**Capital University of Science and Technology, Islamabad**

**Department of Software Engineering**

**Lab 11: GIT (Continuous Integration Tool)**

**Lab 11: GIT (Continuous Integration Tool)**

# Introduction

Continuous integration is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. Continuous integration most often refers to the build or integration stage of the software release process and entails both an automation component (e.g., a CI or build service) and a cultural component (e.g., learning to integrate frequently). The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

# **Relevant Lecture Reading**

https://git-scm.com/

# Activity Time Boxing

|  |  |  |  |
| --- | --- | --- | --- |
| **Task No.** | **Activity Name** | **Activity time** | **Total Time** |
| Task 1 | Home task evaluation | 40 min | 40 min |
| Task 2 | Concept Map: | 40 min | 40 min |
| Task 3 | Walkthrough Tasks | 40 min | 40 min |
| Task 4 | Practice Tasks | As per time specified for each task | 40 min |
| Task 5 | Evaluation Task (Unseen) | 20 min for assigned task | 20 min |

# Objective of the Experiment

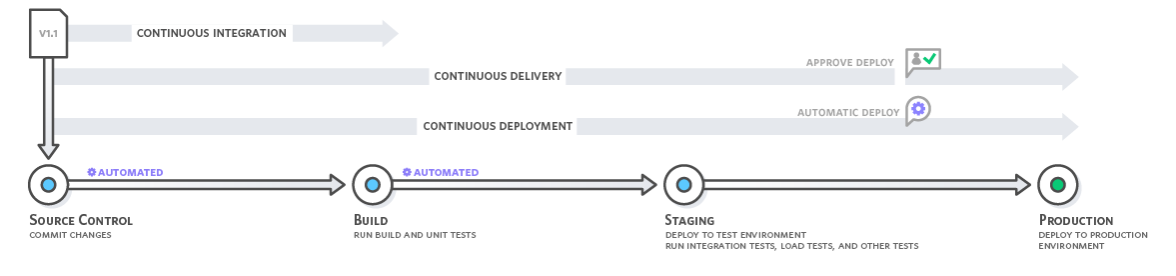
To learn and practice about the git and its commands

# Concept Map

* 1. Why Continuous Integration is needed?

In the past, developers on a team might work in isolation for an extended period of time and only merge their changes to the master branch once their work was completed. This made merging code changes difficult and time-consuming, and also resulted in bugs accumulating for a long time without correction. These factors made it harder to deliver updates to customers quickly.

* 1. How Continuous Integration Works?

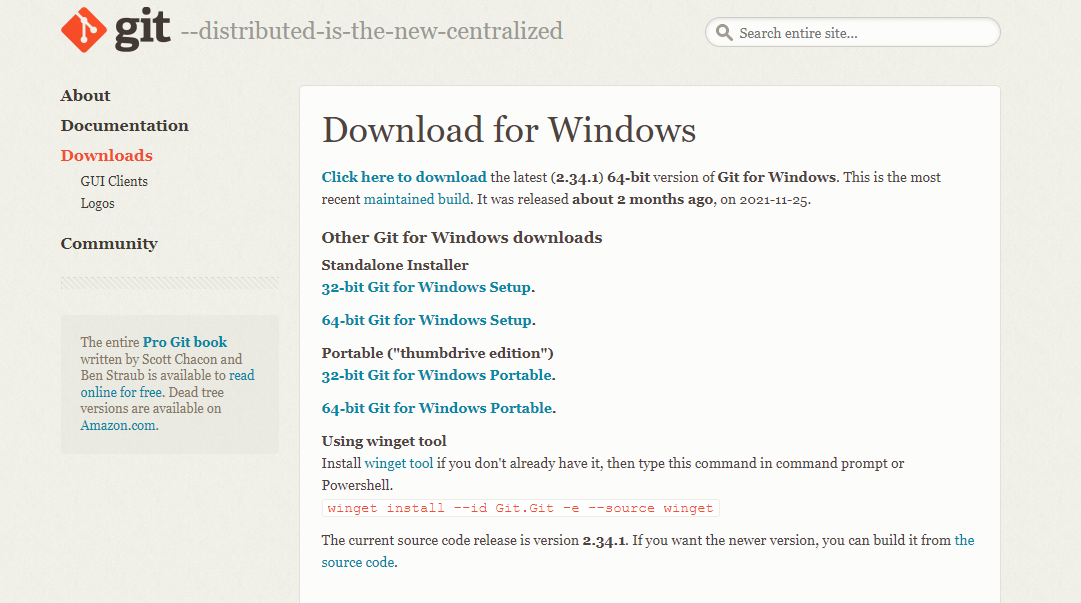
With continuous integration, developers frequently commit to a shared repository using a version control system such as Git. Prior to each commit, developers may choose to run local unit tests on their code as an extra verification layer before integrating. A continuous integration service automatically builds and runs unit tests on the new code changes to immediately surface any errors.

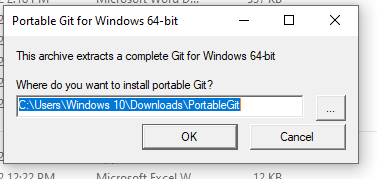
* 1. What is Git?

Git is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).

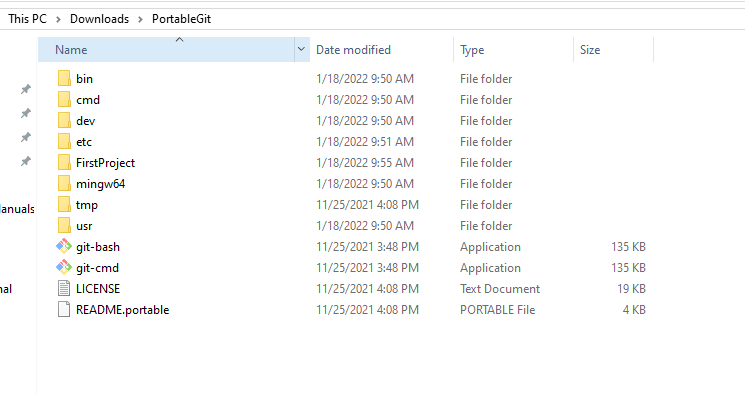
* 1. Download and Install Git

Go to the following link and download the portable edition of Git according to your operating system.

<https://git-scm.com/download/win>

Once download is completed, install the downloaded file.

After installation, a folder named “PortableGit” will be created having following folders.



# Tools and Procedure

* 1. Tools
* Java Development Kit (JDK)
* NetBeans IDE.

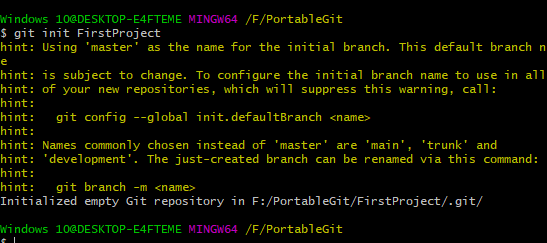
# Walkthrough

* 1. Git Commands
     1. Git init

The git init command creates a new Git repository. It can be used to convert an existing, unversioned project to a Git repository or initialize a new, empty repository. Most other Git commands are not available outside of an initialized repository, so this is usually the first command you'll run in a new project.

Executing git init creates a **.git** subdirectory in the current working directory, which contains all of the necessary Git metadata for the new repository. This metadata includes subdirectories for objects, refs, and template files. A HEAD file is also created which points to the currently checked out commit.

Syntax: **git init RepositoryName**

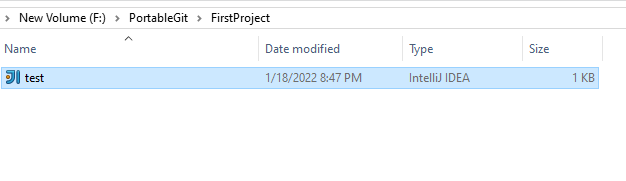


* + 1. touch

This command is used to create a file within the current working directory



Above command will create a file “test.java” in the current directory.



* + 1. git add

The git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded.



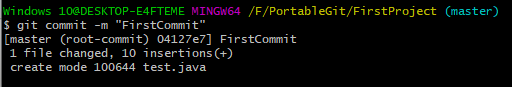
For adding multiple files:



* + 1. **git commit**

The git commit command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as “safe” versions of a project—Git will never change them unless you explicitly ask it to. Prior to the execution of git commit, The git add command is used to promote or 'stage' changes to the project that will be stored in a commit. These two commands git commit and git add are two of the most frequently used.

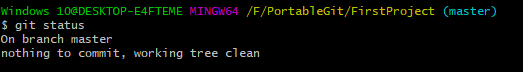
Syntax: **git commit -m “Title for commit”**



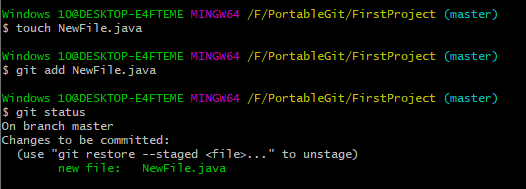
* + 1. **git status**

This command returns the current state of the repository. git status will also return the current working branch. If the file is in the staging area but not committed, it is also shown with git status. If all files are committed, then it will show nothing to commit.

Syntax: **git status**

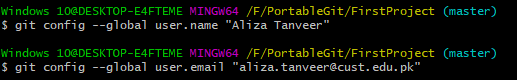


If we create another file and add it only to staging area then following text will be shown:



* + 1. git config

The git config command is a convenience function that is used to set Git configuration values on a global or local project level. These configuration levels correspond to .gitconfig text files. Executing git config will modify a configuration text file. We'll be covering common configuration settings like email, username, and editor.



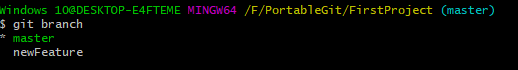
* + 1. **git branch**

This command is used to create a new branch in the repository. A branch is used whenever you want to add a new feature.

Syntax: **git branch branchName**



To view all the branches, write command:

**git branch**

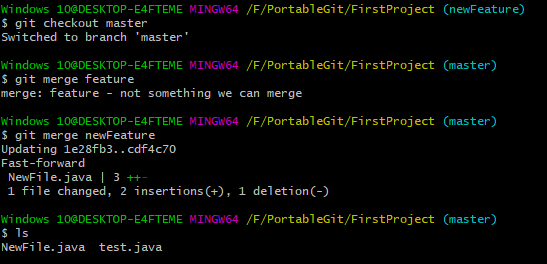
* + 1. **git checkout**

If you want to move from the master branch to any other branch, type:



* + 1. **git merge**

It is used to merge two branches. If you want to merge feature branch with the master branch then first move to master branch then write following command.



* + 1. **git branch -d**

This command is used to delete a branch. Once you have merged a branch with the master branch then you have to delete that branch.



* 1. Connecting to a remote repository

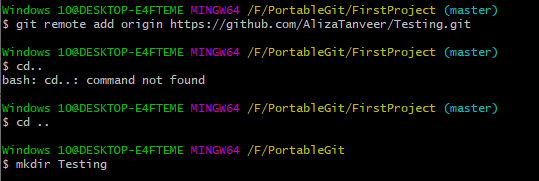
For connecting your local repository to a remote repository use command git remote add

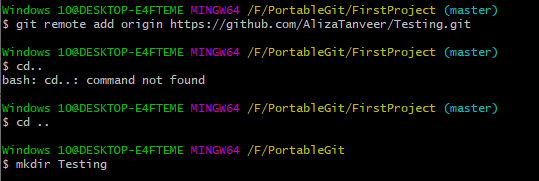
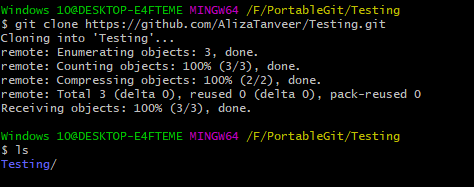


* + 1. **git clone**

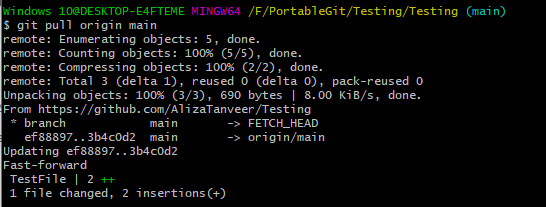
This command creates a local working copy of an existing remote repository.

For that create a new directory and make a clone of your remote repository in that local directory.



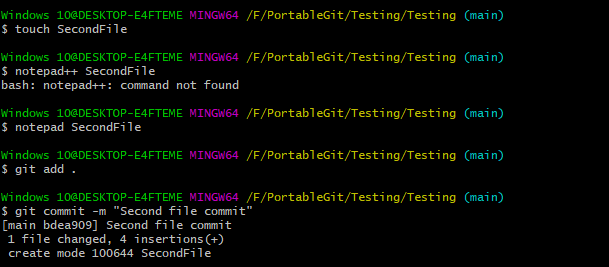
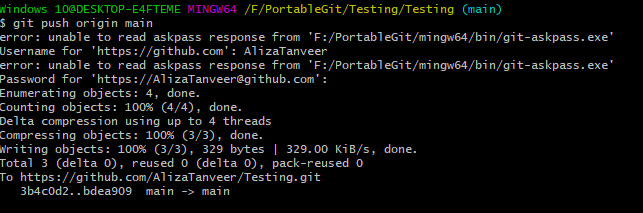


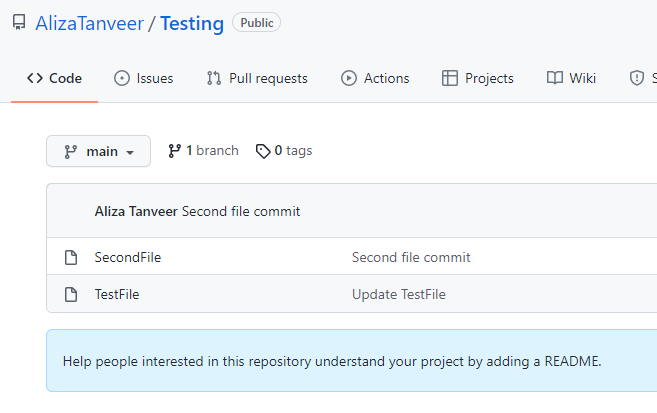
* + 1. **git pull**

This command is used to get all changes in the remote repository into your local clone.

* + 1. **git push**

This command is used to push all the changes made in the local repository to the remote repository.





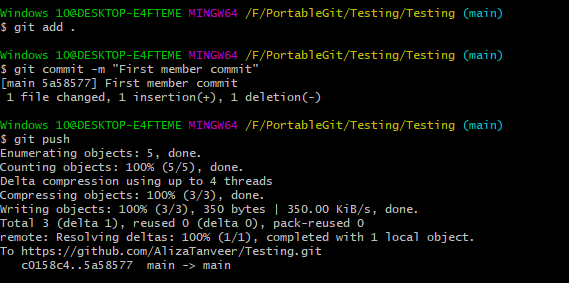
* + 1. **git revert**

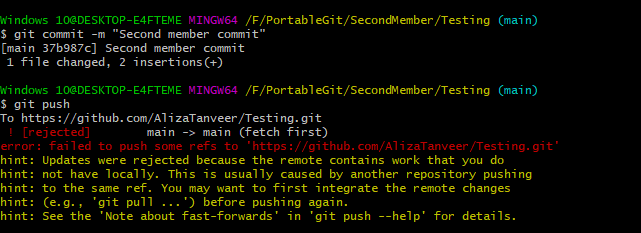
This command helps you to roll back to the previous version of the repository

* 1. Centralized Git workflow

A centralized Git workflow enables all team members to make changes directly to the main branch, with every change logged in a running history. A [centralized workflow](https://about.gitlab.com/topics/version-control/what-is-centralized-version-control-system/) involves every contributor committing to the main branch without using any other branch. This strategy works well for small teams, because team members can communicate so that multiple developers aren’t contributing to the same piece of code simultaneously. Centralized workflow can be seamless if team members communicate well, but there are limitations. If multiple developers commit to the same branch, it’s challenging to find a stable moment to release changes. Consequently, developers must keep unstable changes local until they’re ready for release.

* + 1. What is the benefit of a centralized Git workflow?

After developers apply a stash and solve any merge conflicts, they can just commit as usual without dealing with [automatic merge commits](https://about.gitlab.com/blog/2020/03/25/git-merge-fifteen-year-git-party/), unless someone pushed their changes at the same time. Because this strategy is simple, it is well-suited for small teams, Git beginners, and projects that don’t get a lot of updates.



* 1. **Feature branching Git workflow**

Every feature gets its own branch when developers commit to this workflow. Rather than commit directly to the main branch, developers create a branch, make changes, and then merge it into main.

Ideally, a branch should have a lifespan of a few hours. The longer the branch lives, the higher the risk to find integration conflicts when merging back to main. After all, at this scale, there are plenty of teams working on other branches and directly streaming changes to the main branch, incrementing entropy and chances of running into conflict with local changes.

* + 1. What is the benefit of a feature branching Git workflow?

This Git workflow has the benefit of keeping a clean main branch that isn’t polluted with unfinished features. Teams of any size can use this feature branching, because it permits multiple developers to work on the same feature simultaneously. Software that’s still in development sees the most benefit from feature branching, but this workflow can be used for more mature applications as well.

# **Practice Tasks**

* 1. Practice Task 1

# Evaluation Task

Given by instructor during lab.

# Further Reading

1. Big Java Early Objects

Author: Cay S. Horstmann [6th\_ edition]

1. “Core Java- Volume 1- Fundamentals

Author: Cay S. Horstmann, Gary Cornell,[9th edition]  
Desc: ,2013, ISBN-13: 978-0-13-708189-9

1. Clean code: a handbook of Agile Software Craftsmanship,

Author: Robert C. Martin,