Experiment No 11

Aim: Case Study: GitHub For Version Control.

Theory:

Configuration Management:

Configuration management is a systems engineering process for establishing consistency of a product's attributes throughout life. In the technology world, configuration management is an IT management process that tracks individual configuration items of an IT system. IT systems are composed of IT assets that vary in granularity. An IT asset may represent a piece of software, a server, or a cluster of servers. The following focuses on configuration management as it directly applies to IT software assets and software asset CI/CD.

Software configuration management is a systems engineering process that tracks and monitors software system configuration metadata changes. Configuration management is commonly used in software development alongside version control and CI/CD infrastructure.

Configuration management helps engineering teams build robust and stable systems through the use of tools that automatically manage and monitor updates to configuration data. Complex software systems are composed of components that differ in granularity size and complexity. For a more concrete example consider a microservice architecture. Each service in a microservice architecture uses configuration metadata to register itself and initialize.

Some examples of software configuration metadata are:

- Specifications of computational hardware resource allocations for CPU, RAM, etc.
- Endpoints that specify external connections to other services, databases, or domains
- Secrets like passwords and encryption keys

It's easy for these configuration values to become an afterthought, leading to the configuration becoming disorganized and scattered. Imagine numerous Post-it notes

with passwords and URLs blowing around an office. Configuration management solves this challenge by creating a “sourceof truth” with a central location for configuration.

Git is a fantastic platform for managing configuration data. Moving configuration data into a Git repository enables version control and the repository to act as a source of truth. Version control also solves another configuration problem: unexpected breaking changes. Managing unexpected changes through the use of code review and version control helps to minimize downtime.

Configuration values will often be added, removed, or modified. Without version control, this can cause problems. One team member may tweak a hardware allocation value so that the software runs more efficiently on their personal laptop. When the software is later deployed to a production environment, this new configuration may have a suboptimal effect or may break.

Version control and configuration management solve this problem by adding visibility to configuration modifications. When a change is made to configuration data, the version control system tracks it, which allows team members to review an audit trail of modifications.

Configuration version control enables rollback or “undo” functionality to configuration, which helps avoid unexpected breakage. Version control applied to the configuration can be rapidly reverted to the last known stable state.

Version control:
Version control, also known as source control, is the practice of tracking and managing changes in 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 teams since they help them 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 toolchain 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 everyday 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 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 in how well they handle the renaming and moving 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 change. 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 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, 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 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.

GitHub commands

Getting & Creating Projects

Command	Description
git init	Initialize a local Git repository
git clone ssh://git@github.com/[username]/[repository-name].git	Create a local copy of a remote repository

Basic Snapshotting

Command	Description
git status	Check Status
git add [file-name.txt]	Add a file to the staging area
git add -A	Add all new and changed files to the staging area
git commit -m "[commit message]"	Commit changes
git rm -r [file-name.txt]	Remove a file (or folder)

Branching & Merging

Command	Description
git branch	List branches (the asterisk denotes the current branch)
git branch -a	List all branches (local and remote)
git branch [branch name]	Create a new branch
git branch -d [branch name]	Delete a branch
git push origin --delete [branch name]	Delete a remote branch
git checkout -b [branch name]	Create a new branch and switch to it
git checkout -b [branch name] origin/[branch name]	Clone a remote branch and switch to it

git branch -m [old branch name] [new branch name]	Rename a local branch
git checkout [branch name]	Switch to a branch
git checkout -	Switch to the branch last checked out
git checkout -- [file-name.txt]	Discard changes to a file
git merge [branch name]	Merge a branch into the active branch
git merge [source branch] [target branch]	Merge a branch into a target branch
git stash	Stash changes in a dirty working directory
git stash clear	Remove all stashed entries

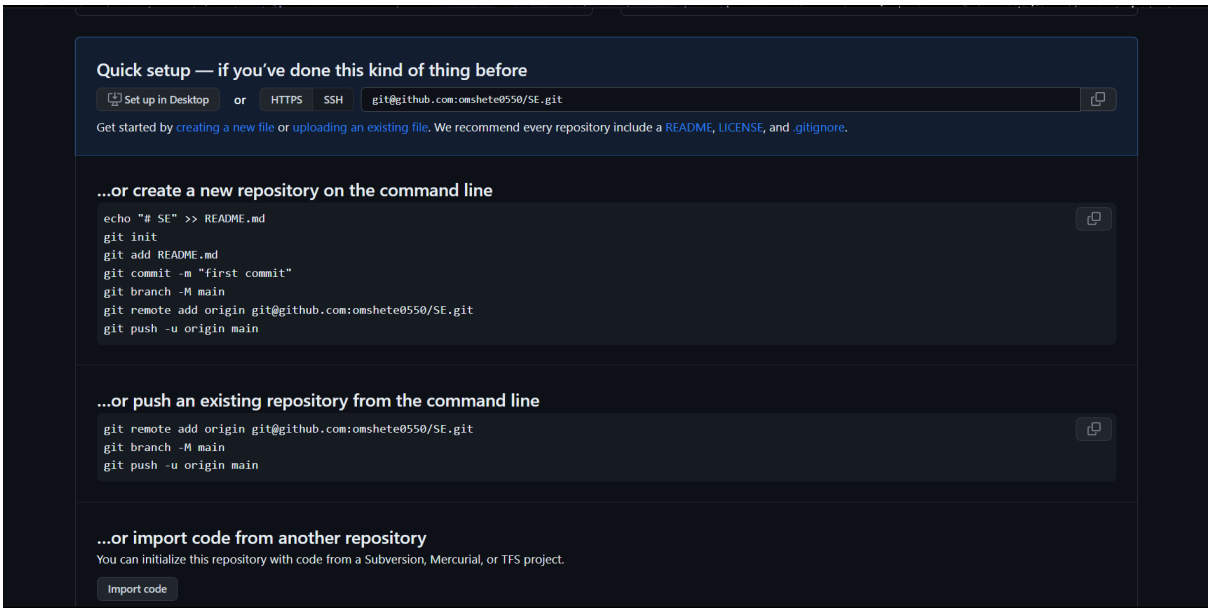
Sharing & Updating Projects

Command	Description
git push origin [branch name]	Push a branch to your remote repository
git push -u origin [branch name]	Push changes to the remote repository (and remember the branch)
git push	Push changes to the remote repository (remembered branch)
git push origin --delete [branch name]	Delete a remote branch
git pull	Update the local repository to the newest commit
git pull origin [branch name]	Pull changes from the remote repository
git remote add origin ssh://git@github.com/[username]/[repository-name].git	Add a remote repository
git remote set-url origin ssh://git@github.com/[username]/[repository-name].git	Set a repository's origin branch to SSH

Inspection & Comparison

Command	Description
git log	View changes
git log --summary	View changes (detailed)
git log --oneline	View changes (briefly)
git diff [source branch] [target branch]	Preview changes before merging

1. Creating a Repository:



2. Initial Commit

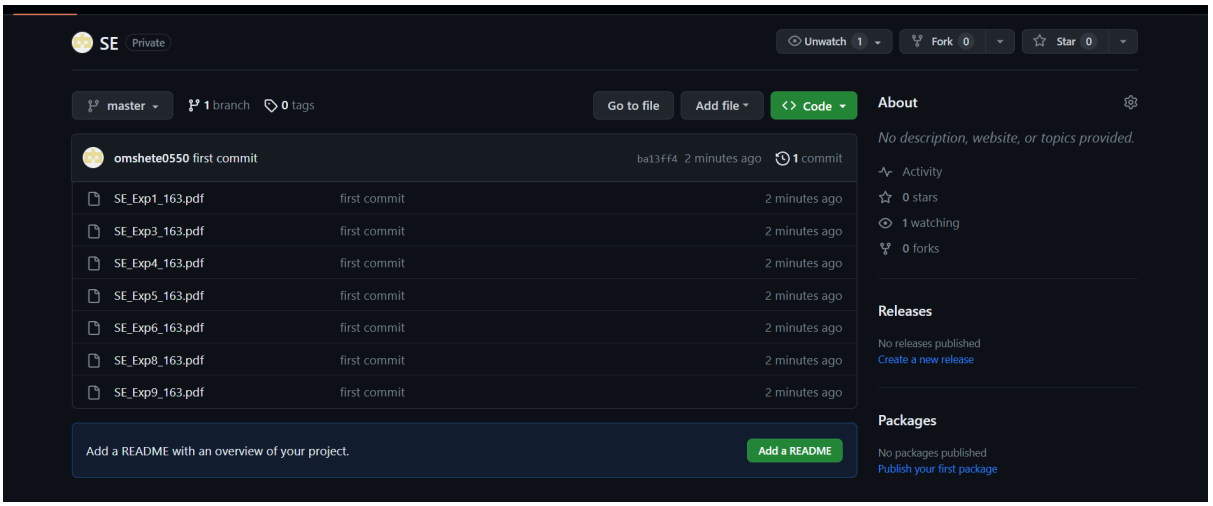
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS E:\College\Sem-V\SE_EXP> git init
Initialized empty Git repository in E:/College/Sem-V/SE_EXP/.git/
PS E:\College\Sem-V\SE_EXP> git add -A
PS E:\College\Sem-V\SE_EXP> git commit -m "first commit"
[master (root-commit) ba13ff4] first commit
 7 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 SE_Exp1_163.pdf
 create mode 100644 SE_Exp3_163.pdf
 create mode 100644 SE_Exp4_163.pdf
 create mode 100644 SE_Exp5_163.pdf
 create mode 100644 SE_Exp6_163.pdf
 create mode 100644 SE_Exp8_163.pdf
 create mode 100644 SE_Exp9_163.pdf
PS E:\College\Sem-V\SE_EXP> git remote add origin git@github.com:omshete0550/SE.git
PS E:\College\Sem-V\SE_EXP> git push -u origin master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 20 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 3.08 MiB | 1.84 MiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:omshete0550/SE.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
PS E:\College\Sem-V\SE_EXP> git log
commit ba13ff4461fde1fa486850b12b08ee7558d5741a (HEAD -> master, origin/master)
Author: Om <omshete0550@gmail.com>
Date:   Fri Oct 13 21:16:19 2023 +0530

    first commit
PS E:\College\Sem-V\SE_EXP>
```

3. Commit History



4. Git Commands

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS E:\College\Sem-V\SE_EXP> git --help
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone     Clone a repository into a new directory
   init      Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add       Add file contents to the index
   mv        Move or rename a file, a directory, or a symlink
   restore   Restore working tree files
   rm        Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect    Use binary search to find the commit that introduced a bug
   diff      Show changes between commits, commit and working tree, etc
   grep      Print lines matching a pattern
   log       Show commit logs
   show      Show various types of objects
   status    Show the working tree status
```

```
grow, mark and tweak your common history
   branch    List, create, or delete branches
   commit    Record changes to the repository
   merge     Join two or more development histories together
   rebase    Reapply commits on top of another base tip
   reset     Reset current HEAD to the specified state
   switch    Switch branches
   tag       Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch     Download objects and refs from another repository
   pull      Fetch from and integrate with another repository or a local branch
   push      Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
PS E:\College\Sem-V\SE_EXP>
```