## Major GIT commands with examples

Major GIT commands:

git config

Sets configuration values for your user name, email, gpg key, preferred diff algorithm, file formats and more.  
  
Example: *git config --global user.name "My Name"  
                 git config --global user.email "user@domain.com"  
  
cat ~/.gitconfig  
[user]  
    name = My Name  
    email = user@domain.com*

git init

Initializes a git repository – creates the initial ‘.git’ directory in a new or in an existing project.  
  
Example: *cd /home/user/my\_new\_git\_folder/  
git init*

git clone

Makes a Git repository copy from a remote source. Also adds the original location as a remote so you can fetch from it again and push to it if you have permissions.  
  
Example: *git clone git@github.com:user/test.git*

git add

Adds files changes in your working directory to your index.  
  
Example: *git add .*

git rm

Removes files from your index and your working directory so they will not be tracked.  
  
Example: *git rm filename*

git commit

Takes all of the changes written in the index, creates a new commit object pointing to it and sets the branch to point to that new commit.  
  
Examples: *git commit -m ‘committing added changes’  
git commit -a -m ‘committing all changes, equals to git add and git commit’*

git status

Shows you the status of files in the index versus the working directory. It will list out files that are untracked (only in your working directory), modified (tracked but not yet updated in your index), and staged (added to your index and ready for committing).  
  
Example: *git status  
# On branch master  
#  
# Initial commit  
#  
# Untracked files:  
#   (use "git add <file>..." to include in what will be committed)  
#  
#    README  
nothing added to commit but untracked files present (use "git add" to track)*

git branch

Lists existing branches, including remote branches if ‘-a’ is provided. Creates a new branch if a branch name is provided.  
  
Example: *git branch -a  
\* master  
  remotes/origin/master*

git checkout

Checks out a different branch – switches branches by updating the index, working tree, and HEAD to reflect the chosen branch.  
  
Example: *git checkout newbranch*

git merge

Merges one or more branches into your current branch and automatically creates a new commit if there are no conflicts.  
  
Example: *git merge newbranchversion*

git reset

Resets your index and working directory to the state of your last commit.  
  
Example: *git reset --hard HEAD*

git stash

Temporarily saves changes that you don’t want to commit immediately. You can apply the changes later.  
  
Example: *git stash  
Saved working directory and index state "WIP on master: 84f241e first commit"  
HEAD is now at 84f241e first commit  
(To restore them type "git stash apply")*

git tag

Tags a specific commit with a simple, human readable handle that never moves.  
  
Example: *git tag -a v1.0 -m 'this is version 1.0 tag'*

git fetch

Fetches all the objects from the remote repository that are not present in the local one.  
  
Example: *git fetch origin*

git pull

Fetches the files from the remote repository and merges it with your local one. This command is equal to the git fetch and the git merge sequence.  
  
Example: *git pull origin*

git push

Pushes all the modified local objects to the remote repository and advances its branches.  
  
Example: *git push origin master*

git remote

Shows all the remote versions of your repository.  
  
Example: *git remote  
origin*

git log

Shows a listing of commits on a branch including the corresponding details.  
  
Example: *git log  
commit 84f241e8a0d768fb37ff7ad40e294b61a99a0abe  
Author: User <user@domain.com>  
Date:   Mon May 3 09:24:05 2010 +0300  
  
    first commit*

git show

Shows information about a git object.  
  
Example: *git show  
commit 84f241e8a0d768fb37ff7ad40e294b61a99a0abe  
Author: User <user@domain.com>  
Date:   Mon May 3 09:24:05 2010 +0300  
  
    first commit  
  
diff --git a/README b/README  
new file mode 100644  
index 0000000..e69de29*

git ls-tree

Shows a tree object, including the mode and the name of each item and the SHA-1 value of the blob or the tree that it points to.  
  
Example: *git ls-tree master^{tree}  
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    README*

git cat-file

Used to view the type of an object through the SHA-1 value.  
  
Example: *git cat-file -t e69de29bb2d1d6434b8b29ae775ad8c2e48c5391  
blob*

git grep

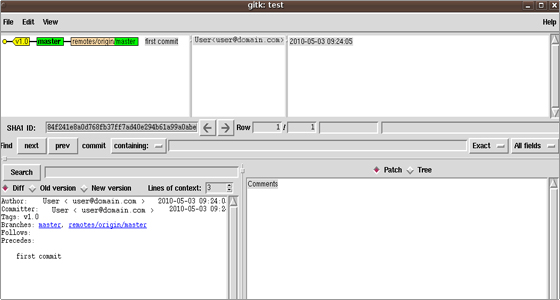
Lets you search through your trees of content for words and phrases.  
  
Example: *git grep "www.siteground.com" -- \*.php*

git diff

Generates patch files or statistics of differences between paths or files in your git repository, or your index or your working directory.  
  
Example: *git diff*

gitk

Graphical Tcl/Tk based interface to a local Git repository.  
  
Example: *gitk*



git instaweb

Runs a web server with an interface into your local repository and automatically directs a web browser to it.  
  
Example: *git instaweb --httpd=webrick  
git instaweb --stop*



git archive

Creates a tar or zip file including the contents of a single tree from your repository.  
  
Example: *git archive --format=zip master^ README >file.zip*

git gc

Garbage collector for your repository. Optimizes your repository. Should be run occasionally.  
  
Example: *git gc  
Counting objects: 7, done.  
Delta compression using up to 2 threads.  
Compressing objects: 100% (5/5), done.  
Writing objects: 100% (7/7), done.  
Total 7 (delta 1), reused 0 (delta 0)*

git fsck

Does an integrity check of the Git file system, identifying corrupted objects.  
  
Example: *git fsck*

git prune

Removes objects that are no longer pointed to by any object in any reachable branch.  
  
Example: *git prune*

**GIT Directory structure**

A .git directory has a structure similar to the following one:

branches

A deprecated way to store shorthands that specify URL to the git fetch, git pull and git push commands is to store a file in branches/<name> and give the name to the command in the place of the repository argument.

COMMIT\_EDITMSG

This is the last commit message. It’s not actually used by Git at all, but it is for your reference after you have made a commit.  
  
*user@user:/GIT/test# cat COMMIT\_EDITMSG  
first commit*

config

This is the main Git configuration file. It keeps specific Git options for your project, such as your remotes, push configurations, tracking branches and more. Your configuration will be loaded first from this file, then from a ~/.gitconfig file and then from an /etc/gitconfig file, if they exist.  
  
A exemplary content of this file is:  
  
*user@user:/GIT/test# cat config  
[core]  
    repositoryformatversion = 0  
    filemode = false  
    bare = false  
    logallrefupdates = true  
[remote "origin"]  
    url = git@github.com:user/test.git  
    fetch = +refs/heads/\*:refs/remotes/origin/\**

description

If you’re using gitweb or invoking git instaweb, this will show when you have viewed your repository or the list of all versioned repositories.

gitweb

A folder with the GIT web scripts. They allow you to browse the git repository using a web browser.

HEAD

This file holds a reference to the branch you are currently on. This tells Git what to use as the parent of your next commit:  
  
*user@user:/GIT/test# cat HEAD  
ref: refs/heads/master*

hooks/

This directory contains shell scripts that are invoked after the corresponding Git commands. For example, after you run a commit, Git will try to execute the post-commit script.

index

The Git index is used as a staging area between your working directory and your repository. You can use the index to build up a set of changes that you want to commit together. When you create a commit, what is committed is what is currently in the index, not what is in your working directory. It is a binary file containing a sorted list of path names, each with permissions and the SHA-1 of a blob object. Its content can be listed through:  
  
*user@user:/GIT/test# git ls-files --stage  
100644 e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 0    README*

info/

Contains additional information about the repository.

logs/

Keeps records of changes made to refs.

objects/

In this directory the data of your Git objects is stored – all the contents of the files you have ever checked in, your commits, trees and tag objects.  
The files are stored by their SHA-1 values. The first two characters are for the subdirectory and the next 38 are the filename.  
  
For example, if the SHA-1 for a blob we’ve checked in is a576fac355dd17e39fd2671b010e36299f713b4d  
the path to the corresponding file is:  
*[GIT\_DIR]/objects/a5/76fac355dd17e39fd2671b010e36299f713b4d*

ORIG\_HEAD

This is the previous state of HEAD.

packed-refs

The file consists of packed heads and tags. It is useful for an efficient repository access.

refs/

This directory normally contains three subfolders – heads, remotes and tags. There you will find the corresponding local branches, remote branches and tags files.

For example, if you create a production branch, the file .git/refs/heads/production will be created and will contain the SHA-1 of the latest branch commit.