Git Cheat Sheet

Git is the free and open source distributed version control system. This cheat sheet saves you from learning all the commands by heart and features the most important and commonly used Git commands for easy reference.

Be free to contribute and update the grammar mistakes.

Table of Contents

Setup

Show configuration:

Current configuration:

```
$ git config --list
```

Repository configuration:

```
$ git config --local --list
```

Global configuration:

```
$ git config --global --list
```

System configuration:

```
$ git config --system --list
```

Repository Compression (range -1 to 9):

-1 is default compression level. 0 value means no compression, and 1 to 9 are various speed/size tradeoffs, 9 being slowest.

```
$ git config --global core.compression 2
```

Diff tool (Visual Studio Code):

```
$ git config --global diff.tool vscode
```

Merge tool (Visual Studio Code):

```
$ git config --global merge.tool vscode
```

Configuring user information used across all local repositories

Set a name that is identifiable for credit when review version history:

```
$ git config --global user.name “Github-username”
```

Set an email address that will be associated with each history marker:

```
$ git config --global user.email “Github-email-id”
```

Set automatic command line coloring for Git for easy reviewing:

```
$ git config --global color.ui auto
```

Set global editor for commit (Visual Studio Code):

```
$ git config --global core.editor “code –wait”
```

💡 **Remember**

- **Ask for help** `git <command> --help`
- **master** is the default development branch
- **origin** is the default upstream repository

Create

Initializing and cloning repositories:

Create a new local repository in the current directory or reinitialize an existing one:

```
$ git init
```

Create a new local repository in a specific directory:

```
$ git init [directory_name]
```

Clone an existing repository:

There are two ways:

- SSH
- HTTP

By SSH: `$ git clone ssh://user@domain.com/repo.git`

By HTTP: `$ git clone http://domain.com/user/repo.git`

If you don't like the name of the repository you are cloning, just type the following command in your terminal:

```
$ git clone [HTTPS or SSH Link] [new_file_name]
```

Local Changes

Working with snapshots and the Git staging area:

Add a file as it looks now to your next commit (stage):

```
$ git add [file_name]
```

Add all current changes to the next commit(root directory):

```
$ git add .
```

Add all current changes to the next commit(root and other directory):

```
$ git add --all   or    $ git add -A    or    $ git add *
```

Changes in working directory, staged for your next commit:

```
$ git status
```

Give the output in the short-format:

```
$ git status -s
```

Unstage a file while retaining the changes in working directory:

```
$ git reset [file_name]
```

OR

```
$ git rm --cached [file_name]
```

Remove untracked files from your repository:

```
$ git clean -f
```

Diff of what is changed but not staged:

```
$ git diff
```

Diff of what is staged but not yet commited:

```
$ git diff --staged
```

See difference of a specific file:

```
$ git diff [file_name]
```

Commit with message:

```
$ git commit -m “[descriptive message]”
```

Branch and Merge

Isolating work in branches, changing context, and integrating changes:

Create a new branch:

```
$ git branch [branch_name]
```

List all local branches ([*]marks represents the current branch):

```
$ git branch
```

List local and remote branches:

```
$ git branch -a
```

List all remote branches:

```
$ git branch -r
```

Switch to a branch:

```
$ git checkout [branch_name]
```

Checkout single file from different branch:

```
$ git checkout [branch_name] -- [file_name]
```

Create and switch to a new branch:

```
$ git checkout -b [branch_name]
```

Create a new branch from an exiting branch and switch to a new branch:

```
$ git checkout -b [new_branch_name] [existing_branch_name]
```

Delete a local branch:

```
$ git branch -d [branch_name]
```

Rename current branch to new branch name:

```
$ git branch -m [new_branch_name]
```

Merge the specified branch's history into the current one:

```
$ git merge [branch]
```

Show all commits in the current branch's history:

```
$ git log
```

Inspect and Compare

Examining logs, diffs and object information:

Show all commits, starting with newest for the active branch (it'll show the hash, author information, date of commit and title of the commit):

```
$ git log
```

Show all the commits (it'll show just the commit hash and the commit message):

```
$ git log --oneline
```

Show all commits of a specific user:

```
$ git log --author="username"
```

Show the commits on branchA that are not on branchB:

```
$ git log branchB..branchA
```

Show changes over time for a specific file:

```
$ git log -p [file_name]
```

Show the commits that changed file, even across renames:

```
$ git log --follow [file_name]
```

Show history of changes for a file with diffs:

```
$ git log -p [file_name] [directory_name]
```

Show the diff of what is in branchA that is not in branchB:

```
$ git diff branchB...branchA
```

Who changed, what and when in a file:

```
$ git blame [file_name]
```

Show the metadata and content changes of the specified commit:

```
$ git show
```

Show any object in Git in human-readable format:

```
$ git show [SHA]
```

A commit identified by ID:

```
$ git show [ID]
```

A specific file from a specific ID:

```
$ git show [ID]:[file]
```

Show Reference log:

```
$ git reflog show
```

Delete Reference log:

```
$ git reflog delete
```

Commit and Revert

Commit with message:

```
$ git commit -m 'message here'
```

Commit all local changes in tracked files:

```
$ git commit -a
```

Commit previously staged changes:

```
$ git commit
```

Commit skipping the staging area and adding message:

```
$ git commit -am 'message here'
```

OR,

```
$ git commit -a -m 'message here'
```

Updates the last commit without creating a new commit(default editor will open - VS Code):

```
$ git commit --amend
```

Fix the last commit (after editing the broken files - default editor will open - VS Code):

```
$ git commit -a --amend
```

Amend a file then commit in terminal (fresh new commit will be created):

```
$ git add [file_name]
```

```
$ git commit --amend -m 'commit message goes here'
```

Create a single commit on top of the current branch:

```
$ git commit --squash
```

Undo your last your commit and move HEAD pointer to a previous commit:

```
$ git reset --soft HEAD^
```

Return to the last commited state(This can't be undone):

```
$ git reset --hard
```

Discard all local changes in your working directory:

```
$ git reset --hard HEAD
```

Revert the last commit (Create a new commit):

```
$ git reset HEAD
```

Revert a commit (by producing a new commit with contrary changes):

```
$ git revert [commit]
```

Reset your HEAD pointer to a previous commit and discard all changes since then:

```
$ git reset --hard [commit]
```

Temporary Commits

Temporarily store modified, tracked files in order to change branches:

Save modified and staged changes:

```
$ git stash
```

List stack-order of stashed file changes:

```
$ git stash list
```

Restore stashed changes back to current branch:

```
$ git stash apply
```

Restore particular stash back to current branch: *{stash_number}* can be obtained from `git stash list`

```
$ git stash apply stash@{stash_number}
```

Write working from top of stash stack:

```
$ git stash pop
```

Remove the last set of stashed changes:

```
$ git stash drop
```

Update & Publish

Retrieving updates from another repository and updating local repos:

List all current configured remote:

```
$ git remote -v
```

Show information about a remote:

```
$ git remote show [remote]
```

Add new remote repository, named (origin):

```
$ git remote add origin [HTTPS URL]
```

Rename a remote repository, from (remote) to (new_remote):

```
$ git remote rename [remote] [new_remote]
```

Remove a remote:

```
$ git remote rm [remote]
```

Note: git remote rm does not delete the remote repository from the server. It simply removes the remote and its references from your local repository.

Download all changes from (remote), but don't integrate into HEAD:

```
$ git fetch [remote]
```

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

```
$ git merge [remote]/[branch_name]
```

Fetch and merge any commits from the tracking remote branch:

```
$ git pull
```

Get all changes from HEAD to local repository:

```
$ git pull origin master
```

Download changes and directly merge/integrate into HEAD:

```
$ git remote pull [remote] [url]
```

Get all changes from HEAD to local repository without a merge:

```
$ git pull --rebase [remote] [branch_name]
```

Publish local changes on a remote:

```
$ git push remote [remote] [branch_name]
```

OR

Here remote server is origin and branch is master

```
$ git push -u origin master
```

By using this command, next time when you publish your local changes to remote then use:
`$ git push`

Delete a branch on the remote:

```
$ git push [remote] :[branch]
```

OR

```
$ git push [remote] --delete [branch_name]
```

Delete a branch on the remote:

```
$ git push [remote] :[branch]
```

OR

```
$ git push [remote] --delete [branch_name]
```

Publish your tags

```
$ git push --tags
```

Publish your tags to remote repository:

```
$ git push [remote] tag [tag_name]
```

Tracking path changes

Versioning file removes and path changes:

A text search on all files in the directory (without count number):

```
$ git grep "Hello"
```

A text search on all files in the directory (with count number):

```
$ git grep -n "Hello"
```

A text search on all files in the directory (only count number):

```
$ git grep -c "Hello"
```

Delete the file from project and stage the removal for commit:

```
$ git rm [file_name}
```

Change an existing file path and stage the move:

```
$ git mv [existing_path] [new_path]
```

Show all commit logs with indication of any paths that moved:

```
$ git log --stat -M
```

Rename a file:

```
$ git mv [old_file_name] [new_file_name]
```

Ignoring Patterns

Preventing unintentional staging or commiting of files:

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

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

System wide ignore pattern for all local repositories:

```
$ git config --global core.excludesfile [file_name]
```

Tags

Mark HEAD with a tag:

```
$ git tag [tag_name]
```

Mark HEAD with a tag and open the editor to include a message:

```
$ git tag -a [tag_name]
```

Mark HEAD with a tag that includes a message:

```
$ git tag [tag_name] -am [message goes here]
```

OR

```
$ git tag -a [tag_name] -m [message goes here]
```

Tag a particular commit with hash number(Commit id) instead of the HEAD pointer:

```
$ git tag [tag_name] [hash_number]
```

List all tags:

```
$ git tag
```

List all tags with their messages (tag message or commit message if tag has no message):

```
$ git tag -n
```

Delete a tag from local repository:

```
$ git tag -d [tag_name]
```

Delete a tag from remote repository:

```
$ git push [remote] :[tag_name]
```

Rewrite History

Rewriting branches, updating commits and clearing history:

Apply any commits of current branch ahead of specified one:

```
$ git rebase [branch_name]
```

Clear staging area, rewrite working tree from specified commit:

```
$ git reset --hard [Commit]
```