Made by Flavius Schmidt
for OshrEQ, August 2024.
home.in.tum.de/scfl

Git Cheat Sheet

Basic Workflow

1. `git add <file1> <file2> ...` *or* `git add .`
   stages changes made to specific files *or* everything in current directory.
2. `git commit -m "<msg>"`
   locally records the changes made to the files added in the previous step.
3. `git push`
   uploads the commited changes to the remote repository (online).

Tip: Use `git status` to repeatedly check the changes you made.

Initial Setup

Installation: Follow the instructions on `git-scm.com/download`.

User Configuration:

- `git config --global user.name "<username>"`
  sets the username of your contributions (usually `"<firstname> <lastname>"`).
- `git config --global user.email "<email>"`
  sets the email address of your contributions.
- `git config --global core.editor "<editor> --wait"`
  sets the default code editor for certain manual inputs (e.g. `code`)
- `git config --global color.ui auto`
  sets automatic command line coloring for git.

Tip: There's many more options available to configure, such as aliases.

Downloading (Cloning) a Remote Repository: `git clone <url>`
To clone via SSH (preferred), you need to add a valid SSH key to your GitHub account.

Terminology

- **Workspace** / **Working Tree**: A local copy of the codebase.
  Any changes made here must be staged manually with `git add ...`.

↓ `git add ...`

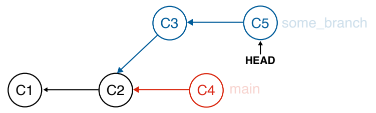
- **Staging Area**: Intermediate zone. Files here can either be commited or removed.
  Used i.a. to review changes before committing.

↓ `git commit ...`

- **(Local) Repository**: Workspace containing necessary version control files.
  This includes branches, the commit history, tracked files etc.

↓ `git push ...`

- **Remote** / **Upstream Repository**: A repository stored online instead of locally.
  The local and remote versions should be in-sync when cloned.



- **Commit**: A save point / check point storing the current state of your codebase.
  Commit Hash: Unique commit ID, primarily used to address it.
- **Branch**: An alternate timeline version of your codebase (default: `main`).
- **HEAD**: A pointer to the current commit / branch of your working directory.

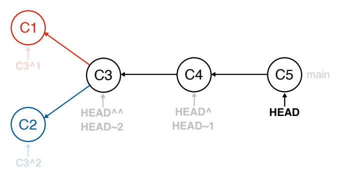
Git-Specific Files

- `.git`: Directory, contains all necessary information for the repo's version control.
  Do **not** manually change files in here unless you know what you are doing.
- `.gitignore`: File, specifies intentionally omitted files that won't be added to the repo.
  Tip: Use an automatic online `.gitignore` generator to create a preset.
- `~/.gitconfig`: File, contains your global git configuration.
  Tip: You can also find samples online for configuration files.

Detached HEADs and Relative Refs

Detached HEAD: HEAD points to a commit; experimental changes can be made.

- **discard changes**: return to some existing branch (e.g. `git checkout main`).
- **keep changes**: create a new branch (e.g. `git checkout -b alternate`).



Relative Ref: Address a commit relative to another commit, a branch or HEAD.

- `^` moves one commit up (e.g. `HEAD^`). Can be stacked (e.g. `HEAD^^`).
  If a commit has multiple parents, `^n` checks out the `n`-th parent.
- `~n` moves `n` commits up (e.g. `HEAD~3`, default: 1).

Tip: The operators can be chained (e.g. `HEAD~^2~^2`).

Snapshotting

- `git add <file1> <file2> ...` *or* `git add .`
  adds (stages) one or more files *or* the current directory to the next commit.
  Tip: You can use *globs* (i.a. wildcards) to add a set of related files.
- `git reset [<file1> <file2> ...]`
  unstages specific files if provided, otherwise unstages all modified files.
- `git commit -m "<msg>"`
  commits staged changes, i.e. creates a new "snapshot". Write something useful!
  Tip: `git commit -am "<msg>"` is the same as `git add .`, `git commit -m "<msg>"`
- `git status`
  shows status of files modified since last commit.
- `git diff [<commit>] [--staged] [<file1> <file2> ...]`
  shows unstaged / staged changes (if provided, between file(s) and / or commit).

Undo

- `git revert <commit>` *Useful if you pushed a faulty change.*
  reverts the repository to a certain commit, creating a new one.
- `git commit --amend -m "<msg>"` *Useful if you typo'd a msesage.*
  changes the message of the previous commit.
- `git checkout -- <file1> <file2>...` *Useful if you want to undo local changes.*
  undoes local changes to the specified files since the last commit (unrecoverable!).

Inspection and Comparison

`git log` shows the commit history (incl. commit hashes) of the current branch.

- `-<limit>` limits the number of commits (newest → oldest).
- `--oneline` condenses each commit and commit hash.
- `--author="<pattern>"` only shows commits made by author matching `<pattern>`.
- `--grep="<pattern>"` only shows commits with messages containing `<pattern>`.
- `-- <file1> <file2>...` only shows commits made on the specified files.

Tip: `git log --oneline --graph --decorate` shows a nice visual representation.

`git diff` can be used to view differences between files, commits or branches.

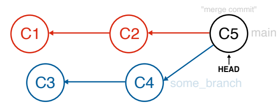
- `<commit1>..<commit2>` shows the difference between two commits.
- `<branch1>..<branch2>` shows the difference between two branches.

Tip: + represents "file a", – represents "file b". They're almost always the same file.

`git reflog` shows a list of all operations performed on the local repository.
In other words, it's a list of commits that HEAD has pointed to (i.e. an *undo history*).

Branching and Merging

- `git branch -a` shows all existing branches (current: `*`).
  `git branch <branchname>` creates a new branch (but doesn't switch to it).
  `git branch -d <branchname>` deletes a branch.
  `git branch -m <branchname>` renames the *current* branch.
- `git checkout <branchname>` switches to a branch.
  `git checkout -b <branchname>` creates a new branch *and* switches to it.
  `git checkout <commit>` switches to a commit (→ detached HEAD!).
  `git checkout -` returns to where you were previously.



Merging merges the contents of two branches into a single branch (usually `main`).

- `git merge <branchname>` merges the current branch with `<branchname>`.

Merge Conflicts

Merge Conflicts occur when git cannot automatically merge two branches.

```
<<<<<<< HEAD remove this line...
Everything above the middle conflict divider represents the conflicting
contents inside the file in the current branch.
======= ...and this line...
Everything below the middle conflict divider represents the conflicting
contents inside the file in the branch to be merged.
>>>>>>> branch_to_merge ...and this line!
```

To resolve a merge conflict, remove all conflict dividers (`<<<...HEAD`, `===...`, `>>>...branch_to_merge`) and only keep the changes you wish to be made.

Temporary Stashing

The **Stash** is a place to hide modifications while working on something else.

- `git stash` temporarily stashes uncommited changes away.
- `git stash pop` retrieves (reapplies) the last changes stashed away.
- `git stash list` lists everything inside of the stash.
- `git stash clear` clears the stash.

Remote Repositories

- `git clone <url>`
  clones a repository from an URL (can be HTTPS or SSH).
- `git pull`
  fetches changes from the remote branch and merges them.
- `git push`
  pushes (uploads) local changes to the remote branch.

Closing Remarks

- Use the official git reference for all available info: `git-scm.com/docs`.
- If you're interested in mastering git, check out the book: `git-scm.com/book/en/v2`
- When in doubt, follow the hints and error messages provided by git.
- Your favorite search engine and StackOverflow are your best friends.

Note: I've purposefully avoided certain topics, such as hard resetting, rebasing or patching. This is to avoid needless complication and frustration, especially for those unfamiliar with these topics. If you still wish to use these tools, you're on your own - I take no responsibility for any messed up repositories.

And remember, GitHub and Git are not the same thing.