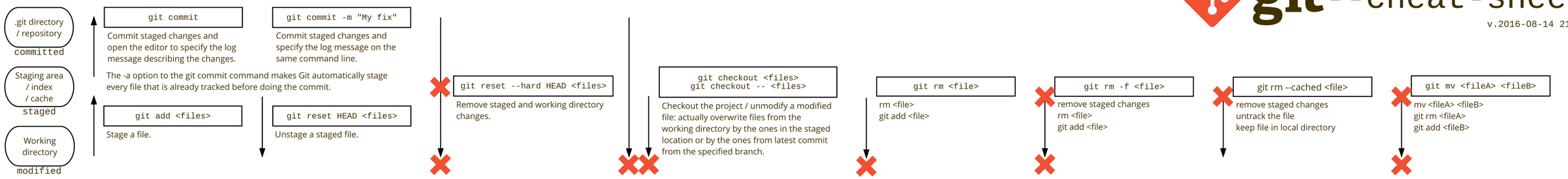
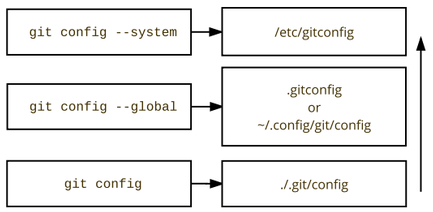
git--cheat-sheet

v.2016-08-14 21:48

Adding and removing files for staged and commit locations



Git configuration



`git config --list`

Some keys may appear more than once, because reading in all configuration files.

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
git config --global alias.<shortcut> <git command>
git config --global alias.<shortcut> '!<command>'
```

Tagging

`git tag <tag string identifier>`

Create a lightweight tag just a pointer to the last commit 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>]`

Create 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.

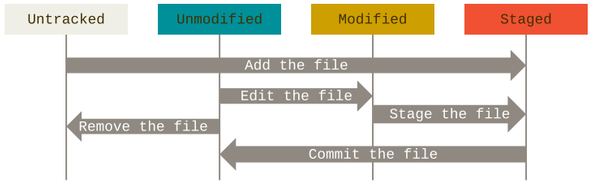
`git tag`

List all tags of the current branch.

`git tag -l <search pattern>`

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

Vocabulary



Cloning and pushing

`git clone <git repo url>`

`git clone <git repo url> <directory to clone to>`

Implicitly create a default remote repository shortname called "`origin`", download the default branch content and name that branch "master".

`git pull`

If the current branch is tracking a remote branch, automatically fetch and merge that remote branch into the current branch. Tags are pulled too.

`git fetch <remote repo shortname>`

Download all the changes made on the remote shortname location, i.e. all objects (and thus commit history) are downloaded, all remote pointers will be updated (`refs/remotes/*`). It doesn't automatically merge the changes with files from working directory nor does it modify these files.

`git push`

Push the default branch master (the default one from remote.*.push) to the default remote repo called origin.

`git push <remote repo name> <local branch>`

As local branchs are not pushed by default, push the local branch to the remote location specified.
If we do not have write access or if someone has pushed data in the meanwhile, 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 by default. Push the tag/all tags manually.

`git remote -v`

List the shortnames of each remote handle you've specified. -v adds the remote URL (pull and push) next to it.

`git remote show <remote repo shortname>`

Show the branches the remote repository has, which one we track and which ones are concerned (tracked) when doing a pull or a push. Untracked (aka "new") branches will be downloaded with "`git fetch <remote>`".

`git remote add <shortname> <remote url>`

Add a new remote explicitely (contrary to the implicit "`origin`" made by the "`git clone`" command) and reference to it as the specified shortname.

`git remote rm <shortname>`

Remove the specified remote either because the server has changed, the mirror is not used anymore or no one uses that mirror any more.

`git remote rename <shortname> <new shortname>`

Change the remote shortname to the new one you specified.
Remote tracked branches are renamed in the process.

Branching

`git branch <new branch name>`

Create a new branch of the specified name.

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

List all branches
List all branches with last commit (short hash + log message)
List all branches that have been merged (branches safe to remove)
List all branches that have not been merged
The * prefix indicates the branch that HEAD points to.

`git branch -d <branch name>`

Remove specified branch if it has been merged.

`git branch -D <branch name>`

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

`git checkout <branch name>`
`git checkout -`

Move the HEAD pointer to the specified branch name/last 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>`

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

`git checkout -b <branch name> <branch name on remote>`

Create a new branch of the specified name, download the branch content locally and switch to it. This command is useful as `fetch` only gets pointers to new remote-tracking branches but does not take a local copy of them.

`git merge <branch name>`

Merge the specified branch to the current branch.
If the specified branch is ahead of the current one, the merge will be "Fast-forward": the pointer is moved forward.
If there is a conflict, the merge is paused. The user needs to fix it manually by removing lines and commit the changes manually.

```
<<<<<<< HEAD:<filename>
<content from branch we are merging into>
=======
<content of branch we are merging from>
>>>>>>> <branch name we are merging from>:<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.

Differences and logging changes

`git log`

List in reverse chronological order (paging the result), the ID, author, date and message of commits made in the repository.

`git log -p -2`

Same but showing the difference introduced in each commit with a limit to only the last two entries.

`git log --since=2.weeks --until=2016-08-10`

To specify limits in the commits research. Lots of time formats and research patterns available.

`git log -- <file/directory>`

Display the commits having modified the specified files or directories.

`git log -S"somecode to search for"`

Search for commits where changes to the specified code string were introduced.

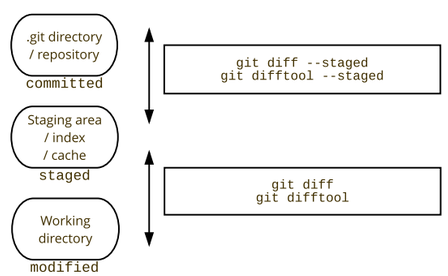
`git show <commit id>:file`

Show the file content like it was at the commit ID. Redirect the output to a temporary file with ">" if you want to see a finary file.

```
$ 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".
States are displayed according two columns.
The left-hand column indicates the status of the staging area
The right-hand column indicates the status of the working tree.

In this 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.



Caption:

✖ The red cross means typing the command may involve data loss.

Schemas and texts inspired from the book progit2 written by Scott Chacon and Ben Straub and made available under the license Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Made by William Gathoye (@wget42) and licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
http://creativecommons.org/licenses/by-nc-sa/4.0/