**SOURCE CODE MANAGEMENT**

>LECTURE: 1

Git: It is an open-source version-controlled tool created in 2005 by the developers working on Linux operating system.

GitHub: It is a company founded in 2008 that makes tools which integrate with git. You don’t need GitHub to use the git but you can’t use GitHub without using git. It is used for:

1. Tracking code changes.
2. Tracking who made the changes.
3. Coding collaboration.

What is GitHub?

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

* Git is not same as GitHub.
* GitHub makes the tools that used git.
* GitHub is a largest host of source code in the world and has been owned by Microsoft since 2018.

Why we use Git?

* Over 70% of developers around the world use git. Developers can work together form anywhere in the world.
* Developers can see the full history of the project.
* Developers can revert to the early versions of the project.

What is Version Controlled System (VCS)?

A Version Controlled System tracks the history of the changes as people and the teams collaborate on the project together. As developers make the changes to the project any earlier version of the project can be recovered at any time.

Types of Versions Controlled System:

* Local Version Control System: It is located in your local machine. If the local machine crashes it wouldn’t be possible to retrieve the files and all the information will be the lost. If anything happens to a single version, all the versions made after will be lost.
* Centralized Version Control System: In this system, there will be a single central server that contains all the files related to the project and many collaborations check out the files from this single server (You will have only working copy). The problem with the centralized version control system is if the central server crashes almost everything related to project will be lost.
* Distributed Version Control System: In a distributed version control system there will be one or more servers and many collaborators similar to the centralized system. But the difference is not only do they check out the latest version, but each collaborator will have an exact copy (the mirror image). Exact copy of main repository on their local machines (including its entire history).

Each user has their own repository and working copy, this is very useful because even if server crashes, we would not lose everything as several copies are residing in several other computers.

>LECTURE: 2

What is Repository?

A repository or git project and compasses the entire classes of files and folders associated with a project along with each files revision history.

Repositories in git contains a collection of files of various different versions of a project. These files are important from the repository into local server of the user for further updations and modifications in the content of the file. The file history appears as snapshots in the time called **commits.** The commits can be organized into multiple lines of development called **branches** because git is a distributiveversion control system, repositories are self-controlled units and anyone who has a copy of the repository can access the entire code base and its history. Using the command line or other easy of use interfaces git repository also allows for interaction with the history, cloning the repository, creating branches, committing merging, changes across versions of code and many more. Platforms like GitHub, git also provides many opportunities for project transparency and collaboration.

Public repositories help teamwork together to build the best possible final product.

You typically obtained a git repository in one of the two ways:

1. We can take the local directory that is currently not under the version control and turn it into a git repository.
2. We can clone an existing git repository from elsewhere. In either case we end up with a git repository on our local machine ready for work.

A git repository allows performing various operations on it to create different versions of a project. These operations include the addition of files, creating new repositories, committing an action and deleting a repository. These modifications will result in the creation of different versions of project.

>LECTURE: 3

**Benefits of VCS:**

Using version control software is a best practice for high performing software and 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.

**Branching & Merging:** Creating a branch in VCS tools peeps multiple streams of work independent from each other while also providing the facility to merge that work that together. Enabling developers to verify that the changes on each branch does not conflict.

**Traceability**:Being able to trace each change made to the software and connect it to the project management and buy tracking software and being able to allocate 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.

**Version Comparison**: A Complete long term change history of every file. This means every change made by many individuals over the years. Changes include the creations and deletion of files as well as added to their contents. Different VCS tools differ on how well they handle renaming and moving of files.

**Support Developers Work flow**: Having the annotated history of the code at your fingertips when and why you are reading the code. Trying to understand what it is doing and why it is to design can enable developers to make correct and harmonious changes that are in according with the intended long-term design of the system.

-- Examples of some of the best VCS:

•GitHub

•Apache Sub version

•Team foundation series

•Congruent version control

•Bitbucket

What is Source Code Management?

•All version control systems using a programming environment to provide the whole class of software solutions with source code management. Software configuration management is a generic term for the ability to manage multiple version of a document or a collection of document.

•Source code management is an integral part of any development project in the current IT world.

•SCM supports storing the work result and sharing the details with colleagues is essential part.

•SCM is used to track modifications to a source code repository. SCM is also synonyms with version control.

•SCM tracks a running history of changes to a code base and helps to resolve the conflicts when merging the updates from multiple contributors.

•As software projects grow in lines of code and contributors head count, the best of communication overhead and management complexity also grow.

•SCM is a critical tool to elevate the organizational strain of growing development cost.

**Importance of SCM tools:**

•When multiple developers are working within a shared code base. It is a common occurrence to make edits to a shared piece of code.

•Separate developers may be working or a seemingly isolated feature however this feature may use a shared code module.

•Therefore, developer 1 working on feature-1 make some edits and find out later then developer 2 working on feature-2 hence has conflicting

•Before the adoption of SCM this was nightmare scenario. Developers would edit text files directly and move them around to remote locations using file transfer protocols of other protocols.

•Developer 1 would make edits and developer 2 would unknowingly save over the developer 1 work and wipe out the changes.

•SCM’s role as protection mechanism against this specific scenario is known as version control.

•SCM brought version control safeguards to prevent loss of work due to conflict over writing.

•These safeguards work by tracking the charges from each individual developer & identifying areas of conflict and preventing overwrites.

•SCM will then communicate these points of conflict back to the developers so that they can safely review and address.

•This foundational conflict prevention mechanism has the side effect of providing passive communication for the development team.

•The team can then monitor and discuss the work in progress that the SCM is monitoring.

•The SCM tracks the entire history of changes to the code base.

•This allows developers to examine and review the edits that may have introduced bugs or regressions.

>Lecture: 4

Benefits Of SCM:

In addition to version control SCM provides a suit of other helpful features to make collaborating code development. A more user-friendly experience. Once SCM has started tracking all the changes to a project over the times. A detail historical record of a project file is created. This historical record can then be used to undo the changes to the code base. The SCM can instantly revert the code base that back to a previous point in time. This is extremely valuable for preventing regressions on the updates and undoing mistakes.

The most common benefits are as follows:

**History log:**

* The SCM archive of every change over a project’s lifetimes provides valuable report keeping for a project’s release version notes.
* A clean and maintain SCM history log can be used interchangeably as release notes.
* This offers inside and transparency into the progress of a project that can be shared with n user or non-development teams.

**Non overlapping:**

* SCM will reduce a team’s communication over head and increase release velocity.
* Without SCM development is slower because contributors have to take extra efforts to plan non-overlapping sequence of a develop for release.
* With SCM developers can work independently on separate branches of fisher development eventually merging them together.

**Reduce development cost:**

* Overall SCM is a huge aid to engineering teams that will lower the development cost by allowing engineering resources to execute more efficiently.
* SCM is a must have in the modern age of software development. Professional teams use version control and our team should use this.

**Best practices of SCM:**

**Commit of ten:** Commits are cheap and easy to make. They should be made frequently to capture the updates to a code base. Each commit is a snap shot that the code base can be reverted if needed. Frequent commits give many opportunities to revert or undo the work. A group of commits can be combined into a single commit using a rebase to clarify the development log.

**Ensure you are working from the latest version:** SCM enables rapid updates from multiple developers. It is easy to have a local copy of the code base for behind global copy. Make sure to git pull or fetch the latest code before making the updates. This will help avoid the conflicts at merge time.

**Make detail notes:** Each commit has a corresponding log entry. At the time creation this log entry is populated with a message. It is important to leave descriptive exploratory commit log messages. These commits log messages should explain why and what that can compass the commits content. These log messages become the canonical history of the projects development and leave a trail for future communication.

**Use the branches:** Branching is a powerful SCM mechanism that allows the developers to create a separate line of development. Branches should be used frequently as they are quick and expensive. Branches enable multi developers who work in parallel on separate lines of developers are generally different product feature. When development is complete on a branch it is then merge into the main line of development.

**Review changes before Committing:** The SCM offer a staging area. The staging area can be used to collect a group of edits before writing them to a commit. The staging area can be used to manage and review the changes before creating the commits snapshots. Utilizing the staging area in this manner provide a buffer area to help refine the contents of the commit.

**Agree on a Workflow:** By default, SCM offers very free-form methods of contribution. It is important that teams established shared patterns of collaborations. SCM work flows establish patterns and processes for merging the branches. If a team doesn’t agree on a shared workflow it can lead to inefficient communication overhead when it comes times to merge the branches.

>Lecture: 5

**Features Of GIT:** There are various features of git but most popular features are as follows:

**Performance:**

* The raw performance characteristics of git are very strong when compared to many other alternatives.
* Committing new changes, branching, merging and comparing past versions are all optimized for performance.
* The algorithm implemented inside the git take advantage of deep knowledge about common attributes of real source code file trees, how they are usually modified over the times and what the excess patterns are.
* Unlike some version control software, git is not fooled by the names of the files when determining what the storage and version history of the file tree should be instead git focuses on the file content itself.
* After all source code files are frequently renamed, split and rearranged.
* The object format of GIT’s repository files uses a combination of delta encoding, completion in explicitly stores directory contents and versions meta data.
* Examples: Suppose a developer Mark make changes to source code adding a feature for the upcoming 2.0 release then commits those changes with descriptive messages, he then works on a second feature and commits those changes to naturally and these are stored as separate piece of work in the version history.
* Mark then switches to the version 1.3 branch of the same software to fix a bug that affects only that older version. The purpose of this is enable Mark’s team to shift a bug fix release, version 1.3.1 before version 2.0 is ready. Mark can then return to the 2.0 branch to continue working on new features for 2.0 and all of this can occur without any network axis and is therefore fast and reliable. Mark could even do it on an airplane when he is ready to sent all of the individually committed changes to the remote repository, Mark can push them one in single command.

Security:

* Git has been designed with the integrity of managed source code as a top priority.
* The content of the files as well as the true relationships between files and the directories, versions, texts & commits. All of these objects in the git repository are secured with a crypto graphically secured hashing algorithm called SHA1.
* This protects the code at the chain history against both accidental and malicious change and ensures that the history is fully traceable.

Flexibility:

* One of the GIT’s key designed objectives is flexibility.
* GIT is flexible in several aspects like in support of various kinds of nonlinear development work flows, in its efficiency in both small and large projects and in its compatibility with many existing systems and protocols.
* GIT has been designed to support branching and tagging as operations that affects the branches and a part of the change history, not all the version control systems feature this level of tracking.

🡪lecture:6

Git is valuable for every department of organization. It is valuable from development in to the marketing team and everything in between. We can categorize the usefulness of git for various departments:

1. **Git for developers :**

* One of the biggest advantages of git is its branching capabilities.
* Unlike centralized version control system, git branches are cheap and easy to merge.
* This facilitates the feature branch workflow popular with many git users.
* Feature branches provide an isolated environment for every change to your code base.
* When a developer wants to start working on something, no matter how big or small they just create a new branch.
* This ensures that the main branch always contains production quality code.
* Using feature branches is not only more reliable then directly editing production code but it also provides organizational benefits.

**Distributed development**:

* Git however is a distributive control system instead of a working copy each developer gets there on local repository, complete with a full history of commits
* Having a full local history makes git fast since it means you don’t need a network connection to create commits, inspect previous versions of a file, or perform difference between commits.
* Similar to feature branches distributed development creates a more reliable environment.

**Pull request:**

* Many source code management rules such as bit bucket, enhance core git functionality with pull request.
* A pull request is a way to ask another developer to merge one of your branches into their repository. This not only makes it easier for project leads to keep track of the changes but also let the developers’ initiate discussions around their work before integrating it with the rest of the code base.
* When a developer gets stuck with a hard problem, they can open a pull request to ask for the help from the rest of the team. Alternatively, junior developers can also be confident that they are not destroying the entire project by creating pull request as a formal code review.

**Committed:** In many sentences, git has come to be expected version control system for new projects. If your team is using git then you won’t have to create new joints because they are already familiar with the distributed development.

**Faster release cycle:** The ultimate result of feature branches distributed development, pull request and a stable community is a faster release cycle. For example, you might want to configure git to deploy the most recent commit from the developed branch to a test server whenever anyone merge a pull request into it. Combining this kind of bill automation with peer review means that you have the highest possible confidence in your code as it moves from development to staging to the production.

**IMPORTANCE OF GIT:**

Git is valuable for every department of organization. It is valuable from development in to the marketing team and everything in between. We can categorize the usefulness of git for various departments.

1. **GIT FOR DEVELOPERS:**

**•Feature Branch Workflow:**

* One of the biggest advantages of git is its branching capabilities.
* Unlike centralized version control system, git branches are cheap and easy to merge.
* This facilitates the feature branch workflow popular with many git users.
* Feature branches provide an isolated environment for every change to your code base.
* When a developer wants to start working on something, no matter how big or small they just create a new branch.
* This ensures that the main branch always contains production quality code.
* Using feature branches is not only more reliable then directly editing production code but it also provides organizational benefits.

•**Distributive Development:**

* Git however is a distributive control system instead of a working copy each developer gets there on local repository, complete with a full history of commits.
* Having a full local history makes git fast since it means you don’t need a network connection to create commits, inspect previous versions of a file, or perform difference between commits.
* Similar to feature branches disturbed development creates a more reliable environment

•**Pull Request:**

* Many source code management rules such as a bit bucket, enhance core git functionally with pull request.
* A pull request is a way to ask another developer to merge one of your branches into their repository. This not only makes it easier for project leads to keep track of the changes but also let the developers’ initiate discussions around their work before integrating it with the rest of the code base.
* When a developer gets stuck with a hard problem, they can open a pull request to ask for the help from the rest of the team. Alternatively, junior developers can also be confident that they are not destroying the entire project by creating pull request as a formal code review.

**•Committed:**

In many sentences, git has come to be expected version control system for new projects. If your team is using git then you won’t have to create new joints because they are already familiar with the distributed development.

**•Faster Release Cycle:**

The ultimate result of feature branches distributed development, pull request and a stable community is a faster release cycle. For example, You might want to configure git to deploy the most recent commit from the developed branch to a test server whenever anyone merge a pull request into it. Combining this kind of bill automation with peer review mean that you have the highest possible confidence in your code as it moves from developed to staging to the production.

1. **Git for marketing team**
2. **For product management**
3. **Git for designers**
4. **Git for costumer support**
5. **Git for human resource**
6. **Git for managing a budget**

**Branching** – A branch in git is an independent line of work (a pointer to a specific commit)

It allows users to create a branch to a code (master branch) and isolate their work. Branches allow you to work on different parts of a project without impacting the main branch.

**Create branches** – The main branch in git is called master branch but we can make branches out of this master branch. All the files present in master can be shown in branch but the file which are created in branch are not shown in master branch. We can also merge both the parent and child branch.

->lecture:5

**Git Lifecycle description**

**Stages in Git lifecycle:** Files in a git project have various stages like creation, modification, refactoring and deletion and so on irrespective of whether this project is tracked by git or not, these phases are still prevalent. However, when a project is under git version control system, they are present in three major git states in addition to these basic ones.

The three git states are as follows:

* **Working directory:** Consider a project in your residing local system. This project may or may not be tracked by it in either case this project directory is called your working directory.
* **Staging area:** It is the playground where you group and organize the files to be committed to git for tracking their versions.
* **Git directory:** Now that files to be committed are grouped and ready in the staging area we can commit these files so we commit this group of files along with a commit message explaining what is a commit about. Apart from the commit messages this step also records the other and a time of a commit. Now a snapshot of the files in the commit is recorded by git. The information related to this commit is stored in the git directory.

|  |  |  |
| --- | --- | --- |
| Working Directory | Staging Area | Git Directory (Rep.) |

CHECK OUT THE PROJECT

|  |  |
| --- | --- |
| Staging Area | Comment |

**REMOTE REPOSITORIES:** It means the clone of the local git repository in the GitHub. And pushing means uploading the commits from local git repository to remote repository hosted in the GitHub.

{Git init} {Git add} {Git commit} {Git push}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Working directory (Untracked by Git) | Working directory (Tracked by Git) | Staging Area | Local Repository | Remote Repository |

(Local) (Remote)