A Contextual Git Cheatsheet

Places in git
working copy your local copy of the code
index a staging area holding a snapshot of the code that will form the next commit
stash a place to hold code that isn't ready to be committed, while you work on other things
local repository a set of commits, branches, tags etc in a .git folder on your computer
remote repository a repository other than the one you are running commands in (often on another computer)

Terms
remote a named non-local repository, stored with a path, or a ssh, git, http, https, ftp, sftp or rsync URL
commit a stored snapshot of a repository, referred to by a long hash
parent the commit that another set of changes adds to
branch an independent thread of development
tag a human-readable name for a specific commit
merge a commit that joins two threads of commits

Keywords
master the default branch, which will be assumed if you don't specify one
origin the default upstream repository, which will be assumed if you don't specify one
HEAD the parent to your working copy, can be the name of a branch or a specific commit.
HEAD^ the parent commit to HEAD

Notes
All commands should be typed with 'git' before them
[] = [optional] and <> = <replace with the name of a branch / remote / commit / stash>

Creating repositories
init creates an empty repository where you are
clone <remote> makes a local copy of a remote repository

Prepare the index before storing changes
add <files and directories> adds the current content of the files to the index
add -u adds all modified (not new) files to the index
rm <files> removes a file from the working copy and the index
mv <source> <destination> moves a file in the working copy and the index
reset HEAD <file list> undoes changes to the files in the index (not the working copy)
reset [<commit or branch>] Makes HEAD point to a different commit, and resets the index to that, but doesn't change the working copy

Prepare HEAD before storing changes
checkout [-b] <branch> switches to the branch (changing the index but not the working copy), changing the parent of your next commit; use -b to create the branch at the same time
reset --soft HEAD^ sets the last commit but one as the parent of your next commit (forgets changes in the last commit without affecting the index or the working copy)
reset --soft [<commit or branch>] makes HEAD point to a different commit or branch without changing the index or working copy

Storing changes
stash [save <message>] saves working copy modifications to a new stash and removes them from the working copy
commit -a -m <message> creates a new commit of changes to all tracked files
commit -m <message> creates a new commit from the index
commit --amend alters the last commit to the current state of the index

Reapply some changes
stash apply [<stash>] applies changes from the most recent / named stash to working copy
stash pop applies the most recent stash to the working copy, then deletes the stash
stash branch <branch> [<stash>] creates a new branch from where the stash was created, applies the stash and then deletes the stash
cherry-pick <commit> creates a new commit with changes from another commit, without having to merge in any of its parents

Undo changes to files
clean recursively deletes all files that aren't being tracked by git
checkout <files or directories> changes the files / directories to be as they are in the index
reset --hard [<commit or branch>] resets the index and the working copy back to the state at HEAD or at a specified commit / branch
revert <commit> creates a new commit that reverses the changes of an existing commit

Moving commits
remote add <remote> <url> adds a new named remote
pull [<remote> <branch or commit>] fetches commits from the remote repository and merges them into the local repository
fetch [<remote> <branch or commit>] copies commits from the remote repository into the local repository
push [<remote> <branch>: <branch>] copies commits from the local to the remote repository; if 2 branches are given, the first is the local branch and the second is the remote branch

Branches / tags
branch <branch> creates a new branch
branch --track <branch> <remote>/<branch on remote> creates a new branch that tracks a remote branch
tag <tag> creates a new tag

Manipulate commits
merge <commit or branch> merges other changes into the current branch
git diff (--base or --ours or --theirs) <file> see file differences between the last common ancestor, local version and remote version during merging
mergetool opens your chosen GUI for merging
rebase [<remote> <branch>] changes the parent of your existing commits; if the remote branch has been added to since you made commits, rebase rewrites your commits so their parent is the tip of the remote branch, rather than coming off part-way along the branch and being merged into the tip. Each of your commits in turn is merged onto the branch tip, and any conflicts are resolved
rebase --continue after doing adds and rms that resolve conflicting file changes, this continues with the rebasing
rebase --abort undoes rebasing

Delete some stored changes
stash drop [<name>] deletes a stash
stash clear deletes all stashes
branch -d <branch> deletes a branch
push <remote> :<branch> deletes a branch from a remote

Status
status shows which files have differences between the index and HEAD and between the working copy and the index
diff shows differences between the working copy and the index
diff <commit or branch> shows differences between the working copy and a commit or branch
diff <commit> <commit> shows differences between any two commits
diff --cached [<commit>] shows differences between the index and HEAD or the given commit
stash list lists all stashes
stash show [<stash>] shows differences between a stash and its parent commit
log shows recent commits
blame <file> shows which commit and author last changed each line of a file
branch lists existing branches
remote -v shows details of all remotes
remote show <remote> shows all details about a remote

Common problems
Detatched head: HEAD isn't pointing to a branch but to an individual commit. New commits won't belong to any branch, so won't be pushed / pulled to other repos. If you haven't made any commits, then do a checkout <branch>. If you have, then create a branch for your local commit. Checkout another branch you want your commit to belong to, and merge in the branch you just created.
Undo a merge / pull: if you're happy to lose your local changes, do a git reset --hard to the commit you were working on.
Undo an add: do reset HEAD <file> to the file you didn't want to add.
Amend a commit: make the changes you want (with rm and add), then do a commit --amend.
Undo commit: do reset --soft "HEAD^" to go to the previous commit.
Changes to the working copy are stopping a git operation: stash them, do the operation, then do stash pop.

Made by Hywel Carver at www.londonstartuptech.com : software consulting for startups