

ZOHAIR BANOORI

# GIT CHEAT SHEET

---

28 JULY 2025 / 11:44 PM

## INSTALLATION & GUIs

**GitHub for Windows**

<https://windows.github.com>

**GitHub for Mac:**

<https://mac.github.com>

**Git for All Platforms:**

<http://git-scm.com>

## SETUP

*Configuring user information used across all local repositories*

- `git config --global user.name "[firstname lastname]"`  
  
# Set a name that is identifiable for credit when reviewing version history
  - `git config --global user.email "[valid-email]"`  
  
# Set an email address that will be associated with each history marker
  - `git config --global color.ui auto`  
  
# Set automatic command line coloring for Git for easy reviewing
-

# CREATE & INITIALIZE

*Creating and cloning Git repositories*

- `git init`  
`# Initialize an existing directory as a Git repository`
  - `git clone [url]`  
`# Retrieve an entire repository from a hosted location via URL`
- 

# STAGE & SNAPSHOT

*Working with snapshots and the Git staging area*

- `git status`  
`# Show modified files in working directory, staged for your next commit`
- `git add [file]`  
`# Add a file as it looks now to your next commit (stage)`
- `git reset [file]`  
`# Unstage a file while retaining the changes in working directory`
- `git diff`  
`# Diff of what is changed but not staged`
- `git diff --staged`  
`# Diff of what is staged but not yet committed`
- `git commit -m "[descriptive message]"`  
`# Commit your staged content as a new commit snapshot`

---

## UNDOING CHANGES

*Mistake recovery and safe rollback options*

- `git checkout <branch>`  
`# Switch to another branch (e.g., git checkout main)`
- `git reset`  
`# Unstage staged files (after git add)`
- `git reset --hard [commit]`  
`# Reset to specific commit (dangerous – loses local changes)`
- `git stash`  
`# Temporarily shelve your changes to clean your working directory`
- `git commit --amend`  
`# Modify the most recent commit (do not use on published commits)`
- `git revert [commit]`  
`# Revert changes by creating a new commit that undoes a specific commit`

Refer to [Git Basics – Undoing Things](#) for further details.

---

# BRANCH & MERGE

## *Branch operations and history tracking*

- `git branch`  
  
# List your branches. A \* will appear next to the currently active branch
- `git branch [branch-name]`  
  
# Create a new branch at the current commit
- `git branch -d [name]`  
  
# Delete a branch from your repository
- `git branch -D [name]`  
  
# Force delete a branch from your repository
- `git checkout [branch-name]`  
  
# Switch to another branch and check it out into your working directory
- `git checkout -b [new-branch]`  
  
# Create a new branch and switch to it
- `git merge [branch]`  
  
# Merge the specified branch's history into the current one
- `git merge --abort`  
  
# Abort a merge and return to the pre-merge state (use after merge conflicts)
- `git log`  
  
# Show all commits in the current branch's history

- `git log --graph`  
# Print an ASCII graph of the commit and merge history
  - `git log --oneline`  
# Print each commit on a single line
- 

## SHARE & UPDATE

*Synchronizing your repository with remotes*

- `git remote`  
# Manage the set of tracked remote repositories
- `git remote -v`  
# Show remote URLs for fetch/push
- `git remote show <name>`  
# Display detailed information about a specific remote
- `git remote update`  
# Fetch updates from all remotes or a group of remotes
- `git remote add [alias] [url]`  
# Add a git URL as an alias
- `git branch -r`  
# List all remote-tracking branches
- `git fetch [alias]`  
# Fetch all the branches from that Git remote

- `git merge [alias]/[branch]`

# Merge a remote branch into your current branch to bring it up to date

- `git push [alias] [branch]`

# Transmit local branch commits to the remote repository branch

- `git pull`

# Fetch and merge any commits from the tracking remote branch

---

## TEMPORARY COMMITS

*Preserve work-in-progress using stash*

- `git stash`

# Save modified and staged changes

- `git stash list`

# List stack-order of stashed file changes

- `git stash pop`

# Write working from top of stash stack

- `git stash drop`

# Discard the changes from top of stash stack

---

# TRACKING PATH CHANGES

*Tracking file renames and deletions*

- `git rm [file]`  
`# Delete the file from project and stage the removal for commit`
  - `git mv [existing-path] [new-path]`  
`# Change an existing file path and stage the move`
  - `git log --stat -M`  
`# Show all commit logs with indication of any paths that moved`
- 

# INSPECT & COMPARE

*Comparing branches, diffs, and commit history*

- `git log`  
`# Show the commit history for the currently active branch`
- `git log branchB..branchA`  
`# Show the commits on branchA that are not on branchB`
- `git log --follow [file]`  
`# Show the commits that changed file, even across renames`
- `git diff branchB...branchA`  
`# Show the diff of what is in branchA that is not in branchB`
- `git show [SHA]`  
`# Show any object in Git in human-readable format`

---

## REWRITE HISTORY

*Rewriting and cleaning up commit history*

- `git rebase [branch]`  
# Apply any commits of current branch ahead of specified one
- `git reset --hard [commit]`  
# Clear staging area, rewrite working tree from specified commit

Use with caution. **Do not** rewrite history on shared/public branches.

---

## IGNORING PATTERNS

*Preventing unintentional staging or committing of files*

- `git config --global core.excludesfile [file]`  
# System-wide ignore pattern for all local repositories

`.gitignore` example:

```
logs/  
  
*.notes  
  
pattern*/
```

Save a file with desired patterns as `.gitignore` with either direct string matches or wildcard globs.

---



# SHA-1 & OBJECTS

*Git internal structure – commit identification and integrity*

**Git uses SHA-1 hashes for commit identification.**

- SHA-1 is a cryptographic hash function
- It generates a unique digital fingerprint for each file/commit
- Ensures file integrity and serves as a reference (e.g., in `git revert [SHA]`)

Hashes are visible in `git log` or on GitHub pages and are used across many Git commands.

## GIT TERMS & DEFINITIONS

- **Branch:** A pointer to a particular commit, representing an independent line of development in a project.
- **Commit:** A command to make edits to multiple files and treat that collection of edits as a single change.
- **Commit files:** A stage where the changes made to files are safely stored in a snapshot in the Git directory.
- **Commit ID:** An identifier next to the word `commit` in the log.
- **Commit message:** A summary and description with contextual information on the parts of the code or configuration of the commit change.
- **Diff:** A command to find the differences between two files.
- **DNS zone file:** A configuration file that specifies the mappings between IP addresses and host names in your network.
- **Fast-forward merge:** A merge when all the commits in the checked out branch are also in the branch that's being merged.
- **Git:** A free open source version control system available for installation on Unix-based platforms, Windows and macOS.

- **Git directory:** A database for a Git project that stores the changes and the change history.
  - **Git log:** A log that displays commit messages.
  - **Git staging area:** A file maintained by Git that contains all the information about what files and changes are going to go into the next commit.
  - **Head:** This points to the top of the branch that is being used.
  - **Master:** The default branch that Git creates when a new repository is initialized; commonly used to place the approved pieces of a project.
  - **Merge conflict:** This occurs when the changes are made on the same part of the same file, and Git won't know how to merge those changes.
  - **Modified files:** A stage where changes have been made to a file, but they have not been stored or committed.
  - **Patch:** A command that can detect that there were changes made to the file and will do its best to apply the changes.
  - **Repository:** An organization system of files that contain separate software projects.
  - **Rollback:** The act of reverting changes made to software to a previous state.
  - **Source Control Management (SCM):** A tool similar to VCS to store source code.
  - **Stage files:** A stage where the changes to files are ready to be committed.
  - **Three-way merge:** A merge that uses the snapshots at the two branch tips along with their most recent common ancestor (the commit before the divergence).
  - **Tracked:** A file's changes are recorded.
  - **Untracked:** A file's changes are not recorded.
  - **Version control systems (VCS):** A tool to safely test code before releasing it, allowing multiple people to collaborate on the same coding projects together, and stores the history of that code and configuration.
-

# EDUCATION

GitHub is **free** for students and teachers. Discounts available for other educational uses.

- **Email:** [education@github.com](mailto:education@github.com)
- **Website:** <https://education.github.com>