

Main Commands

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

Status

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

Commits

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

Commits (cont)

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

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

Resetting

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

Maintenance

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

Remote Management

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

