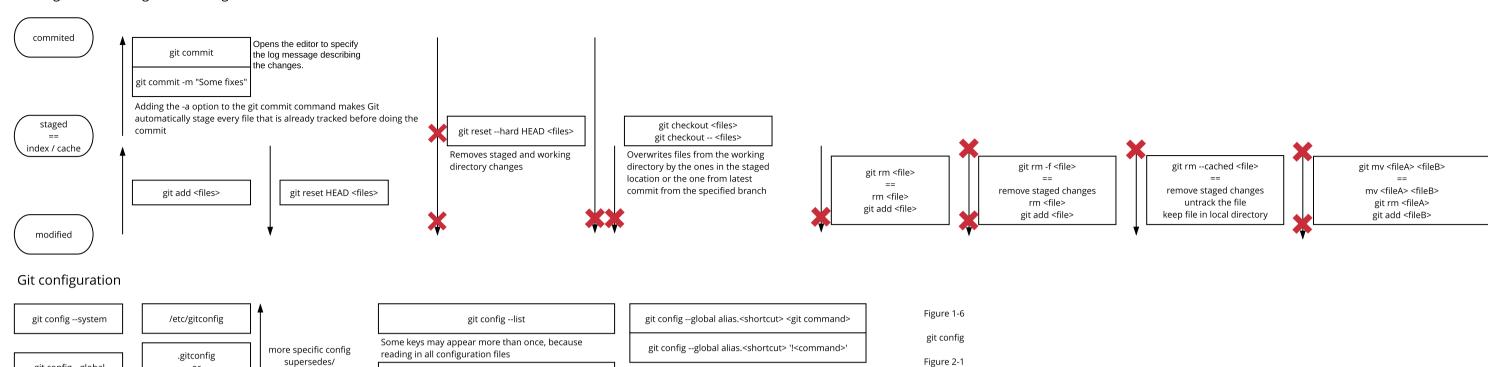
Adding and removing files for staged and commit location



Cloning and pushing

git config --global

git config

git clone https://github.com/libgit2/libgit2

~/.config/git/config

./.git/config

Downloads branch, tracks it and implicitely creates a remote reponame called "origin" (the default name Git gives to the server you cloned from) and creates a local branch named "master".

git clone https://github.com/libgit2/libgit2 someNewName

git remote -v

git config --global user.name "John Doe"

git config --global user.email johndoe@example.com

overrides

higher levels

Lists the shortnames of each remote handle you've specified. -v adds the remote URL next to it.

git remote show <remote repo name>

Show the branches the rremote repository has, which one we track and which ones are concerned (tracked) when doing a pull or a push.

git fetch <local branch>

Downloads all the data we do not have yet from remote branches. It doesn't automatically merge it with any of your work or modify what you're currently working on.

git pull

If the current branch is tracking a remote branch, automatically fetches and merges that remote branch into the current branch.

Tags are pulled too.

git push <remote repo name> <local branch>

e.g.: git push origin master
Try to push the local branch to the remote location, if

we have write access and if someone has not pushed data in the meanwhile, otherwise, the push is rejected and we will have to fetch and merge first to continue.

git push <remote repo name> <tag number>

git push <remote repo name> --tags

Tags are not pushed. Pushing the tag/all tags manually.

git remote add <local repo name> <remote repo url>

git remote rename <local repo name> <new local repo name>

git remote rm <local repo name>

Tagging

git tag <tag string identifier>

Creates a lightweight tag, just a pointer to the last commit $\ensuremath{\mathsf{ID}}$ of the current branch.

git tag -a <tag string identifier> [<commit id>]

git tag -a <tag string identifier> -m "Tagging log message" [<commit id>]

Creates a tag (a commit) with a pointer to the last commit ID of the current branch and a message log. Has a dedicated SHA-1 ID.

git tag

List all tags of the current branch.

git tag -l <search pattern>

Search for a tag matching the search pattern (wildcard ~regex supported).

git checkout <branch name>

Moves the HEAD pointer to the specified branch name and changes working directory files to the last snapchot/commit of the specified branch.

Changing branch is rejected if not all changes to files in working directory have been committed (staged is still uncommitted). That rejection is only true if both branches do not point to the same location (i.e. no commit to one branch or the other).

git checkout -b
branch name>

Creates a new branch of the specified name and switch to it.

git merge <branch name>

Merges the specified branch to the current branch.

If the specified branch is ahead of the current one, the merge will be "Fast-foward": the pointer is moved forward.

If there is a conflict, the merge is pausedd. Theuser needs to fix it manually by removing lins and commit the changes manually.

<<<<< HEAD:<filename>

<content from branch we are merging into>

<content of branch we are merging from>
>>>>> <hranch>'<filename>

git mergetool

Use an external tool like vimdiff to help resolve merge conflicts. User answers Y/N if conflict have been solved and then commit changes manually.

git branch <new branch name>

Creates a new branch of the specified name.

git branch git branch -v git branch --merged git branch --no-merged

List all branches

Branching

List all branches with last commit (short hash + log message) List al branches that have been merged (branches safe to remove) List al branches that have not been merged

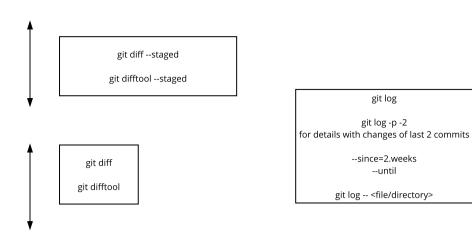
The * prefix indicates the branch that HEAD points to.

git branch -d <branch name>

Removes specified branch if it has been merged.

git branch -D <branch name>

Removes specified branch and remove its content even if it has not been merged.



\$ git status -s M README MM Rakefile A lib/git.rb M lib/simplegit.rb ?? LICENSE.txt

New files that aren't tracked have a ?? next to them, new files that have been added to the staging area have an A, modified files have an M and so on. There are two columns to the output - the left-hand column indicates the status of the staging area and the right-hand column indicates the status of the working tree. So for example in that output, the README file is modified in the working directory but not yet staged, while the lib/simplegit.rb file is modified and staged. The Rakefile was modified, staged and then modified again, so there are changes to it that are both staged and unstaged.