Opensource.com: Git Cheat Sheet

BY MATT BROBERG

Git is the dominant version control utility these days. Here's how to be effective using it.

The Essentials — When working with git on your own or with others.

`git status`	To remind you of where you left off. See a summary of local changes, remote commits, and untracked files.
`git diff`	To see the specific local changes to tracked files. Use `--name-only` to see changed filenames.
`git add`	To stage changes to your tracked and untracked files. Use `-u`, `-a`, and `.` strategically.
`git commit`	To create a new commit with changes previously added. Use `-m` and add a meaningful commit message.
`git push`	To send changes to your configured `remote` repository, most commonly GitLab or GitHub.

Basic Flow — Daily usage of git, including flags

`git init`
`git status`
`git add --all`
`git status`
`git commit -m "meaningful initial commit message"`
`git show`
	cd to your local project that you want to start versioning with git. You only have to run `git init` the first time to set up the directory for version tracking.

`git diff`
`git commit -a -m "Another commit message. -a performs the add step for you"`
`git status`
`git log --graph --pretty=oneline --abbrev-commit`
	And you begin to hack on your local files, then commit at regular intervals

`git log --graph --pretty=oneline --abbrev-commit`
`git reset --soft HEAD~3`
`git diff --cached`
`git commit -a -m "Better commit message for last 3 commits"`
	After a while, you have 3 commits that would be more meaningful as a single commit

`git status`
`git diff --cached`
`git add -u`
`git commit -m "Another commit message. -u adds updates, including deleted files"`
`git status`
`git log --graph --pretty=oneline --abbrev-commit`
`git push origin master`
	Lastly, you delete some unneeded files in the current directory

Basic Branching — Branches represent a series of commits.

`git branch --all`	list all local and remote branches
`git checkout <branch>`	change to an existing branch
`git checkout -b <branch> master`	make a branch based off of `master` and check it out
`git checkout master && git merge <branch>`	merge branch changes onto `master`

Getting Help

`git <cmd> -h`	great for quick review of command flags
`git <cmd> --help`	to dig into the full man pages of the command

Important Flags — These are my personal favorites for keeping everything organized.

`git reset HEAD --`	get back to the last known commit and unstage others
`git add -u`	add only the updated, previously-committed files
`git log --graph --pretty=oneline --abbrev-commit`	for a pretty branch history. Create a shell or git alias for easy access, such as `git lg`

Working with a Remote Repository — Once you get into the flow, you'll frequently contribute back to larger projects, and possibly managing forks of forks. Here are some tips for doing so.

`git fetch --all`	downloads all commits, files, and references to branches on all remote repositories so you can then `git checkout` or `pull` what you want to work on.
`git pull --rebase <remote> <branch>`	Merge all commits since your last common commit from the remote branch without creating a merge commit
`git stash`	Use this as needed to save uncommitted changes so you can `git stash pop` them onto a different branch.
`git stash pop`	bring it back
`git add [-A or . or -- <filename>]`	Be intentional about what files you add to your commits, especially if you want to open a request to merge them into an upstream project.
`git commit -m "commit message"`	Most projects have a format they prefer for commit messages. Look at CONTRIBUTING.md files in the project or review previous commits to get an idea of their format.
`git push origin <branch>`	Push your current branch to your remote titled "origin" and branch named <branch>
`git checkout -b <new_branch>`	A shortcut for `git branch <branch> && git checkout branch`. It's great for when you want to experiment with an idea and have a new branch to try it out on that can later be merged or deleted.
`git checkout master && git pull --rebase`	Great to get to the most recent commit for a project you only infrequently follow.
`git reset --hard origin/master`	For when you inevitably get lost in all the git-fu and need to get to a known state. WARNING: this erases all changes, even commits, since the last commit pushed to the remote `origin` on branch `master`.
`git push origin master`	For when you inevitably do something right! Send your changes up to your remote titled `origin` on branch `master`.

Helpful Reads

- Read this excellent guide to your first git repository
- Learn more about git branching
- Dig deeper into reset and rebase

 CC BY-SA 4.0