Git Repository Management in 30 Days

Learn to manage code repositories like a pro

Sumit Jaiswal

www.bpbonline.com

First published: 2023

Published by BPB Online
WeWork
119 Marylebone Road
London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55518-071

www.bpbonline.com

Dedicated to

My beloved wife

Kanika

&

My daughter

Anika

About the Author

Sumit Jaiswal has been engaged in software development for over 11 years, serving as a technical leader and software engineer on several projects utilizing Open-Source Technologies. He is currently a Principal Engineer at Ansible by RedHat. Meanwhile, he has obtained multiple Kubernetes and Security certifications. Furthermore, the author speaks at international conferences and writes technical blogs on Open-Source-related topics.

About the Reviewer

Paul Oluyege is an Innovative, Result and Data-driven Software Engineer and Manager. His professional experience of about 10 years cut across software development, process management, people management, project management and product management. He currently leads and manages a cross-functional development team of 20 members building and maintaining both new and existing products in multiple domains (Fintech, E-commerce B2B, B2C, SAAS, Logistics). Alongside, he is a tech author, tech coach and mentor.

Acknowledgement

I want to express my deepest gratitude to my family and friends for their unwavering support and encouragement throughout this book's writing, especially my wife Kanika and my daughter Anika.

I am also grateful to BPB Publications for their guidance and expertise in bringing this book to fruition. It was a long journey of revising this book, with valuable participation and collaboration of reviewers, technical experts, and editors.

I would also like to acknowledge the valuable contributions of my colleagues and co-worker during many years working in the tech industry, who have taught me so much and provided valuable feedback on my work.

Finally, I would like to thank all the readers who have taken an interest in my book and for their support in making it a reality. Your encouragement has been invaluable.

Preface

Git Repository Management in 30 Days welcomes you!

This book will give you a complete and practical approach to managing Git repositories. This book will help you grasp Git and take control of you are code, whether you're a newbie or an experienced developer.

Git is a critical tool for code management in modern software development. It lets engineers effectively track changes, collaborate with others, and manage code versions. Git, on the other hand, can be difficult and overwhelming for people who are new to it. This is where this book comes into play. This guide has been created to help you learn Git systematically and logically, with lessons that will take you from novice to expert in 30 days.

Each chapter of this book delves into a different facet of Git, beginning with the fundamentals of version control and progressing to more advanced topics like branching, merging, and rebasing. Collaboration, troubleshooting, and best practices for optimizing your productivity will also be covered. By the end of this book, you'll be able to confidently manage code repositories, interact with others, and streamline your development process.

With clear explanations, real-world examples, and step-by-step directions, this book is intended to be practical and approachable. We've also included challenges to help you put what you've learned into practise and improve your skills. This book is for you if you are a student, a nonexpert, or a professional developer.

Thank you for your interest in "Git Repository Management in 30 Days". We hope you find it useful and educational, and we look forward to assisting you in mastering Git and advancing your development abilities!

Chapter 1: Introduction to Git and GitHub - This is the introductory chapter. Source control is one of the key concepts and tools that is widely used in the software development process and without which DevOps makes little sense as it helps to bring collaboration and transparency between the development and operation teams. One of the most popular and well-liked source control systems is GIT, which is elaborated and extended by GitHub. This chapter covers the configuration and setup of GIT on various operating systems, as well as the creation of a GitHub account.

Chapter 2: Getting Started and Understanding Git and GitHub - Git and GitHub go hand in hand, but users should be aware of the differences that define each other's roles in the software development process. Any system or tool used to store and manage changes to projects over time is referred to as version control. The key advantages of source control include standardizing coding practices, parallelizing development activities, and eliminating dependencies. This chapter covers all the details around version control and goes on to examine Git in depth and detail, allowing you to clear a few basics about Git and make the learning process go more smoothly. Discussing Git gradually leads to the distinctions between Git and GitHub.

Chapter 3: Git Branching, Merging, and Rebasing - This chapter focuses on the essential capabilities of Git and GitHub, as well as how they complement each other in the software development and DevOps processes. It addresses the essential ideas of GIT as well as the basic day-to-day processes and commands that you may encounter while using the Git source control.

Chapter 4: Deleting, Renaming, and Ignoring Files in Git - This chapter builds on what readers learned in the previous chapter and allows you to make the final decision before pushing and committing changes to source control. This process of committing changes to the GitHub repo may include renaming, deleting, and ignoring files in the project.

Chapter 5: Collaborating Towards Your/Other Larger Projects over GitHub - This chapter discusses all of the process-related and critical aspects that should be kept in mind and followed before attempting to contribute to an open-source project that is being followed and used by a larger community from all over the world, as opposed to maintaining and contributing to a repo maintained by a single user.

Chapter 6: Contributing Towards Open-Source Project Repo - As one of the most important applications of using Git and GitHub together is how users can contribute to open-source projects that are part of GitHub, and having worked in open-source projects for quite some time, I've gained insights into how one should approach their contributions towards an open-source way of working and process, and one very important aspect of this is raising PR and issues over GitHub open source projects in a way that can get the most traction and help.

Chapter 7: Tags and Releases Using Git - This chapter goes over all of the Git and GitHub processes and important points to remember. Git only stores four kinds of objects in its object store: blobs, trees, commits, and tags. Managing releases using

Git and GitHub is a basic and straightforward procedure, and we will learn all about the principles and underlying commands involved.

Chapter 8: Undo or Refresh all the Work Done - This chapter focuses on Git's undo/refresh functionality, discussing all of the principles that Git exposes to assist users to achieve similar functionalities, as well as how employing GitHub processes and workflows can aid and make the corresponding job seamless and efficient.

Chapter 9: Most Commonly Used Git Commands - This chapter is essentially a summary of all of the chapters that we have gone through together. It will assist all users, whether beginner, intermediate, or advanced, in referring to this chapter whenever they may find it useful.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/aqascyr

The code bundle for the book is also hosted on GitHub at **https://github.com/bpbpublications/Git-Repository-Management-in-30-Days**. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

1. Introduction to Git and GitHub 1
Structure 1
Objectives 2
What is version control 2
Local Version Control Systems 3
Centralized Version Control Systems 4
Distributed Version Control Systems 4
Git History 5
What is Git 6
Git three States 8
Getting started with Git 9
Linux/Unix 10
Mac OS 15
Windows 16
Introducing GitHub 24
Creating and configuring the GitHub account 25
Conclusion 30
Multiple choice questions 30
Answers 32
Key terms 32
Points to remember 33

2. Getting Started and Understanding Git and GitHub 35
Structure 35
Objectives 36
Difference between Git and GitHub 36
GitHub fundamental 39
Creating a repository on GitHub 41
Committing changes to your repository 46

Conclusion 48
Multiple choice questions 49
Answers *49*
Key terms 49
Points to remember 50
Further reading 50

3. Git Branching, Merging, and Rebasing 51
Structure 51
Objectives 52
Introducing Git options 52
Git options 52
Git commands 55
Starting a working area *56*
Git init - Initialize Git repository *57*
Git clone - Clone a Git repository into a new directory *60*
Work on the current change *66*
Git add - Adding file contents to the index *66*
Mv - Move or rename a file, a directory, or a symlink *68*
Restore - Restore working tree files *69*
rm - Remove files from the working tree and from the index *72*
sparse-checkout - Initialize and modify the sparse-checkout *74*
To examine the history and state of the repository *74*
bisect *74*
diff *76*
grep *79*
log *80*
show *81*
status *82*
To grow, mark and tweak your repo history *86*
branch *86*
Commit *90*

Merge 92
Rebase 93
Tag 97
To collaborate over repository 102
fetch 102
Pull 105
Push 107
Conclusion 111
Multiple choice questions 111
Answers 112
Key terms 112
Points to remember 113
Further reading 114

4. Deleting, Renaming, and Ignoring Files in Git 115
Structure 115
Objectives 116
Delete the Git file 116
Options 116
Examples 117
Git rm cached 118
Undo before Commit command 119
Git rename files 120
Method 1 120
Method 2 122
Git branching 123
Local vs remote Git branch 123
Working of Git commit 125
Ignoring the files using .gitignore 125
The .gitignore files 126
The .gitignore patterns, that is, file structure 126
.gitignore sample 129

Global .gitignore 129
Ignoring a previously committed file 129
Stashing an ignored file 131
Debugging .gitignore File 131
Git commit: save the staged changes 132
How Git commits differs from SVNs 132
Options 133
Examples 134
Conclusion 136
Multiple choice questions 136
Answers 138
Key terms 138
Points to remember 138
Further reading 139

5. Collaborating Towards Your/Other Larger Projects over GitHub 141
Structure 141
Objectives 142
Clone and fork the GitHub repository 142
Cloning, forking, and duplicating 142
Cloning repository 143
Forking repository 144
Duplicating repository 145
Why forking repository is needed 147
Creating a Pull request from forked repository 149
Contributing to single repository 150
Moving your changes to new branch 151
Make the source repository the upstream remote setting 152
Fork the repo 152
Set your forked repository as the origin remote: 152
Send your branch to the forked copy 153
Create a new pull request 153

Collaborating on pull request *153*
Collaborators' involvement in the pull request *154*
Pull request review process *154*
Commenting over a pull request *155*
Contributing to a pull request *155*
Testing pull request *156*
Merging pull request *157*
Who should merge the pull request *157*
Git Aliases 158
Conclusion 159
Multiple choice questions 159
Answers *160*
Key terms 160
Points to remember 161
Further reading 162

6. Contributing Towards Open-Source Project Repo 163
Introduction 163
Structure 163
Objectives 164
Understanding a pull request 164
Nature of a pull request *164*
Git pull *165*
Git pull from remote branch *169*
Git force pull *169*
A complete GitHub workflow *170*
GitHub Workflow with pull requests *171*
Fork Workflow with pull requests *171*
GitHub for Code distribution *172*
Open a pull request over GitHub 172
Opening a pull request *175*
Describing the pull request *176*

Adding reviewers *177*
Adding assignees *177*
Adding labels *178*
Adding projects and milestones *178*
Creating the pull request *178*
Writing a good pull request *179*
Maintaining the focus *180*
Suggesting changes *181*
Finish review *183*
Merging Pull Request *184*
Writing a great bug report 185
Characteristics of a quality software bug report *186*
Effective bug reporting *188*
Pushing code and opening a pull request over GitHub 189
Summary 190
Conclusion 190
Multiple choice questions 190
Answers *191*
Further readings 192

7. Tags and Releases Using Git 193
Structure 193
Objectives 194
Release tags versus release branches 194
Git Tag 195
Git Create tag *196*
Annotated tag *196*
Light-weighted tag *197*
Git list tag 198
Tagging old commits *199*
Git Push tag *200*
Git Delete tag *202*

Delete remote repository tag *203*
Delete multiple tags *203*
Git checkout tags *204*
Retagging/Replacing old tags *204*
Git branch 205
Git main branch *206*
Operations on branches *206*
Cherry-Pick commit for reuse 209
Need for Cherry-Picking *210*
Git Stash for code reusability 212
Git stash branch *213*
Save Git Stash *214*
List Git Stash *214*
Apply Git Stash *214*
Git stash changes *215*
Re-applying your stashed changes *216*
Git stash branch *217*
Git stash cleaning *218*
Conclusion 218
Multiple choice questions 218
Answers *219*
Key terms 220
Points to remember 220
Further reading 221

8. Undo or Refresh all the Work Done 223
Structure 223
Objectives 224
Undo and refresh changes in Git 224
Navigating log *226*
Git log Oneline *226*
Git log Log-Size *227*

Git log Stat 227
Git log graph 228
Filtering the commit history 229
Git reflog versus Git log 233
Git revert 233
Git revert to previous commit 234
Git reset 236
Git reset hard 237
Git reset mixed 238
Git reset soft 239
Git reset to commit 240
Resetting versus reverting 240
Amend Git commit 240
Changing most recent Git commit message 241
Changing committed files 241
Interactive rebase 242
Interactive rebasing at work 242
Squash commits together 244
Rebase on top of main 246
Re-writing history risks 247
Conclusion 247
Multiple choice questions 247
Answers 248
Points to remember 249
Further readings 249

9. Most Commonly Used Git Commands 251
Structure 251
Objectives 252
Git config 252
Git init 253
Git clone 253

Git status 254
Git add 254
Git commit 255
Git push 255
Git branch 255
Git checkout 256
Git merge 257
Git pull 257
Git log 258
Git show 258
Git diff 258
Git tag 258
Git rm 259
Git stash 259
Git reset 260
Git revert 260
Git remote 261
Git fetch 261
Conclusion 261
Multiple choice questions 262
Answers 263
Key terms 263
Further reading 264

Index ..... 265 -269

Chapter 1
Introduction to Git and GitHub

Source control is one of the key concepts and tools used extensively in software development. With it, DevOps makes more sense as it helps bring collaboration and transparency between the development and the operation teams. The tracking and management of code changes are known as source control, and it ensures that developers are constantly working on the correct version of the source. One of the most used and loved by community source control is Git, which is elaborated and extended by GitHub. This chapter is about the configuration and setup of Git over different flavors of **Operating System** (**OS**) and setting up an account over GitHub.

Structure

In this chapter, we will cover the following topics:

- Version Control
- Introducing Git and GitHub
- Getting started with Git
    - Linus/Unix
    - Mac OS
    - Windows
- Creating and configuring the GitHub account

Objectives

After reading this chapter, you will get an understanding of What is source version control, and the Git version control. You will also get equipped with introducing Git and GitHub. You will also understand the running instance of Git and the difference between Git and GitHub. By the end of this chapter, you will have learned how to create an account on GitHub.

Once completed, you will learn about different types of version control systems, and how they evolved and resulted in the creation of Git. And as we progress through the chapter, we will go through all the information and requirements needed to follow along and complete all the examples and concepts discussed in the upcoming chapters, making the reading and development process easily consumable.

What is version control

Version control, also known as source control, refers to tracking and managing changes to code. This ensures that developers are always working on the right version of the source code.

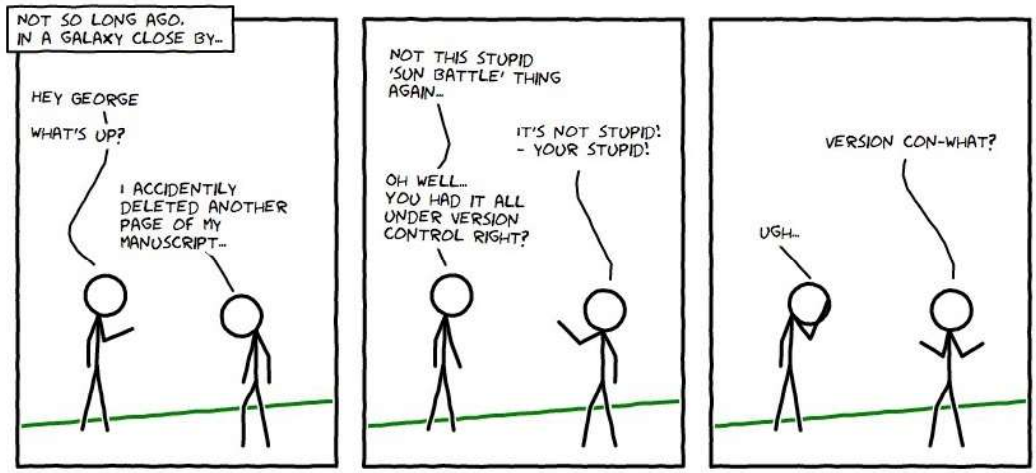

***Figure 1.1**: Why version control (credit: smutch)*

Why should you care? Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

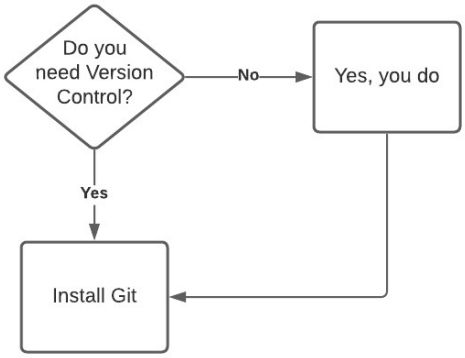

***Figure 1.2**: Version control importance*

Version control allows the developers the flexibility of making mistakes without worrying that they will have to start over the project/work. Basically, version control keeps track of all the changes at any time. If there is a need to undo any particular change, it can be done on the fly. Version control systems went through a series of evolutions with time and as project complexity grew.

Local Version Control Systems

The local version control system approach is very basic and simple, but it is also incredibly error prone. That is because it is extremely easy for the user to forget which directory they are in, and thus, they can mistakenly either write to the wrong file or copy over the entire files they do not mean to.

To avoid the above discussed issue, developers worked on the concept of local **Version Control Systems** (**VCSs**), which is a local database located on your local computer, in which every file change is stored as a patch. Every patch set contains only the changes made to the file since its last version.

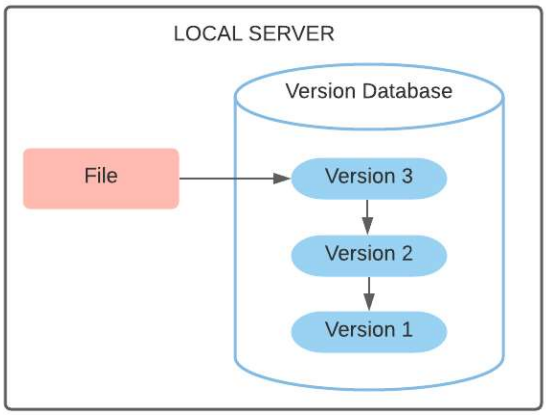

***Figure 1.3**: Local version control*

Centralized Version Control Systems

Centralized Version Management Systems (**CVCSs**) resolve the likely issues and challenges local version control systems face. The requirement to collaborate with developers on alternative systems became a recurring issue that developers and creators noticed over time. Systems (such as CVCS, Subversion, and Perforce) have a single server that contains all the versioned files. Various users use it to check out the files from a central location, so CVCS is still popular and is alternatively used instead of the local version control system.

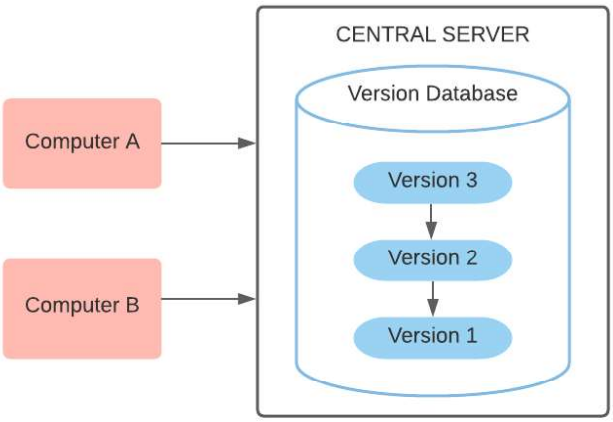

Figure 1.4: Centralized version control

Distributed Version Control Systems

Distributed Version Control Systems (**DVCSs**), such as Git, Mercurial, Bazaar, or Darcs, each clone of the repository is the full backup of the repository data. This, in turn, means that when the user takes the latest snapshot of the files, DVCS takes the full mirror back of the repository. This includes the complete history of the repository. It helps when the server hosting the repository crashes, and any of the users' repositories can be copied back up to the server to help restore the repository content onto the server.

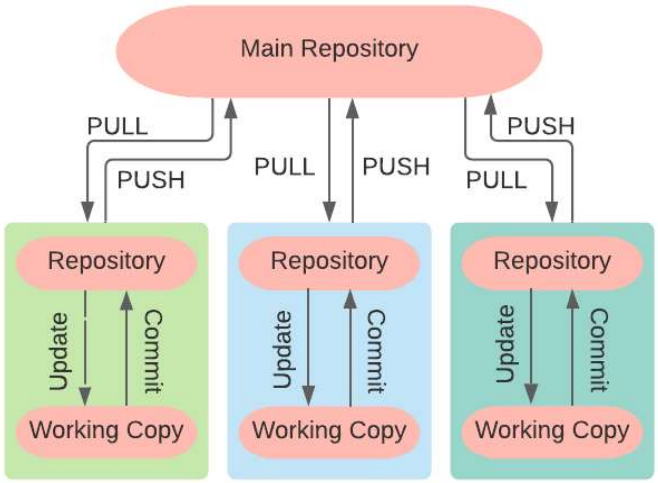

***Figure 1.5**: Distributed version control*

Git History

Like numerous extraordinary things in everyday life, Git started with a touch of innovative obliteration and blazing debate.

The Linux OS kernel is a free and open-source OS with extreme opportunities. For a large portion of the lifetime of Linux piece support (1991–2002), changes to the operating system were passed around as patches and archived files. In 2002, the Linux OS project started utilizing a restrictive DVCS called BitKeeper.

In 2005, the agreement between the DVCS system BitKeeper and the community that had worked on its Kernel got revoked, and thus BitKeeper being used as a free tool got renounced too. This resulted in the Linux community (particularly Linus Torvalds, the creator of Linux) working and developing their own tool based on the learnings when using BitKeeper. Linux community also prioritized the goals which they wanted in the new system. They are as follows:

Design that is simpler and easier to use:

Well-rounded support for non-linear development (that is, working on thousands of parallel branches)

Distributed Completely:

Should be able to handle large projects like the Linux kernel efficiently and with zero tolerance (speed and data size)

Git became self-hosted on April 7 with this commit:

```
commit e83c5163316f89bfbde7d9ab23ca2e25604af29
Author: Linus Torvalds <torvalds@ppc970.osdl.org>
Date:   Thu Apr 7 15:13:13 2005 -0700

Initial revision of "git", the information manager from hell
```

***Figure 1.6**: Git first commit*

Shortly thereafter, the first Linux commit was made:

```
commit 1da177e4c3f41524e886b7f1b8a0c1fc7321cac2
Author: Linus Torvalds <torvalds@ppc970.osdl.org>
Date:   Sat Apr 16 15:20:36 2005 -0700

Linux-2.6.12-rc2

Initial git repository build. I'm not bothering with the full history, even though we have it. We can create a
separate "historical" git archive of that later if we want to, and in the meantime it's about 3.2GB when
imported into git - space that would just make the early git days unnecessarily complicated, when we don't
have a lot of good infrastructure for it.

Let it rip!
```

***Figure 1.7**: Linux first commit*

From the time Git came into existence around 2005, it has evolved and matured into a tool that is easy to use and yet inherited and extends all the capabilities of DVCS. It also ticked all the initial use cases and principles it was built upon, which is why it is lightning-fast and very efficient for large projects. It also has an incredible branching system for non-linear development.

What is Git

This section is a key to understanding the underlying concept and principles upon which Git is built. If you follow this section keenly, you can learn how Git works fundamentally and use the concepts and knowledge to use Git effectively when you start using the same for your projects. As discussed previously in the chapter Git is a distributed version control system. Other version control systems are also available in the market, but Git functions differently and stores the information differently.

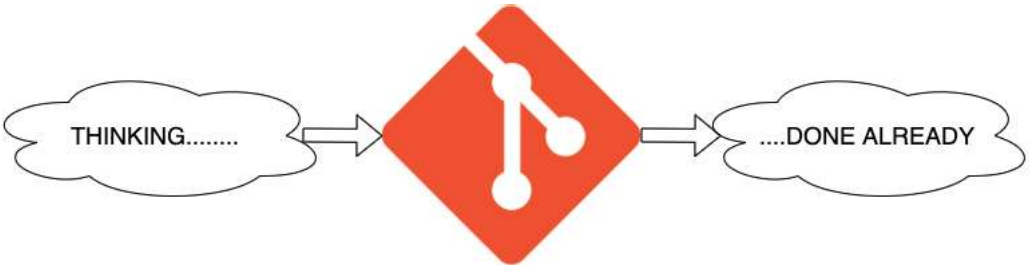

***Figure 1.8**: Git User experience*

A major difference between Git and any other **version control system** (**VCS**) is how they store information. Other VCS store data as a rundown of record-based changes. These different frameworks like (Central Version control system, Subversion, Perforce, Bazaar, and so on) think about the data they store like a bunch of records and the progressions made to each of the records over the long haul (this is more commonly represented as delta-based variant control).

On the other hand, Git considers its record information more like a series of smaller snapshots of the filesystem. With Git, each time users commit or save the state of the user's project, Git essentially snaps a photo of what every one of your records resembles at that point and stores a reference to that depiction. To be effective, if records have not changed, Git does not store them again. Simply a connection to the past indistinguishable records has already been effectively stored. Git considers its data to be more like a stream of snapshots.

Simply put, every time a change is made to the filesystem, Git just merges those changes with the already present ones instead of replacing or overriding the existing content.

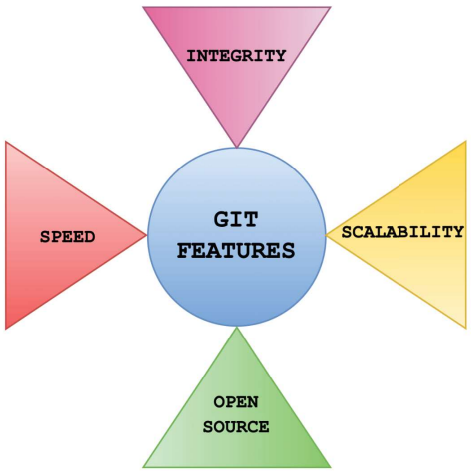

***Figure 1.9**: Git Features*

Git has a high level of performance and integrity:

Most Git operations are done locally and only need local files and objects. On the contrary, other VCSs have network latency overhead which gives Git a major performance boost as the entire history of the project files is stored over your local disk and is thus available instantaneously.

This also gives you the freedom to work remotely or without network connectivity as you can save your changes to your local copy. Once back online, you can push the required changes to the repo, whereas other VCSs do not have this flexibility.

When it comes to Integrity and Git knowing things, there is no way any file can be updated or modified without Git knowing it, and this is taken care of by Git's checksum in place. Everything in Git is check summed before it is stored and is then referred to by that checksum.

SHA-1 hash is used for the checksum by Git, and you will see these hash values all the time as, Git stores everything in its database not by file name but by the hash value of its contents.

Git three States

Git principally works based on three stages, as depicted in *Figure 1.10*, and the files stored under Git can either be in a **modified**, **staged,** or **committed** state:

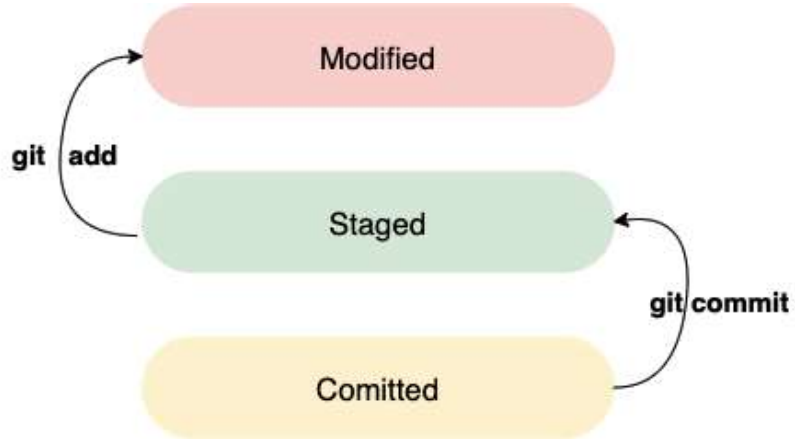

***Figure 1.10**: GIT 3 states*

Modified: It means that the user has changed/edited the file but has not made the changes to their GitHub repository database yet.

Staged: It means that the user has marked a modified file in its current version, which is supposed or will go into the user's next commit snapshot.

Committed: The users' GitHub repository data changes are safely stored in the users' local database.

This leads us to the three main sections of a Git project: the working tree, the staging area, and the Git directory.

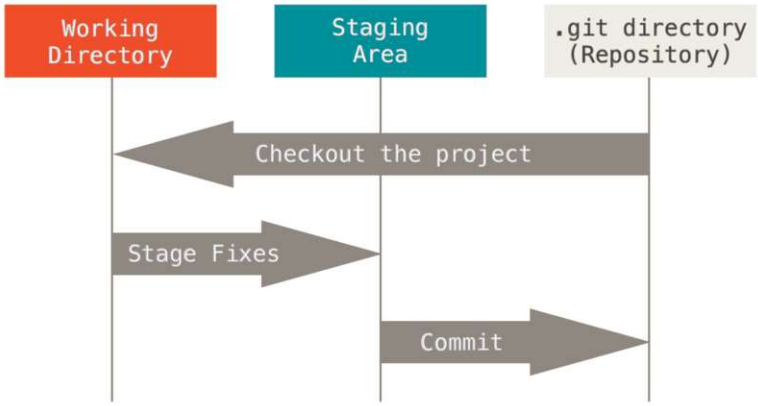

***Figure 1.11**: Working tree, staging area, and Git directory (source:* Git*)*

The Git working tree is a single checked-out version of the project, where the files are pulled out from a compressed database from the Git directory. And this is then placed over the user's disk for them to use, update and modify.

The staging area is a file, generally contained in the user's Git directory, which generally stores the information about what would go into the next commit. Technically and more precisely, it is called Git parlance, which is the "index," but the phrase "staging area" also works.

The Git repository directory is the place where Git stores the metadata and item information base for the user's project. This is the most important aspect of Git, duplicated when users clone a Git repository from a different computer.

The usual Git workflow is a process that goes as follows:

Users modify the files in their working tree. Users then selectively stage those changes they want to be part of their next commit, adding only those changes to their respective staging area. Users do a commit, which will take the files as they are in the staging area and then stores that particular snapshot permanently in their Git repository directory.

Getting started with Git

In this section of the chapter, we will go through some of the basics of Git, its installation and setup procedure on your work machine.