***Version control terminology***

You’ll encounter many terms when reading about version control. This section offers

a brief listing of those terms and a short definition for each:

■ *Branch*—Another path, or line, of development in the repository.

■ *Check out*—To request a copy of a file so you can work on it; a typical feature of

centralized version control systems.

■ *Clone*—To make a copy of a repository that exists somewhere locally (in another

directory) or remotely (on another server, or Git hosting site such as GitHub).

■ *Commit*—A change that’s saved to a repository, recording itself into the timeline.

■ *Distributed*—A characteristic of a system such that its operations can be performed

without the need of a server (as opposed to *centralized*).

■ *Repository*—A storage area for files; in the context of version control, this storage

area is usually a directory or folder with special operations for viewing the

timeline, committing files, and branching.

■ *Staging area*—A feature of Git that enables the developer to commit certain

parts of files instead of the whole file.

■ *Timeline*—A set of events ordered by time, from the earliest to the most recent

event; also known as a *history.*

■ *Version control*—The practice of keeping track of changes such that you can

always go back to a known state.

|  |  |
| --- | --- |
| git config --global user.name "*Your Name*" | Add your name to the global Git configuration |
| git config --global user.email "*Your E-mail Address*" | Add your email address to the global Git configuration |
| git config --list | Display all the Git configurations |
| git config user.name | Display the user.name configuration value |
| git config user.email | Display the user.email configuration value |
| git help help | Ask Git for help about its help system |
| git help -a | Print all available Git commands |
| git --paginate help -a | Paginate the display of all Git commands |
| git help -g | Print all available Git guides |
| git help glossary | Display the Git glossary |
| git init | Initialize a Git repository in the current directory |
| git status | Display status of current directory, as it relates to Git |
| git add FILE | Start tracking FILE in Git; adds FILE to the staging area |
| git commit -m MSG | Commit changes to the Git repository, with a message (in quotes) |
| git log | Display the log (history) of the Git repository |
| git log --stat | Display the log with the files that were modified |
| git ls-files | List the files in the repository |
| git gui | Start Git GUI |
| git citool | Start Git GUI to commit changes |
| gitk | Start gitk (git log viewer) |
| git commit -m "Message" | Commit changes with the log message entered on command line via the -m switch |
| git diff | Show any changes between the tracked files in the current directory and the repository |
| git commit -a -m "Message" | Perform a git add, and then a git commit with the supplied log message |
| git diff --staged | Show any changes between the staging area and the  repository |
| git add --dry-run . | Show what git add would do |
| git add . | Add all new files in the current directory (use git status afterward to see what was added) |
| git log --shortstat --oneline | Show history using one line per commit, and listing each file changed per commit |
| git rm file | Remove *file* from staging area |
| git mv file1 file2 | Rename *file1* to *file2* in the staging area |
| git add -p | Pick parts of your changes to add to the staging area |
| git checkout file | Check out the latest committed version of the file into your  working directory |
| git reset file | Reset your staging area, removing any changes you’ve added with git add |
| git log --patch | Display the history, showing the file differences between each commit. |
| git log --stat | Display the history, showing a summary of the file changes  between each commit. |
| git log --patch-with-stat | Display the history, combining patch and stat output. |
| git log --oneline file\_one. | Display the history for file\_one |
| git rev-parse | Translate a branch name or a tag name to a specific SHA1 ID. |
| git checkout YOUR\_SHA1ID | Change your working directory to match the version specified in YOUR\_SHA1ID. |
| git tag TAG\_NAME -m "MESSAGE" YOUR\_SHA1ID | Create a tag named TAG\_NAME, pointing to YOUR\_SHA1ID.  The tag will have a short MESSAGE associated with it. |
| git tag | List all tags. |
| git show TAG\_NAME | Show information about the tag named TAG\_NAME. |
| git branch | List all branches. |
| git branch dev | Create a new branch named dev. (This branch points to the same commit as HEAD.) |
| git checkout dev | Change your working directory to the branch named dev. |
| git branch -d master | Delete the branch named master. |
| git log --graph --decorate  --pretty=oneline --all --abbrev-commit | View history of the repository across all branches. |
| git config --global alias.lol "log  --graph --decorate --pretty=oneline  --all --abbrev-commit" | Make an alias named lol for the git log command in the previous row (see the Above and Beyond sidebar). git branch -v List all branches with SHA1 ID information. |
| git branch fixing\_readme YOUR\_SHA1ID | Make a branch using YOUR\_SHA1ID as the starting point. |
| git checkout -b another\_fix\_branch fixing\_readme | Make a branch named another\_fix\_branch using branch fixing\_readme as the starting point. |
| git reflog | Show a record of all the times you changed branches (via git checkout). |
| git stash | Set the current work in progress (WIP) to a stash (holding area), so you can perform a git checkout. |
| git stash list | List works in progress that you’ve stashed away. |
| git stash pop | Apply the most recently saved stash to the current working directory; remove it from the stash. |
| git diff BRANCH1...BRANCH2 | Indicate the difference between BRANCH1 and BRANCH2 relative to when they first became different. |
| git diff --name-status BRANCH1...BRANCH2 | Summarize the difference between BRANCH1 and BRANCH2, by listing each file and its status. |
| git merge BRANCH2 | Merge BRANCH2 into the current branch that you’re on. |
| git log -1 | A shorthand for git log -n 1 (show only the most recent commit). |
| git mergetool | Open a tool to help perform a merge between two conflicted branches. |
| git merge --abort | Abandon a merge between two conflicted branches. |
| git merge-base BRANCH1 BRANCH2 | Show the base commit between BRANCH1 and BRANCH2. |
| git clone source destination\_dir | Clone the Git repository at source to the destination\_dir. |
| git log --oneline --all | Display all commit log entries from all branches. (Normally, git log displays only entries from the current branch.) |
| git log --simplify-by-decoration --decorate --all --oneline | Display the history in a simplified form. |
| git branch --all | Show remote-tracking branches in addition to local branches. |
| git clone --bare source destination\_dir | Clone the bare directory of the source repository into the destination\_dir. By convention, destination\_dir should end with .git. |
| git ls-tree HEAD | Display all the files for HEAD (the current branch). |
| git checkout -f master | Check out the master branch, throwing away any changes in your current branch. |
| git remote | Display the name of the remote(s) in the current repository. |
| git remote -v | show Display the names of the remotes along with the corresponding remote URL. |
| git remote add bob ../math.bob | Add a remote named bob that points to the local repository in ../math.bob. |
| git ls-remote REMOTE | Display the references of a remote repository (use . as the REMOTE when you want the current local repository). |
| GIT\_TRACE\_PACKET git ls-remote REMOTE | Display the underlying network interaction. |
| git tag -a TAG\_NAME -m  TAG\_MESSAGE SHA1ID | Create a tag to the SHA1ID with the name TAG\_NAME and the message TAG\_MESSAGE. |
| git push origin TAGNAME | Push the tag named TAGNAME to the remote named *origin*. |
| git push --tags | Push all tags to the default remote. |
| git push origin :TAGNAME | Delete the tag named TAGNAME on the remote named *origin*. |
| git tag -d TAGNAME | Remove the tag named TAGNAME from your local repository. |
| git config –global push.default simple | Set the push.default configuration variable to simple for all repositories that you have access to (globally). |
| git pull | Sync your repository with the repository that you cloned from (a.k.a. the  upstream repository). This command comprises two commands: git fetch and git merge. |
| git fetch | The first part of git pull. This brings in new commits from the remote  repository and updates the remote-tracking branch. |
| git merge FETCH\_HEAD | Merge the new commits from FETCH\_HEAD into the current branch. |
| git pull --ff-only | The --ff-only switch will allow a merge only if FETCH\_HEAD is a  descendant of the current branch (a fast-forward merge). |
| git log --merges | List commits that are the result of merges. |
| git log --oneline FILE. | List commits that affect FILE |
| git log --grep=STRING | List commits that have STRING in the commit message. |
| git log --since MM/DD/YYYY --until MM/DD/YYYY | List commits between two dates. |
| git shortlog | Summarize commits by authors. |
| git shortlog -e | Summarize commits by authors (and show email address). |
| git log --author=AUTHOR | List commits by AUTHOR (name or email). |
| git log --stat HEAD^..HEAD | List commits (with files) between the current commit and its immediate parent. |
| git log --patch HEAD^..HEAD | List commits (with text changes) between the current  commit and its immediate parent. |
| git branch --column | List all branches in columns |
| git name-rev SHA1\_ID | Print a name for the specified SHA ID, based on the closest branch. |
| git branch -r --contains SHA1\_ID | Similar to the preceding command, in that it will identify all the branches that contain this SHA1 ID (-r specifies remote-tracking branches; omit this to print local branches). |
| git grep STRING | Find all files that contain STRING. |
| git gui blame FILE | Bring up a FILE in the Git GUI showing git blame output (each line showing what commit it’s from). |
| git gui browser REV | List all files at REV (use HEAD for the current directory) in the GUI browser. |
| git blame FILE | Display blame output of FILE on the command line. |
| git --no-pager blame FILE > FILE-annotate | Save the blame output of FILE to FILE-annotate on the  command line. |
| git log --oneline master..new\_feature | Show the commits between the master branch and the new\_feature branch. |
| git rebase master | Rebase your current branch with the latest commit from master. |
| git reflog | Display the reflog (the internal history of all the times that you changed HEAD). |
| git reset --hard HEAD@{4} | Reset HEAD to point to the SHA1 ID represented by HEAD@{4}. The --hard switch says to reset both the staging area and the working directory. |
| git rebase –interactive master | Interactively rebase your current branch with the latest commit from master. This opens an editor, allowing you to pick and choose which commits will be included in the rebase. |
| git cherry-pick SHA1 ID | Copy the commit to the current branch that you’re on. |
| git commit --allow-empty –m "Initial commit" | Create a commit without adding any files. |
| git merge --no-ff BRANCH | Merge BRANCH into the current branch, creating a merge commit even if it’s a fast-forward commit. |
| git flow | A Git command that becomes available after installing gitflow. |
| git config --local --list | List the local (repository-specific) Git configuration. |
| git config --global --list | List the global (user-specific) Git configuration. |
| git config --system --list | List the system (server-specific) Git configuration. |
| git -c log.date=relative log -n 2 | Show the last two commits using the relative date format. |
| git config --local log.date relative | Save the relative date format in the local Git configuration. |
| git config --local --edit | Edit the local (repository-specific) Git configuration. |
| git config --global --edit | Edit the global (user-specific) Git configuration. |
| git config --system --edit | Edit the system (server-specific) Git configuration. |
| git -c core.editor=echo config --local --edit | Print the name of the local Git configuration file. |
| git -c core.editor=nano config --local --edit | Edit the local Git configuration file using nano. |
| git config core.excludesfile | Print the value of the core.excludesfile Git configuration setting. |
| <https://git-scm.com/downloads/guis> | Several different guis recommended by Git |
| <https://confluence.atlassian.com/bitbucket/resolve-merge-conflicts-704414003.html> | Resolve merge conflicts |