## Unit-1

Software Development Methodologies: High-level overview, Waterfall, Agile, Scrum, Kanban, Extreme Programming. Version Control with GIT: Version Control concept, Centralized vs. distributed version control systems, Download, install and configure GIT, GitHub, SSH-KEY, GIT internals, undoing changes (Git Revert & Git Reset), Branching and merge, Tags, Stash, Remotes, Branching strategies, Practical tasks.

## Software Development Methodologies

The goal of most software development companies and their clients is software production at the lowest cost, with the best quality, in the shortest time. Proper planning and management of the development process with the right methodology is important to achieve such a goal.

There are lots of software development methodologies that propose different ways to achieve the desired result at a fair cost and short delivery time. These methodologies set up the framework that structures planning, controlling, and developing information systems.



**What Is a Good Software Development Methodology?**

While the term might be self-explanatory, it’s still a good idea for us to get on the same page about the meaning of software engineering methodologies and what might constitute a good one for your particular undertaking.

In short, an IT development method is a series of steps needed to produce a certain piece of software. Typically, it includes research, planning, design, and development phases that all constitute the life cycle of the software you’re working on. However, depending on the chosen approach, the way of carrying out these stages might differ.

So, what makes the most suitable methodology? Unsurprisingly, it all depends on your unique business and [industry needs](https://www.velvetech.com/industry-expertise/), capabilities, and end goals. If you base your choice of the development process on these three elements, you’ll be more likely to pick the one that will deliver the best results.

## Software Development Methodologies: a Comparison of Pros & Cons

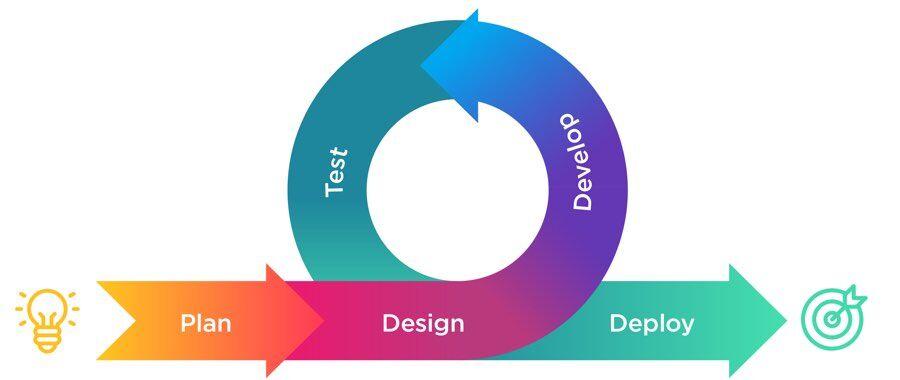
What drives the choice of a software development methodology? This choice is always relative to the requirements of a project. In addition to this, project type and size, the skills of team members, financial resources and preferences are also valuable considerations.

Let’s take a look at our top list of methodologies and see why [software development companies](https://www.velvetech.com/) opt for these software development models.

1

### Agile Methodology

Agile development is one of the most popular approaches in the IT world today. Besides that, numerous software production methodologies are based on agile principles. However, they slightly differ. Keep pace, some of them we will describe in this article.



By itself, the [agile software development](https://www.velvetech.com/blog/agile-software-development/) approach is mainly focused on a finished product with collaborative efforts. It involves short-term software development cycles called iterations. Each iteration is well organized and similar to a mini software project usually lasting from one to four weeks.

The iterations contain tasks such as adding new functionalities, analyzing and planning, designing, coding, testing, and documenting. As the number of iterations increases, the development team reassesses the project and prioritizes the backlog. It is a good practice that helps to ensure the product release and have a better version at the end of every iteration. However, the agile model is not a panacea and its advantages go along with disadvantages.

Benefits of Agile

* The agile methodology delivers a high-quality output because small iterations involve easy test and maintenance with fewer errors.
* It allows for creative improvements and modifications while working on a software product. Developers get to explore various modifications with the code.
* The approach of agile methodology is adaptive with low dependence on initial documentation. Applied changes don’t disrupt or have setbacks on the project.
* [Software costing](https://www.velvetech.com/blog/custom-software-development-cost/) and budgets are well planned. The agile approach requires project estimation before each iteration.
* Due to the focus on clarity, there are regular interactions and communication amidst the client, developers and members of the production process leading to a good working relationship.

Considerations in Agile

* Lack of initial clarity and project vision due to inconsistency in product specifications.
* The challenge of estimating the resources required for a project. Unpredictable changes make it hard to estimate the costs and resources.
* Lacks efficiency in documentation.
* No set or strict deadlines. Changes in specifications and requirements will mean no accurate estimate deadlines for project completion.

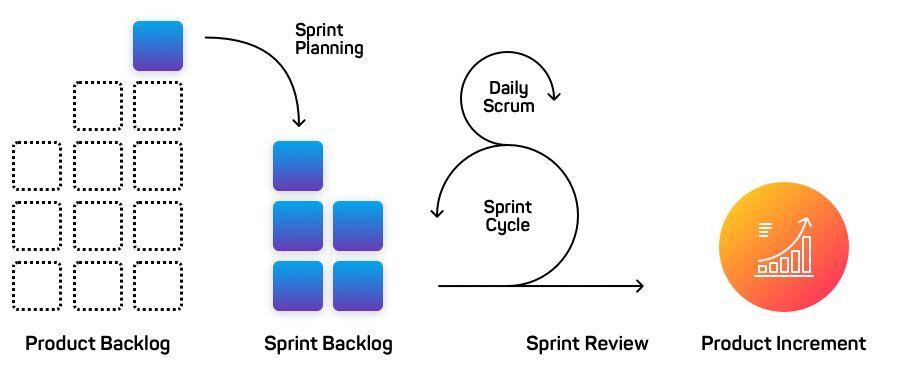
Agile Methodology Steps

* Development iteration
* QA iteration
* Retrospective
* Scope adjustment

2

### Scrum Methodology

Scrum is the most famous framework that is based on Agile methodology. It is empirical in nature and can be applied to any fast-changing or priority emerging requirements.



In scrum, iterations are named sprints. As a software development process, it kicks off with brief planning for each sprint, followed by daily scrum meetings that highlight the progress, and ends with a final review.

Scrum methodology is ideal for managing projects with not well-defined requirements and feedback from the client. Teamwork, transparency, and regular status updates speed up the development.

Benefits of Scrum

* The team makes the principal project decisions.
* The daily meetings promote the measure of individual productivity leading to the improvement in efforts of every team member.
* Scrum methodology discovers problems fast, resulting in short meetings and easy focus by the team.
* Prioritizing customer-driven features with Scrum is flexible. Business necessities documentation is not compulsory for successful development.
* Clients are in the production cycle as there is always something to assess after every sprint.
* The Feedback cycle is fast, helping the project to stay focused.

Considerations in Scrum

* It is not an effective method for junior or middle-skill team members.
* The estimation of time and cost demands high accuracy for a project to be successful.
* This methodology is less effective for large project types.

Scrum Methodology Steps

* Sprint planning
* Sprint execution
* Sprint review meetings
* Retrospective
* Release

3

### Feature Driven Development Methodology

This is another iterative methodology but in combination with object modeling. It is effective for large team projects. The first step in this method is reviewing the scope of the system. After the review, next is the creation of domain models in detail for every feature, and then the scope of the system is reviewed all over again.



The merging of all the domain models leads to an overall single model. Software development with this method involves five steps which are: developing the complete model, making the list of features, planning, designing, and building by feature.

Benefits of FDD

* This methodology is suitable for big projects with a guarantee of success.
* Its five-step process of development helps in accelerating software delivery with ease.
* FDD supports multiple teams working in parallel.
* The standards of this method rank with the best industry practices.
* Feature-driven development covers for all projects that need sequential updates.
* The resulting features are always greater than the inputs.

Considerations in FDD

* It provides almost no documentation for project owners.
* Based on the efforts required, it’s not suitable for small projects.
* It’s too much a complex pattern of development for individual software developers.
* The success of this methodology solely depends on the Team Lead and his skills.

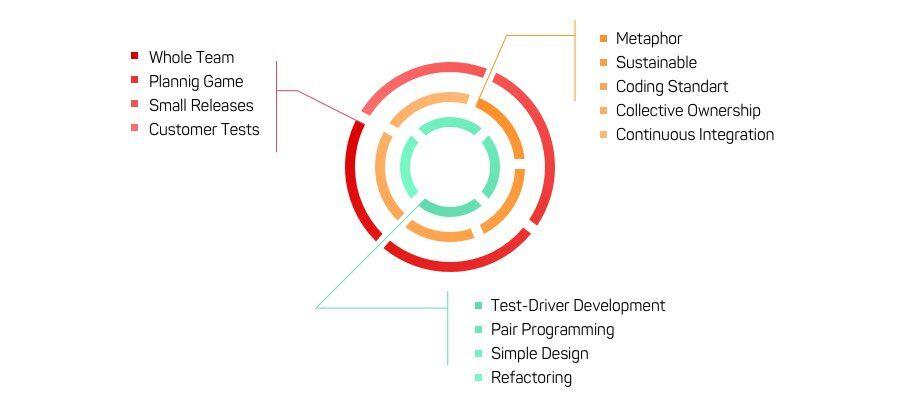
FDD Methodology Steps

* Development of an overall project model
* List of features
* Feature-focused iterations with milestones
* Design by feature
* Build by feature

4

### Extreme Programming Methodology

Extreme Programming or XP also refers to agile methodology. The main aim of this model is to create a fully-functional product and cut the cost of software non-essentialities. It is a perfect fit for complex projects with fixed deadlines and not clearly determined requirements. Continuous planning and testing are one of the core principles here.



Extreme programming is most suitable for creating software in an unstable environment. This method gives the provision for developers to deliver a final product at a lower cost. At the same time, it is time-consuming with a lot of human effort due to the frequent meetings, test-driven approach, and pair programming.

Benefits of Extreme Programming

* It is a method cost-effective for software development.
* Customer’s involvement and interaction are parts of the production process.
* There is a focus on practical planning and task scheduling. This makes developers stay committed to a project.
* Works well for small and large teams.
* Effective risk management increases the chances of success.

Considerations in Extreme Programming

* A practical quote for this method is almost impossible because of undetermined and changing project requirements.
* It requires frequent meetings and reviews between all stakeholders leading to expenses and time consumption.
* It involves too many changes in code which are tedious for some developers.
* Changing initial requirements at a later stage with this model has a high cost.

Extreme Programming Steps

* Project requirements
* Iteration planning
* Stories
* Test cases
* Development tasks
* Acceptance tests

5

### Waterfall Methodology

With its [introduction](http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf) in 1970 by Dr. Winston W. Royce, the waterfall is the most traditional methodology in the IT industry. It is a classic approach and a very popular version of the system development life cycle in software engineering. The goals are pre-defined for each development phase.



It is linear in nature with all projects progressing in stages. The completion of one stage precedes the other. This methodology is rigid because once a stage is complete, no reversible changes can be made based on new requirements.

Benefits of Waterfall Methodology

* Straightforward with no complexity in use. It requires little or no experience.
* Testing is simple because it’s based on the use-cases defined in the technical specification.
* Time-saving. In this methodology, the stages are processed and completed one at a time.
* It is an effective method for small projects in cases where the requirements are well defined.
* There is easy management due to the rigid nature of the model with each stage having a separate review process.
* There is a fixed deadline for each of the development stages.

Considerations in Waterfall Methodology

* Maintenance type of projects does not apply to this method.
* The software under production only becomes functional at the last stage of the cycle.
* It is best used with only well-defined requirements available up-front.
* There can be no edits or changes once a project advances to the testing stage.
* It’s an ideal method for small and medium-size projects but not long-term or research and development projects.
* Not applicable for projects that tend to have modifications along the way.

Waterfall Methodology Steps

* Requirements
* System design
* Development
* QA
* Deployment & maintenance

6

### Spiral Methodology

The Spiral lifecycle model seems highly sophisticated. It functions by the early identification and reduction of risks in a project. Beginning on a small scale, it encompasses risks exploration and the provision of plans to eliminate such risks.



It then indicates if the next level of spiral iteration can begin. In contrast to the waterfall model, the spiral method enables the developer to make changes in the code or design even in the testing stage. It is suitable for most kinds of projects but requires good management.

Benefits of Spiral Methodology

* It involves extensive risk analysis leading to very minimal risks.
* This method is effective for developing high-risk and large projects in general.
* Features and functions can still be added even at the late stages of testing. The Spiral model revolves all the phases repeatedly, thereby enabling changes.

Considerations in Spiral Methodology

* It is a waste of resources for projects with low-risk factors.
* The success of the approach is dependent on risk analysis. If the risk analysis is faulty, then the results can be flawed.
* There is a risk factor of the development being inconclusive, resulting in a spiral manner without a conclusion.

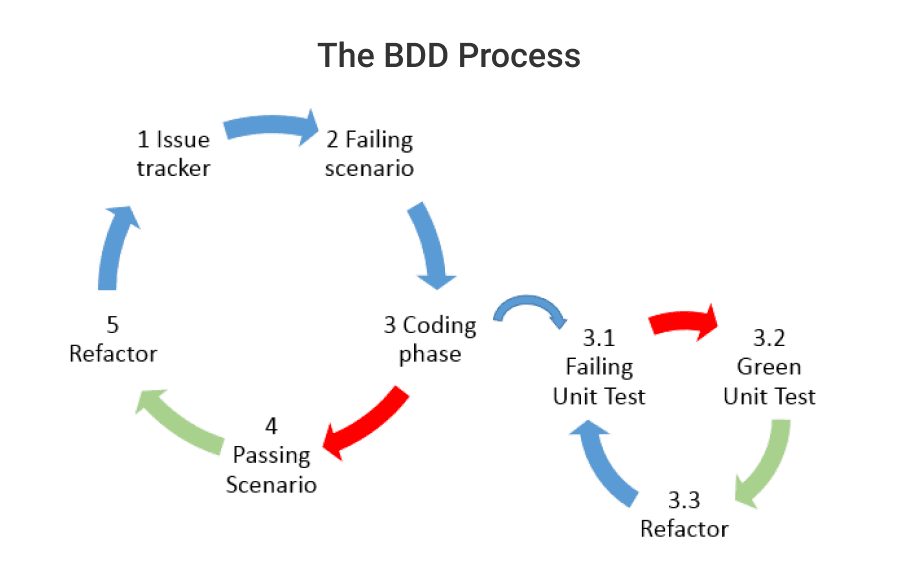
Spiral Methodology Steps

* Analysis
* Evaluation
* Development
* Planning

7

### Behavior Driven Development (BDD)

Behavior Driven Development is a variation of an agile methodology that formalizes a shared vision between all participants of how an app should behave. It aims to enable non-technical people to take an active role in the implementation of technical functionality.



The approach is based on evolved Test-Driven Development (TDD). Essentially, it drives the execution of only those system behaviors that are crucial for business outcomes. On top of that, it calls for ongoing conversation and the use of concrete examples to minimize resource waste.

Benefits of BDD

* The method provides enhanced opportunities for collaboration between developers, QA specialists, and domain experts to deliver desired outcomes.
* BDD automates the generation of technical and end-user documentation based on specifications.

Considerations in BDD

* The approach implies a good understanding of TDD methodology, thus it suits better for experienced developers.
* Creating scenarios requires a lot of effort and time. Consequently, it’s not a good option for short-term projects.

BDD Methodology Steps

* Identifying the scenario for the feature
* Writing a failing feature test
* Passing the test
* Refactoring the code if needed

### Differences between Agile and Waterfall

Waterfall is different from Agile in the same way as it is from Lean because as we’ve established, there’s no real difference between Agile and Lean. Just like with Lean, Waterfall differs from Agile in interim steps and the pace of the project.

#### Agile vs. Lean vs. Waterfall

|  | Agile | Lean | Waterfall |
| --- | --- | --- | --- |
| **Focus on** | Delivering the product as fast as possible | Delivering the product that the market buys | Staying within budget and time frames |
| **Development Process** | Phases are following in cycle | Phases are following in cycle | Linear: one phase after another |
| **Development Pace** | Iterations of 2 to 4 weeks | Extra short iterations | Development with no interim customer / market feedback |
| **Customer Involvement** | Throughout the project | Throughout the project | At milestones |
| **Scope** | Praises new changes that improve the product-market fit | Keeps changing by removing waste: everything market is not ready to pay for | Detailed from start |
| **Team** | Smaller teams with tight communication are preferable | Smaller teams with tight communication are preferable | Hierarchical organization with clearly defined roles; not much communication |

## History of the Agile Manifesto :

The [Agile Manifesto](http://www.agilemanifesto.org/) and the [Twelve Principles of Agile Software](http://agilemanifesto.org/principles.html) were the consequences of industry frustration in the 1990s. The enormous time lag between business requirements (the applications and features customers were requesting) and the delivery of technology that answered those needs, led to the cancelling of many projects. Business, requirements, and customer requisites changed during this lag time, and the final product did not meet the then current needs. The software development models of the day, led by the [Waterfall model](https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban), were not meeting the demand for speed and did not take advantage of just how quickly software could be altered.

In 2000, a group of seventeen “thought leaders,” including [Jon Kern](https://agilemanifesto.org/authors.html), [Kent Beck](https://en.wikipedia.org/wiki/Kent_Beck), [Ward Cunningham](https://en.wikipedia.org/wiki/Ward_Cunningham), [Arie van Bennekum](http://www.acm-software.com/en/trainer/arie-van-bennekum-2/), and [Alistair Cockburn](https://en.wikipedia.org/wiki/Alistair_Cockburn), met first at a resort in Oregon and later, in 2001, at The Lodge at Snowbird ski resort in Utah. It was at the second meeting where the Agile Manifesto and the Twelve Principles were formally written. The Manifesto reads:

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

“Individuals and interactions over processes and tools Working software over comprehensive documentation Customer collaboration over contract negotiation Responding to change over following a plan

“That is, while there is value in the items on the right, we value the items on the left more.”

**The Four Values of the Agile Manifesto :**

The Agile Manifesto is comprised of four foundational values and 12 supporting principles which lead the Agile approach to software development. Each Agile methodology applies the four values in different ways, but all of them rely on them to guide the development and delivery of high-quality, working software.

**1. Individuals and Interactions Over Processes and Tools**  
The first value in the Agile Manifesto is “Individuals and interactions over processes and tools.” Valuing people more highly than processes or tools is easy to understand because it is the people who respond to business needs and drive the development process. If the process or the tools drive development, the team is less responsive to change and less likely to meet customer needs. Communication is an example of the difference between valuing individuals versus process. In the case of individuals, communication is fluid and happens when a need arises. In the case of process, communication is scheduled and requires specific content.

**2. Working Software Over Comprehensive Documentation**  
Historically, enormous amounts of time were spent on documenting the product for development and ultimate delivery. Technical specifications, technical requirements, technical prospectus, interface design documents, test plans, documentation plans, and approvals required for each. The list was extensive and was a cause for the long delays in development. Agile does not eliminate documentation, but it streamlines it in a form that gives the developer what is needed to do the work without getting bogged down in minutiae. Agile documents requirements as user stories, which are sufficient for a software developer to begin the task of building a new function.  
The Agile Manifesto values documentation, but it values working software more.

**3. Customer Collaboration Over Contract Negotiation**  
Negotiation is the period when the customer and the product manager work out the details of a delivery, with points along the way where the details may be renegotiated. Collaboration is a different creature entirely. With development models such as Waterfall, customers negotiate the requirements for the product, often in great detail, prior to any work starting. This meant the customer was involved in the process of development before development began and after it was completed, but not during the process. The Agile Manifesto describes a customer who is engaged and collaborates throughout the development process, making. This makes it far easier for development to meet their needs of the customer. Agile methods may include the customer at intervals for periodic demos, but a project could just as easily have an end-user as a daily part of the team and attending all meetings, ensuring the product meets the business needs of the customer.

**4. Responding to Change Over Following a Plan**  
Traditional software development regarded change as an expense, so it was to be avoided. The intention was to develop detailed, elaborate plans, with a defined set of features and with everything, generally, having as high a priority as everything else, and with a large number of many dependencies on delivering in a certain order so that the team can work on the next piece of the puzzle.

With Agile, the shortness of an iteration means priorities can be shifted from iteration to iteration and new features can be added into the next iteration. Agile’s view is that changes always improve a project; changes provide additional value.

Perhaps nothing illustrates Agile’s positive approach to change better than the concept of Method Tailoring, defined in [An Agile Information Systems Development Method](https://journals.tubitak.gov.tr/elektrik/issues/elk-04-12-2/elk-12-2-5-0404-6.pdf) in use as: “A process or capability in which human agents determine a system development approach for a specific project situation through responsive changes in, and dynamic interplays between contexts, intentions, and method fragments.”[Agile methodologies](https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban) allow the Agile team to modify the process and make it fit the team rather than the other way around.

**The Twelve Agile Manifesto Principles:**

The Twelve Principles are the guiding principles for the methodologies that are included under the title “The Agile Movement.” They describe a culture in which change is welcome, and the customer is the focus of the work. They also demonstrate the movement’s intent as described by Alistair Cockburn, one of the signatories to the Agile Manifesto, which is to bring development into alignment with business needs.

The twelve principles of agile development include:

1. **Customer satisfaction through early and continuous software delivery**– Customers are happier when they receive working software at regular intervals, rather than waiting extended periods of time between releases.
2. **Accommodate changing requirements throughout the development process**– The ability to avoid delays when a requirement or feature request changes.
3. **Frequent delivery of working software**– Scrum accommodates this principle since the team operates in software sprints or iterations that ensure regular delivery of working software.
4. **Collaboration between the business stakeholders and developers throughout the project**– Better decisions are made when the business and technical team are aligned.
5. **Support, trust, and motivate the people involved** – Motivated teams are more likely to deliver their best work than unhappy teams.
6. **Enable face-to-face interactions**– Communication is more successful when development teams are co-located.
7. **Working software is the primary measure of progress** – Delivering functional software to the customer is the ultimate factor that measures progress.
8. **Agile processes to support a consistent development pace**–Teams establish a repeatable and maintainable speed at which they can deliver working software, and they repeat it with each release.
9. **Attention to technical detail and design enhances agility**– The right skills and good design ensures the team can maintain the pace, constantly improve the product, and sustain change.
10. **Simplicity** – Develop just enough to get the job done for right now.
11. **Self-organizing teams encourage great architectures, requirements, and designs**– Skilled and motivated team members who have decision-making power, take ownership, communicate regularly with other team members, and share ideas that deliver quality products.
12. **Regular reflections on how to become more effective**– Self-improvement, process improvement, advancing skills, and techniques help team members work more efficiently.

The intention of Agile is to align development with business needs, and the success of Agile is apparent. Agile projects are customer focused and encourage customer guidance and participation. As a result, Agile has grown to be an overarching view of software development throughout the software industry and an industry all by itself.

## GIT

# What is version control?

###### **How version control helps high performing development and DevOps teams prosper**

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time. As development environments have accelerated, version control systems help software teams’ work faster and smarter. They are especially useful for [DevOps](https://www.atlassian.com/devops/what-is-devops) teams since they help them to reduce development time and increase successful deployments.

Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

For almost all software projects, the source code is like the crown jewels - a precious asset whose value must be protected. For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful effort. Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences.

Software developers working in teams are continually writing new source code and changing existing source code. The code for a project, app or software component is typically organized in a folder structure or "file tree". One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.

Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting. Changes made in one part of the software can be incompatible with those made by another developer working at the same time. This problem should be discovered and solved in an orderly manner without blocking the work of the rest of the team. Further, in all software development, any change can introduce new bugs on its own and new software can't be trusted until it's tested. So testing and development proceed together until a new version is ready.

Good version control software supports a developer's preferred workflow without imposing one particular way of working. Ideally it also works on any platform, rather than dictate what operating system or tool chain developers must use. Great version control systems facilitate a smooth and continuous flow of changes to the code rather than the frustrating and clumsy mechanism of file locking - giving the green light to one developer at the expense of blocking the progress of others.

Software teams that do not use any form of version control often run into problems like not knowing which changes that have been made are available to users or the creation of incompatible changes between two unrelated pieces of work that must then be painstakingly untangled and reworked. If you're a developer who has never used version control you may have added versions to your files, perhaps with suffixes like "final" or "latest" and then had to later deal with a new final version. Perhaps you've commented out code blocks because you want to disable certain functionality without deleting the code, fearing that there may be a use for it later. Version control is a way out of these problems.

Version control software is an essential part of the every-day of the modern software team's professional practices. Individual software developers who are accustomed to working with a capable version control system in their teams typically recognize the incredible value version control also gives them even on small solo projects. Once accustomed to the powerful benefits of version control systems, many developers wouldn't consider working without it even for non-software projects.

## Benefits of version control systems:

Using version control software is a best practice for high performing software and [DevOps](https://www.atlassian.com/devops/what-is-devops) teams. Version control also helps developers move faster and allows software teams to preserve efficiency and agility as the team scales to include more developers.

Version Control Systems (VCS) have seen great improvements over the past few decades and some are better than others. VCS are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System). One of the most popular VCS tools in use today is called Git. Git is a *Distributed* VCS, a category known as DVCS, more on that later. Like many of the most popular VCS systems available today, Git is free and open source. Regardless of what they are called, or which system is used, the primary benefits you should expect from version control are as follows.

1. A complete long-term change history of every file. This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents. Different VCS tools differ on how well they handle renaming and moving of files. This history should also include the author, date and written notes on the purpose of each change. Having the complete history enables going back to previous versions to help in root cause analysis for bugs and it is crucial when needing to fix problems in older versions of software. If the software is being actively worked on, almost everything can be considered an "older version" of the software.
2. Branching and merging. Having team members work concurrently is a no-brainer, but even individuals working on their own can benefit from the ability to work on independent streams of changes. Creating a "branch" in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict. Many software teams adopt a practice of branching for each feature or perhaps branching for each release, or both. There are many different workflows that teams can choose from when they decide how to make use of branching and merging facilities in VCS.
3. Traceability. Being able to trace each change made to the software and connect it to project management and bug tracking software such as [Jira](https://www.atlassian.com/software/jira), and being able to annotate each change with a message describing the purpose and intent of the change can help not only with root cause analysis and other forensics. Having the annotated history of the code at your fingertips when you are reading the code, trying to understand what it is doing and why it is so designed can enable developers to make correct and harmonious changes that are in accord with the intended long-term design of the system. This can be especially important for working effectively with legacy code and is crucial in enabling developers to estimate future work with any accuracy.

While it is possible to develop software without using any version control, doing so subjects the project to a huge risk that no professional team would be advised to accept. So the question is not whether to use version control but which version control system to use.

## What is Git?

Git is a version control system which lets you track changes you make to your files over time. With Git, you can revert to various states of your files (like a time traveling machine). You can also make a copy of your file, make changes to that copy, and then merge these changes to the original copy.

For example, you could be working on a website's landing page and discover that you do not like the navigation bar. But at the same time, you might not want to start altering its components because it might get worse.

With Git, you can create an identical copy of that file and play around with the navigation bar. Then, when you are satisfied with your changes, you can merge the copy to the original file.

You are not limited to using Git just for source code files – you can also use it to keep track of text files or even images. This means that Git is not just for developers – anyone can find it helpful.

### How to install Git

In order to use Git, you have to install it on your computer. To do this, you can download the latest version on the [official website](https://git-scm.com/downloads). You can download for your operating system from the options given.

You can also install Git using the command line, but since the commands vary with each operating system, we'll focus on the more general approach.

### How to configure Git

I will assume that at this point you have installed Git. To verify this, you can run this command on the command line: git --version. This shows you the current version installed on you PC.

The next thing you'll need to do is to set your username and email address. Git will use this information to identify who made specific changes to files.

To set your username, type and execute these commands: git config --global user.name "YOUR\_USERNAME" and git config --global user.email "YOUR\_EMAIL". Just make sure to replace "YOUR\_USERNAME" and "YOUR\_EMAIL" with the values you choose.

## How to Create and Initialize a Project in Git

We are finally done with installing and setting up Git. It is now time to create our project.

I have created a folder on my desktop called Git and GitHub tutorial. Using the command line, navigate to your new project's location. For me, I would run the following commands:

cd desktop

cd Git and GitHub tutorial

If you are new to the command line and are still learning how to use it to navigate around your PC, then I would suggest using Microsoft's Visual Studio Code. It is a code editor which has an inbuilt terminal for executing commands. You can download it [here](https://code.visualstudio.com/download).

After installing VS Code, open your project in the editor and open a new terminal for your project. This automatically points the terminal/command line to your project's path.

Now to initialize your project, simply run git init. This will tell Git to get ready to start watching your files for every change that occurs. It looks like this:



git init

The first line has information about my PC and the path to where the folder exists. The second line is the command git init, and the third line is the response sent back telling me that my repository (repo) has been initialized. It is considered empty because we have not told Git what files to track.

A repository is just another way to define a project being watched/tracked by Git.

### Git project files

I have created only one file called todo.txt. This is what the file looks like:

MY TO-DO LIST

1. Write an article.

2. Code.

3. Study books.

4. Attend classes on time.

5. Visit aunt.

6. Apply for remote jobs.

Before we proceed with learning other Git commands, let's talk about GitHub.

## What is GitHub?

GitHub is an online hosting service for Git repositories. Imagine working on a project at home and while you are away, maybe at a friend's place, you suddenly remember the solution to a code error that has kept you restless for days.

You cannot make these changes because your PC is not with you. But if you have your project hosted on GitHub, you can access and download that project with a command on whatever computer you have access to. Then you can make your changes and push the latest version back to GitHub.

In summary, GitHub lets you store your repo on their platform. Another awesome feature that comes with GitHub is the ability to collaborate with other developers from any location.

Now that we have created and initialized our project locally, let's push it to GitHub.

If you are a beginner, you will come across some new terms like push, commit, add, and so on – but do not be overwhelmed by them. With some practice you will be able to remember these terms and what they do.

## How to push a repository to GitHub

I will divide this section into steps to help you understand the process more clearly.

### Step 1 – Create a GitHub account

To be able to use GitHub, you will have to create an account first. You can do that on their [website](https://github.com/).

### Step 2 – Create a repository

You can click on the + symbol on the top right corner of the page then choose "New repository". Give your repo a name then scroll down and click on "Create repository".

### Step 3 – Add and commit file(s)

Before we "add" and "commit" our files, you need to understand the stages of a file being tracked by Git.

#### Committed state

A file is in the **committed** state when all the changes made to the file have been saved in the local repo. Files in the committed stage are files ready to be pushed to the remote repo (on GitHub).

#### Modified state

A file in the **modified** state has some changes made to it but it's not yet saved. This means that the state of the file has been altered from its previous state in the committed state.

#### Staged state

A file in the **staged** state means it is ready to be committed. In this state, all necessary changes have been made so the next step is to move the file to the commit state.

You can understand this better by imagining Git as a camera. The camera will only take a snapshot when the file reaches the commit state. After this state, the camera starts comparing changes being made to the same file with the last snapshot (this is the modified state). And when the required changes have been made, the file is staged and moved to the commit state for a new snapshot.

This might be a lot of information to take in at the moment, but do not be discouraged – it gets easier with practice.

### How to add files in Git

When we first initialized our project, the file was not being tracked by Git. To do that, we use this command git add . The period or dot that comes after add means all the files that exist in the repository. If you want to add a specific file, maybe one named about.txt, you use git add about.txt.

Now our file is in the staged state. You will not get a response after this command, but to know what state your file is in, you can run the git status command.

### How to commit files in Git

The next state for a file after the staged state is the committed state. To commit our file, we use the git commit -m "first commit" command.

The first part of the command git commit tells Git that all the files staged are ready to be committed so it is time to take a snapshot. The second part -m "first commit" is the commit message. -m is shorthand for message while the text inside the parenthesis is the commit message.

After executing this command, you should get a response similar to this:

git commit

Now our file is in the committed state.

### Step 4 – Push the repository to GitHub

After you create the repo, you should be redirected to a page that tells you how to create a repo locally or push an existing one.

In our case, the project already exists locally so we will use commands in the "…or push an existing repository from the command line" section. These are the commands:

git remote add origin https://github.com/ihechikara/git-and-github-tutorial.git

git branch -M main

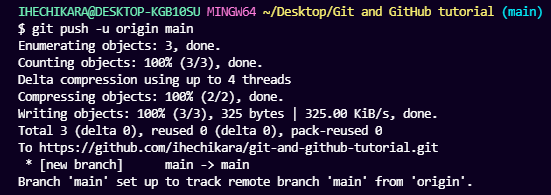
git push -u origin main

The first command git remote add origin <https://github.com/ihechikara/git-and-github-tutorial.git> creates a connection between your local repo and the remote repo on Github.

The URL for your remote project should be entirely different from the one above. So to follow along, make sure you are following the steps and working with your own remote repo. You won't usually get a response after executing this command but make sure you have an internet connection.

The second command git branch -M main changes your main branch's name to "main". The default branch might be created as "master", but "main" is the standard name for this repo now. There is usually no response here.

The last command git push -u origin main pushes your repo from your local device to GitHub. You should get a response similar to this:

git push

To help you deepen your understanding of file stages, I will make changes to the file and then push the new version to GitHub.

Recall that our file is now in the committed state. Let's make changes to the file and take note of the states.

I am going to add a new task to the to-do list:

MY TO-DO LIST

1. Write an article.

2. Code.

3. Study books.

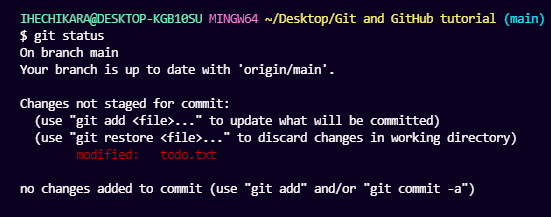
4. Attend classes on time.

5. Visit aunt.

6. Apply for remote jobs.

7. Practice code

After adding the new task, run the git status command. This is what you should see:

git status

After making changes to the file, it moved to the modified state – but it's not yet staged for commit, so you can't push it to GitHub yet. Git has not taken a final snapshot of this current state as it's only comparing the changes we have made now with the last snapshot.

Now we are going to add (stage) this file and then commit and push it. This is the same as in the last section.

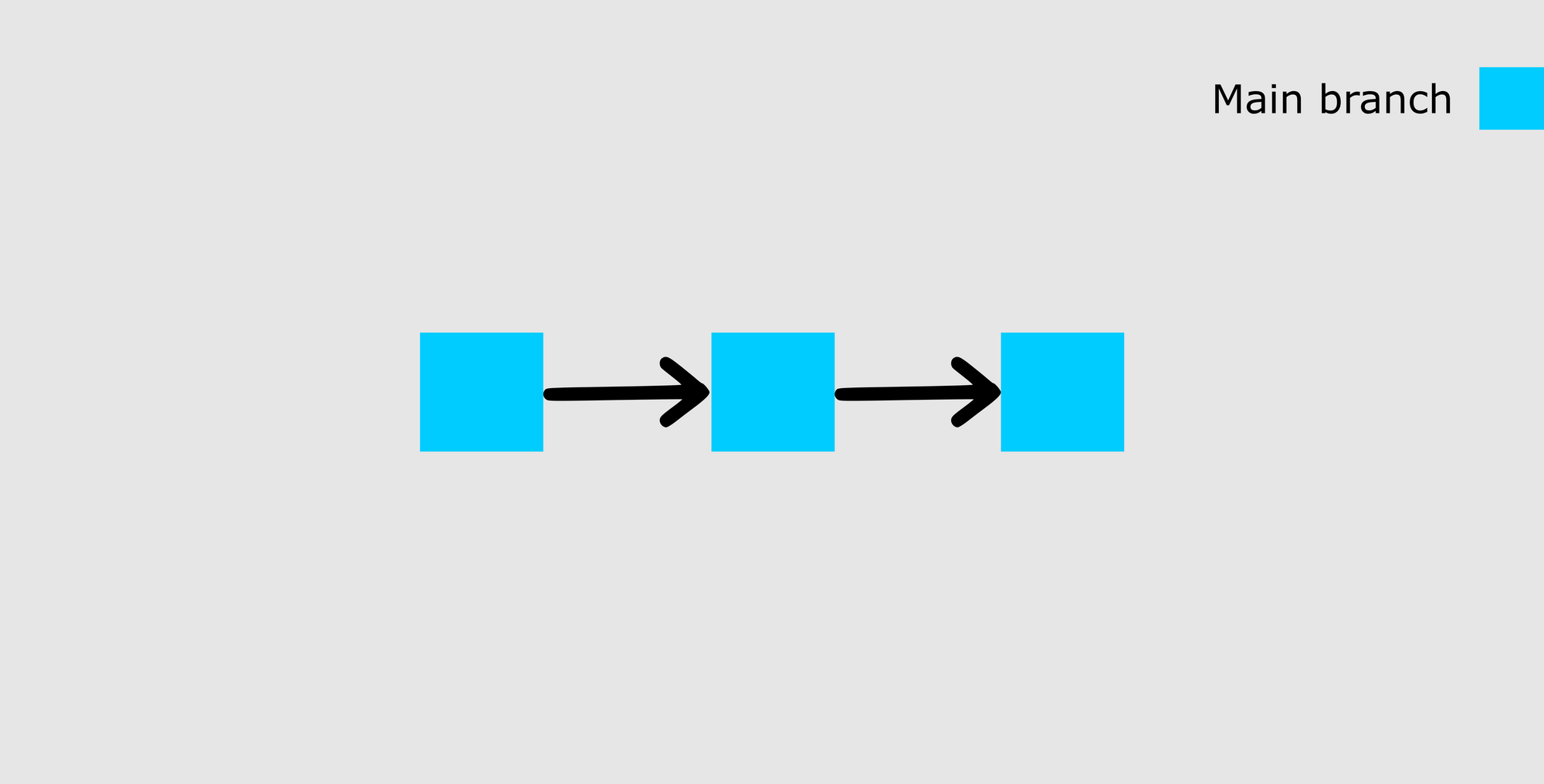
We first add the file by using git add . which adds all the files in the folder (one file in our case). Then we commit the file by running git commit -m "added new task" followed by git push -u origin main.

Those are the three steps to pushing your modified files to GitHub. You add, commit, and then push. I hope you now understand file stages and the commands associated with them.

## How to Use Branches in Git

With branches, you can create a copy of a file you would like to work on without messing up the original copy. You can either merge these changes to the original copy or just let the branch remain independent.

Before we go into using branches, I want to show you a visual representation of our repo which looks like this:

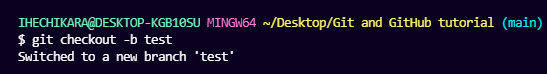
main banch

The image above shows our main branch with the last two commits (the first commit and the added new task commit).

At this point, I want to add more tasks to the list but I am not yet sure whether I want them on my main list. So I will create a new branch called test to see what my list would look like with more tasks included.

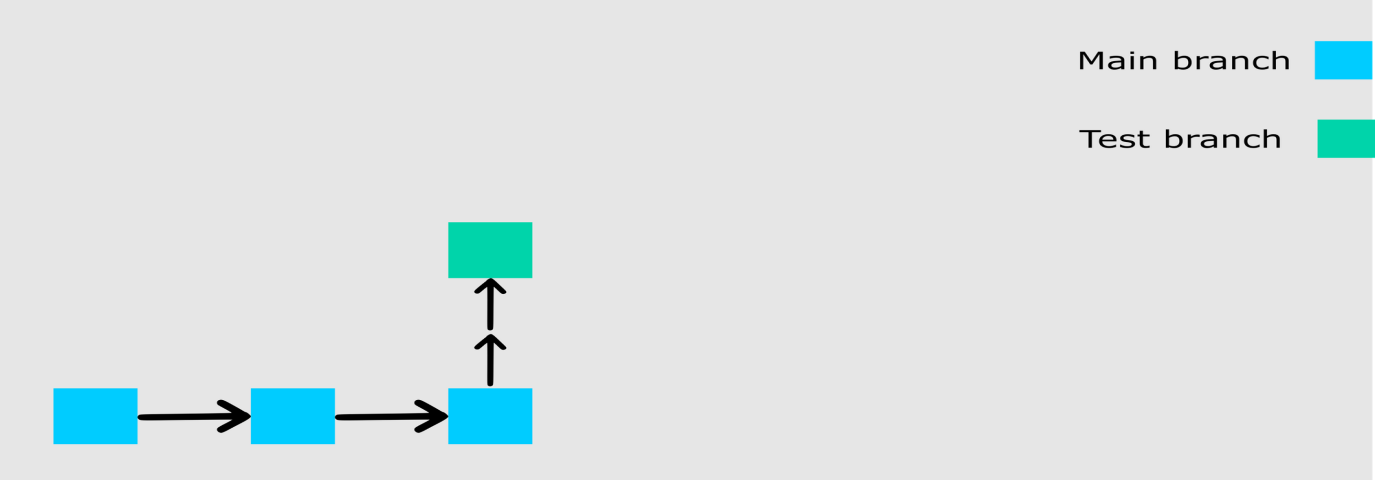
To create a new branch, run this command: git checkout -b test. I will break it down.

checkout tells Git it is supposed to switch to a new branch. -b tells Git to create a new branch. test is the name of the branch to be created and switched to. Here is the response you should get:



git checkout -b

Now that we have a new branch created, this is what our repo will look like:



git branch

We created the new branch from the state of our last commit. Let's now add more tasks to this new branch.

MY TO-DO LIST

1. Write an article.

2. Code.

3. Study books.

4. Attend classes on time.

5. Visit aunt.

6. Apply for remote jobs.

7. Practice code

8. Complete internship task.

9. Practice chess openings.

10. Solve chess puzzles.

11. Check exam schedule.

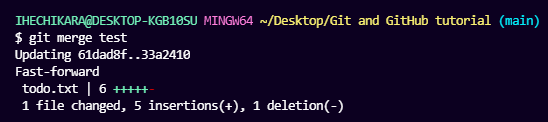
I have added four new tasks. To merge the new state to the main branch, you have to first stage and commit this branch. I will not go into details about how to do this as we did it twice in the last section.

You should try doing it yourself so you understand how it works. As a hint, add the file and then commit with a message (refer to the previous section for details showing you how to do that).

After committing your test branch, switch back to the main branch by running this command: git checkout main.

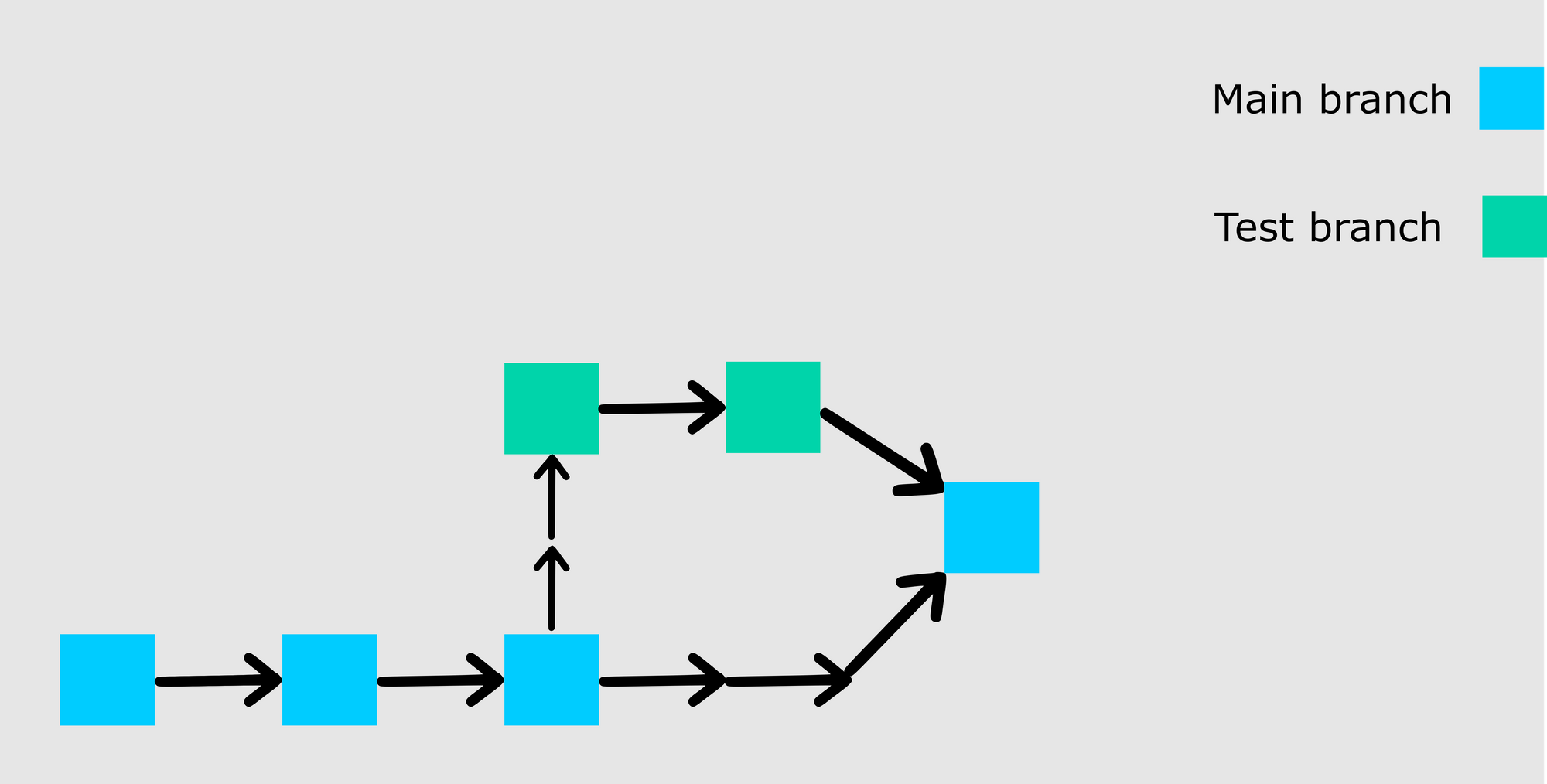
Did you notice that we did not add -b ? This is because we are not creating a new branch but rather switching to an existing one. You can check all the branches that exist in your repo by running the git branch command.

Now we can merge the changes we made in the test branch into the main branch by running git merge test. At this point, you will see all the changes made in the test branch reflected on the main branch. You should also receive a response similar to this:



git merge

Here is a visual representation of our repo:



git merge

If you go on to push your repo to GitHub, you'll see that the test branch will not be pushed. It will only remain in your local repo. If you would like to push your test branch, switch to the branch using git checkout test and then run git push -u origin test.

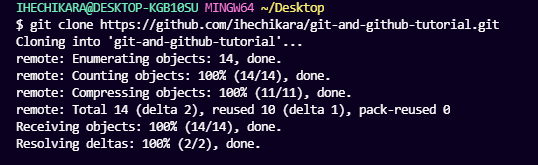
## How to Pull a Repository in Git

To pull in Git means to clone a remote repository's current state into your computer/repository. This comes in handy when you want to work on your repo from a different computer or when you are contributing to an open source project online.

To test this, don't worry about switching to a new computer. Just run cd .. to leave the current directory and go back one step. In my own case, I have navigated back to my desktop.

Go to GitHub, and on your repository's main page you should see a green button that says "Code". When you click on the button, you should see some options in a dropdown menu. Go on and copy the HTTPS URL.

After that, run git clone YOUR\_HTTPS\_URL. This command pulls the remote repository into your local computer in a folder called git-and-git-tutorial. That is:

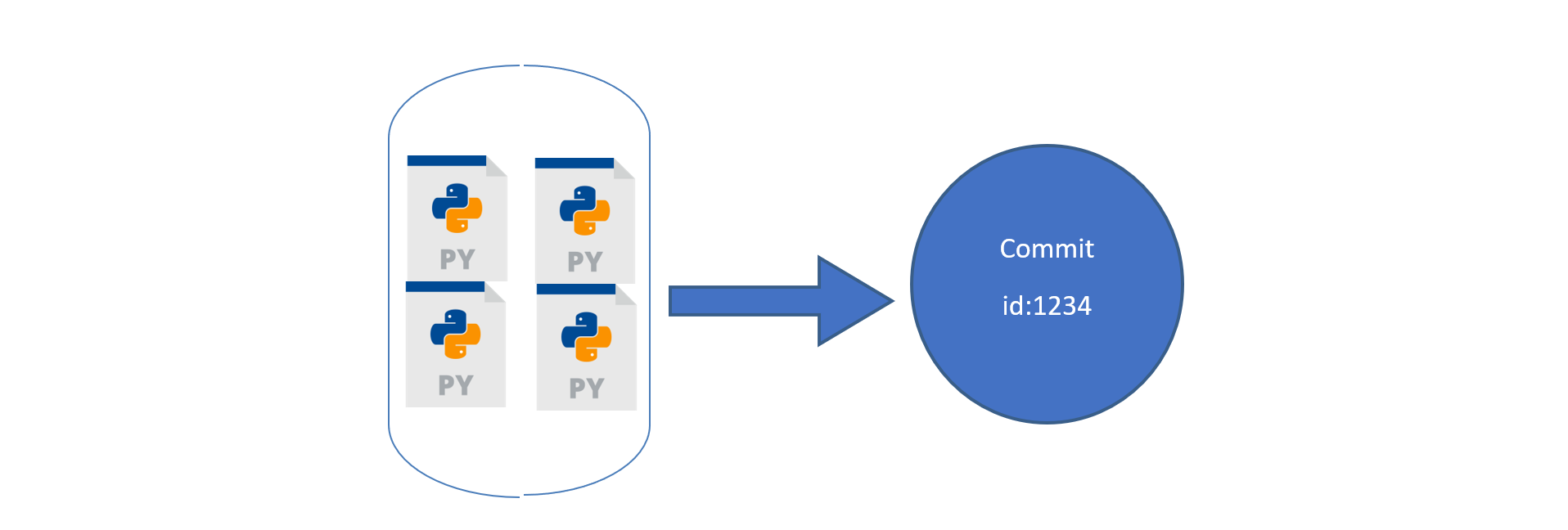


**How does Git Work?**

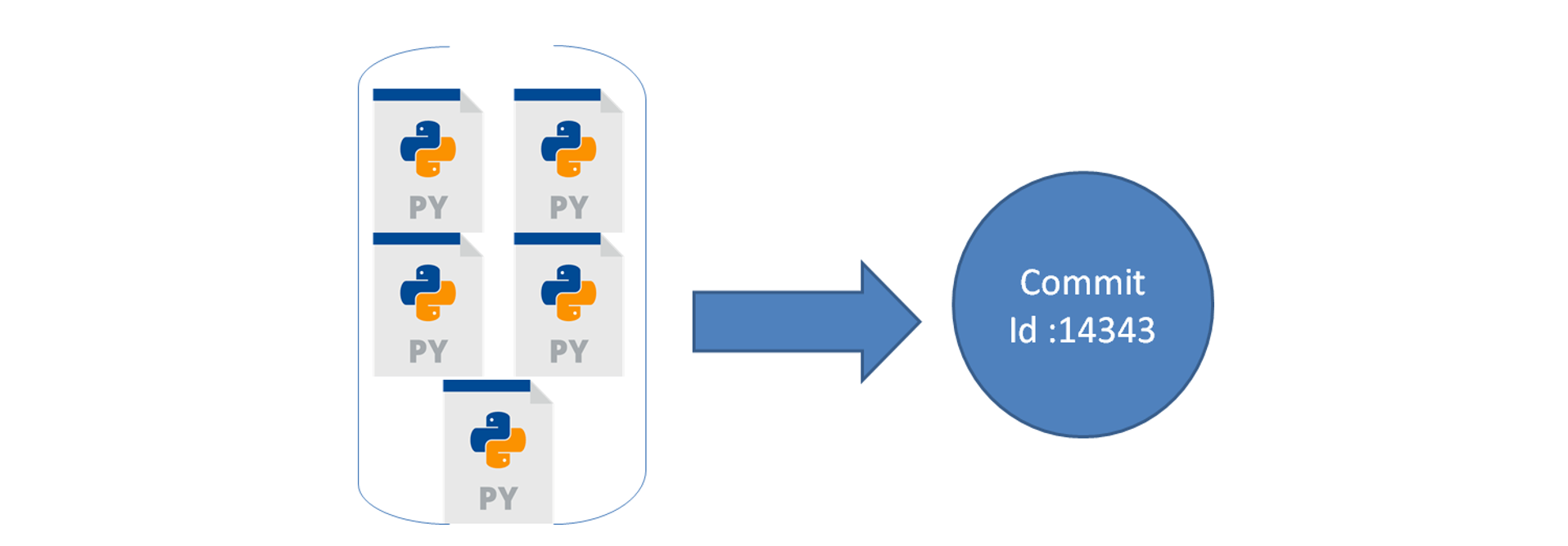
For understanding the working of git, we need to understand the two fundamental concepts in git which is git commit and git branch. Let’s understand these two terms respectively.

### **What is a Commit?**

Commit is defined as the location where the code and its changes are stored. Let us take an example and discuss in brief from the diagram shown below:

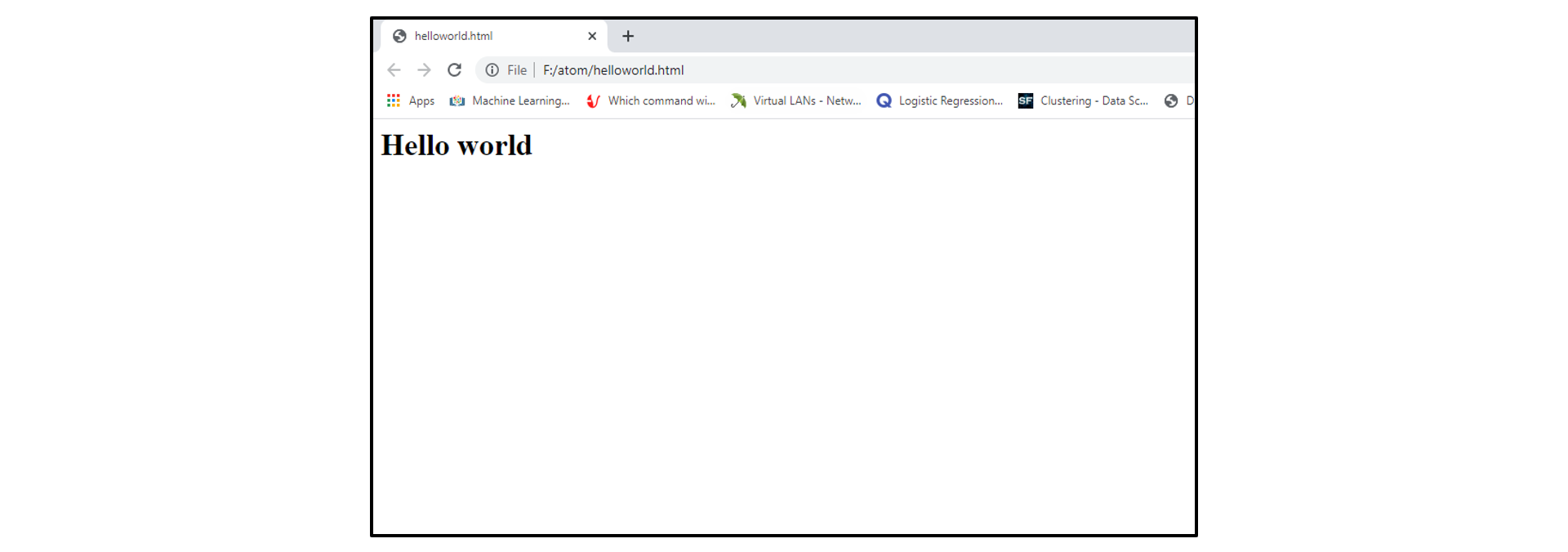
**Fig 1: The changes made are saved in a commit**

In Fig: 1, Let us assume that we have four python files. We saved them on git. These four python files will be saved inside a commit. Each commit has a commit-id, let it be 1234 in our case. Now let us say we have made some changes to the code by adding another python file. These changes in git will be saved as another commit with another commit-id 14343. This can be seen in Fig 2 . So whenever we make any change, it generates a new commit.

**Fig 2: The changes made in the repository will be saved as a new commit with a new commit id**

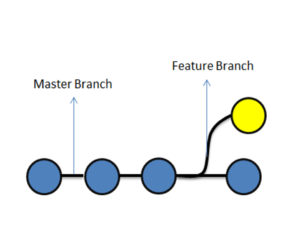
### **What is a Branch?**

A Branch is a representation of different isolated versions of code. Let us take an example to understand this, Let’s say you have a website that is currently running. The website looks something like this.

  
**Fig 3: Sample example of the website before changes**

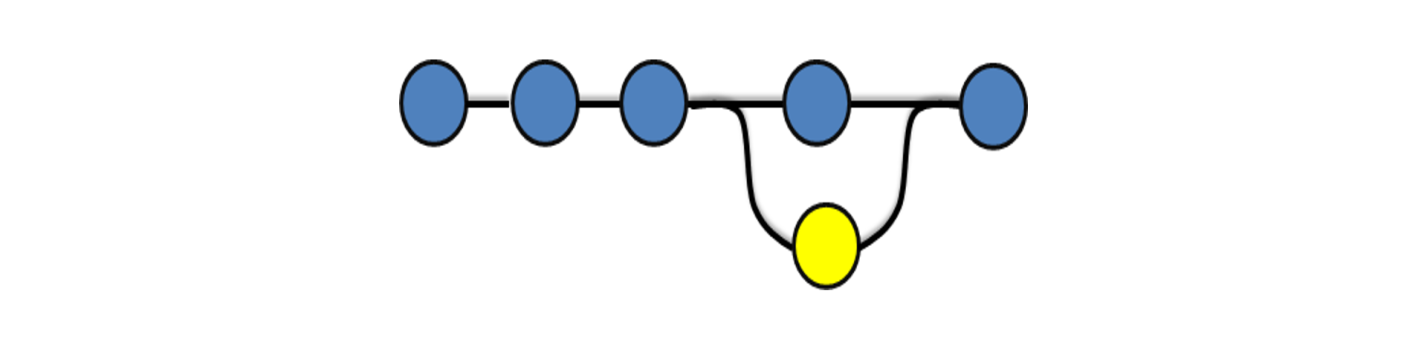
You want to add more features to this website. For this, you will have to change the code of the website. But if you are changing the code, you do not want the changes to be reflected in the main website which is deployed. So, what do you do? Ideally, you will copy the code of this website in a new folder. Make changes in the code, and then once the changes are finalised, you will replace the code in the main folder right? Let us understand how we can do the above thing in git.

So, in git to isolate different versions of code we have branches. By default, all your code is stored on a master branch. So, the website we showed you above is the master branch. Now, we don’t want to touch the master branch code, we want to copy the code of the master branch to a new place, where we can experiment or change the code. Hence, not affecting the master  branch. So, we create a new branch from the master branch, let’s call it a ‘feature branch. Now, any new changes that you will do on the feature branch, will not affect the code on the master branch.



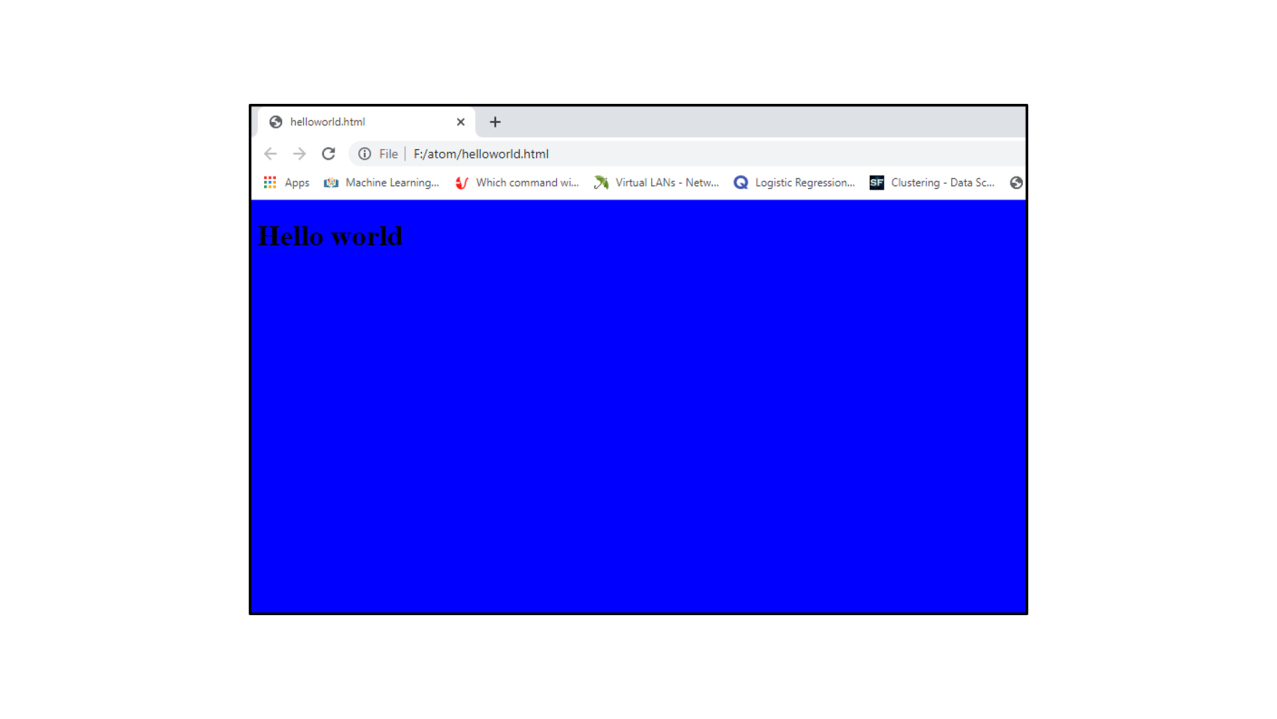
**Fig 4: Master branch and the Feature branch**

Once you are done with the changes, we simply ‘merge’ the changes of the ‘feature’ branch to the ‘master branch’. And now, the changes which were made in the feature branch will exist on the master branch as well.



**Fig 5: Merging of feature branch with the master branch**

If we consider the above website example after merging changes can be seen in the diagram below:

  
**Fig 6: After merging the changes into the master branch changes were depicted on the website.**

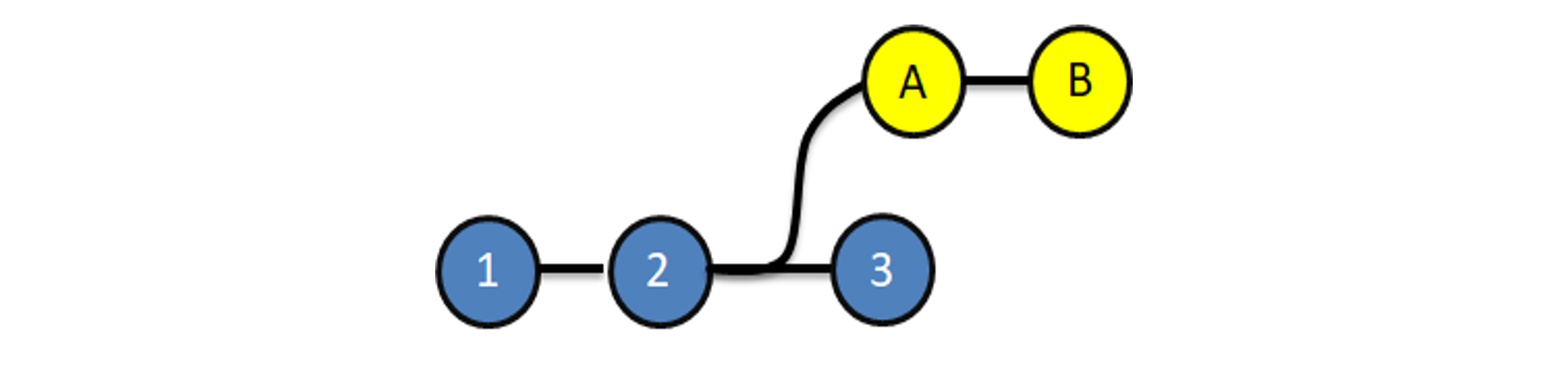
If you don’t understand merging don’t worry everything will be clear in the next section.

## **What is Merging?**

In general, merging means combining something into a single entity.

Git Merge is a technique that is used to include the changes from one branch to the other branch.

So let us take an example from the diagram below, which shows the status before and after merging of two branches, feature and master branch. Blue commits are on the master branch, and yellow commits are on the feature branch.

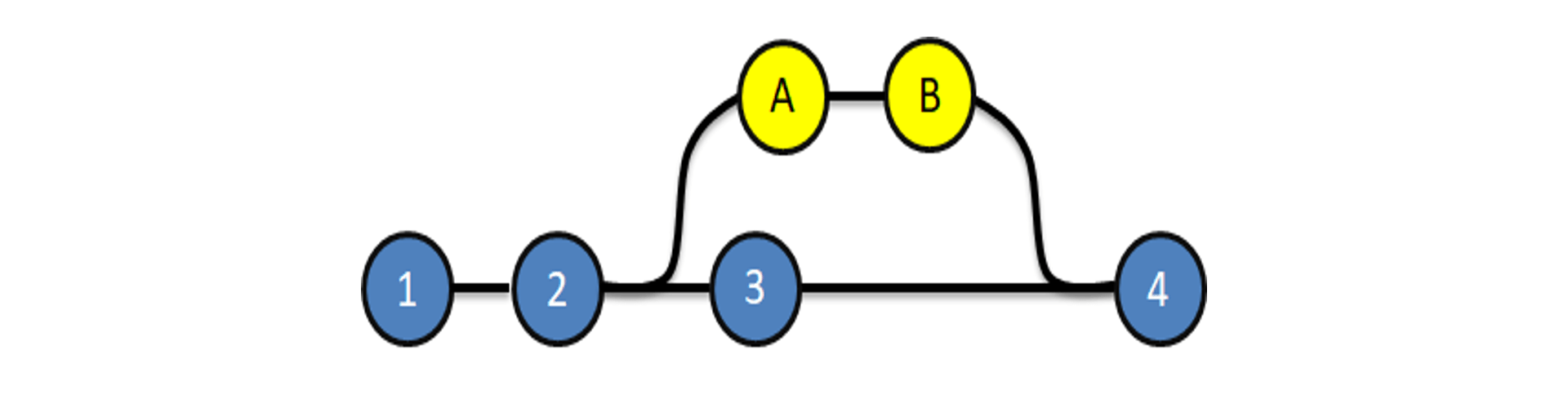
  
**Fig 7:Before merging**

In Fig 7: before merging, we can see that there are some commits on the master branch. We created a feature branch, after commit ‘2’ on master. Later, we did some changes in the feature branch in the form of commits A and B. We also did some changes on the master branch, in the form of commit 3.

In the current scenario, the master branch has the code of Commit 1,2 and 3, but it does not have changes of Commit A and B from the feature branch.

Similarly, since the feature branch was branched out from master on commit ‘2’, it has code changes from Commit 1,2, A, and B, but it does not have changes from Commit 3 on master.

Below, we have merged the feature branch on master, let us understand what happened.

**Fig 8: After merging**

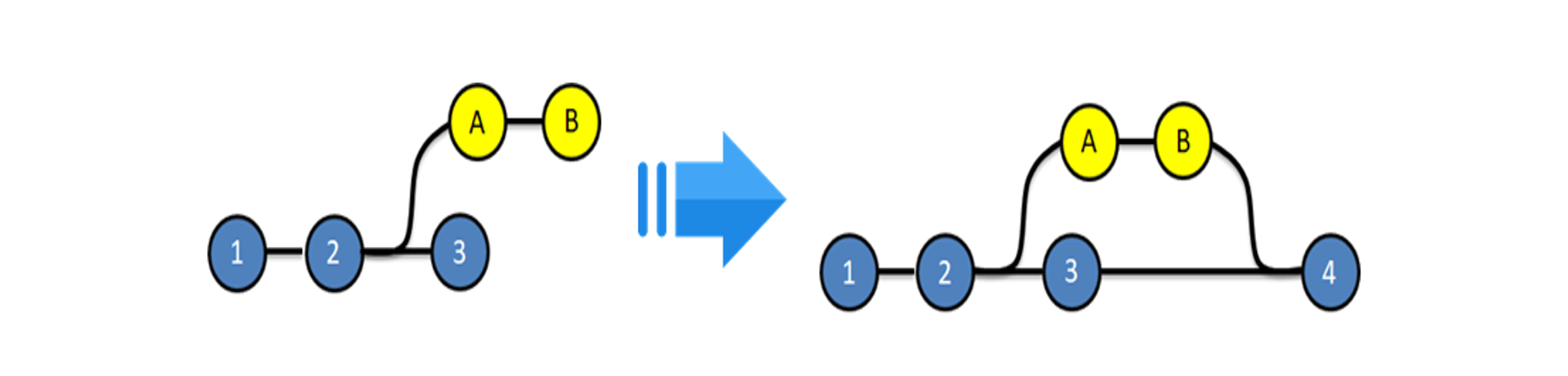
We merged the feature branch on master, resulting in commit 4. Commit 4 on the master, has all the changes of the code i.e Commit 1,2, 3, A, and B. Now, that we understand merging, let’s understand the different types of merging that we can perform in git. In Git, merging is of two types:

* Git Merge
* Git Rebase

Let’s understand both of them in detail

**GitMerge**  
Git merge is one of the merging techniques in git, in which the logs of commits on branches are intact.

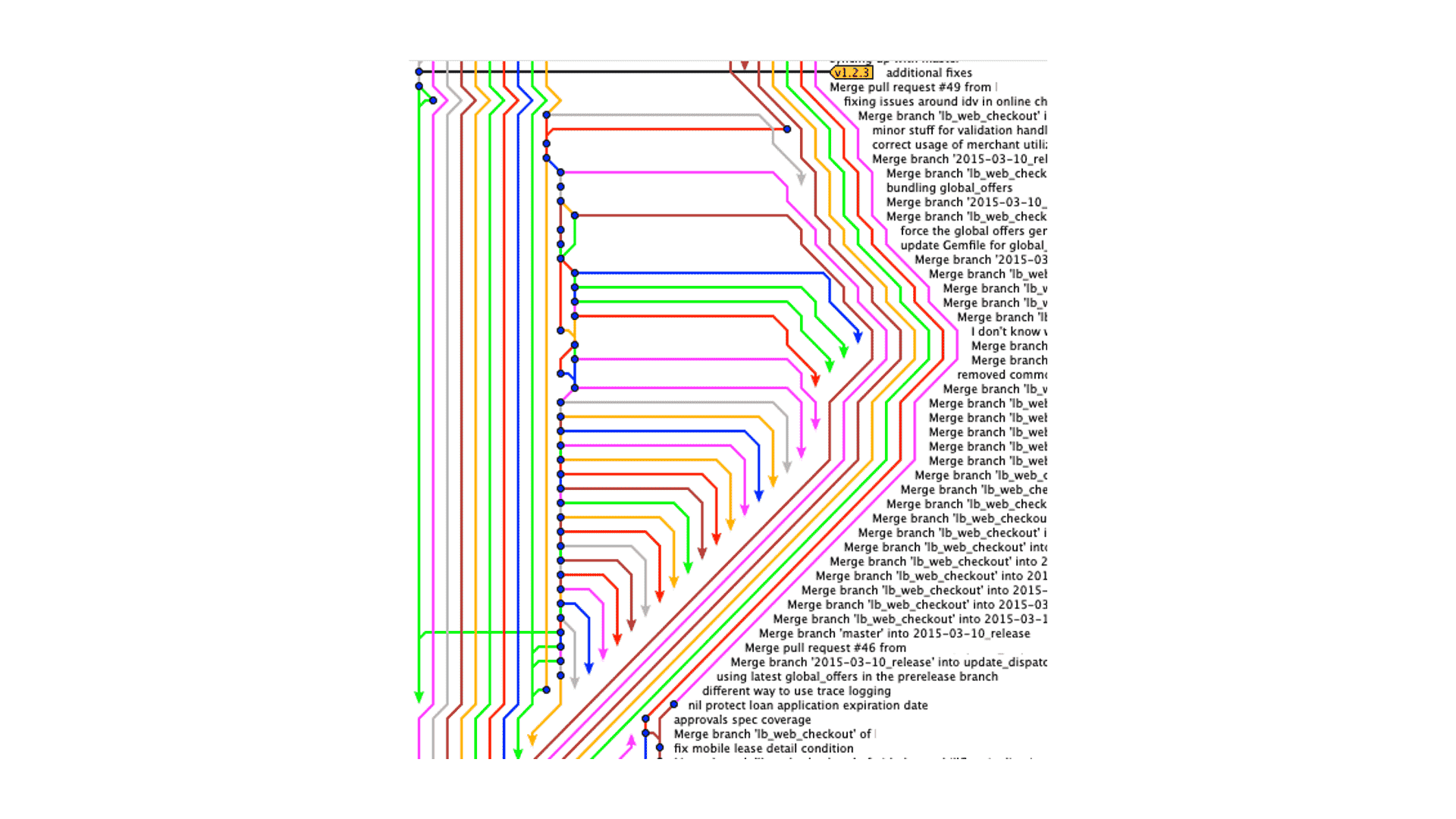
Let us take an example if we have a project with 3 commits on the master branch as commit 1,2,3 and feature branch commits as commit A and B. If we perform a git merge operation then commits A and B will be merged as commit 4 onto the master branch. This is depicted in Fig:9.



**Advantages:**

* The logs are very exhaustive and can help in understanding the complete history of how and when each merge happened
* It is easy to find mistakes and resolve them.

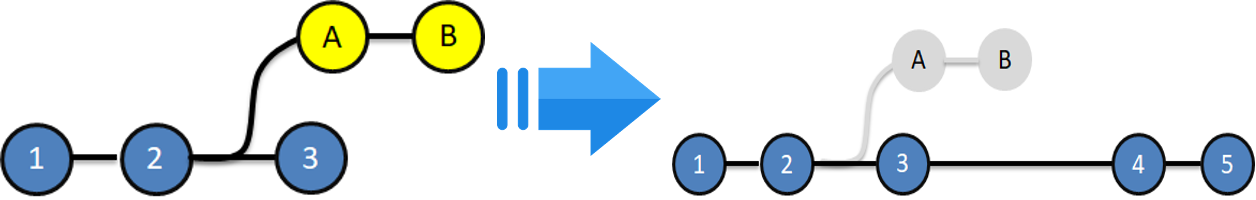
**Disadvantages:**

* Results in a clumsy log / history
* Notveryuser-friendly

**Fig 10: Example showing a repository with multiple branches which were merged using git-merge**

**Git-Rebase**  
Git Rebase is similar to git merge, but the logs are modified after merge in this technique. Git rebase was introduced to overcome the limitation of merging, i.e., to make logs of repository history look linear.

Let us take an example if we have a project with 3 commits on the master branch as commit 1,2,3 and feature branch commits as commit A and B. If we perform a git rebase operation then the commits A and B will be rebased on to the master branch as commit 4 and 5 and there will be no logs of the feature branches. This is depicted in Fig 12.

  
**Fig 12: Before and After git rebase**

**Advantages:**

* The logs are linear
* It’s easy to move through the project.

**Disadvantages:**

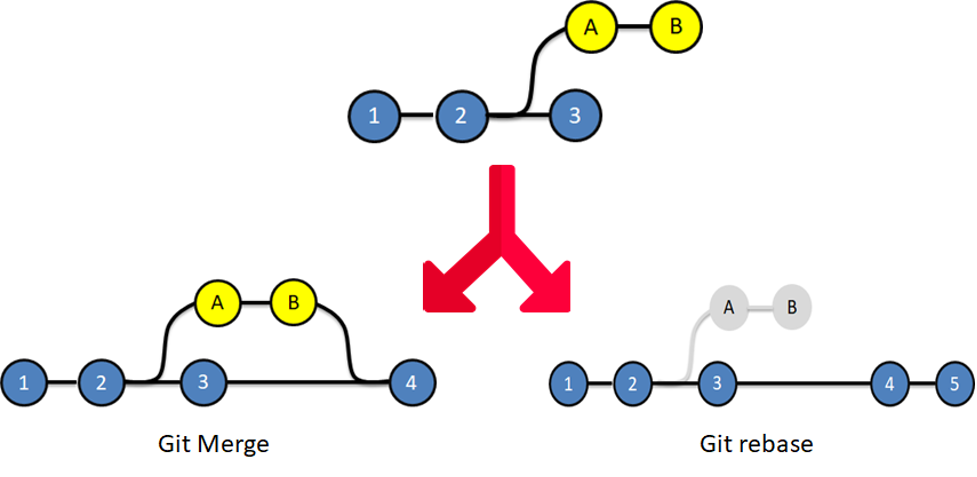
* We cannot track, when and how the commits were merged on the target branch

## **Git Merge Vs Git Rebase:**

| Merge | Rebase |
| --- | --- |
| Git merge is a command that allows you to merge branches from Git. | Git rebase is a command that allows developers to integrate changes from one branch to another. |
| In Git Merge logs will be showing the complete history of the merging of commits. | Logs are linear in Git rebase as the commits are rebased |
| All the commits on the feature branch will be combined as a single commit in the master branch. | All the commits will be rebased and the same number of commits will be added to the master branch. |
| Git Merge is used when the target branch is shared branch | Git Rebase should be used when the target branch is private branch |

## **When do we use Git Merge vs Git Rebase?**

Let’s understand it using an example, please refer diagram below:

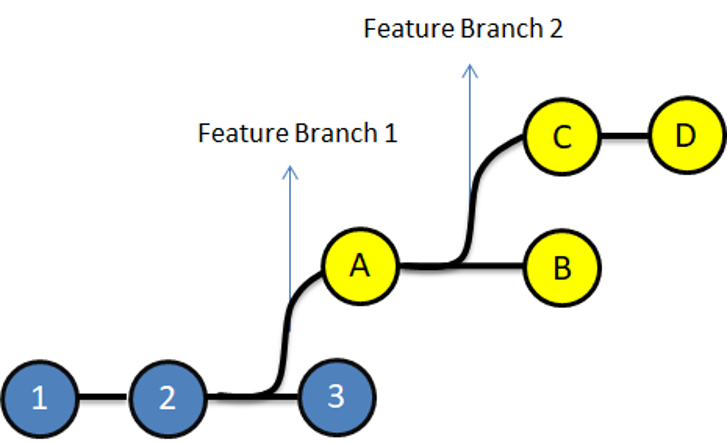
****

**Fig 13: Depicts the difference in the repository after a Git Merge vs Git Rebase**

In git merge, looking at the end diagram, one can make out Commit A and B came from the feature branch. However, in the git-rebase diagram, it is difficult to understand where commits A and B came from. Therefore, we do git merge when we want our team to understand logs in a way where they can identify where each commit is coming from. We use Git Rebase when the logs of the repository will not be referred by anyone else. To summarise, we can use Git Rebase, when we are working on branches, which cannot be seen by other developers. And we use Git Merge when the target and source branch can be viewed by other developers.

## **How can Git Rebase and Git Merge be used together?**

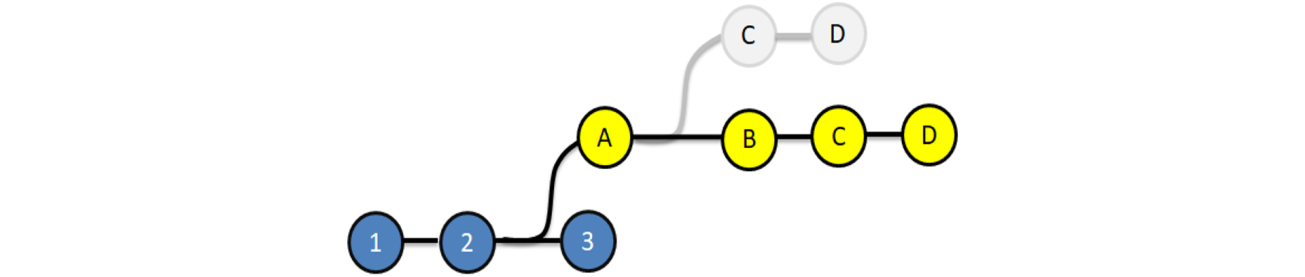
Let us take an example if we have a project which consists of a master branch with 3 commits as 1,2,3, and 2 feature branches with commits A, B on feature branch 1 and commits C, D on feature branch 2 which is depicted in Fig 14.

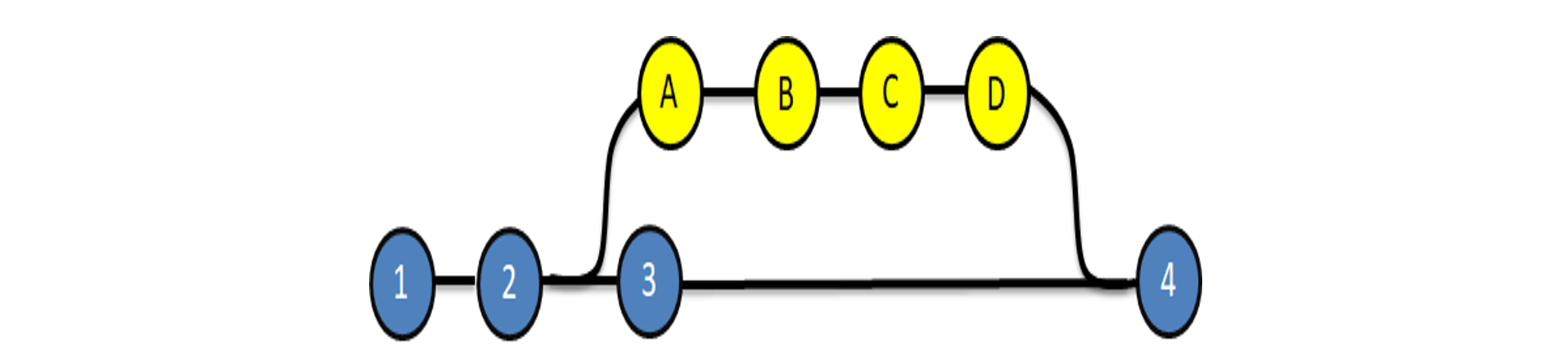
****

**Fig 14: Representation of the above example**

Let us assume that Developer A is working on feature branch 1.

To experiment with the code, he creates another branch Feature branch 2, does some changes in it, and finalizes it with Commits C and D.  
He does not want anyone to know about his private branch, because it’s unnecessary. So, he can rebase Feature 2 on Feature 1.

  
Now he can finally, merge feature 1 branch on master resulting in the below diagram.

The above is how the final log will look like to other developers. They can only see commits being made on feature branch 1, which was then finally merged on the master branch, with commit 4.