Git

This is my git cheat-sheet, which grew with time ..

cloning

```
git clone git@github.com:/<GHUSER>/<PROJECT>.git
git clone https://[<USER>:<PWD>@]github.com/<GHUSER>/<PROJECT>

# only specific branch
git clone --branch <BRANCH> git@github.com:/<GHUSER>/<PROJECT>.git

# clone including submodules
git clone --recursive git@github.com:/<GHUSER>/<PROJECT>.git
git clone --recursive https://[<USER>:<PWD>@]github.com/<GHUSER>/<PROJECT>
```

shallow option

saves local space - by just getting last (few) commits

```
git clone --depth 1 ...
git clone --branch <BRANCH> --depth 1 <URL>
```

work with branches

list existing branches

```
git branch -l  # local only
git branch -a  # all - including remotes
```

create branch

create local branch and switch to that one

```
git checkout -b <BRANCH>
```

create private dev branch - might require configuration of git server

```
git push origin HEAD:dev/<USER>/<BRANCH>
```

work on local branch

- edit files locally
- `git commit`
- NO `git push`, if BRANCH should stay in local clone

get updates from remote

```
git pull --rebase   # alias: u

git fetch origin master
```

overwrite/force push

```
git push --force             # alias: p   - definitely overwite history on
remote
git push --force-with-lease  # alias: pf  - overwrite, BUT: stop, if remote
has changed meanwhile
```

delete branch

```
git branch -d <LOCAL-BRANCH>                # alias: brdl / brDl
git push origin --delete <REMOTE-BRANCH>    # alias: brdr
```

rename local branch

```
git branch -m <NEW_NAME>
```

push to different name

```
git push origin <LOCAL-NAME>:<REMOTE-NAME>
git push origin HEAD:<REMOTE-NAME>    # HEAD for the local/active branch
git push -f origin HEAD:dev/<USER>/<BRANCH>
```

see https://penandpants.com/2013/02/07/git-pushing-to-a-remote-branch-with-a-different-name/

push current/local branch to new upstream branch name. option -u is short for –set-upstream. see https://forum.freecodecamp.org/t/push-a-new-local-branch-to-a-remote-git-repository-and-track-it-too/13222

```
git push -u origin <BRANCH>

git push --force-with-lease -u origin <BRANCH>   # when having a modified
commit
```

```
gpf -u origin <BRANCH>         # with alias gpf for git push --force-with-
lease
```

rebase (update/move start point (base) of) a branch

commits, which didn't exist when creating of the feature_branch,
are inserted (put/pulled) in front of first commit of the feature_branch.
then, the patches(commits) of the feature_branche are applied.

usually, the feature branch was created from a master.
master got additions/patches ..
now, we want to 'update' the feature branch - as if started from the updated/new master.

have master updated before

```
git checkout <BRANCH>
git rebase master    # if BRANCH was created from master
```

squash apply commits of branch onto master

```
git checkout master

git merge <BRANCH>           # get each commit from feature_branch
git merge --squash <BRANCH>  # get all commits as one single new commit

git push
```

submodules

more actions with submodules, see https://git-scm.com/book/en/v2/Git-Tools-Submodules

retrieve submodules automatically when cloning

```
git clone --recursive git@github.com:/<GHUSER>/<PROJECT>.git
git clone --recursive https://[<USER>:<PWD>@]github.com/<GHUSER>/<PROJECT>
```

retrieve submodules later

```
git submodule init
git submodule update --recursive

# or in one step
git submodule update --init
```

```
git submodule update --init --recursive
```

add submodule

a submodule can be added any time and everywhere within an existing repository. it retrieves the new repository as with `git clone`.

```
git submodule add https://github.com/<GHUSER>/<PROJECT>

# e.g.
git submodule add https://github.com/DLTcollab/sse2neon
```

stashing

```
git stash save "<NAME>"        # alias: ss
git stash list                 # alias: sl
git stash apply stash@{<N>}    # alias: sa <no>
git stash pop stash@{<N>}
git stash drop stash@{<N>}     # alias: sd <no>
git stash show -p stash@{<N>}  # alias: sshow <no>
```

backup stash contents into a branch

```
git branch <FEATURE_BRANCH> stash@{0}  # creates local branch
git push origin <FEATURE_BRANCH>       # push branch
```

push except last commit(s)

https://stackoverflow.com/questions/8879375/git-push-push-all-commits-except-the-last-one

```
# git push <remote> <commit_hash>:<branch>
git push origin 555555:master          # push including commit 555555
```

cherry picking

http://stackoverflow.com/questions/5304391/how-can-i-cherry-pick-from-1-branch-to-another

```
git checkout master
git log                    # remember/copy commit/SHA ID
git checkout <BRANCH>

git cherry-pick -x <ID>    # for taking including comment
git cherry-pick <ID>       # for new comment
```

tagging

see
https://stackoverflow.com/questions/11514075/what-is-the-difference-between-an-annotated-and-unannotated-tag

annotated tags are pushed, when using git push explicitly with option `–follow-tags`:

```
git push --follow-tags
```

or when having previously configured

```
git config --global push.followTags true
```

Above option allows pushing a commit together with it's tag in one step - without additional `git push origin <TAG>`. Unannotated tags are kept local - and not shared/pushed. The difference gets important, when pushing does automatically trigger an action - conditioned with a tag - or not.

```
git tag -a -m "<MESSAGE>" <TAG-NAME>  # creates tag with annotatation
git tag -a <TAG-NAME>                 # create
git tag -a <TAG-NAME> <SHA-ID>        # create from commit/SHA-ID
                                      # git log --pretty=oneline
git push origin <TAG-NAME>            # share tag with others

git tag -d <TAG-NAME>                 # delete local
git push origin :refs/tags/<TAG-NAME>
git push origin :<TAG-NAME>
git push --delete origin <TAG-NAME>

git tag                               # lists tags
git show <TAG-NAME>                   # show details on tag

git checkout -b <BRANCH> <TAG-NAME>   # checkout tag with new BRANCH!
```

move tag

```
git tag -d <TAG-NAME>
git tag -a <TAG-NAME> -m "<MESSAGE>"
git push -f origin <TAG-NAME>
```

resolving conflicts

recovering from disasterous git rebase mistake
see https://blog.screensteps.com/recovering-from-a-disastrous-git-rebase-mistake

```
git fetch --all
```

```
git reset --hard origin/master
```

work with remotes

show all remotes

```
git remote -v
```

example output

```
origin  https://github.com/hayguen/eti-stuff.git (fetch)
origin  https://github.com/hayguen/eti-stuff.git (push)
```

add a remote

```
git remote add <NEW-REMOTE-NAME> https://github.com/<GHUSER>/<PROJECT>

# e.g. git remote add upstream https://github.com/librtlsdr/librtlsdr

git fetch <REMOTE-NAME>
git branch -a
```

remove/delete a remote

```
git remote rm <REMOTE-NAME>
```

create branch from a remote

```
git checkout -b <NEW_BRANCH> <REMOTE-NAME>/<BRANCH>
```

get log of a remote branch

```
git log <REMOTE-NAME>/<BRANCH>   | grep -B 5 -A 10 search_for
```

pull / rebase onto remote

```
# git pull [--rebase] {repo(remote)} {remotebranchname}:{localbranchname}

git pull --rebase <REMOTE> <BRANCH_OF_REMOTE>[:<LOCAL_BRANCH>]

# e.g. git pull --rebase upstream development
```

change remote's https source to git

```
git remote set-url <REMOTE> git@github.com:/<GHUSER>/<PROJECT>.git

# e.g. git remote set-url origin git@github.com:hayguen/eti-stuff.git
```

push specific commit to remote

```
git push <REMOTE-NAME> <COMMIT-SHA>:<REMOTE-BRANCHNAME>
```

misc operations

set upstream

set default push target to upstream gloabally - needed just once, that local branch name needn't exactly fit remote name

```
git config --global push.default upstream
```

determine top level directory

```
git rev-parse --show-toplevel
```

see
https://stackoverflow.com/questions/957928/is-there-a-way-to-get-the-git-root-directory-in-one-command

hide subdirectory and its contents in git gui

create `.gitignore` with * in that directory

```
echo "*" >.gitignore
```

loose objects warning with git gui

git-gui reports it has N loose objects each time it is run; then try

```
git gc --aggressive
```

or, if the problem remains

```
git config --global gui.gcwarning false
```

show specific commit

```
git show <sha-id>
```

see https://stackoverflow.com/questions/42996907/is-it-possible-to-search-by-commit-hash-in-gitk for gitk

Git Tools

- git
  - Text/Console
    - git `sudo apt install git`
    - tig `sudo apt install tig`
      - https://github.com/jonas/tig
    - https://github.com/tj/git-extras
    - https://github.com/unixorn/git-extra-commands
  - GUI
    - git-gui, gitk `sudo apt install git-gui gitk`
    - gitg for nice browsing `sudo apt install gitg`
- GitHub
  - https://cli.github.com/
    - https://github.com/cli/cli
    - https://cli.github.com/manual/gh
  - https://github.com/github/hub
  - https://github.com/cli/cli/blob/trunk/docs/gh-vs-hub.md
- Tool Overview
  - https://git-scm.com/download/gui/linux
  - giggle, git-cola and qgit look interesting
    - `sudo apt install giggle`
    - `sudo apt install git-cola`
    - `sudo apt install qgit`
- Setting up and using Meld as your git difftool and mergetool
  - https://stackoverflow.com/questions/34119866/setting-up-and-using-meld-as-your-git-difftool-and-mergetool

From:
https://codingspirit.de/dokuwiki/ - **coding spirit**

Permanent link:
https://codingspirit.de/dokuwiki/doku.php?id=development:git

Last update: **2025/08/22**