GIT Command Reference Document

Santha Kumar K

GitHub can be accessed and manipulated using the standard Git command-line interface and all of the standard Git commands work with it.

Gusto Techno Solutions
Siri Sampada Hitech Building,
Opp. Bajaj Electronics,
Kavuri Hills, Phase – I,
Madhapur, Hyderabad

+91 96661 18111
+91 96662 82555

GIT Commands Reference

I. Setup

Show current configuration:

```
$ git config –list
```

Show repository configuration:

```
$ git config --local –list
```

Show global configuration:

```
$ git config --global –list
```

Show system configuration:

```
$ git config --system –list
```

Set a name that is identifiable for credit when review version history:

```
$ git config --global user.name “[firstname lastname]”
```

Set an email address that will be associated with each history marker:

```
$ git config --global user.email “[valid-email]”
```

Set automatic command line coloring for Git for easy reviewing:

```
$ git config --global color.ui auto
```

Set global editor for commit

```
$ git config --global core.editor vi
```

II. Configuration Files

Repository specific configuration file [--local]:

```
<repo>/.git/config
```

User-specific configuration file [--global]:

```
~/.gitconfig
```

System-wide configuration file [--system]:

```
/etc/gitconfig
```

III. Create

Clone an existing repository:

There are two ways:

1. Via SSH
```
$ git clone ssh://user@domain.com/repo.git
```
2. Via HTTP
```
$ git clone http://domain.com/user/repo.git
```

Create a new local repository:

```
$ git init
```

Local Changes

Changes in working directory:

```
$ git status
```

Changes to tracked files:

```
$ git diff
```

Add all current changes to the next commit:

```
$ git add .
```

Add some changes in <file> to the next commit:

```
$ git add -p <file>
```

Commit all local changes in tracked files:

```
$ git commit -a
```

Commit previously staged changes:

```
$ git commit
```

Commit with message:

```
$ git commit -m 'message here'
```

Commit skipping the staging area and adding message:

```
$ git commit -am 'message here'
```

Commit to some previous date:

```
git commit --date="`date --date='n day ago'`" -am "Commit Message"
```

Change last commit:

```
$ git commit -a --amend
```

Change committer date of last commit:

```
GIT_COMMITTER_DATE="date" git commit –amend
```

Change Author date of last commit:

```
git commit --amend --date="date"
```

Move uncommitted changes from current branch to some other branch:

```
git stash
git checkout branch2
git stash pop
```

Restore stashed changes back to current branch:

```
git stash apply
```

Remove the last set of stashed changes:

```
git stash drop
```

IV. Search

A text search on all files in the directory:

```
$ git grep "Hello"
```

In any version of a text search:

```
$ git grep "Hello" v2.5
```

V. Commit History

Show all commits, starting with newest (it'll show the hash, author information, date of commit and title of the commit):

```
$ git log
```

Show all the commits(it'll show just the commit hash and the commit message):

```
$ git log --oneline
```

Show all commits of a specific user:

```
$ git log --author="username"
```

Show changes over time for a specific file:

```
$ git log -p <file>
```

Display commits that are present only in remote/branch in right side

```
$ git log --oneline <origin/master>..<remote/master> --left-right
```

Who changed, what and when in <file>:

```
$ git blame <file>
```

Show Reference log:

```
$ git reflog show
```

Delete Reference log:

```
$ git reflog delete
```

VI. Branches & Tags

List all local branches:

```
$ git branch
```

List all remote branches:

```
$ git branch -r
```

Switch HEAD branch:

```
$ git checkout <branch>
```

Create and switch new branch:

```
$ git checkout -b <branch>
```

Create a new branch based on your current HEAD:

```
$ git branch <new-branch>
```

Create a new tracking branch based on a remote branch:

```
$ git branch --track <new-branch> <remote-branch>
```

Delete a local branch:

```
$ git branch -d <branch>
```

Force delete a local branch:

You will lose unmerged changes!

```
$ git branch -D <branch>
```

Mark the current commit with a tag:

```
$ git tag <tag-name>
```

Mark the current commit with a tag that includes a message:

```
$ git tag -a <tag-name>
```

VII. Update & Publish

List all current configured remotes:

```
$ git remote -v
```

Show information about a remote:

```
$ git remote show <remote>
```

Add new remote repository, named <remote>:

```
$ git remote add <remote> <url>
```

Download all changes from <remote>, but don't integrate into HEAD:

```
$ git fetch <remote>
```

Download changes and directly merge/integrate into HEAD:

```
$ git remote pull <remote> <url>
```

Get all changes from HEAD to local repository:

```
$ git pull origin master
```

Get all changes from HEAD to local repository without a merge:

```
git pull --rebase <remote> <branch>
```

Publish local changes on a remote:

```
$ git push remote <remote> <branch>
```

Delete a branch on the remote:

```
$ git push <remote> :<branch> (since Git v1.5.0)

or

git push <remote> --delete <branch> (since Git v1.7.0)
```

Publish your tags:

```
$ git push --tags
```

VIII. Merge & Rebase

Merge branch into your current HEAD:

```
$ git merge <branch>
```

Rebase your current HEAD onto <branch>:

Don't rebase published commit!

```
$ git rebase <branch>
```

Abort a rebase:

```
$ git rebase --abort
```

Continue a rebase after resolving conflicts:

```
$ git rebase --continue
```

Use your configured merge tool to solve conflicts:

```
$ git mergetool
```

Use your editor to manually solve conflicts and (after resolving) mark file as resolved:

```
$ git add <resolved-file>
$ git rm <resolved-file>
```

Squashing commits:

```
$ git rebase -i <commit-just-before-first>
```

Now replace this,

```
pick <commit_id>
pick <commit_id2>
pick <commit_id3>
```

to this,

```
pick <commit_id>
squash <commit_id2>
squash <commit_id3>
```

IX. Undo

Discard all local changes in your working directory:

```
$ git reset --hard HEAD
```

Get all the files out of the staging area(i.e. undo the last `git add`):

```
$ git reset HEAD
```

Discard local changes in a specific file:

```
$ git checkout HEAD <file>
```

Revert a commit (by producing a new commit with contrary changes):

```
$ git revert <commit>
```

Reset your HEAD pointer to a previous commit and discard all changes since then:

```
$ git reset --hard <commit>
```

Reset your HEAD pointer to a remote branch current state.

```
git reset --hard <remote/branch> e.g., upstream/master, origin/my-feature
```

Reset your HEAD pointer to a previous commit and preserve all changes as unstaged changes:

```
$ git reset <commit>
```

Reset your HEAD pointer to a previous commit and preserve uncommitted local changes:

```
$ git reset --keep <commit>
```

Remove files that were accidentally committed before they were added to .gitignore

```
$ git rm -r --cached .
$ git add .
$ git commit -m "remove xyz file"
```