GitNoon Cheat Sheet

Part of the TechNoon.org Movement

Standard Machine Configuration

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
git config --global core.editor nano
git config --global init.defaultBranch main
git config --global alias.graph \
    "log --all --graph --oneline --decorate"
# For Linux or macOS only:
git config --global core.autocrlf input
# For Windows only:
git config --global core.autocrlf true
# Only if you're behind a corporate proxy on Windows:
git config --global http.sslBackend schannel
# Check configuration:
git config -l
```

Basic Versioning

`git <command> --help` Show help for any `<command>`.

`git init` Setup repository by creating `.git/` in current directory.

`git status` Show the current state of the branch, uncommitted changes, and staged changes.

`git add <files>` Add uncommitted changes in `<files>` to the staging area. Adding `--patch` prompts for each change.

`git commit` Create a new commit from staged changes.

`git tag -a <name> -m "<message>"` Add an annotated tag to the current commit with the given `<message>`.

`git diff` Compare uncommitted changes to the last commit. Optionally add `<commit>` to compare to that commit.

`git diff --staged` Show differences between the staging area and the last commit (useful before committing).

`git restore <files>` Update `<files>` to match the last commit.

`git restore --source=<commit> <files>` Update `<files>` to match the specified `<commit>` (e.g. `HEAD`, SHA, branch, or tag).

`git revert` Create a commit that reverses changes in `<commit>`.

Exploring History

`git log` Show the history of commits in the current branch.

`git log <branch>` Show commits in the specified `<branch>`, or specify `--all` for all branches.

`git log <files>` Show commits that changed the specified `<files>`. Add `--follow` to trace history across file renames.

`git log --stat` Include summary of changes for each commit.

`git log --patch` Show the full diff for each commit.

`git log --author "<author>"` Show commits by `<author>`.

`git log --since "<start-date>" --until "<end-date>"` Show commits made between `<start-date>` and `<end-date>`.

`git log --grep "<search-term>"` Show commits with a message that contains `<search-term>`.

`git log -S "<search-term>"` Show commits with a diff that contains `<search-term>`. AKA The Git Pickaxe.

`git blame <file>` Show the commit, date, and author of each line in `<file>`. Ignore whitespace changes with `-w`, and optionally trace copies within the file (`-M`) or across files (`-C`).

`git bisect start <bad> <good>` Binary search to find the first "bad" commit after `<good>` commit and before `<bad>` commit.

`git bisect good` During bisect, mark current commit as "good".

`git bisect bad` During bisect, mark current commit as "bad".

`git bisect skip` During bisect, mark current commit as "unsure".

`git bisect reset` Restore the state of the repository prior to running `git bisect start`.

Working with Remotes

`git clone <remote-url>` Clone the specified remote repoistory as a local repository in a new directory.

`git remote add origin <remote-url>` Add a remote named `origin` to an existing local repository.

`git remote -v` Show remotes that can be pushed and fetched.

`git push -u origin <branch>` Push the local state of `<branch>` to the `origin` remote. Often shortened to `git push` after the first push to a particular branch. Include tags with `--tags`.

`git fetch` Update the local references to remote branches for the default remote (usually `origin`). Include tags with `--tags`.

`git merge --ff-only origin/main` Safely update the local branch with fetched commits on the `origin/main` remote reference. Fails if local commits prevent fast-forwarding.

Branching and Merging

`git graph` Graphically log branches (alias in standard config).

`git branch <branch-name>` Create a new branch named `<branch-name>` from the currently checked-out branch/commit.

`git switch <branch-name>` Check-out the specified branch. Updates `HEAD` to point to the branch.

`git merge <branch-name>` Update the current branch by merging in commits from `<branch-name>`. Creates a merge commit if fast-forwarding is not possible.

`git merge --abort` Restore repo state prior to `git merge`.

Rewriting History (Danger Zone!)

`git rebase <branch-name>` Update the current branch by replaying its commits on top of `<branch-name>`.

`git rebase --abort` Restore repo state prior to `git rebase`.

`git rebase --interactive HEAD~5` Rewrite the last 5 commits of the current branch by deleting, editing, or re-ordering commits.

`git cherry-pick <commit>` Replay `<commit>` on the current branch.

`git commit --amend` Rewrite the last commit with staged changes.

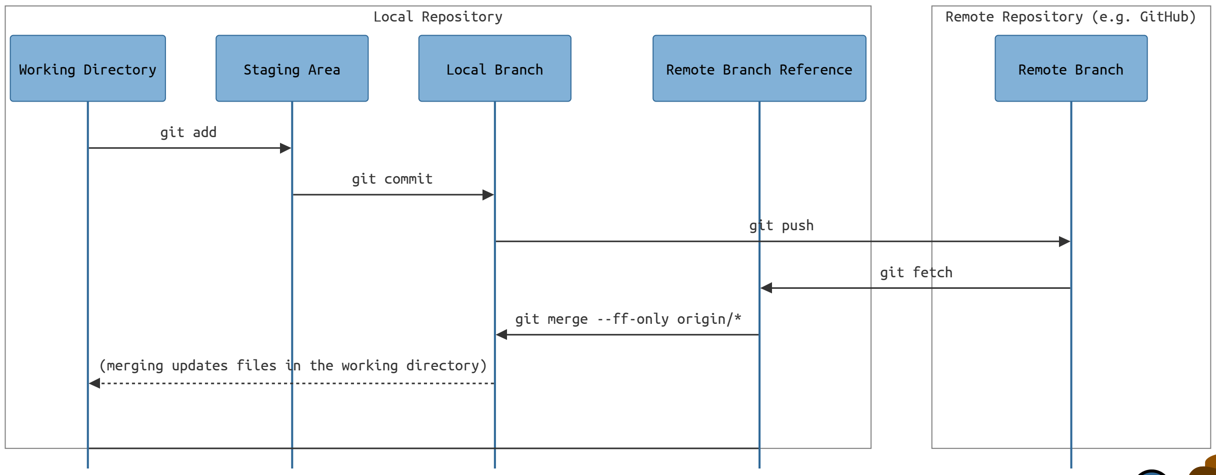
`git reset <commit>` Force current branch to point to `<commit>`, potentially losing commits. `--soft`, `--mixed`, and `--hard` options control what happens to the working directory and staging area.

`git filter-repo` Power tool for rewriting history, e.g. removing a file from every commit in history. Installed independently of Git.

`git push --force` Push the current local state of the current branch to its remote equivalent, even if commits on the remote would be lost.

`git reflog` Show commits that `HEAD` has recently pointed to. Useful for finding a lost commit to `reset` to.

This cheat sheet is licensed under CC BY-SA 4.0.


Local Repository
Remote Repository (e.g. GitHub)
Working Directory
Staging Area
Local Branch
Remote Branch Reference
Remote Branch
git add
git commit
git push
git fetch
git merge --ff-only origin/*
(merging updates files in the working directory)