

Main Commands

git init	Create git repository here (decentralized)
git init --bare	Create bare git repository (centralized)
git clone <i>repo</i>	Clone repository at address <i>repo</i>
git clone -b <i>branch</i> <i>repo</i>	Clone <i>branch</i> from repository

Status

git status	List changes
git status --short --branch	List changes (compact)
git log	Display commit history
git log --oneline --decorate --graph -all	Display visual commit history
git config --global core.autocrlf true	Ignore OS-specific line endings

Commits

git add <i>hello.py</i>	Add <i>hello.py</i> (or stage changes) for next commit
git rm --cached <i>hello.py</i>	Remove <i>hello.py</i> from git tracking
git checkout -- <i>hello.py</i>	Discard all local changes to <i>hello.py</i>
git add -u	Stage all changes under tracking
git add -A	Stage all changes (new files and deletions included)
git update-index --assume-unchanged <i>hello.py</i>	Ignore changes to <i>hello.py</i> locally
git add --interactive	Review and stage changes
git commit	Create a commit (snapshot)
git commit -m " <i>message</i> "	Create a commit with message
git commit -a	Commit every change under tracking
git commit -amend	Combine current and last commit (correction)
git diff	Show all unstaged differences
git diff --cached	Show all differences including staged ones

Commits (cont)

git diff	See differences between <i>branch1</i> and <i>branch1..branch2</i>
	<i>branch2</i>

You can use commits instead of branch names. *HEAD* refers to current point in git.

Resetting

git reset --soft <i>commit</i>	Stage all changes since <i>commit</i> (used for squashing)
git reset --hard	Discard all local changes and reset to last commit

Maintenance

git revert <i>commit</i>	Creates a new commit out of an old commit (turn back in history)
git clean -dfx	Delete all untracked files and directories ! Dangerous
git clean -dfxn	Dry run - Show which files will be deleted
git gc --aggressive --prune=now	Reduce file size of repository by pruning
git checkout <i>file</i>	Discard local changes to <i>file</i>
bfg -delete-files <i>file</i>	BFG cleans sensitive data <i>file</i>

Remote Management

git remote add <i>target</i> <i>repo</i>	Add remote <i>target</i> at address <i>repo</i>
git remote -v	List all remotes
git remote set-url <i>target</i> <i>repo</i>	Changes the address of remote <i>target</i>
git push -f <i>origin</i> <i>commit.master</i>	Forces <i>origin/master</i> branch to be moved to <i>commit</i>
git pull (--rebase) <i>target</i> <i>branch</i>	Push existing commit to <i>target/branch</i>
git push <i>target</i> <i>branch</i>	Push <i>branch</i> to <i>target/branch</i>
git push <i>target</i> <i>branch1:branch2</i>	Push local <i>branch1</i> to <i>target/branch2</i>
git push -u <i>target</i> <i>branch</i>	Push and track <i>target/branch</i> . It sets upstream for default pull / push
git push <i>target</i> --delete <i>branch</i>	Deletes <i>target/branch</i> at remote

Remote Management (cont)

`git fetch target` Update branch information of *target* remote

Note: *repo* might start with `git@` or `https`. When it starts with `git@` an SSH key is needed for pull/push operations.

Branch management

`git checkout target` Switch to *target* branch

`git checkout -b feature` Create new branch from current commit

`git checkout -b feature master` Create new branch *feature* from *master* branch

`git branch -a` List all branches

`git branch -r --merged master` List remote branches that are merged with *master*

`git merge feature` Merge *feature* branch into current branch

`git merge --no-ff feature -m "message"` Merge *feature* with a commit (no fast-forward)

`git branch -d feature` Delete *feature* branch (if already merged)

`git branch -D feature` Force delete *feature* branch
! Dangerous

`git branch -f feature commit` Move branch head *feature* to *commit*
! Dangerous

Use

```
git for-each-ref --sort =committerdate refs/heads/ --format= '%(HEAD) %(refname:short) - %(objectname:short) - %(contents:subject) - %(authorname) (%(commitdate:relative))'
```

to print summary of each branch, last commit, and other info

Tag management

`git tag -a v1.0.0 -m "message"` Creates an annotated tag with message

`git push target --tags` Push local tags to remote *target*

`git tag -d v1.0.0` Delete local tag **v1.0.0*

`git push target :refs/tags/v1.0.0` Delete remote tag *target/v1.0.0*

Semantic Versioning

`git describe --tags --dirty` Describe current commit using tags

Submodules

`git submodule update --init --recursive .` Pull all submodules (linked repositories)

`git submodule status folder` Show the status of submodule under *folder*