JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY

BIT 2321 SOFTWARE ENGINEERING 11

GIT ASSIGNMENT

**GROUP MEMBERS**

1. JAMES VITALIS ANYONA : SCT221-C004-0025/2019.
2. MOHAMED NOOR HUSSEIN : SCT221-C004-0750/2017
3. ABDIRIZAK HASHI : SCT221-C004-0747/2017

**INTRODUCTION**

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

The version Control System (VCS) allows you to track the iterative changes you make to your code. This makes it easier to experiment with new ideas but always have the option to revert to a specific past version of the code that was used to generate particular results. Furthermore, you can record messages as you save each successive version so that anyone else reviewing the development history of the cod is able to understand the rational for the given edits.

The VCS also facilitates collaboration. Using a VCS, your collaborators can make and save changes to the code and one can automatically incorporate these changes to the main code base. The collaborative aspect is enhanced with the emergence of websites that host version-controlled code.

Some of the VCS tools features are;

* Concurrency management.
* Version Control
* Synchronization.

Some of the reasons why we need a software tool like Git are;

* Useful when multiple developers are working on software that’s continually updating.
* Where multiple branches are introduced in a software project
* Where the team is geographically distributed and working concurrently.
* If we have changes in user requirements, policy, budget and schedules that needs to be accommodated.
* The software is supposed to run on multiple machines and different operating System
* Where co-ordination among stakeholders is needed.
* If there’s need to control cost that arise as a result of changes.

Some of the key tasks that take place in a Version Control System are;

* Configuration Identification – Method of determining the size of software e.g. source code, modules, test cases.
* Identifying each CSCI (Computer Software Configuration Items)
* Baseline – This is the accepted version of the software configuration item. It is designated and fixed at a specific time while adhering the SCM process.
* Change Control – Procedural method that ensures quality and consistency when changes are made in the configuration object.
* Configuration Status Accounting
* Traces each release during SCM process.
* Monitors status of change requests.
* Complete Listing of all changes since the last baseline.
* Allow tracking of changes
* Allows checking previous release.
* Configuration audit and review – All the software products satisfy the baseline needs.

Some of the key participants in the Version Control System Process are:

* Configuration Manager.
* Developer.
* Auditor.
* Project Manager – recognizes issues in the scm process
* Users

**Common terms used while using GIT.**

* **Version Control System (VCS)**: a program that tracks changes to specified files over time and maintains a library of all past versions of those files
* **Git**: a version control system
* **repository (repo)**: folder containing all tracked files as well as the version control history
* **commit**: a snapshot of changes made to the staged file(s); to save a snapshot of changes made to the staged file(s)
* **stage**: the staging area holds the files to be included in the next commit; to mark a file to be included in the next commit
* **track**: a tracked file is one that is recognized by the Git repository
* **branch**: a parallel version of the files in a repository
* **local**: the version of your repository that is stored on your personal computer
* **remote**: the version of your repository that is stored on a remote server; for instance, on GitHub
* **clone**: to create a local copy of a remote repository on your personal computer
* **fork**: a copy of another user’s repository on GitHub; to copy a repository; for instance, from one user’s GitHub account to your own
* **merge**: to update files by incorporating the changes introduced in new commits
* **pull**: to retrieve commits from a remote repository and merge them into a local repository
* **push**: to send commits from a local repository to a remote repository
* **pull request**: a message sent by one GitHub user to merge the commits in their remote repository into another user’s remote repository

**ABOUT GIT**

**Branching and Merging**

The Git feature that really makes it stand apart from nearly every other SCM out there is its branching model.

Git allows and encourages you to have multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds.

This means that you can do things like:

* **Frictionless Context Switching**. Create a branch to try out an idea, commit a few times, switch back to where you branched from, apply a patch, switch back to where you are experimenting, and merge it in.
* **Role-Based code lines**. Have a branch that always contains only what goes to production, another that you merge work into for testing, and several smaller ones for day to day work.
* **Feature Based Workflow**. Create new branches for each new feature you're working on so you can seamlessly switch back and forth between them, then delete each branch when that feature gets merged into your main line.
* **Disposable Experimentation**. Create a branch to experiment in, realize it's not going to work, and just delete it - abandoning the work—with nobody else ever seeing it (even if you've pushed other branches in the meantime).

## Small and Fast

**Git is fast**. With Git, nearly all operations are performed locally, giving it a huge speed advantage on centralized systems that constantly have to communicate with a server somewhere.

Git was built to work on the Linux kernel, meaning that it has had to effectively handle large repositories from day one. Git is written in C, reducing the overhead of runtimes associated with higher-level languages. Speed and performance has been a primary design goal of the Git from the start.

## Distributed

One of the nicest features of any Distributed SCM, Git included, is that it's distributed. This means that instead of doing a "checkout" of the current tip of the source code, you do a "clone" of the entire repository.

### Multiple Backups

This means that even if you're using a centralized workflow, every user essentially has a full backup of the main server. Each of these copies could be pushed up to replace the main server in the event of a crash or corruption. In effect, there is no single point of failure with Git unless there is only a single copy of the repository.

### Any Workflow

Because of Git's distributed nature and superb branching system, an almost endless number of workflows can be implemented with relative ease.

#### Subversion-Style Workflow

A centralized workflow is very common, especially from people transitioning from a centralized system. Git will not allow you to push if someone has pushed since the last time you fetched, so a centralized model where all developers push to the same server works just fine.

## Data Assurance

The data model that Git uses ensures the cryptographic integrity of every bit of your project. Every file and commit is check summed and retrieved by its checksum when checked back out. It's impossible to get anything out of Git other than the **exact bits you put in**.

## Staging Area

Unlike the other systems, Git has something called the "staging area" or "index". This is an intermediate area where commits can be formatted and reviewed before completing the commit.

One thing that sets Git apart from other tools is that it's possible to quickly stage some of your files and commit them without committing all of the other modified files in your working directory or having to list them on the command line during the commit.



This allows you to stage only portions of a modified file. Gone are the days of making two logically unrelated modifications to a file before you realized that you forgot to commit one of them. Now you can just stage the change you need for the current commit and stage the other change for the next commit. This feature scales up to as many different changes to your file as needed.

Of course, Git also makes it easy to ignore this feature if you don't want that kind of control — just add a '-a' to your commit command in order to add all changes to all files to the staging area.



## Free and Open Source

Git is released under the [GNU General Public License version 2.0](http://opensource.org/licenses/GPL-2.0), which is an [open source license](http://www.opensource.org/docs/osd). The Git project chose to use GPLv2 to guarantee your freedom to share and change free software---to make sure the software is free for all its users.

**REFERENCES**

* <https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004668>
* <https://git-scm.com/about>
* <https://guides.github.com/introduction/git-handbook/>
* <https://try.github.io/>
* <https://www.atlassian.com/git/tutorials>