Git cheat sheet

Git installation

For GNU/Linux distributions, Git should be available in the standard system repository. For example, in Debian/ Ubuntu please type

```
$ sudo apt-get install git
```

Git config

```
$ git config --global user.name "My Name"
$ git config --global user.email "me@cusy.io"
```

Set name and email address that will be attached to your commits and tags.

```
$ git config --global color.ui auto
```

Set colorisation of Git output

.gitignore

Some files usually shouldn't be tracked by git. They are written to a special file named `.gitignore`. You can find helpful templates at `github.com/veit/dotfiles/`.

Start a project

```
$ git init [my_project]
```

Create a new local repository.

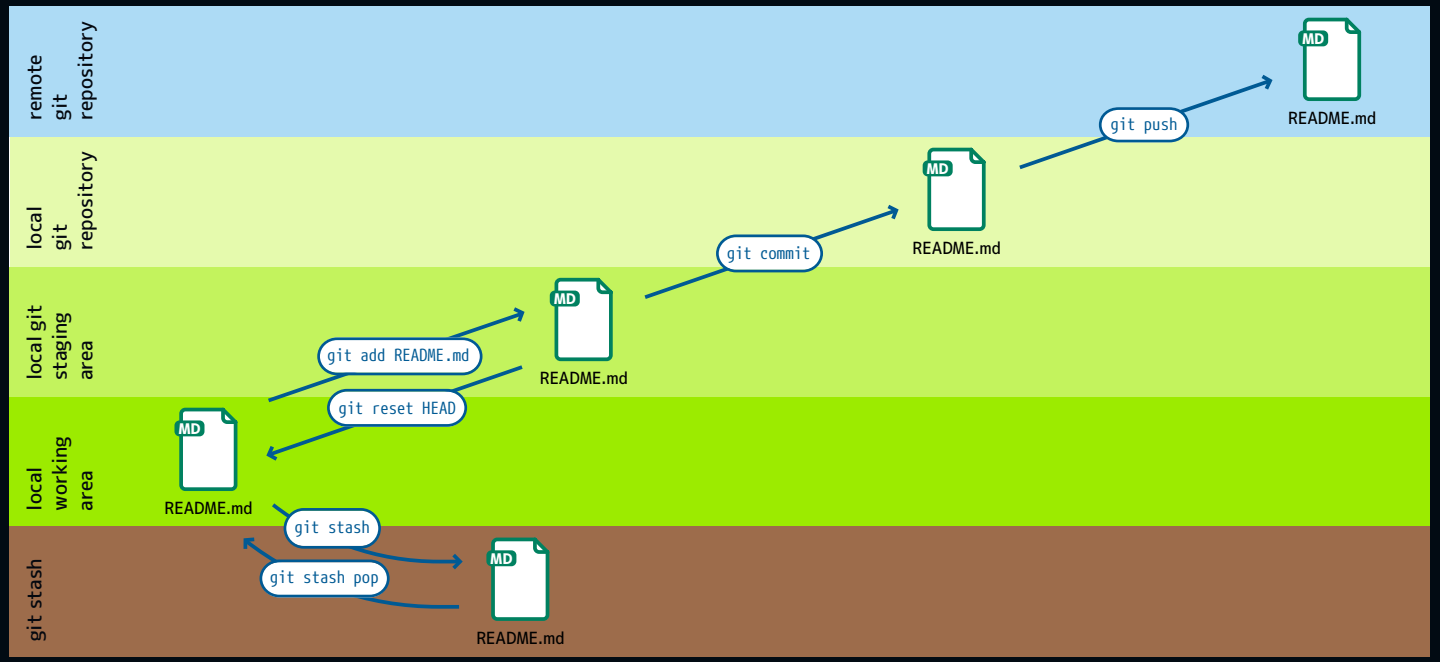
If `[my_project]` is provided, Git will create a new directory and initialise it as a repository.

If `[my_project]` is not provided, the new repository is initialised in the current directory.

```
$ git clone [project_url]
```

Downloads a project with all branches and the entire history from the remote repository.

Work on a project




```
$ git status
```

Display the status of the working directory with new, staged and modified files for the current branch.

```
$ git add [file]
```

Add a file to the staging area.

```
$ git add -p [file]
```

Add only parts of a file to the staging area.

```
$ git diff [file]
```

Show changes between working and staging area.

```
$ git diff --staged [file]
```

Show changes between staging area and repository.

```
$ git checkout -- [file]
```

Irrevocably discard changes in the working directory.

```
$ git commit -m 'Commit message'
```

Create a new commit from added changes.

```
$ git reset [file]
```

Revert the file to the last commited version.

```
$ git rm [file]
```

Remove the file from working directory and staging area.

```
$ git stash
```

Put current changes from your working directory into stash for later use.

```
$ git stash list
```

List the modifications stashed away with `git stash`.

```
$ git stash show [<stash>]
```

List the modifications stashed away with `git stash`.

`$ git stash pop [<stash>]`

Inspect the modifications stashed away.

`$ git stash drop [<stash>]`

Delete a specific stash from all your previous stashes.

Git branching

`$ git branch [-a]`

List all local branches in the repository.

`[-a]` shows also the remote branches.

`$ git branch [branch_name]`

Creates a new branch, referencing to the current `HEAD`.

`$ git checkout [-b] [branch_name]`

Switch the working directory to the specified branch.

`-b` will create the specified branch if it does not exist.

`$ git merge [from name]`

Join specified `[from name]` branch into your current branch (the one you are on currently).

`$ git branch -d [name]`

Remove selected branch, if it is already merged into any other.

`-D` instead of `-d` forces deletion.

Review

`$ git log [-n count]`

List the commit history of the current branch.

`-n` limits the list to the last n commits.

`$ git log --oneline --graph --decorate`

An overview with reference labels and history graph – one commit per line.

`$ git log ref..`

List commits that are present on the current branch and not merged into `ref`. `ref` can be a branch name or a tag. name.

`$ git log ..ref`

List commits that are present on `ref` and not merged into the current branch.

`$ git reflog`

List operations (e. g. checkouts or commits) made on the local repository.

Tagging

`$ git tag`

List all tags.

`$ git tag [name] [commit sha]`

Create a tag reference named `name` for current commit.

With `sha` the specific commit is tagged instead of the current one.

`$ git tag -a [name] [commit sha]`

Create a tag named `name` for current commit.

Reverting

`$ git reset [--hard] [target reference]`

Switches the current branch to the target reference, leaving a difference as an uncommitted change.

When `--hard` is used, all changes are discarded.

`$ git revert [commit sha]`

Create a new commit, reverting the changes from the specified commit. It generates an inversion of changes.

`$ git fetch [remote]`

Fetch changes from the remote, but not update tracking branches.

`$ git fetch --prune [remote]`

Delete remote Refs that were removed from the remote repository.

`$ git push --prune [remote]`

Remove remote branches that don’t have a local counterpart.

Synchronising repositories

`$ git pull [remote]`

Fetch changes from the remote and merge the current branch with its upstream.

`$ git push [--tags] [remote]`

Push local changes to the remote. Use `--tags` to push tags.

`$ git push -u [remote] [branch]`

Push the local branch to a remote repository. Set its copy as an upstream.

https://cusy.io/de/seminare

>cusy_