**About the Tutorial**

Git is a distributed revision control and source code management system with an emphasis on speed. Git was initially designed and developed by Linus Torvalds for Linux kernel development. Git is a free software distributed under the terms of the GNU General Public License version 2.

This tutorial explains how to use Git for project version control in a distributed environment while working on web-based and non-web-based applications development.

**Audience**

This tutorial will help beginners learn the basic functionality of Git version control system. After completing this tutorial, you will find yourself at a moderate level of expertise in using Git version control system from where you can take yourself to the next levels.

**Prerequisites**

We assume that you are going to use Git to handle all levels of Java and non-Java projects. So it will be good if you have some amount of exposure to software development life cycle and working knowledge of developing web-based and non-web-based applications.

**Copyright & Disclaimer**

© Copyright 2018 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com
# Table of Contents

1. **GIT — BASIC CONCEPTS** ................................................................. 1
   - Version Control System ................................................................ 1
   - Distributed Version Control System ............................................ 1
   - Advantages of Git ......................................................................... 2
   - DVCS Terminologies .................................................................... 3

2. **GIT — ENVIRONMENT SETUP** ..................................................... 7
   - Installation of Git Client .............................................................. 7
   - Customize Git Environment .......................................................... 7

3. **GIT — LIFE CYCLE** ................................................................... 10

4. **GIT — CREATE OPERATION** ...................................................... 11
   - Create New User ......................................................................... 11
   - Create a Bare Repository .............................................................. 11
   - Generate Public/Private RSA Key Pair ........................................ 12
   - Adding Keys to authorized_keys ................................................ 13
   - Push Changes to the Repository .................................................. 14

5. **GIT — CLONE OPERATION** ....................................................... 17

6. **GIT — PERFORM CHANGES** .................................................... 18
Shortcut to Create and Switch Branch .......................... 59
Delete a Branch .......................................................... 59
Rename a Branch ......................................................... 60
Merge Two Branches ................................................... 61
Rebase Branches .......................................................... 65

19. GIT—HANDLING CONFLICTS ......................................... 67
   Perform Changes in wchar_support Branch .................. 67
   Perform Changes in Master Branch .............................. 68
   Tackle Conflicts ....................................................... 71
   Resolve Conflicts ..................................................... 71

20. GIT—DIFFERENT PLATFORMS ........................................... 75

21. GIT—ONLINE REPOSITORIES .......................................... 76
   Create GitHub Repository ........................................... 76
   Push Operation ........................................................ 76
   Pull Operation ......................................................... 78
**Version Control System**

**Version Control System (VCS)** is a software that helps software developers to work together and maintain a complete history of their work.

Listed below are the functions of a VCS:

- Allows developers to work simultaneously.
- Does not allow overwriting each other’s changes.
- Maintains a history of every version.

Following are the types of VCS:

- Centralized version control system (CVCS).
- Distributed/Decentralized version control system (DVCS).

In this chapter, we will concentrate only on distributed version control system and especially on Git. Git falls under distributed version control system.

**Distributed Version Control System**

Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. But the major drawback of CVCS is its single point of failure, i.e., failure of the central server. Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all. And even in a worst case, if the disk of the central server gets corrupted and proper backup has not been taken, then you will lose the entire history of the project. Here, distributed version control system (DVCS) comes into picture.

DVCS clients not only check out the latest snapshot of the directory but they also fully mirror the repository. If the server goes down, then the repository from any client can be copied back to the server to restore it. Every checkout is a full backup of the repository. Git does not rely on the central server and that is why you can perform many operations when you are offline. You can commit changes, create branches, view logs, and perform other operations when you are offline. You require network connection only to publish your changes and take the latest changes.
Advantages of Git

Free and open source
Git is released under GPL’s open source license. It is available freely over the internet. You can use Git to manage propriety projects without paying a single penny. As it is an open source, you can download its source code and also perform changes according to your requirements.

Fast and small
As most of the operations are performed locally, it gives a huge benefit in terms of speed. Git does not rely on the central server; that is why, there is no need to interact with the remote server for every operation performed. The core part of Git is written in C, which avoids runtime overheads associated with other high-level languages. Though Git mirrors entire repository, the size of the data on the client side is small. This illustrates the efficiency of Git at compressing and storing data on the client side.

Implicit backup
The chances of losing data are very rare when there are multiple copies of it. Data present on any client side mirrors the repository, hence it can be used in the event of a crash or disk corruption.

Security
Git uses a common cryptographic hash function called secure hash function (SHA1), to name and identify objects within its database. Every file and commit is check-summed and retrieved by its checksum at the time of checkout. It implies that it is impossible to change file, date, and commit message and any other data from the Git database without knowing Git.

No need of powerful hardware
In case of CVCS, the central server needs to be powerful enough to serve requests of the entire team. For smaller teams, it is not an issue, but as the team size grows, the hardware limitations of the server can be a performance bottleneck. In case of DVCS, developers don’t interact with the server unless they need to push or pull changes. All the heavy lifting happens on the client side, so the server hardware can be very simple indeed.

Easier branching
CVCS uses cheap copy mechanism. If we create a new branch, it will copy all the codes to the new branch, so it is time-consuming and not efficient. Also, deletion and merging of branches in CVCS is complicated and time-consuming. But branch management with Git is very simple. It takes only a few seconds to create, delete, and merge branches.
DVCS Terminologies

Local Repository

Every VCS tool provides a private workplace as a working copy. Developers make changes in their private workplace and after commit, these changes become a part of the repository. Git takes it one step further by providing them a private copy of the whole repository. Users can perform many operations with this repository such as add file, remove file, rename file, move file, commit changes, and many more.

Working Directory and Staging Area or Index

The working directory is the place where files are checked out. In other CVCS, developers generally make modifications and commit their changes directly to the repository. But Git uses a different strategy. Git doesn’t track each and every modified file. Whenever you do commit an operation, Git looks for the files present in the staging area. Only those files present in the staging area are considered for commit and not all the modified files.

Let us see the basic workflow of Git.

**Step 1:** You modify a file from the working directory.

**Step 2:** You add these files to the staging area.

**Step 3:** You perform commit operation that moves the files from the staging area. After push operation, it stores the changes permanently to the Git repository.

Suppose you modified two files, namely “sort.c” and “search.c” and you want two different commits for each operation. You can add one file in the staging
area and do commit. After the first commit, repeat the same procedure for another file.

```
# First commit
[bash]$ git add sort.c

# adds file to the staging area
[bash]$ git commit -m "Added sort operation"

# Second commit
[bash]$ git add search.c

# adds file to the staging area
[bash]$ git commit -m "Added search operation"
```

**Blobs**

Blob stands for **Binary Large Object**. Each version of a file is represented by blob. A blob holds the file data but doesn’t contain any metadata about the file. It is a binary file and in Git database, it is named as SHA1 hash of that file. In Git, files are not addressed by names. Everything is content-addressed.

**Trees**

Tree is an object, which represents a directory. It holds blobs as well as other sub-directories. A tree is a binary file that stores references to blobs and trees which are also named as **SHA1** hash of the tree object.

**Commits**

Commit holds the current state of the repository. A commit is also named by **SHA1** hash. You can consider a commit object as a node of the linked list. Every commit object has a pointer to the parent commit object. From a given commit, you can traverse back by looking at the parent pointer to view the history of the commit. If a commit has multiple parent commits, then that particular commit has been created by merging two branches.

**Branches**

Branches are used to create another line of development. By default, Git has a master branch, which is same as trunk in Subversion. Usually, a branch is created to work on a new feature. Once the feature is completed, it is merged back with the master branch and we delete the branch. Every branch is
Referenced by HEAD, which points to the latest commit in the branch. Whenever you make a commit, HEAD is updated with the latest commit.

**Tags**

Tag assigns a meaningful name with a specific version in the repository. Tags are very similar to branches, but the difference is that tags are immutable. It means, tag is a branch, which nobody intends to modify. Once a tag is created for a particular commit, even if you create a new commit, it will not be updated. Usually, developers create tags for product releases.

**Clone**

Clone operation creates the instance of the repository. Clone operation not only checks out the working copy, but it also mirrors the complete repository. Users can perform many operations with this local repository. The only time networking gets involved is when the repository instances are being synchronized.

**Pull**

Pull operation copies the changes from a remote repository instance to a local one. The pull operation is used for synchronization between two repository instances. This is same as the update operation in Subversion.

**Push**

Push operation copies changes from a local repository instance to a remote one. This is used to store the changes permanently into the Git repository. This is same as the commit operation in Subversion.

**HEAD**

HEAD is a pointer, which always points to the latest commit in the branch. Whenever you make a commit, HEAD is updated with the latest commit. The heads of the branches are stored in `.git/refs/heads/` directory.

```
[CentOS]$ ls -1 .git/refs/heads/
master

[CentOS]$ cat .git/refs/heads/master
570837e7d58fa4bccd86cb575d884502188b0c49
```
**Revision**

Revision represents the version of the source code. Revisions in Git are represented by commits. These commits are identified by **SHA1** secure hashes.

**URL**

URL represents the location of the Git repository. Git URL is stored in config file.

```
[tom@CentOS tom_repo]$ pwd
/home/tom/tom_repo

[tom@CentOS tom_repo]$ cat .git/config
[core]
repositoryformatversion = 0
filemode = true
bare = false
logallrefupdates = true
[remote "origin"]
url = gituser@git.server.com:project.git
fetch = +refs/heads/*:refs/remotes/origin/*
```
Before you can use Git, you have to install and do some basic configuration changes. Below are the steps to install Git client on Ubuntu and Centos Linux.

### Installation of Git Client

If you are using Debian base GNU/Linux distribution, then `apt-get` command will do the needful.

```
[ubuntu ~]$ sudo apt-get install git-core
[sudo] password for ubuntu:

[ubuntu ~]$ git --version
git version 1.8.1.2
```

And if you are using RPM based GNU/Linux distribution, then use `yum` command as given.

```
[CentOS ~]$ su -
Password:

[CentOS ~]$ yum -y install git-core

[CentOS ~]$ git --version
git version 1.7.1
```

### Customize Git Environment

Git provides the `git config` tool, which allows you to set configuration variables. Git stores all global configurations in `.gitconfig` file, which is located in your home directory. To set these configuration values as global, add the `-global` option, and if you omit `-global` option, then your configurations are specific for the current Git repository.

You can also set up system wide configuration. Git stores these values in the `/etc/gitconfig` file, which contains the configuration for every user and
repository on the system. To set these values, you must have the root rights and use the **--system** option.

When the above code is compiled and executed, it produces the following result:

**Setting username**
This information is used by Git for each commit.

```
[jerry@CentOS project]$ git config --global user.name "Jerry Mouse"
```

**Setting email id**
This information is used by Git for each commit.

```
[jerry@CentOS project]$ git config --global user.email "jerry@tutorialspoint.com"
```

**Avoid merge commits for pulling**
You pull the latest changes from a remote repository, and if these changes are divergent, then by default Git creates merge commits. We can avoid this via following settings.

```
jerry@CentOS project]$ git config --global branch.autosetuprebase always
```

**Color highlighting**
The following commands enable color highlighting for Git in the console.

```
[jerry@CentOS project]$ git config --global color.ui true
[jerry@CentOS project]$ git config --global color.status auto
[jerry@CentOS project]$ git config --global color.branch auto
```

**Setting default editor**
By default, Git uses the system default editor, which is taken from the VISUAL or EDITOR environment variable. We can configure a different one by using git config.
End of ebook preview
If you liked what you saw...
Buy it from our store @ https://store.tutorialspoint.com