Introduction to Git and GitHub

Introduction to Git and GitHub

Q. What we will learn here?
A. Firstly, we will learn here what is version control and why git is so popular version control tool.
The course will show you how to configure Git on your local computer.
You will also be introduced to GitHub.
This course will teach you how to build remote repositories and local repositories in addition to familiarising you with GitHub.
You should have a firm grasp on several key Git concept by the end of this course, including:

1. Commits
2. Branches
3. Merging
4. Rebasing

Q. What is Version Control?
A. A version control system (VCS) is a collection of resources for keeping track of the changes made to a series of files. This means you can instruct your VCS (in our case, Git) to save the current state of your files at any time. After that, you will continue to edit the files and save their current state. Creating a backup copy of your working directory is equivalent to saving the state. Making a commit is how we relate to this saving of state in Git.

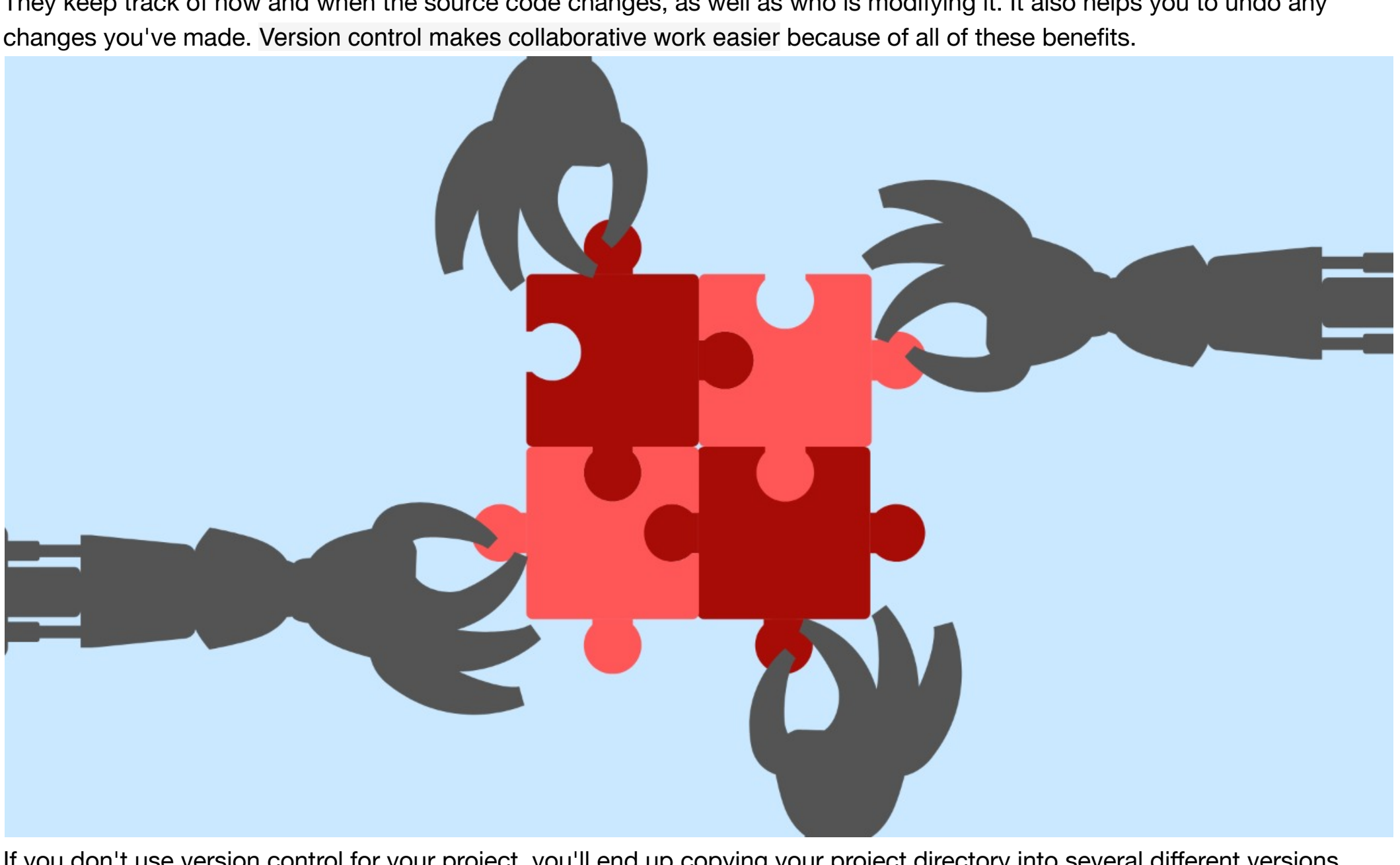
When you commit to Git, you add a commit message that describes the improvements you made in this commit at a high level. Git will show the history of all commits as well as the commit messages. This provides a valuable record of your work and can be extremely useful in determining when a bug entered the system.

Git allows you to compare files between commits in addition to showing you the log of changes you've made. As I previously stated, Git will allow you to easily return any file (or all files) to a previous commit.

Q. Why do we need Version Control?
A. Let's say you've been working on a project for over a month. You remember when introducing a new feature that something you did a week ago would prevent your new feature from working properly. You decide to undo those changes, but you're not sure whether similar changes were made in other files.

What if there was a way to see exactly what improvements you made and where they were made in your code? That's where version control comes in handy.

1. **To keep track of changes to the codebase:-** In its most basic form, version control can be thought of as a log of all the changes made to your software code. Version control systems allow you to monitor how the project evolves over time, as well as provide details about why and when each change was made.
2. **Ease in collaborative work:-** Many issues that you will encounter as a developer can be solved with version control systems. They keep track of how and when the source code changes, as well as who is modifying it. It also helps you to undo any changes you've made. Version control makes collaborative work easier because of all of these benefits.

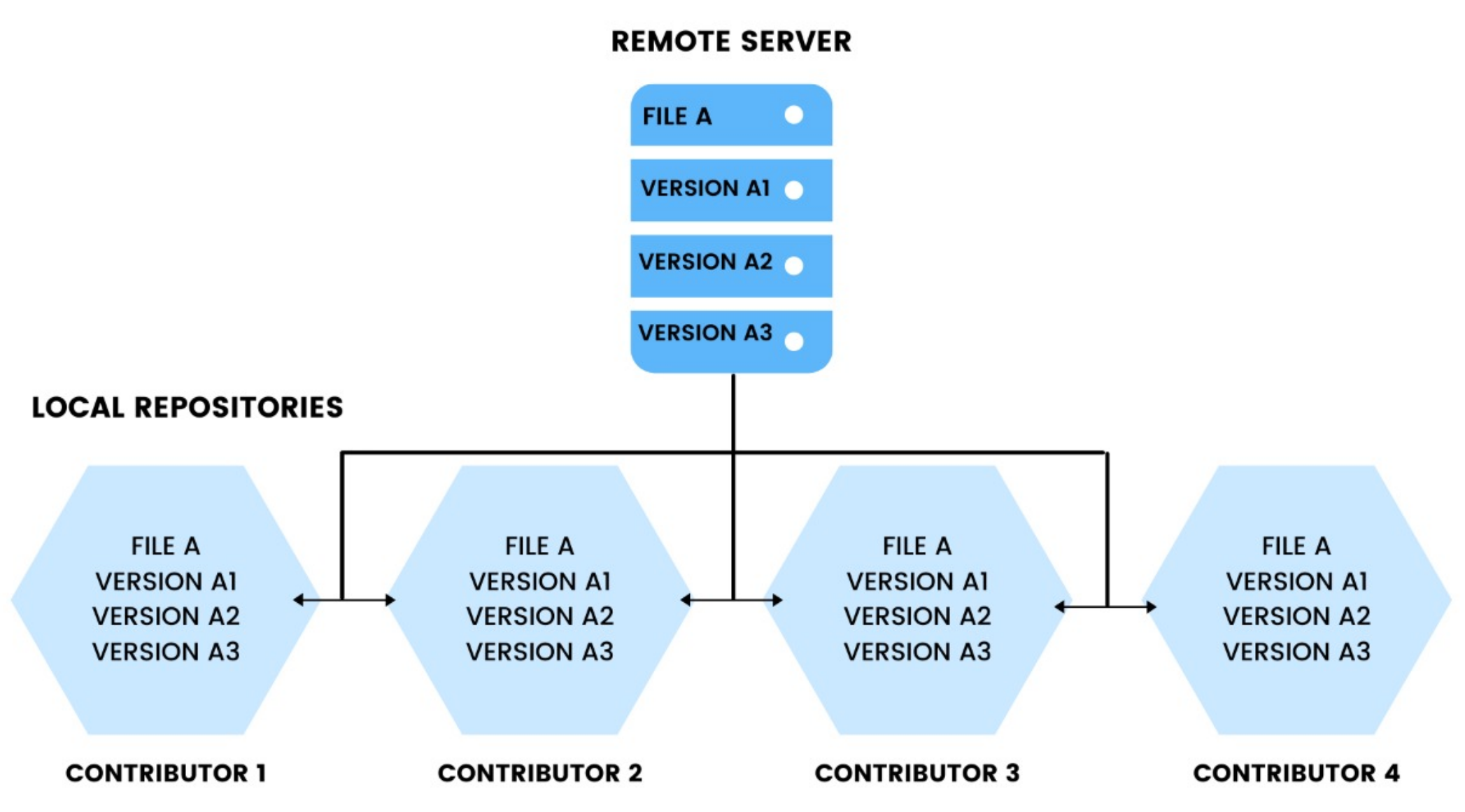


If you don't use version control for your project, you'll end up copying your project directory into several different versions sooner or later. You might call them 'final_version,' 'latest_version,' or something along those lines, depending on the priority you choose to give. However, it is not a long-term strategy, particularly when working in a group. This method is basically formalised by version control systems, so you don't have to do it manually.
3. **Version control tools:-** Version management systems have been modified on a regular basis to keep up with the ever-changing workflows that developers use. Git has recently risen to prominence as the preferred version control tool.

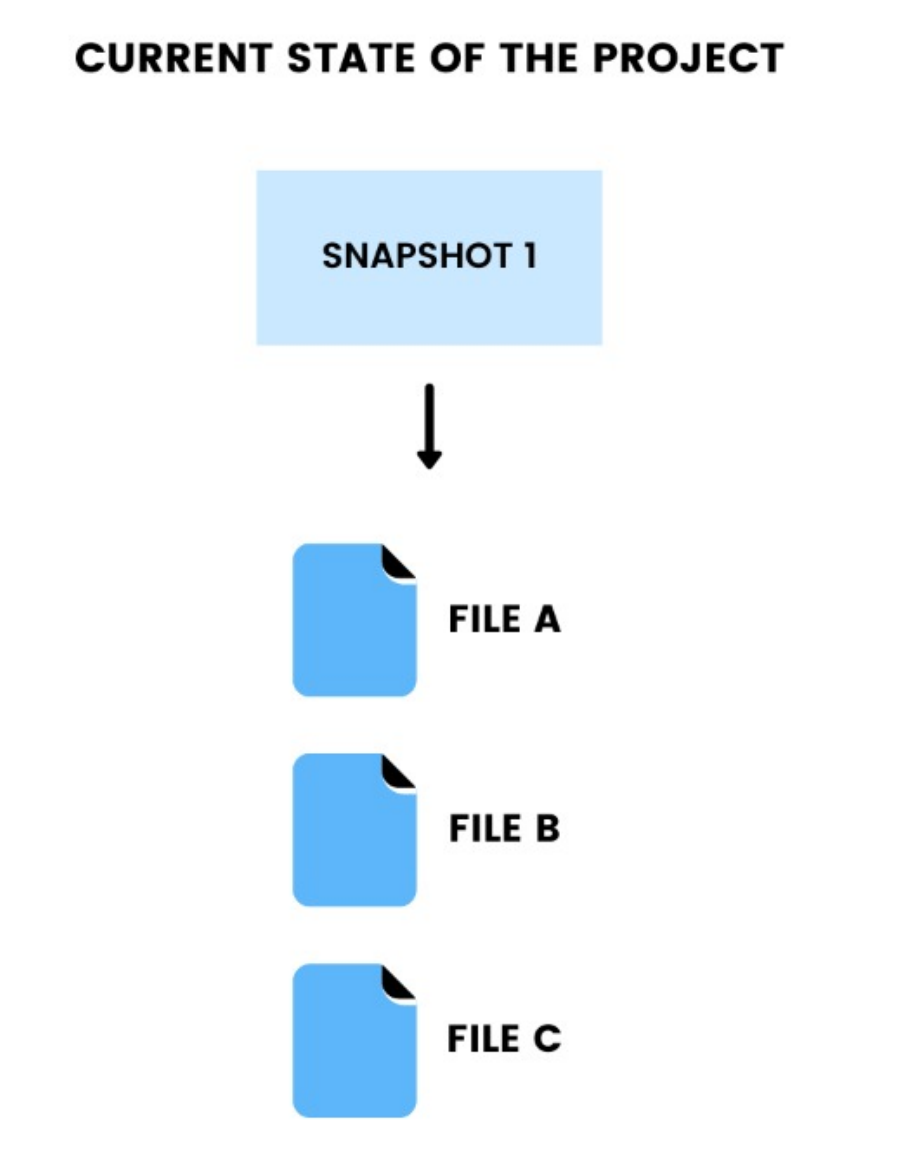
Q. What is Git?
A. Git is a version control system that is distributed. As a result, any contributor's machine would have a full copy of the entire codebase; this codebase is also known as a local repository.

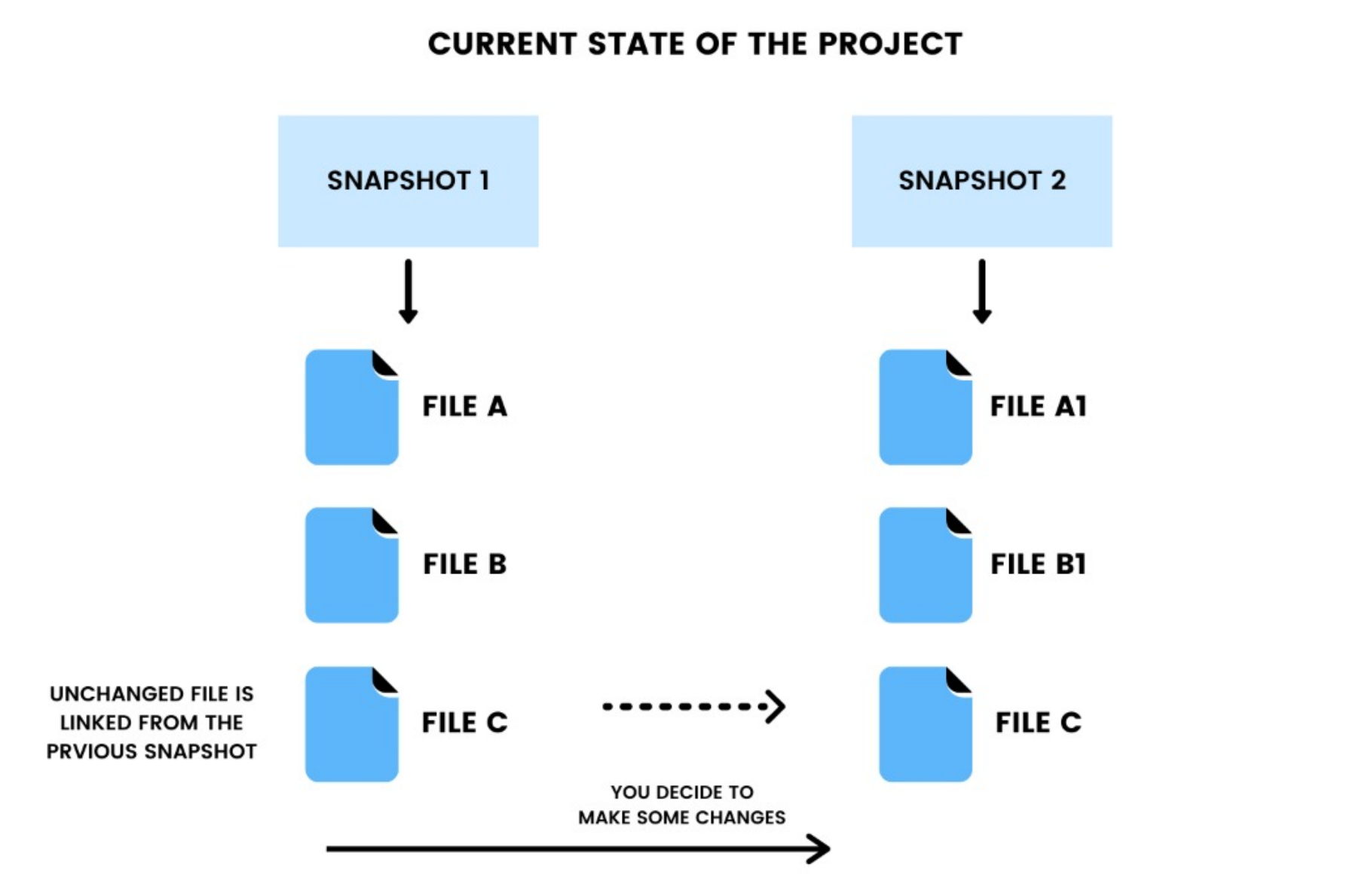
Git keeps track of the changes made to the local repository and keeps a record of them. It eliminates the hassle of maintaining different versions of the project in separate folders and allows collaborators to share changes to the source code in a seamless and fast manner.

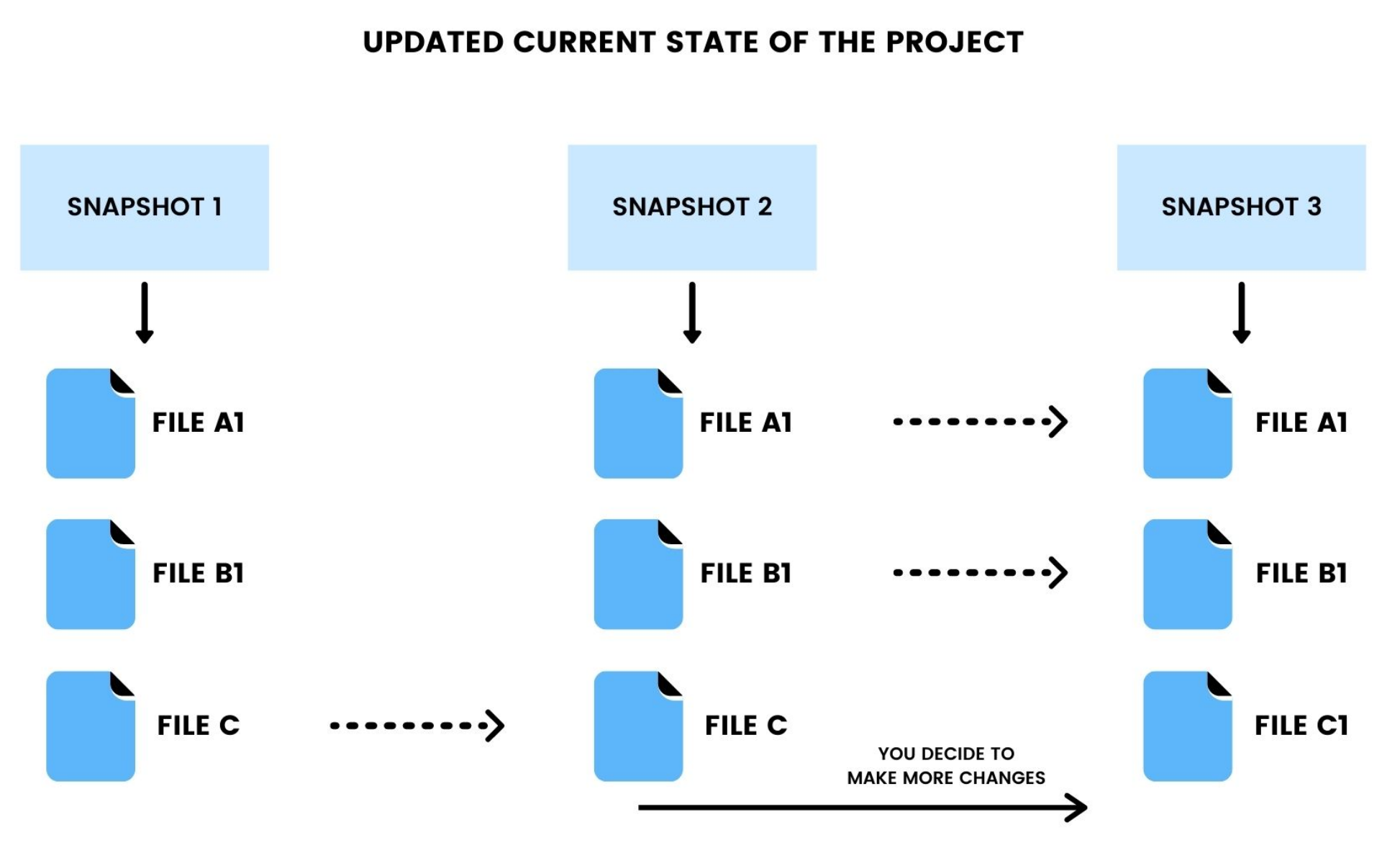
The distributed nature of Git makes it unique and efficient. Since every contributor or collaborator has their own '**clone**' of the source code in their **local repository,** it allows Git to track, add, or revert changes very quickly. Git uses the local database to keep a record of the changes to the project files.



Q. What is Git snapshot?
A. A Git snapshot is the state of the project files at a particular point in time. In other words, rather than keeping track of file-based changes over time, Git can save the information as a snapshot, which represents the current state of the project. To prevent duplicates, it will only hold references to files that haven't changed since the previous snapshot while saving the snapshot.







How a snapshot keeps track of new changes.

Q. What are different states we have in git?
A. Git has three main states in which the project's files can be acquired. The three states are as follows:

Modified:- files that have been modified but have not yet been labelled as such by Git. These modifications will not be included in the next screenshot.
Staged:- Git has registered the changes and will mark them as such in the next snapshot.
Committed:- Modified files that have successfully become part of the most recent snapshot are referred to as committed.

Local System

Working Directory	Staging/Indexing Area	Local Repository(.git)
• Here we keep the files and make changes to the file. • Files in this stage are in a modified state and are not tracked by git. • Here we use all OS-related operations for creating a file or folder and it is similar to doing general operation in windows or mac OS using UI.	• Staging or index area is an intermediate stage in git. • Files in this stage are tracked by git. • We use the "git add " command to move the files from the modified stage to the indexing area. • This stage is only available in git but not available in other VCS.	• It is the last state where changes are committed. • A normal working directory can be made to work as a local repository when we initiate that repository as a git repo using the "**git init**" command. • "**.git**" folder works as a local repository. • We use the "**git commit**" command to commit to the local repo. • git commit is nothing but saving the changes to the local repo. • It is the last stage from where the changes are pushed to cloud repo using the "**git push**" command.

Git States

Q. Discuss few basic linux commands.
A.

```
$ cat file1 file2 …
It prints the contents of file1, file2. The program is called cat because it performs concatenation when it
prints the contents of more than one file.
```

```
$ ls -l
The ls command displays a directory's contents. The current directory is the default, but you can specify any
directory or file as an argument, and there are numerous useful choices. Use ls -l for a comprehensive (long)
listing and ls -F for file type details, for example. The owner of the file (column 3), the group (column 4), the
file size (column 5) and the modification date/time (between column 5 and the filename) are all included in this
example of long listing. Ignore column 1 and 2, we will discuss about them later.


$ ls -l
total 272
-rw-r--r--  1 sysuser  staff  25956 May  1 14:58 Class_01_Git_GitHub_intro.ipynb
-rw-r--r--  1 sysuser  staff  13843 May  8 12:39 Class_01_Python_Introduction.ipynb
-rw-r--r--  1 sysuser  staff  65484 May  8 15:57 Class_02_Python_Introduction-II.ipynb
-rw-r--r--  1 sysuser  staff  19395 May  9 15:03 Class_03_List_Tuple.ipynb
-rw-r--r--  1 sysuser  staff     59 May  2 13:45 f1.py
-rw-r--r--@ 1 sysuser  staff    112 May  6 20:42 first_file.py


$ ls -l file
It gives same information as above but for a specific file
```

```
Standard Input and Standard Output
I/O streams are used by Unix processes to read and write data. Data is read from input streams and written
to output streams in the processes. Streams are extremely adaptable. A file, a computer, a terminal window, or
even the output stream from another process may be the source of an input stream.

Enter cat (no arguments) and press ENTER to see an input stream at work. Since cat is still working, you won't
get any immediate output, and you won't get your shell prompt back. Now type whatever you want, and at the end
of each line, press ENTER. The cat command, when used in this manner, will repeat any line you type. Click
CTRL-D on an empty line to return to the shell prompt once you are done.

Streams are the reason cat adopts an interactive activity here. When you don't define an input filename, cat
uses the Linux kernel's standard input stream rather than a stream attached to a file. The standard input is
connected to the terminal where cat is running in this case.

The standard output is similar. Each process is given a standard output stream to which it can write its
output by the kernel. The output of the cat command is always written to the standard output. The standard
output was connected to that terminal when you ran cat, so that's where you saw the output.

stdin and stdout are common abbreviations for standard input and output.

[Book reference:- How Linux Works by Brian Ward]
```

```
$ cp file1 file2
Above command is used to copy file1 to file2.


$ cp file dir
This command copies a file to another directory, keeping the same file name in that directory.


$ cp file1 file2 file3 dir
This command copies three files(file1, file2, file3) to a folder called dir.
```

```
$ mv file1 file2
The mv (move) command is similar to the cp command. It renames a file in its most basic form. For example, To
rename file1 to file2, use above command.
In the same way that cp can transfer files to different folders, mv can do the same.
```

```
$ touch file
The touch command has the ability to generate a file. Touch does not modify the target file if it already exists,
but it does update the file's modification timestamp. Above command create an empty file.
```

```
$ rm file
A file is deleted (removed) with the rm command. When you delete a file, it is usually permanently deleted from
your device and cannot be recovered unless you retrieve it from a backup.
```

```
$ echo Hello World
The echo command prints its arguments to the standard output.
```

```
# Directory Mechanics
1. The root directory, also known as /, is at the top of the Unix directory hierarchy.
2. The forward-slash (/) is used to separate directories, not the backslash (\).
3. The root directory contains some regular subdirectories, such as /usr.
4. You define a path or pathname when referring to a file or directory.
5. When a path begins with / (for example, /usr/lib), it is referred to as a complete or absolute path.
6. The parent of a directory is indicated by a path component with two dots (..).
7. If you're working in /usr/lib, for example, the route (..) will refer to /usr. (..)/bin, on the other hand,
refers to /usr/bin.
8. The current directory is indicated by a single dot (.). For example, if you're in /usr/lib, the route (.) is
still /usr/lib, and (.)/X11 is /usr/lib/X11.
9. Since most commands default to the current directory if a path does not begin with /, you won't need to use
(.) quite much (in the preceding example, you could just use X11 instead of (.)/X11).
10. A relative route is one that does not start with /.
11. You'll be working with relative pathnames most of the time since you'll be in or near the directory you need.
```

```
$ pwd
The pwd (print working directory) program simply outputs the name of the current working directory.

$ pwd -P
The symbolic links can sometimes obscure the true full path of the current working directory. Use above command
to eliminate this confusion.
```

```
$ diff file1 file2
To see the differences between two text files, use diff.

$ diff -u file1 file2
When sending output to someone else, most programmers prefer the output from above command because automated
tools have an easier time with this format.
```

```
$ cd dir
The cd command changes the shell's current working directory to dir. The current working directory is the
directory that a process (such as the shell) is currently in.

$ cd
It brings you to your home directory.
```

```
$ mkdir dir
This command is used to create a new directory dir.
```

```
$ rmdir dir
It removes the directory with name dir. This command fails if directory is not empty.

$ rm -r dir
It removes the directory with name dir recursively. Be careful while using this.
```

```
Vim is default editor for linux based system. We use this editor to edit a file.

$ vim file
This command opens file using vim editor. Few commands to use with vim editor:-
    i -> to get into insert mode.
    esc -> to exit insert mode.
    :wq -> to save and exit the file.
```