Git Essentials

Since its inception, Git has attracted skilled developers due to its robust, powerful, and reliable features. Its incredibly fast branching ability transformed a piece of code from a niche tool for former Linux Kernel developers into a mainstream distributed versioning system. Like most powerful tools, Git can be hard to approach since it has a lot of commands, subcommands, and options that easily confuse newcomers. The main purpose of Git Essentials is to help you overcome this fear and become adept at all basic tasks in Git. This book starts with an introduction to version control systems, and after this you’ll delve deeply into the essentials of Git. This serves as a primer for the topics to follow such as branching and merging, creating and managing a GitHub personal repository, and fork and pull requests. You will then learn the art of cherry-picking, taking only commits you like, followed by Git blame. Finally, you’ll learn how to migrate from SVN using Git tools or TortoiseGit and migrate from other VCSs, concluding with a collection of resources, links, and appendices.

Who this book is written for

If you are a software developer with little or no experience of versioning systems, or are familiar with other centralized versioning systems, then this book is for you. If you have some experience working with command lines or just using Unix and want to know more about Git, then this book is ideal for you.

What you will learn from this book

- Create, branch, and merge a local repository
- Transport your work to a remote repository working in a centralized manner
- Organize your source code using feature branches
- Organize the development flow
- Get a meaningful changelog using clear commit messages
- Collaborate with teammates without any impediments
- Experiment with your code without affecting functional code files
- Explore some tools used to migrate to Git from other versioning systems without losing your development history
- Extend your knowledge of Git with some useful resources


Forewords by Arialdo Martini, Solutions Architect at Aduno Gruppe
Nicola Paolucci, Developer Advocate at Atlassian
Dr. André Roberge, Former President and Vice-Chancellor of Université Sainte-Anne

$ 24.99 US
£ 16.99 UK
Prices do not include local sales tax or VAT

Ferdinando Santacroce

Free Sample
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Getting Started with Git'
- A synopsis of the book’s content
- More information on Git Essentials

About the Author

Ferdinando Santacroce is a developer, author, and teacher who loves learning new things.

As a software developer, Ferdinando mainly works on the .NET platform using C#, which bridges the gaps between oldstyle systems and new technologies. Over the span of his career, he has allowed COBOL applications to talk with remote services, databases, and electronic devices, such as cash handlers, scanners, and electronic shelf labels. At the moment, he's focusing on continuously improving the Agile Movement, which he follows with great care, in conjunction with XP foundations and lean manufacturing.

While working as a teacher, he developed the first web-based school register in a school of his district, which allowed parents to take a look at grades and absences on a regular basis.

Ferdinando loves to teach people, everyone from children to grandpas, and introduces them to the use of computers, the Internet, and new mobile devices.

He enjoys writing too; after a hiatus of a few years, he started blogging again about his work and passions, which, according to him, is more or less the same thing.

Most of what he's learned over the years has been with the help of his friends and colleagues. This has encouraged him to read books and attend inspiring conferences, such as XP Conference, Italian Agile Day, Codemotion, and others, that help his growth by working on the same code with him.
Git Essentials

If you are reading this book, you are probably a software developer and a professional. What makes a good professional? Sure, culture and experience are a part of the answer, but there's more: a good professional is one who can master different tools, choose the best tool for the job at hand, and has the necessary discipline to develop good working habits.

Version control is one of the base skills for developers and Git is one of the right tools for the job. However, Git can't be compared to a screwdriver, a simple tool with only a basic function; Git provides a complete toolbox that can help you manage your own code, within which there are also other sharp tools that should be handled with caution.

The final aim of this book is to help you to start using Git and its commands in the safest way possible, to get things done without causing any injuries. Having said this, you will not get the most from Git commands if you do not acquire the right habits; just as is the case with other tools, in the end, it is the craftsman who makes all the difference.

This book will cover all the basic topics of Git, thereby letting you start using it even if you have little or no experience with versioning systems; you only need to know about versioning in general and the Wikipedia-related page is enough for this purpose.

What This Book Covers

Chapter 1, Getting Started with Git shows you all the (simple) steps you need in order to install Git and do your first commit.

Chapter 2, Git Fundamentals – Working Locally reveals the essence of Git, how it takes care of your files, and how you can manage and organize your code.

Chapter 3, Git Fundamentals – Working Remotely moves your attention to the collaborating side of the tool, explaining the basic commands individually and the options you use when working with remote repositories.

Chapter 4, Git Fundamentals – Niche Concepts, Configurations, and Commands completes the basic set of Git commands you need to know, giving you some more weapons to use in difficult situations.
Chapter 5, Obtaining the Most – Good Commits and Workflows gives you some hints about common methods you can use to organize the source code within Git, thereby helping you to develop good habits that every developer should possess.

Chapter 6, Migrating to Git is a way to give you a hand if you're used to other versioning systems, such as Subversion, to manage the transitional phase from another piece of software to Git.

Chapter 7, Git Resources offers you some hints, which I've taken from my experience, and could prove interesting for you.
Getting Started with Git

Whether you are a professional or an amateur developer, you've likely heard about the concept of version control. You may know that adding a new feature, fixing broken ones, or stepping back to a previous condition is a daily routine.

This implies the use of a powerful tool that can help you take care of your work, allowing you to move around your project quickly and without friction.

There are many tools for this job on the market, both proprietary and open source. Usually, you will find Version Control Systems (VCS) and Distributed Version Control Systems (DVCS). Some examples of centralized tools are Concurrent Version System (CVS), Subversion (SVN), Team Foundation Server (TFS) and Perforce, while in DVCS, you can find Bazaar, Mercurial, and Git. The main difference between the two families is the constraint—in the centralized system—to have a remote server where get and put your files; if the network is down, you are in trouble. In DVCS instead, you can have or not a remote server (even more than one), but you can work offline, too. All your modifications are locally recorded so that you can sync them some other time. Today, Git is the DVCS that has gained public favor more than others, growing quickly from a niche tool to mainstream.

Git has rapidly grown as the de facto to version source code. It is the second famous child of Linus Torvalds, who, after creating the Linux kernel, forged this versioning software to keep trace of his millions lines of code.

In this first chapter, we will start at the very beginning, assuming that you do not have Git on your machine. This book is intended for developers who never used Git or used it a little bit, but who are scared to throw themselves headlong into it.

If you have never installed Git, this is your chapter. If you already have a working Git box, you can quickly read through it to check whether everything is right.
Installing Git

Git is open source software. If you are running a Windows machine, you can
download it for free from http://git-scm.com. At the time of writing this book,
the current version of Git is 1.9.5. If you are a Mac user, you can download it from
http://git-scm.com/downloads; here, you can find the *.dmg file, too. Finally,
if you are a Linux user, you probably have Git out of the box (if not, use the apt-
get install git command or equivalent). Here, I won’t go into too much detail
about the installation process itself; I will only provide a few recommendations for
Windows users shown in the following screenshot:

Enabling Windows Explorer integration is generally useful; you will benefit
from a convenient way to open a Git prompt in any folder by right-clicking on
the contextual menu.

Let Git be used in classic DOS command prompt, too, as shown in the following
screenshot:
Git is provided with an embedded Windows-compatible version of the famous Bash shell from Linux, which we will use extensively. By doing this, we will also make Git available to third-party applications, such as GUIs and so on. It will come in handy when we give some GUI tools a try.
Use defaults for line endings. This will protect you from future annoyances.

At the end of the process, we will have Git installed, and all its *nix friends will be ready to use it.

**Running our first Git command**

Now, we have to test our installation. Is Git ready to rock? Let's find out!

Open a prompt and simply type `git` (or the equivalent, `git --help`), as shown in the following screenshot:
If Git has been installed correctly, typing `git` without specifying anything else will result in a short help page, with a list of common commands. If not, try reinstalling Git, ensuring that you have checked the **Use Git from the Windows Command Prompt** option. Otherwise, Git will be available only within the embedded Bash shell.

So, we have Git up and running! Are you excited? Let's begin to type!

### Setting up a new repository

The first step is to set up a new repository (or repo, for short). A **repo** is a container for your entire project; every file or subfolder within it belongs to that repository, in a consistent manner. Physically, a repository is nothing other than a folder that contains a special `.git` folder, the folder where the magic happens.

Let's try to make our first repo. Choose a folder you like, and type the `git init` command, as shown here:

Whoa! What just happened? Git created a `.git` subfolder. The subfolder (normally hidden in Windows) contains some files and folders, as shown in the next screenshot:

At this point, it is not important for us to understand what is inside this folder. The only thing you have to know is that you do not have to touch it, ever! If you delete it or if you modify files inside by hand, you could get into trouble. Have I frightened you enough?
Now that we have a repo, we can start to put files inside it. Git can trace the history of any gender of files, text based or binary, small or large, with the same efficiency (more or less, large files are always a problem).

**Adding a file**

Let's create a text file just to give it a try.

And now what? Is that all? No! We have to tell Git to put this file in your repo, *explicitly*. Git doesn't do anything that you don't want it to. If you have some spare files or temp ones in your repo, Git will not be compatible with them, but will only remind you that there are some files in your repo that are not under version control (in the next chapter, we will see how to instruct Git to ignore them when necessary).

Ok, back to the topic. I want *MyFile.txt* under the control of Git, so let's add it, as shown here:

The `git add` command tells Git that we want it to take care of that file and check it for future modifications.

Has Git obeyed us? Let's see.
Using the `git status` command, we can check the status of the repo, as shown in the following screenshot:

As we can see, Git has accomplished its work as expected. In this image, we can read words such as branch, master, commit and unstage. We will look at them briefly, but for the moment, let's ignore them.

**Commit the added file**

At this point, Git knows about `MyFile.txt`, but we have to perform another step to fix the snapshot of its content. We have to commit it using the appropriate `git commit` command. This time, we will add some flavor to our command, using the `--message` (or `-m`) subcommand, as shown here:

Press the Enter key.
Getting Started with Git

Feel the magic—a new branch is born!

With the commit of MyFile.txt, we have finally fired up our repo. It now has a master branch with a file within it. We will play with branches in the forthcoming chapters. Right now, think of it as a course of our repository, and keep in mind that a repository can have multiple courses that often cross each other.

Modify a committed file

Now, we can try to make some modifications to the file and see how to deal with it in the following screenshot:

As you can see, Bash shell warns us that there are some modifications painting the name of the modified files in red. Here, the git status command informs us that there is a file with some modifications and that we need to commit it if we want to save this modification step in the repository history.

However, what does no changes added to commit mean? It is simple. Git makes you take a second look at what you want to include in the next commit. If you have touched two files but you want to commit only one, you can add only that one.

If you try to do a commit without skipping the add step, nothing will happen. We will see this behavior in depth in the next chapter.
So, let's add the file again for the purpose of getting things ready for the next commit.

Let's do another commit, this time, avoiding the \texttt{--message} subcommand. So, type \texttt{git commit} and hit the \textit{Enter} key.

Fasten your seatbelts! You are now entering into a piece of code history!
What is that? It's Vim (Vi Improved), an ancient and powerful text editor. You can configure Git to use your own preferred editor, but if you don't do it, this is what you have to deal with. Vim is powerful, but for newcomers, it can be a pain to use. It has a strange way of dealing with text, so to start typing, you have to press *i* for inserting text, as shown in the following screenshot:

Once you have typed your commit message, you can press *Esc* to get out of the editing mode. Then, you can type the `:w` command to write changes and the `:q` command to quit. You can also type the command in pairs as `:wq`.

After that, press *Enter* and another commit is done, as shown here:
Note that when you saved the commit message in Vim, Git automatically dispatches the commit work, as you can see in the preceding screenshot.

Well done! Now, it's time to recap.

**Summary**

In this chapter, you learned that Git is not so difficult to install, even on a non-Unix platform such as Windows.

Once you have chosen a directory to include in a Git repository, you can see that initializing a new Git repository is as easy as executing a `git init` command, and nothing more. Don't worry now about saving it on a remote server and so on. It's not mandatory to save it; you can do this when you need to, preserving the entire history of your repo. This is a killer feature of Git and DVCS in general. You can comfortably work offline and push your work to a remote location when the network is available, without hassle.

In the end, we discovered one of the most important character traits of Git: it will do nothing if you don't mention it explicitly. You also learned a little bit about the `add` command. We were obliged to perform a `git add` command for a file when we committed it to Git the very first time. Then, we used another command when we modified it. This is because if you modify a file, Git does not expect that you want it to be automatically added to the next commit (and it's right, I'll say).

In the next chapter, we will discover some fundamentals of Git.
Where to buy this book
You can buy Git Essentials from the Packt Publishing website. Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers. Click here for ordering and shipping details.