```
Steffen's Git Cheat Sheet - Version 1.1 (2023-05-12)
(printing recommendation: print with line numbers, 89 lines per page)

git help           | git --help           : global usage and help
git help <command> | git <command> --help : get help to specific command
man git <command>                         : RTFM

git config --global core.pager 'less -x4' : set git diff to 4 spaces for tab
git config --global merge.tool meld       : setup meld as mergetool
git config core.fileMode false            : have git ignoring file mode

git clone <repo>             : clone repository into a new directory (of same name)
git clone <repo> <dir>       : clone repository into a new directory named <dir>
git clone --recursive <repo> : clone repository and all its sub-modules
git clone -b <branch> <repo> : clone repository and checkout <branch>

git status      : get current status
git status -uno : don't show untracked files
 -> make "gitst" an alias for "git status -uno"

git ls-files    : show files under git in current directory
git ls-files -m : show modified files

git log                  : showing the log (commit messages)
git log <file>           : showing the log of particular file
git log --graph          : showing nice graphical view
git log --all            : including all branches into this view
git log --follow         : listing history beyond renames (only for single file)
git log --oneline        : one line per commit only
git log --author="Jon"   : showing by author
git log --grep something : searching for "something" in the logs
git log -n x             : show last x commits only
 -> make "gitlog" an alias for "git log --oneline --graph"

git show <commit>                 : show detailed modifications of that commit
git show <commit> --word-diff=color : highlight word change in-line
git show <commit> --name-status     : affected files only
git show --name-only                : show only names of changed files

git add <file>          : add file to index (staging area) to be committed with next commit
git add <file1> <file2> : add multiple files to index
git add .               : add all files in current directory to index
git add <pattern>       : add all files matching <pattern> to index, e.g. "*.tex"
git add -u              : add all updated (incl. deleted) files to index
git add -f              : force adding (override ignore)
git add -i              : interactive adding

git diff                   : show modifications to all files
git diff <file>            : show modifications to file
git diff <commit>          : show modifications in commit
git diff <c1> <ct2>        : show diff between 2 commits
git diff --cached          : show diff for staged files
git diff --name-only       : show affected file names
git diff --name-status     : show affected file names and type of modification
git diff -w                : --ignore-all-spaces (in line)
git diff -b                : --ignore-space-change (soft)
git diff --word-diff=color : show word diffs in-line with colour highlighting

git difftool : show changes using a diff tool (configured via diff.tool)

git reset HEAD <path>     : unstage path (file/directory) from index ("git un-add")
git reset <commit>        : reset to that commit (modifications are unstaged)
git reset --soft <commit> : "undo" commit: changes get staged, keep modified files
git reset --hard <commit> : reset tree and files to that commit (lose current changes)
git reset --keep <commit> : abort reset if reset would change files with current changes

git commit              : commit index (popping up editor to enter commit message)
git commit -a           : auto-stage all changed and deleted files and commit
git commit -m "message" : commit and use message
git commit --amend      : re-use last commit (append...)

git tag [--list]       : list all local tags
git tag <tag_name>     : create new simple tag
git tag -a <tag_name>  : create annotated tag (with message)
git tag -d             : delete local tag

git ls-remote          : list remote references (branches, tags)
git ls-remote --heads  : list available remote branches
git ls-remote --tags   : list available remote tags (in an existing git repo)

git branch             : list local branches
git branch -r          : list available remote branches
git branch -a          : list all branches (may contain no longer existing remote branches)
git branch -d          : delete local branch
git branch -D          : force deletion of local (unmerged) branch
git branch -m <branch> : rename current branch to <branch>


```

```
git checkout <t/b/c>          : checkout tag/branch/commit
git checkout <t/b/c> -f       : force checkout of tag/branch/commit (discard local changes)
git checkout <t/b/c> <path>   : checkout path (file/directory) from tag/branch/commit
git checkout -b <b> [<t/b/c>] : start new branch <b> from HEAD [or tag/branch/commit]
git checkout -B <b> [<t/b/c>] : force start new branch <b> from HEAD [or tag/branch/commit]
git checkout -- <path>        : checkout path (file/directory) as it was in index
git checkout -- .             : checkout the current directory as it was in index

Example 1): you're on branch 'dev', but want to start a quick 'fix' from master again:
  git checkout -b fix master (without the need to checkout master ...)

  before: A-B-C <- HEAD = dev           after : A-B-C
            \ <- master                         \ <- fix = HEAD (= master)

Example 2): working from a dedicated commit on (in detached HEAD)
  git checkout B (from source branch <name>); then do some work: commit D, E

  before: A-B-C <- HEAD                 after : A-B-C <- <name>
                                                      \D-E <- HEAD
  Problem: E needs reference (other than the commit label): Detached HEAD -> create it via
   one of these: git branch name, git tag name, git checkout -b name

Alternative to git checkout (branches):
git switch <b>    : checkout (switch) to branch <b>
git switch -c <b> : start new branch <b> from HEAD [or tag/branch/commit]
git switch -C <b> : force start new branch <b> from HEAD [or tag/branch/commit]
-> git switch is experimental, behavior may change [as of git 2.40.1]

Alternative to git checkout (files):
git restore <path>            : restore path (file/directory) to what it is in index
git restore --staged <path>   : unstage path (file/directory) from index ("git un-add")
git restore -s <t/b/c> <path> : restore path to what it is in tag/branch/commit
-> git restore is experimental, behavior may change [as of git 2.40.1]

git stash                     : save changes without committing
git stash list                : show all the stored stashed states
git stash apply               : apply the last stashed state without removing it from the list
git stash pop                 : apply the last stashed state and remove it from the list
git stash drop                : remove the last stashed state
git stash drop stash@{<rev>} : remove dedicated stash state
git stash clean               : remove all stashed states

git rm <file>              : remove file from the working tree and the index
git rm -r <directory>      : recursive (give directory)
git rm -f <file/directory> : force removal
git rm --cached <file>     : remove file from git index but keep on file system

git mv <file> <path>       : Move or rename a file, a directory, or a symlink
git mv -f <file> <path>    : force move

git remote -v                       : show remote url
git remote [show]                   : show name(s) of remote(s) [usually 'origin']
git remote show <name>              : show tracking information (and branches) for remote <name>
git remote prune [--dry-run] <name> : Delete all stale remote-tracking branches
git remote set-url <name> <url>     : set url of remote <name> to <url>
git remote add <name> <url>         : add new remote <name> with <url>

git push                       : push current branch to remote
git push -f                    : force push to remote
git push origin <tag>          : push tag to remote
git push origin <branch>       : push branch to remote (*)
git push -u origin <branch>    : push branch to remote and set tracking
git push origin --delete <b/t> : delete branch/tag on remote
(*) to set up tracking afterwards:
git branch --set-upstream <branch> <name>/<branch>

git fetch                          : fetch (new/updated) branches and tags from remote
git fetch -p                       : fetch and remove remote references that don't exist anymore
git fetch <name> <branch>:<branch> : fetch <branch> from remote <name> without checkout
git fetch origin master:master     : fetch remote master (origin) and update local master to it

git merge <branch> : merge current branch with <branch>
 (the new created commit is the combination of all commits in <branch>)
git merge --abort  : abort current merge

git mergetool : launch merge tool to solve conflicts (configured via merge.tool)

git pull                           : fetch and merge current branch from remote
git pull --rebase                  : fetch and rebase current branch from remote
git pull -f                        : forced pull (overrides local divergences)
git pull --all                     : fetch all remotes
git pull <name> <branch>:<branch> : pull <branch> from remote <name> without checkout

git revert <commit>           : revert an existing <commit>
git revert -n HEAD~5..HEAD~2  : revert fifth last (included) to third last commit
                                without creating a new commit yet (option -n)
```

```
git rebase -i HEAD~<n>  : interactive rebasing of most recent n commits
git rebase <ref> [<b>]  : rebase current branch [or <b>] to reference <ref>
 Example 1): commits in branch <b> and <ref> are entirely distinct

  before: A-B-C-D <- <ref>              after: A-B-C-D <- <ref>
             \E-F <- HEAD = <b>                       \E'-F' <- HEAD = <b>

 Example 2): a commit in branch <b> is already taken in <ref> (e.g. by patching)

  before: A-B-E'-C-D <- <ref>           after: A-B-E'-C-D <- <ref>
             \E-F <- HEAD = <b>                         \F' <- HEAD = <b>

 Rule: 'git rebase' (without --onto) re-applies entire history, starting from a new base
  reference, possibly cloning several commits (copy & paste)

git rebase --onto <r1> <r2> <r3> : rebase everything after <r2> until <r3> onto <r1>
 Example 1): moving start of <r3> at the end of <r1> and not <r2>

  before:                               after:
          A-B-C-D <- <r1>                    A-B-C-D <- <r1>
             \E-F-G <- <r2>                    |    \H'-I' <- HEAD = <r3>
                   \H-I <- HEAD = <r3>          \E-F-G <- <r2>

 Example 2): git rebase --onto <r~5> <r~2> <r> = getting rid of some commits

  before: A-B-C-D-E-F-G-H-I <- <r>      after: A-B-C-D-G'-H'-I' <- <r>
                ^     ^                           ^     ^
              <r~5> <r~2>                       <r~5> <r~2>

 Rule: 'git rebase --onto' transplants sections of the history to a new starting reference
  possibly moving several commits (cut & paste)

git rebase --abort      : abort rebase and reset HEAD to the original branch
git rebase --continue   : restart rebasing process after having resolved a merge conflict

git cherry-pick <commit> ..: apply (single) commit to current branch
git cherry-pick ..<branch> : apply all commits that are ancestors of master but not HEAD

git reflog        : show list of local references
git reflog --all  : show entire list of references (multiple refs point to the same commit)

git describe [<commit-ish>] : finds the most recent tag that is reachable from a commit

git submodule add <repository> [<path>] : Add a sub-module at <path>
git submodule init             : initialize submodules recorded in index
git submodule deinit <path>    : unregister given submodules
git submodule update [<path>]  : update registered submodules to match what
 superproject expects by cloning missing submodules and updating the working
 tree of the submodules (e.g. of git checkout ... of superproject)
git submodule foreach <command> : run arbitrary shell command in each submodule

Setting aliases:
a) via git (config) directly, e.g.:
git config --global alias.df "diff --word-diff=color"
-> sets "git df" as alias for "git diff --word-diff=color"
-> known to git only
b) via editing ~/.bashrc, adding the alias, e.g.:
"alias gitlog='git log --oneline --graph'"
-> sets "gitlog" as alias for "git log --oneline --graph"
-> known to the terminal as separate command
```

```
Various examples:

1) Reset all staged files
git reset HEAD *

2) Want to merge with (remote) master, but working on feature.
  1: git pull origin master:master
  2: git merge master
 NB: without 1, the master will not contain the remote changes

3) Add forgotten files to most recent commit (if not yet pushed):
git add <forgotten file>
git commit --amend -C HEAD : re-uses the same commit message right away

4) Modify commit messages of past 3 commits:
git rebase -i HEAD~3
 And then follow the instructions

5) Synchronise local branch to modified (e.g. rebased) branch on remote,
   in particular "throwing away" local history
git fetch --all
git reset --hard origin/<branch>

6) Combine commits of an on old branch after branching off of master
   at [commit] and rebase it to master:
git checkout -b tmp (on top of old branch)
git reset --soft [commit]
git commit -m "message"
git rebase master

7) Interrupted work-flow

git checkout feature             # you were working in "feature" branch and
work add commit work             # got interrupted
git stash                        # backup your current work in progress
git checkout master              # checkout master branch
git pull                         # just make sure your master is up to date
git checkout -b HotFix           # create the hotfix branch
fix add commit fix               # do the work needed
git commit                       # commit with real log
git push origin HotFix           # push to remote
 open merge request to master    # to have it taken into account
git checkout feature             # come back to your feature
git stash pop                    # recover your stashed changes
work add commit work             # continue your work

8) Rebase local branch to (updated remote) master and push back
   8.1) The save way: via direct checkout of master

git checkout master              # save way: check out the (local) master
git pull                         # pull in recent changes from remote
git checkout <branch>            # switch back to (local) working branch
git rebase master                # rebase working branch to (updated) master
git push -f                      # force push working branch to remote

   8.2) The quick way: via indirect checkout of master
git fetch origin master:master   # update local master to remote master
git rebase master                # rebase working branch to (updated) master
git push -f                      # force push working branch to remote

9) Update local branch to (updated remote) branch that was rebased
git pull -r                      # Short for --rebase:
                                 # In case you have new local commits, they
                                 # will be rebased on top of the new remote branch
```