Git cheat sheet

Initialization

```
$ git init <directory>
```
Creates new repo in specified directory

```
$ git clone <url>
```
Copies repo from specified url

```
$ git config user.name <user_name>
```
Sets username for commits in current repo use --global to apply it globally

```
$ git config user.email <user_email>
```
Sets email for commits in current repo use --global to apply it globally

```
$ git config color.ui auto
```
Enables helpfull colorization of command line output

```
$ git config --global --edit
```
Opens the global configuration file in text editor for manual editing

```
$ git remote add <remote> <link>
```
Connect your local repo to the remote one. Usually the default value for <remote> is origin

```
$ vi .gitignore
```
Opens .gitignore file. This file is used for a list of files that have to be excluded. Ensure that this file is in the root of the local repo. You can change **vi** to your favorite text editor

Commits

```
$ git add <path>
```
Adds path into staging. Path can be file or directory

```
$ git restore --staged <path>
```
Removes path from staging back to unstaged area

```
$ git rm -r <path>
```
Removes path and adds that change into staging

```
$ git commit -m <message>
```
Commits the stage with specified message

```
$ git commit --amend -m <message>
```
Repairs last commit with specified new message Change **-m <message>** to **--no-edit** to repair without editing commit message

```
$ git status
```
Lists which files are staged, unstaged, or untracked

```
$ git push <remote> <branch>
```
Uploads **<branch>** branch to same branch in **<remote>**

```
$ git pull -r
```
Updates local branch with all new commits from remote branch with rebasing, avoiding the conflict with changes from remote

Change review

```
$ git log
```
Lists version history for the current branch. add **--pretty=oneline** to show commit hashes and messages only

```
$ git diff <commit1> <commit2>
```
Shows difference between two commits. It is also applied to comparing two branches. Add **--name-only** to show the file names only

```
$ git stash
```
Saves current changes into stash stack. Usually used when current changes don't want to be commited

```
$ git stash pop
```
Applies last changes stored in stash stack onto current working HEAD

```
$ git stash list
```
Shows stash stack

```
$ git revert <commit>
```
Creates new commit that undoes all of the changes in <commit>

```
$ git reset <commit>
```
Undoes the commits after <commit>, keep the changes locally. Add --hard to discard the changes

```
$ git blame -- <file>
```
Shows revision in <file> line by line

Branch & Rebase

```
$ git checkout <branch>
```
Switches to the specified branch

```
$ git checkout -
```
Switched to the previous visited branch

```
$ git checkout -b <name>
```
Creates a new branch with specified name and switch in that branch

```
$ git checkout <path>
```
Restores changes of <path> back into latest revision

```
$ git branch
```
Lists all branches

```
$ git branch -m <old> <new>
```
Renames branch from <old> to <new>

```
$ git branch -d <branch>
$ git push <remote> --delete <branch>
```
Deletes the specified branch in local and remote correspondingly

```
$ git rebase -i <base>
```
Interactively rebases the current branch onto base. It can be branch, comit, or relative reference to HEAD

```
$ git pusah --force-with-lease
```
Uploads all commits to remote branches with force. Usually used when there are conflicts when rebasing. Do not try this unless you know what you are doing

Advanced

```
$ git checkout -R <old_branch>
<new_branch>
$ git push <remote :<old_branch>
<new_branch>
```
Rename branch in local and remote correspondingly

```
$ git tag <tag_name>
$ git push <remote> --tags
```
Create tag and push all created tags

```
$ git tag -d <tag_name>
$ git push <remote> --delete
<tag_name>
```
Delete tag and push deleted tags

```
$ git remote set-url <remote> <url>
```
Changes remote url. Usually used after repository migration

```
$ git cherry-pick <commit>
```
Creates new commit by applying changes in <commit> into current working HEAD

```
$ git gc --prune=now --aggresive
```
Cleanups and optimizes all files in local repository

```
$ git bisect {start,good,bad}
```
Finds the commit that contains bug with binary search. Use start to begin the bisect, good to mark good commits, bad to mark bad commits

RabbitBinary/datasheets-and-schemes @16bitRabbit 16bitrabbit