integrations. You can learn more about branches by reading the [Git documentation](https://git-scm.com/book/en/v1/Git-Branching-What-a-Branch-Is).

List all current branches with the branch command. An asterisk (\*) will appear next to your currently active branch:

1. git branch

Create a new branch. You will remain on your currently active branch until you switch to the new one:

1. git branch new-branch

Switch to any existing branch and check it out into your current working directory:

1. git checkout another-branch

You can consolidate the creation and checkout of a new branch by using the -b flag:

1. git checkout -b new-branch

Rename your branch name:

1. git branch -m current-branch-name new-branch-name

Merge the specified branch’s history into the one you’re currently working in:

1. git merge branch-name

Abort the merge, in case there are conflicts:

1. git merge --abort

You can also select a particular commit to merge with cherry-pick with the string that references the specific commit:

1. git cherry-pick f7649d0

When you have merged a branch and no longer need the branch, you can delete it:

1. git branch -d branch-name

If you have not merged a branch to main, but are sure you want to delete it, you can **force** delete a branch:

1. git branch -D branch-name

**Collaborate and Update**

To download changes from another repository, such as the remote upstream, you’ll use fetch:

1. git fetch upstream

Merge the fetched commits. Note that some repositories may use master instead of main:

1. git merge upstream/main

Push or transmit your local branch commits to the remote repository branch:

1. git push origin main

Fetch and merge any commits from the tracking remote branch:

1. git pull

**Inspecting**

Display the commit history for the currently active branch:

1. git log

Show the commits that changed a particular file. This follows the file regardless of file renaming:

1. git log --follow my\_script.py

Show the commits that are on one branch and not on the other. This will show commits on a-branch that are not on b-branch:

1. git log a-branch..b-branch

Look at reference logs (reflog) to see when the tips of branches and other references were last updated within the repository:

1. git reflog

Show any object in Git via its commit string or hash in a more human-readable format:

1. git show de754f5

**Show Changes**

The git diff command shows changes between commits, branches, and more. You can read more fully about it through the [Git documentation](https://git-scm.com/docs/git-diff).

Compare modified files that are on the staging area:

1. git diff --staged

Display the diff of what is in a-branch but is not in b-branch:

1. git diff a-branch..b-branch

Show the diff between two specific commits:

1. git diff 61ce3e6..e1d9c

Track path changes by deleting a file from your project and stage this removal for commit:

1. git rm file

Or change an existing file path and then stage the move:

1. git mv existing-path new-path

Check the commit log to see if any paths have been moved:

1. git log --stat -M

**Stashing**

Sometimes you’ll find that you made changes to some code, but before you finish you have to begin working on something else. You’re not quite ready to commit the changes you have made so far, but you don’t want to lose your work. The git stash command will allow you to save your local modifications and revert back to the working directory that is in line with the most recent HEAD commit.

Stash your current work:

1. git stash

See what you currently have stashed:

1. git stash list

Your stashes will be named stash@{0}, stash@{1}, and so on.

Show information about a particular stash:

1. git stash show stash@{0}

To bring the files in a current stash out of the stash while still retaining the stash, use apply:

1. git stash apply stash@{0}

If you want to bring files out of a stash, and no longer need the stash, use pop:

1. git stash pop stash@{0}

If you no longer need the files saved in a particular stash, you can drop the stash:

1. git stash drop stash@{0}

If you have multiple stashes saved and no longer need to use any of them, you can use clear to remove them:

1. git stash clear

**Ignoring Files**

If you want to keep files in your local Git directory, but do not want to commit them to the project, you can add these files to your .gitignore file so that they do not cause conflicts.

Use a text editor such as nano to add files to the .gitignore file:

1. nano .gitignore

To see examples of .gitignore files, you can look at GitHub’s [.gitignore template repo](https://github.com/github/gitignore).

**Rebasing**

A rebase allows us to move branches around by changing the commit that they are based on. With rebasing, you can squash or reword commits.

You can start a rebase by either calling the number of commits you have made that you want to rebase (5 in the case below):

1. git rebase -i HEAD~5

Alternatively, you can rebase based on a particular commit string or hash:

1. git rebase -i 074a4e5

Once you have squashed or reworded commits, you can complete the rebase of your branch on top of the latest version of the project’s upstream code. Note that some repositories may use master instead of main:

1. git rebase upstream/main

To learn more about rebasing and updating, you can read [How To Rebase and Update a Pull Request](https://www.digitalocean.com/community/tutorials/how-to-rebase-and-update-a-pull-request), which is also applicable to any type of commit.

**Reverting and Resetting**

You can revert back the changes that you made on a given commit by using revert. Your working tree will need to be clean in order for this to be achieved:

1. git revert 1fc6665

Sometimes, including after a rebase, you need to reset your working tree. You can reset to a particular commit, and **delete all changes**, with the following command:

1. git reset --hard 1fc6665

To force push your last known non-conflicting commit to the origin repository, you’ll need to use --force:

**Warning**: Force pushing to the main (sometimes master) branch is often frowned upon unless there is a really important reason for doing it. Use sparingly when working on your own repositories, and work to avoid this when you’re collaborating.

1. git push --force origin main

To remove local untracked files and subdirectories from the Git directory for a clean working branch, you can use git clean:

1. git clean -f -d

If you need to modify your local repository so that it looks like the current upstream main branch (that is, there are too many conflicts), you can perform a hard reset:

**Note**: Performing this command will make your local repository look exactly like the upstream. Any commits you have made but that were not pulled into the upstream **will be destroyed**.

1. git reset --hard upstream/main

**Conclusion**

This guide covers some of the more common Git commands you may use when managing repositories and collaborating on software.

You can learn more about open-source software and collaboration in our [Introduction to Open Source tutorial series](https://www.digitalocean.com/community/tutorial_series/an-introduction-to-open-source):

* [How To Contribute to Open Source: Getting Started with Git](https://www.digitalocean.com/community/tutorials/how-to-contribute-to-open-source-getting-started-with-git)
* [How To Create a Pull Request on GitHub](https://www.digitalocean.com/community/tutorials/how-to-create-a-pull-request-on-github)
* [How To Rebase and Update a Pull Request](https://www.digitalocean.com/community/tutorials/how-to-rebase-and-update-a-pull-request)
* [How To Maintain Open-Source Software Projects](https://www.digitalocean.com/community/tutorials/how-to-maintain-open-source-software-projects)

There are many more commands and variations that you may find useful as part of your work with Git. To learn more about all of your available options, you can run the following to receive useful information:

1. git --help

You can also read more about Git and look at Git’s documentation from the [official Git website](https://git-scm.com/).