GIT commands

|  |  |  |
| --- | --- | --- |
| **Git task** | **Notes** | **Git commands** |
| [**Tell Git who you are**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-config) | Configure the author name and email address to be used with your commits.  Note that Git [strips some characters](http://stackoverflow.com/questions/26159274/is-it-possible-to-have-a-trailing-period-in-user-name-in-git/26219423#26219423) (for example trailing periods) from user.name. | 1. **System level(/etc/gitconfig): git config –system user.name “om”** 2. **User level ($HOME/.gitconfig): git config –globaluser.name “om”** 3. **Project level (my\_proj/.git/config): git config user.name “om”** |
| [**Create a new local repository**](http://atlassian.com/git/tutorial/git-basics#!init) |  | git init |
| [**Check out a repository**](http://atlassian.com/git/tutorial/git-basics#!clone) | Create a working copy of a local repository: | git clone /path/to/repository |
| For a remote server, use: | git clone username@host:/path/to/repository |
| [**Add file**](http://atlassian.com/git/tutorial/git-basics#!add)**s in staging area** | Add one or more files to staging (index): | git add <filename>  git add [-A, -u, \*, .] |
| [**Commit**](http://atlassian.com/git/tutorial/git-basics#!commit) | Commit changes to head (but not yet to the remote repository): | git commit -m "Commit message" |
| Commit any files you've added with git add, and also commit any files you've changed since then: | git commit –a |
| See detailed info about a commit | Git show <commit-id> |
| [**Push**](http://atlassian.com/git/tutorial/remote-repositories#!push) | Send changes to the master branch of your remote repository: | git push origin master |
| [**Status**](http://atlassian.com/git/tutorial/git-basics#!status) | List the files you've changed and those you still need to add or commit: | git status |
| [**Connect to a remote repository**](http://atlassian.com/git/tutorial/remote-repositories#!remote) | If you haven't connected your local repository to a remote server, add the server to be able to push to it: | git remote add origin <server> |
| List all currently configured remote repositories: | git remote –v |
| [**Branches**](http://atlassian.com/git/tutorial/git-branches) | Create a new branch and switch to it: | git checkout -b <branchname> |
| Switch from one branch to another: | git checkout <branchname> |
| List all the branches in your repo, and also tell you what branch you're currently in: | git branch |
| Delete the feature branch: | git branch -d <branchname> |
| Push the branch to your remote repository, so others can use it: | git push origin <branchname> |
| Push all branches to your remote repository: | git push --all origin |
| Delete a branch on your remote repository: | git push origin :<branchname> |
| [**Update from the remote repository**](http://atlassian.com/git/tutorial/remote-repositories) | Fetch and merge changes on the remote server to your working directory: | git pull |
| To merge a different branch into your active branch: | git merge <branchname> |
| View all the merge conflicts:  View the conflicts against the base file:  Preview changes, before merging:  Difference using between 2 commits  View the changes in staged file against repo file  Difference between 2 branches  \*git diffmerge (lot easier tool "diffmerge", it need to be installed and configured.) | git diff  git diff --base <filename>  git diff <sourcebranch><targetbranch>  git diff commitID-1 commitID-2  git diff --staged  git diff branch1..branch2 |
| After you have manually resolved any conflicts, you mark the changed file: | git add <filename> |
| **Tags** | You can use tagging to mark a significant changeset, such as a release: | git tag 1.0.0 <commitID> |
| CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using: | git log |
| Push all tags to remote repository: | git push --tags origin |
| [**Undo local changes**](http://atlassian.com/git/tutorial/undoing-changes) | If you mess up, you can replace the changes in your working tree with the last content in head:  Changes already added to the index, as well as new files, will be kept. | git checkout -- <filename> |
| Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this: | git fetch origin  git reset --hard origin/master |
| **Search** | Search the working directory for foo(): | git grep "foo()" |
| **Ignore files** | 1. Ignore untracked files 2. Ignore untracked files globally 3. Ignoring tracked files 4. Tracking empty directories 5. Remove all untracked file | 1. Create .gitignore inside REPO and list all files which should be ignored 2. Git config –global core.exludesfile ~/.gitignore\_global (put files entry in gitignore\_global which should be ignored) 3. Git rm –cached <filename> and commit it. 4. Create .gitkeep dummy file inside that directory and commit. 5. Git clean -df |

**History:**

1. **Source Code Control System (SCCS) (1972): Single user at a time, Original file+snapshot of delta**
2. **Revision control system (RCS) (1982): Single user at a time, Latest file+snapshot of delta**
3. **Concurrent version system(CVS)(1986-1990): Multi user at a time, Remote repo comes in picture, central REPO**
4. **Apache subversion(SVN) (2000): Multi user at a time, keep eye on directory, central REPO**
5. **Bitkeeper SCM (2000): Close source, Open for linux kernel development**
6. **Git (2005): open source, DVCS (keep eye on change sets rather files/directories)**

**## Git: Revert/Reset Changes**

git checkout -- filename (undo changes in working area)

git reset (move staging area stuffs back to working area)

git reset HEAD~1 (reset commit by HEAD=1)

git reset –soft commitID (reset upto a particular commit. Changes are still there in both working directory+staging area)

git reset –mixed commitID (default)(reset upto a particular commit)(reset upto a particular commit. Changes are still there in working directory only)

git reset –hard commitID(reset upto a particular commit. Changes are lost)

git commit --amend -m "change messgae in previous commit"

git commit --amend (first do some changes and add changes under previous commit)

**## Git Tags**

git tag -a v1.4 -m "my version 1.4"

git tag -a v1.4 <commit-id> (adding tag to committed changes)

git show v1.4

git push origin v1.4

git push origin –tags (pushing many tags at a time)

**## Git Stash**

git stash save "message" (temporary working area stuffs in stash)

git stash list (list all stashed stuffs with status ID stash{#})

git stash show –p stash-ID

git stash pop stash-ID OR git stash applystash-ID(taking back stashed stuffs in working area)(pop removes from stash whereas apply keep a copy on stash)

git stash drop stashID{#}

git stash clear (Removes all stashed items)

How to recover a dropped stash in Git?

1. Find the stash commits:

git log --graph --oneline --decorate ( git fsck --no-reflog | awk '/dangling commit/ {print $3}' )

This will show you all the commits at the tips of your commit graph which are no longer referenced from any branch or tag – every lost commit, including every stash commit you’ve ever created, will be somewhere in that graph.

2. Once you know the hash of the commit you want, you can apply it as a stash:

git stash apply YOUR\_WIP\_COMMIT\_HASH

Note: The commit message will only be in this form (starting with "WIP on") if you did not supply a message when you did git stash.note

**## MERGE and Conflicts handling**

**Basic Merge (Assume, there is no change in master )**

git checkout <branchname> (make some changes and push to it’s repo)

git checkout master

True/Real merge: git merge<branchname>[git merge –no-ff <branchname>](consider all commits done in <branchname> as one commit and add in commit history of master and merge)

Fast-forward merge: git fast-fwd <branchname>[git merge –ff-only <branchname>] (add individual commits done in <branchname> in commit history of master and merge)

**Rebase&Merge (When there is change in master)**

1. make your branch squashed to commit-previousHEAD+1  
2. git checkout master   
3. git pull  
4. git checkout <your-branch>  
5. git rebase master  
6. fix the merge conflicts  
7. git add .  
8. git push -f origin <your-branch>

**Handling merge conflict:**

1. Abort merge (git merge --abort)
2. Resolve the conflicts manually (open the files having conflicts, keep content as per suitability, add file in git and commit.)
3. Using merge tool (git mergetool –tool=emerge)(other tools: vimdiff, bc3, p4merge, diffuse, xxdiff)

**## TIPS & TRICS**  
#Finding common ancestor (commit) of 2 different branch in git:  
commonCommitID=$(git log --oneline master|awk '{print $1}'>/var/tmp/gitlog;count=0;for i in `git log --oneline |awk '{print $1}'`; do if [ $count -eq 0 ];then grep $i /var/tmp/gitlog>/dev/null; if [ $? -eq 0 ];then echo $i;count=$((count+1)); fi;fi; done);echo $commonCommitID  
#See how many commits your branch is behaind from master  
git log --oneline master|grep -n $commonCommitID|awk -F: '{$1=$1-1;print $1}'  
#Compare file in 2 different commits (These 2 commits may belong to 2 different branches)  
git diff <commit-1> <commit-2> <specificFileName>  
#Find all commits which has a particular file modified  
git log -p <filename> |egrep 'commit|Author|Date'  
#Find all the files which were changed in 2 commits  
git diff --name-only <commit-1> <commit-2>  
#Show changed files only in a commit  
git show <commit-ID> --name-only

**## Using Ref Log (Recover master’s deleted commit-ID): Ref log helps to recover commits which has been reset.**

git reflog

git checkout <deleted commit hash from reflog> (to recover already reset commit)

git checkout -b backup (creating a branch from that commit as HEAD was in detached state)

git checkout master

git merge backup (Merging that commit back to master)

git branch –d backup

**## Git Cherry-pick use case (If you mistakenly committed in other branch. Cherry-pick help to get that commit to your desired branch)**

Assume: We committed in feature branch mistakenly (commit-ID: abc123) and wanted that commit in my release branch.

Git checkout release

Git cherry-pick abc123 (Taken that commit in release branch)

Git checkout feature

Git reset –hard abc123 (Removed that commit from feature branch)

**GIT Log:**

To see only the commits of a certain author:  
#git log --author=bob  
To see a very compressed log where each commit is one line:  
#git log --pretty=oneline  
Or maybe you want to see an ASCII art tree of all the branches, decorated with the names of tags and branches:   
#git log --graph --oneline --decorate --all  
See only which files have changed:   
#git log --name-status

See logs between a time period

#git log –oneline –since=2.weeks –until=2.days

See logs between 2 commits

#git log –oneline <commit-1>..<comit-2>

Filter from logs

#git log –grep=”temp”

**## To resolve GIT hooks for other branch**

git status

git push origin HEAD:refs/for/gic\_batch0

gitdir=$(git rev-parse --git-dir); scp -p -P 29418 ewaaddp@gerrit.ericsson.se:hooks/commit-msg ${gitdir}/hooks/

git commit --amend

git push origin HEAD:refs/for/gic\_batch0

**## Revert remote changes:**

* In the server, move the cursor back to the last known good commit:

#git push -f origin <last\_known\_good\_commitID>:<branch\_name>

* Locally, do the same:

#git reset --hard <last\_known\_good\_commitID>

**## Alter commit message of already pushed commit:**

#git fetch ssh://$USER@gerrit.ericsson.se:29418/puppet/hiera-linkoping refs/changes/99/3094499/3 && git checkout FETCH\_HEAD

#git commit –amend

#git push origin HEAD:refs/for/master

**References:**

[**https://git-scm.com/book/en/v2**](https://git-scm.com/book/en/v2)

[**https://git-scm.com/docs**](https://git-scm.com/docs)

**GitLab installation**

1. **Apt-get install git-core postfix gitlab7.1.1-omnibus-1**
2. **Gitlab-ctl reconfigure**
3. **Gitlab-ctl start**
4. **Gitlab-ctl start nginx**
5. **Gitlab-ctl start unicorn**

**Gitlab backup (DB and Repos)**

1. **Cp /**etc/gitlab/gitlab-secrets.json etc/gitlab/gitlab-secrets.json.bkp
2. **Gitlab-rake gitlab:backup:create (DB backup)**
3. tar -cvf $BAKDIR/$(date "+%s\_config\_gitlab.tar") /etc/gitlab (Repos backup in user defined backup directory)
4. **aws s3 sync** $BAKDIR/$(date "+%s\_config\_gitlab.tar") **s3://my-bucket/path --acl public-read (keeping backup in AWS S3 bucket)**

**Gitlab backup restore**

1. **Gitlab-ctl stop nginx**
2. **Gitlab-ctl stop unicorn**
3. **Gitlab-rake gitlab:backup:restore BACKUP=1393513186**