**Introduction**

*This tutorial explains the usage of the distributed version control system Git via the command line.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Primary Responsibility** | **Secondary Responsibility** | **Escalation (Noida)** | **Escalation (Vancouver)** |
| **Name** | Abhishek | Amit Gupta | Vipul Kulshrestha | Rupmeet Singh |
| **Phone** | 9871252531 |  | +91-981-028-6872 | +1-778-320-4927 |
| **Skype** | ashu.abhishek18 |  | vipul.venture | rupmeet |
| **Email** | abhishek.v@optimusinfo.com |  | vipul.k@optimusinfo.com | [rupmeet.singh@optimusinfo.com](mailto:rupmeet.singh@optimusinfo.com) |

**Below mentioned are the basic steps to use Git CLI**

**1.  What is a distributed version control system?**

A distributed version control system has no central server which stores the data. The user can copy an existing *repository*. This copying process is called *cloning*.

Every local copy contains the full history of the collection of files and a cloned repository has the same functionality as the original repository.

**2. What is Git?**

*Git* is a distributed version control system written in the programming language *C*.

Git originates from the Linux kernel development and is used by many popular Open Source projects, e.g. the Android or the Eclipse Open Source projects, as well as by many commercial organizations.

**3. Installation**

**3.1. Ubuntu**

On Ubuntu you can install the Git command line tool via the following command:

**sudo apt-get install git-core**

To install Git on other Linux distributions please check your vendor documentation.

**3.2. Windows**

A windows version of Git can be found on the msysgit Project site. The URL to this webpage is listed below. This website also describes the installation process.

http://code.google.com/p/msysgit/

**3.3. Mac OS**

The easiest way to install Git on a Mac is via a graphical installer. This installer can be found under the following URL.

http://code.google.com/p/git-osx-installer

As this procedure it not an official Apple one, it may change from time to time. The easiest way to find the current procedure is to Google for the "How to install Git on a Mac" search term.

**4. Git Setup**

**4.1. Global configuration file**

Git allows you to store global settings in the .gitconfig file. This file is located in the user home directory. Git stores the committer and author of a change in each commit. This and additional information can be stored in the global settings.

**4.2. User Configuration**

Configure your user and email for Git via the following command.

# configure the user which will be used by git

**git config --global user.name "Example Surname"**

# Same for the email address

**git config --global user.email "your.email@gmail.com"**

To make pushing to remote repositories easier and to avoid unnecessary commits, you can use the following commands.

# set default so that all changes are always pushed to the repository

**git config --global push.default "matching"**

# set default so that you avoid unnecessary commits

**git config --global branch.autosetuprebase always**

**4.3. Color Highlighting**

The following enables highlighting for Git in the console.

**git config --global color.status auto**

**git config --global color.branch auto**

**4.4. Setting the default editor**

By default Git uses the system default editor. You can configure this via the following setting.

**git config --global core.editor vim**

**4.5. Query existing global Git settings**

To query your Git settings, execute the following command:

**git config --list**

**5. Clone Git repository**

Every Git repository is stored in the .git folder of the directory in which the Git repository has been created. This directory contains the complete history of the repository. The .git/config file contains the local configuration for the repository.

The following command clone the git repository

**git clone [git URL]**

**6. Add files to Git index**

Before committing to a Git repository you need to mark which changes should be committed by adding the new and changed files to the Git index. i.e. the staging area. This creates a snapshop of the affected files, if you afterwards change one of the files before committing, you need to add it again to the index to commit the new changes.

# add all files to the index of the Git repository

**git add .**

**7. Git Status**

As you saw in the git add section, in order to see what the status of your staging area is compared to the code in your working directory, you can run the git status command. Using the -s option will give you short output. Without that flag, the git status command will give you more context and hints. Here is the same status output with and without the -s. The short output looks like this:

**git status -s**

**8. Commit to Git repository**

After adding the files to the Git index, you can commit them to the Git repository. This creates a new snapshot of all your files in your Git repository.

# commit your file to the local repository

**git commit -m "Initial commit"**

**9. Remove files and adjust the last commit**

**9.1. Remove files**

If you delete a file which is under version control git add . will not pick this file up.

You can use the git rm command to delete the file from your working tree and mark it for the next commit.

# Create a file and commit it

**touch nonsense2.txt**

**git add . && git commit -m "more nonsense"**

# remove the file via Git

**git rm nonsense2.txt**

# commit the removal

**git commit -m "Removes nonsense2.txt file"**

Alternatively you can use the git commit command with the -a flag or the -A flag in the git add command.

For this test, commit a new file and remove it afterwards.

# Create a file and put it under version control

**touch nonsense.txt**

**git add . && git commit -m "a new file has been created"**

# Remove the file

**rm nonsense.txt**

# Try standard way of committing -> will NOT work

**git add . && git commit -m "a new file has been created"**

Now remove it from the Git repository.

# commit the remove with the -a flag

**git commit -a -m "File nonsense.txt is now removed"**

**9.2. Remove a file from the index**

You can use the git reset [filename] command to remove a file from the index, which you added with git add [filename] .

# create a file and add to index

**touch unwantedstaged.txt**

**git add unwantedstaged.txt**

# remove it from the index

**git reset unwantedstaged.txt**

# to cleanup, delete it

**rm unwantedstaged.txt**

**10. Git diff**

The way we're going to use it here is to describe the changes that are staged or modified on disk but unstaged.

Without any extra arguments, a simple git diff will display in unified diff format (a patch) what code or content you've changed in your project since the last commit that are not yet staged for the next commit snapshot.

#show diff of unstaged changes

**git diff**

So where git status will show you what files have changed and/or been staged since your last commit, git diff will show you what those changes actually are, line by line. It's generally a good follow-up command to git status

# show diff of staged changes

**git diff --cached**

If you want to see both staged and unstaged changes together, you can run git diff HEAD - this basically means you want to see the difference between your working directory and the last commit, ignoring the staging area. If we make another change to our hello.rb file then we'll have some changes staged and some changes unstaged. Here are what all three diff commands will show you:

# show diff of all staged or unstaged changes

**git diff HEAD**

If we don't want the full diff output, but we want more than the git status output, we can use the --stat option, which will give us a summary of changes instead. Here is the same example as above, but using the --stat option instead.

# git diff --stat show summary of changes instead of a full diff

**git diff --stat**

**10. Git Fetch and Pull**

You can update your remote branches with the git fetch command.

The git fetch command updates your *remote branches*. The fetch command only updates the *remote branches* and none of the local branches and it does not change the working tree of the Git repository. Therefore you can run the git fetch command at any point in time.

# update your remote tracking branch

**git fetch [remote-name]**

The git pull command performs a git fetch and git merge (or git rebase based on your Git settings). Thegit fetch does not perform any operations on your local branches.

**git pull origin**

**11  Git Push:-**

The git push command allows you to send data to other repositories. By default it sends data from your current branch to the master branch of the remote repository called origin.

# push changes to remote repository origin is automatically created as we cloned original from this repository

**git push origin**

**12  Git References:-**

<http://gitref.org/>

<http://www.vogella.com/articles/Git/article.html#cloneremotes>

<http://git-scm.com/book/commands>