LETTERKENNY INSTITUTE OF TECHNOLOGY

ASSIGNMENT COVER SHEET

To Be Completed By The Student

Lecturer’s Name: Maria Griffin Assessment Title: Source code version control management

Submission Date: 04-12-22

Student’s Name: Surendran Gopinathan Id. Number:\_L00171047

Course / Stage Masters in DevOps

Subject/Module: DevOps Software Engineering

Word Limit: Actual Word Count: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

I confirm that the work submitted has been produced solely through my own efforts.

Student’s signature: Surendran Gopinathan Date: 09/11/2022

|  |
| --- |
| **Note**: **PENALTIES**   * The total marks available for an assessment will be reduced by 15% for work submitted up to one week late. The total marks available are reduced by 30% for work up to two weeks late. * Assessment work received more than two weeks late, without prior approval by the lecturer will receive a mark of zero. * Marks awarded will be reduced by 10 % if submitted work is greater than 10% above or below the assigned word limit. * A further hard or electronic copy of your submitted work may be requested, and therefore you must keep a copy on disc. * Incidents of alleged plagiarism and cheating are dealt with in accordance with the Institute’s Assessment Regulations   **Plagiarism:** Presenting the ideas, words of someone else without proper acknowledgement. Refer to the Institutes’ procedures and guidelines for the assessment of learners. |

Introduction

In this lab the main purpose is to understand how version control works and how

Source code management tools can be used to manage a project effectively.

# Aims/Objectives

1. Understand how version control using git.
2. Explore how gitlab can be used to manage projects.
3. Understand various git commands and their use case.
4. Learn how to use milestones and labels to manage a sprint effectively.
5. Investigate gitlab ci/cd and its functionality.
6. Compare different version control and project management tools.

## Method

1. Installed git on the local machine.
2. Checked git version in terminal to confirm git installation.
3. Created a new public project in gitlab.
4. Created new sprint/milestone and appropriate labels in gitlab.
5. Created multiple issues for the project and allocated a time estimate based on work required.
6. From the terminal, initialized a git repository and linked remote repository with local repository using http url. Used git pull command to fetch and merge remote main branch with local main branch.
7. Created merge request for an issue in gitlab which automatically creates a new branch.
8. Used git commands from terminal to pull remote changes to local machine and checkout to the new branch created.
9. Added necessary java code and used git commands to stage the changes and created a commit with proper commit message and pushed the code to remote repository.
10. Reviewed the changes in gitlab merge request and merge the changes with master if there are no merge conflicts.
11. Created merge request for another issue to work on adding more functionality.
12. Used necessary git commands to remove files from staging area or go back previous commits using revert/reset. Used git stash to store temporary changes and use them later if needed or drop them if they are not needed.
13. Deleted local branches which are no longer present in remote repository as they were deleted during merge request.
14. Created gitignore file and used it to ignore one xml by configuring it in the file.
15. Used IDE present in gitlab to make changes to the code from browser.
16. Analyzed the status of the sprint using issue boards and milestone section.
17. Understood how gitlab CI/CD works through yml file.

## Results

1. Git was downloaded from their website and was installed in the local machine. Used git version command in the terminal to check whether git was installed successfully.
2. New project created in gitlab (figure 1).
3. There are two ways to create local git repository. We can either use git clone (figure 2) which will create a local repository and automatically link it with remote repository, or we can initialize a local repository using git init and then use git remote (figure 3) to link gitlab repository.
4. I have created a new milestone and necessary labels (figure 4) which can used to track the status of issues in the issue board.
5. In figure 5,6 and 7, I have created list of issues for specific functionality, and I have added the time estimate based on difficulty.
6. I have provided the issue board view (figure 8) and milestone preview (figure 9) at the beginning of the sprint.
7. I have created a maven project and I am initializing the repository and pushing the project to gitlab from local machine (figure 10,11,12 and 13).
8. In figure 15, I have created a new merge request which automatically created a branch for me where I can add necessary code.
9. I am using git pull to pull all latest changes in remote and in figure 16, I am using git branch command to view list of branches and I am checking out to new branch.
10. I am pushing all the changes made on new branch to remote repository (figure 17).
11. I am reviewing the merge request in gitlab and merging it in figure 18 and 19.
12. Providing the amount of time taken to code and reviewing the issue board to find the issue I need to work on next (figure 20,21,22 and 23).
13. I am running “git checkout -- .” to remove unstaged changes from repository in figure 24. This reverts all code changes that were made.
14. I am executing “git restore --staged” to remove changes from staging area in figure 25. The code will still be untouched and can be modified.
15. In figure 26, I am using git stash to stash changes made after last commit. This can also be used to remove changes from staging area. I can drop the stash or use them later if needed.
16. Using web IDE present in gitlab to make changes directly from the browser in figure 27.
17. In figure 28 and 29, I am creating a .ignore file in my local repository which will make git skip the files mentioned and not push them to remote repository.
18. Taking look at list of all commits and its metadata using git log command.
19. In figure 32, I am using git reset to go back to a previous commit using the commit id.
20. I am taking look at the milestone status in figure 33 and overall status of the current sprint from the issue board in figure 34. This provides me a clear overview about sprint.

## Conclusion

I found git to be a better version control system when compared with svn and other options. Even though svn has few advantages, The fact that git is distributed and can be worked offline and has better branching logic makes it the perfect choice right now. Most of the companies are already moving away from svn and also git has a vibrant community using it and it is open source as well.

Git installation is very straight forward. I just had to run a setup after downloading windows installer from git website and it installs very quickly. It installs git bash along with it which comes in handy when we need to create local repositories and very good theme when compared to command prompt.

Since git is distributed system, it is a very powerful tool as developers can work on it even when the central system is offline. They can continue to make commits in their local machines and push the code to central repository when it is back up. The local repository also has history about all the commits and developer can view any previous commits by checking out the code using the commit id. Git is also useful if multiple developers need to work on same project at same time.

Git is best worked from the command line as the quality of git GUI Is not that great. The basic git commands to initialize or clone a repository, move changes to the staging area, make commits, pushing and pulling changes are very straight forward and easy to understand. Few important git terminologies are HEAD, master/main, stash, origin, rebase, squash, tag and merge request. Since git has all the history when we clone it in the local machine, developer can always go back to previous commits by referring the logs and look at the code. Developer can even revert the code back to previous commit using commit id if needed. Git also stores the information about the developer who is making the commits to the project. This can be used to track or find out who has worked on the specific piece of functionality. Git also provides us ways to generate tags which represents version of branch at that time. Tags are used so that we can recreate the version when needed. Usually before migrating changes to higher environments, a tag will be created for the project and this tag number will be used throughout the deployment steps. In future, if we need to fix a bug in production, developer can just checkout the tag that was deployed and make changes and deploy the tags to production. The master branch should always be protected and only the application owner or maintainer of the project should have access to makes changes to master.

I have used gitlab as my project management tool and have used the features present in it to effectively manage a project and track changes. It is an open-source project and has a very good build-in CI/CD pipeline which can be leveraged later if it is needed. The UI of the tool also looks very neat and does not flood us with too much information. The interface used to view issue boards and milestones/sprints are very smooth and can be used to track the status of the sprint during stand-up meetings. Every changes made to the project code base should always be done via merge requests and the merge request should be reviewed by reviewer and can either be merged or assigned back to developer. If there are any merge conflicts, they will be shown in the merge request section and developer can resolve the conflicts and assign the merge request to the reviewer. We can create issues and we can attach a label and assign a milestone to the issue. We can also provide a time estimate for the issue. It is good practice to mention the requirements/purpose clearly in the issue description. Gitlab offers set of labels, but we can also create custom labels such as in progress, bug, documentation and review. These labels can be assigned to an issue and can be viewed in the issue board. These features add value to the tool as they can be used for many other reasons not just to store projects. Gitlab issues can also be linked with each other, and this can be used to link issues that are related or based on same user story. Gitlab also provides us a way to mention developer usernames in the issue or any other section. The developer will get an email and they can always use this to provide quick updates on the issue. Gitlab also provides us with web IDE which developer can use and make changes to the code directly from the browser.

Github is also another project management tool which is used by many companies. Gitlab and github both offer same functionality but there are few key differences. Github is a bit fast, but it does not have really good CI/CD pipeline when compared to gitlab. Gitlab has very good integrated CI/CD pipeline and has been in market for some time now. Github actions provides same CI/CD functionality, but they are not as good as gitlab. Gitlab is also open source whereas github is owned by microsoft which matters to few companies.

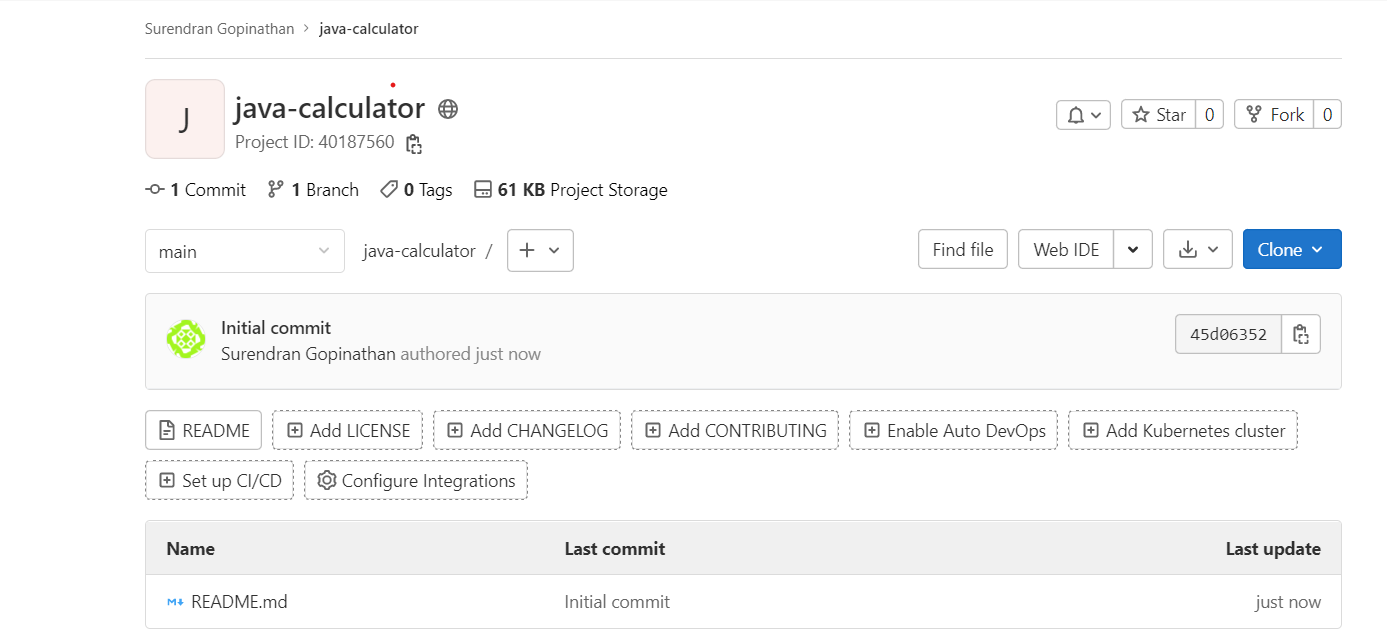
Gitlab project creation also provides us with option to create a pre-populated project with necessary files to get started. This feature is not present in github. This feature adds value and helps the new developers working on the project as they will get clear understanding about folder structure. I felt the way gitlab handles merge requests is very simple and easy compared to github. Gitlab creates a branch for us when we create a merge request but in github we have create branch manually and then add it to the pull request. There is no option to specify time spent in github issues which is a very good feature to track time spent. I need to manually delete the branch used in pull request whereas in gitlab, I can set the flag when I close the merge request and the branch will be deleted. Gitlab also has options to open a project directly in vscode or intellij which is not present in github.

Gitlab also offers CI/CD templates that are already made for specific applications/deployment types such Docker or Maven. Developers can just use the template and change or add anything they want as per requirement. Git commits and merge requests can be made to trigger CI/CD pipelines automatically to deploy code to necessary environments. The CI/CD jobs can also be scheduled periodically as per requirement. Gitlab CI/CD works using gitlab runners and executors. The runners can be shared runner provided by gitlab or we can run these runners in our machine which provides us better performance and security.

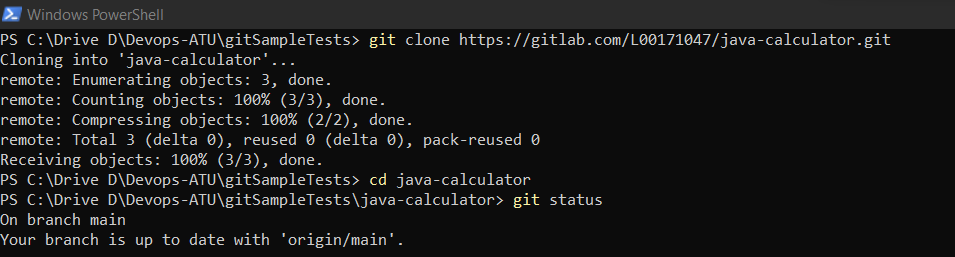
The development of large applications has become a bit simpler to manage because of using git repositories and using necessary project management tools such as gitlab or github to host their project. This also goes hand in hand with agile methodology as we make changes to code in sprints. Since gitlab board offers us similar functionality as kanban boards or any other incident management applications such as jira, this eliminates the need for additional tools which might decrease the cost. The CI/CD is good in gitlab which also eliminates the need to use deployment applications such as Jenkins. Gitlab is not just a source code management tool. It aims to be a one point stop for software development life cycle and devops tools in single application.

## **Appendices**

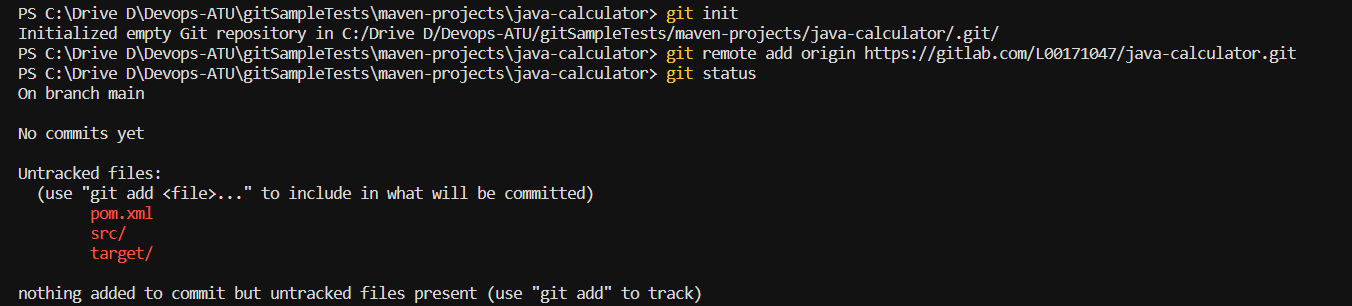
### Figure 1 – New gitlab project



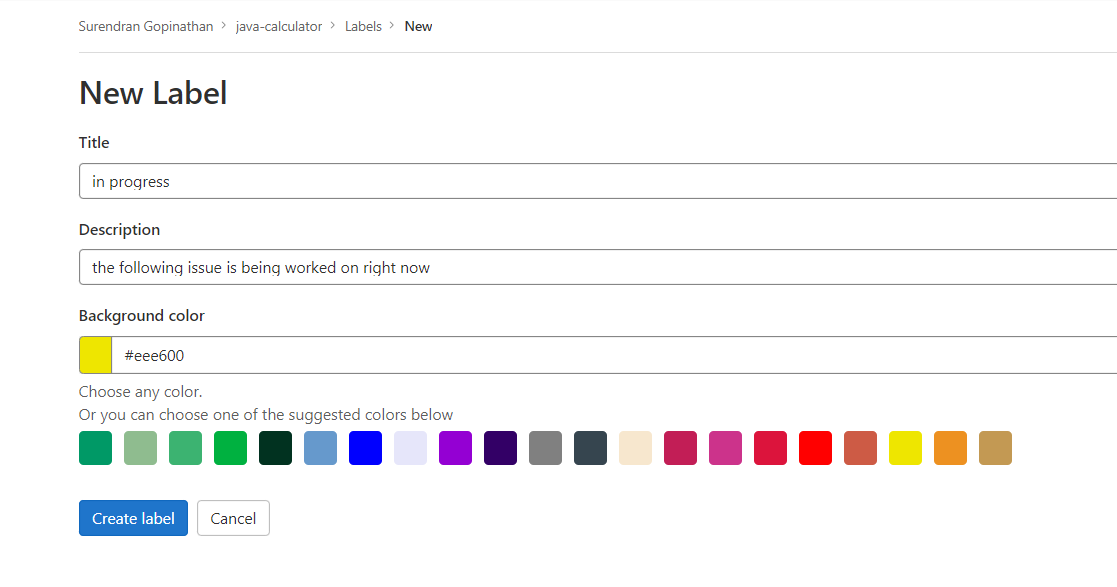
### Figure 2 – Using git clone to create local repository



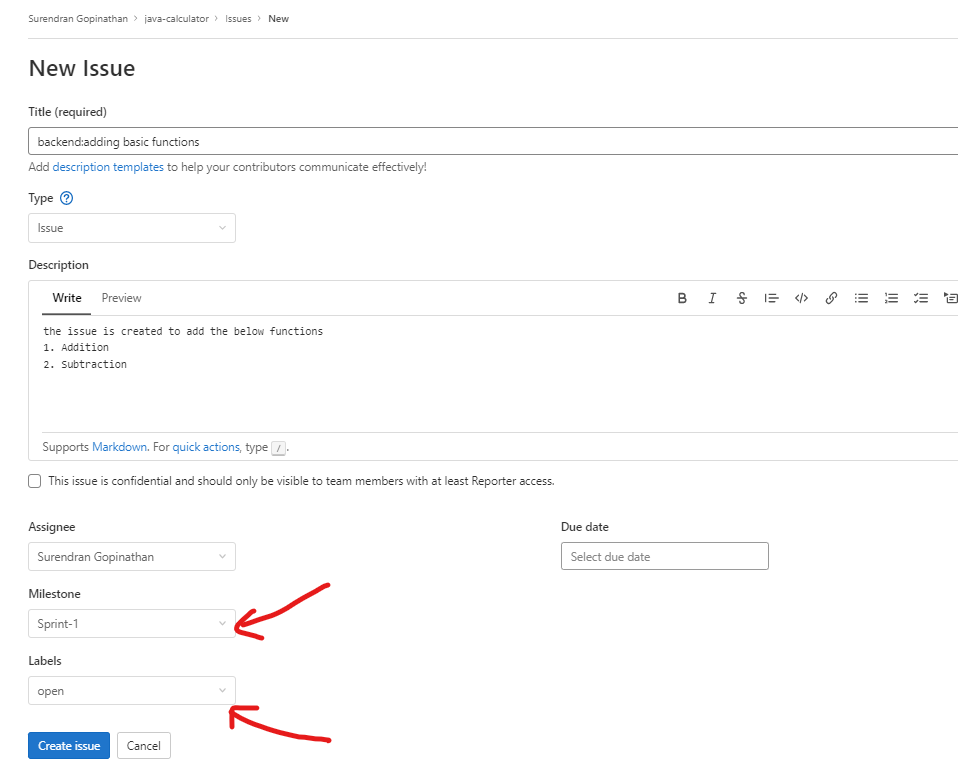
### Figure 3 – Using git init and git remote to create local repository



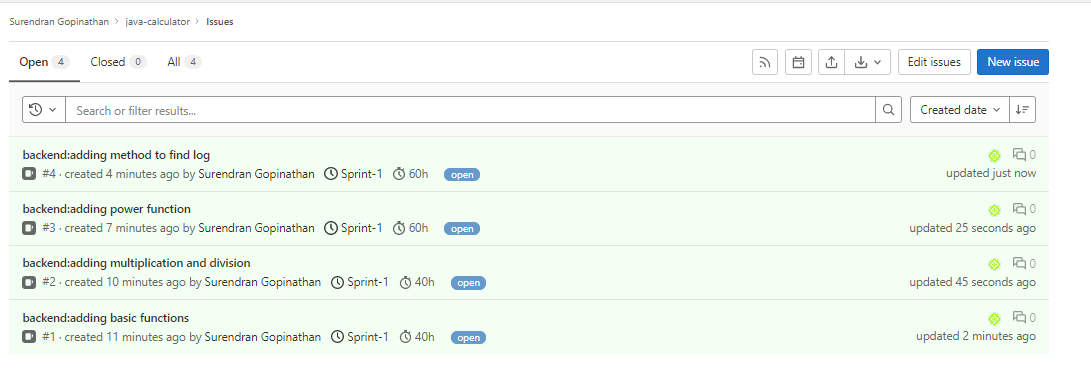
### Figure 4 – Creating labels in gitlab



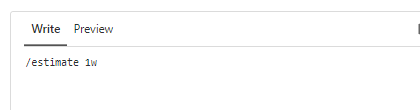
### Figure 5 – Creating new issue in gitlab



### Figure 6 – List of issues created

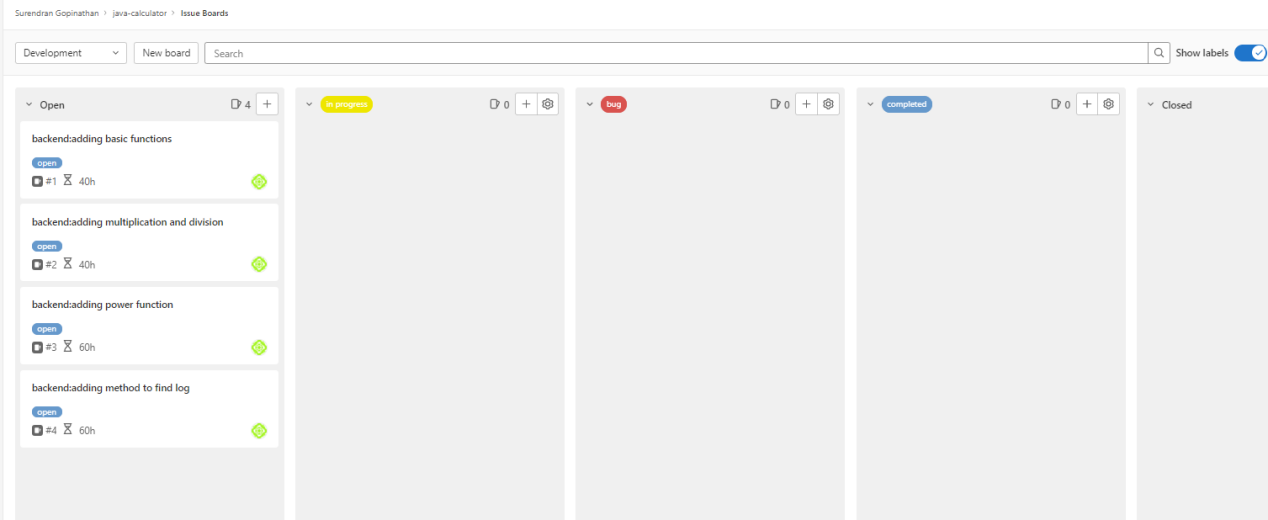


### Figure 7 – Adding time estimate for an issue

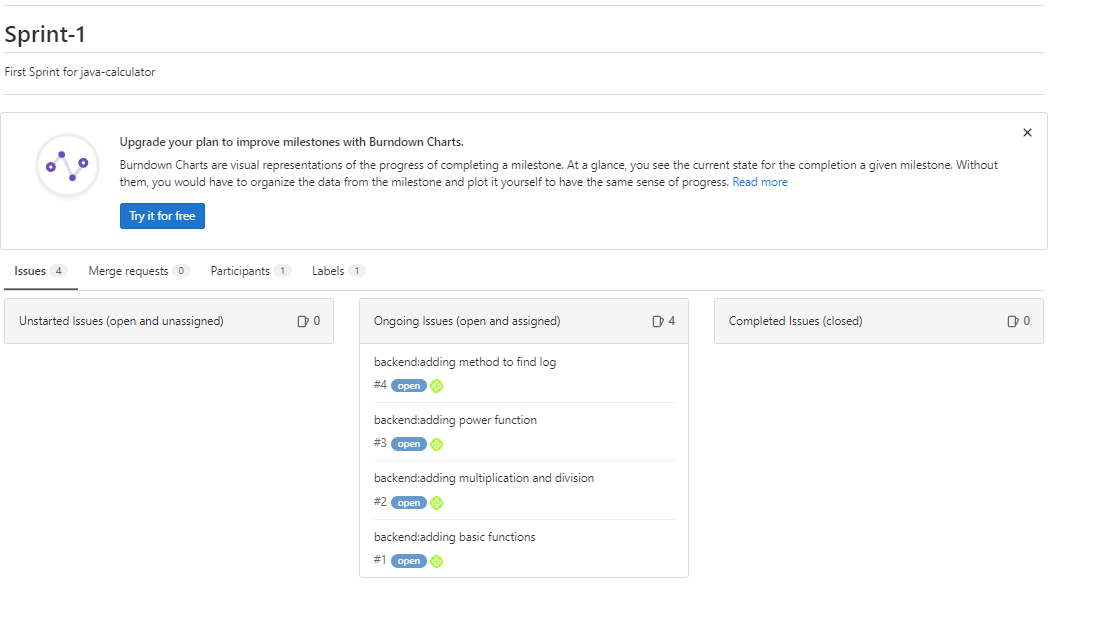


gitlab-issue-added-time-estimate

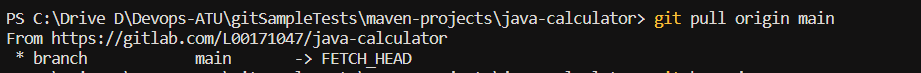
### Figure 8 – Issue board view at beginning of sprint



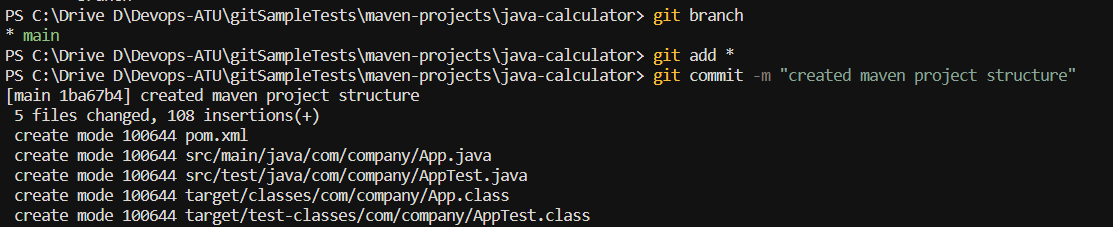
### Figure 9 – Milestone status at beginning of sprint



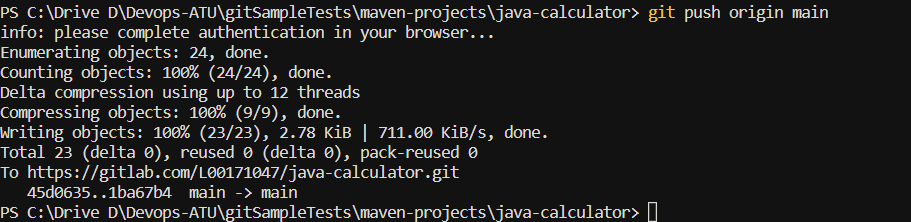
### Figure 10 – Using git pull to integrate remote Main with local HEAD



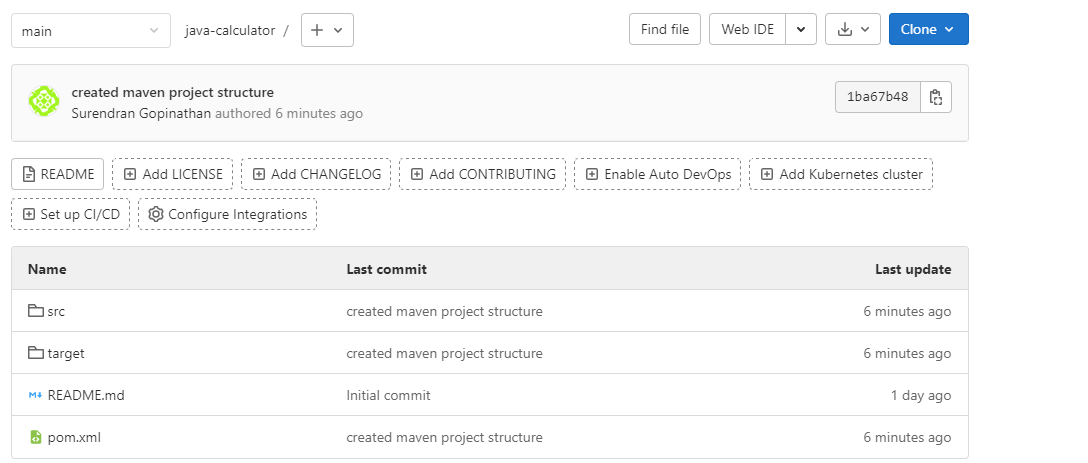
### Figure 11 – Staging and committing changes



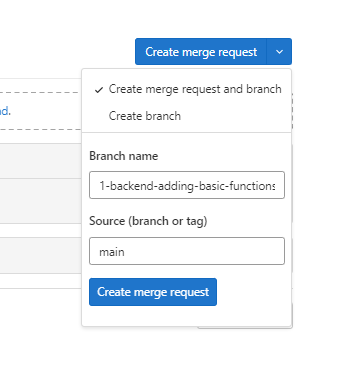
### Figure 12 – Pushing changes to main in remote repository



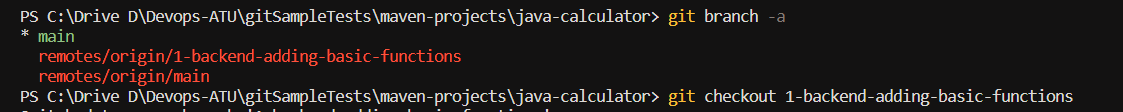
### Figure 13 – Maven folder structure created in gitlab



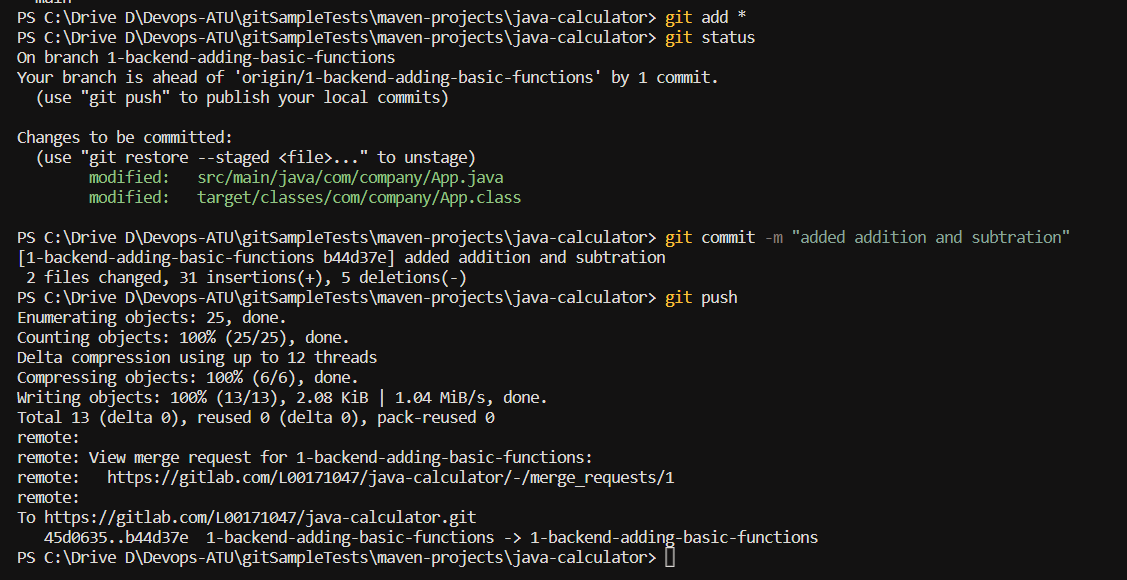
### Figure 15 – Creating a new merge request



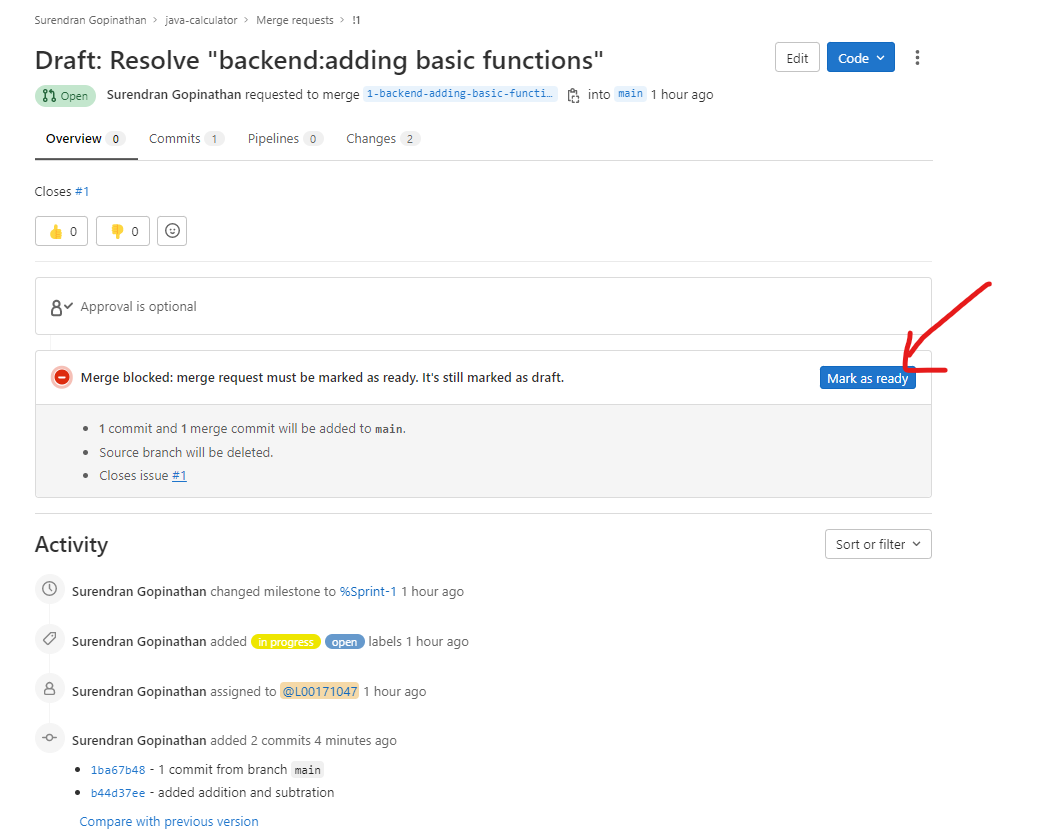
### Figure 16 – Checking out to new branch created via merge request after git pull



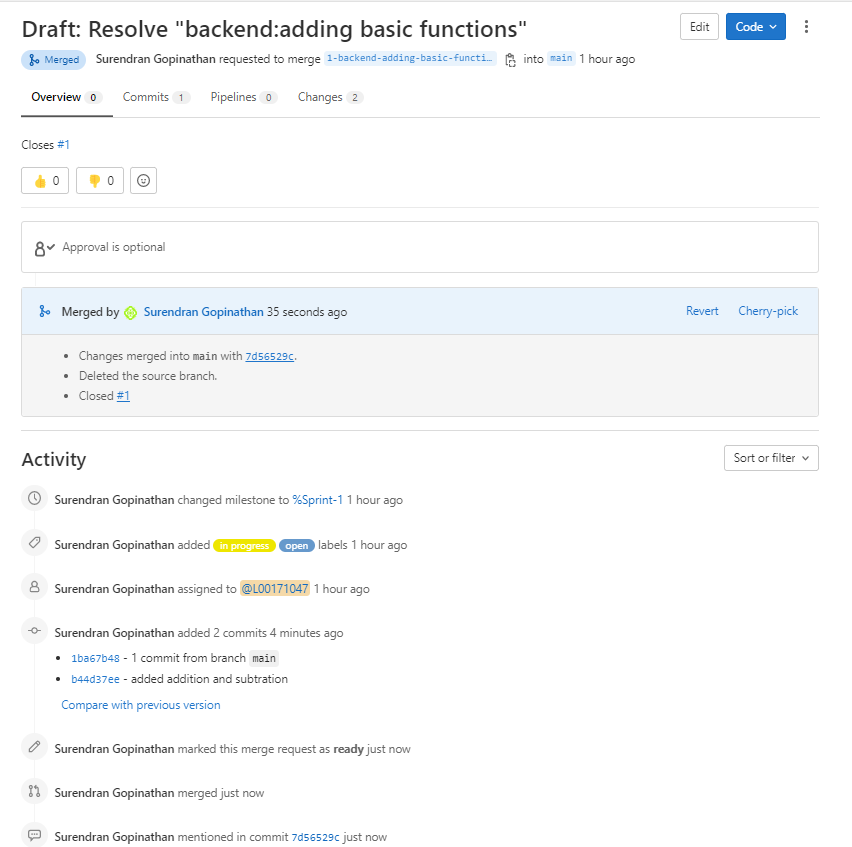
### Figure 17 – Staging,committing and pushing changes



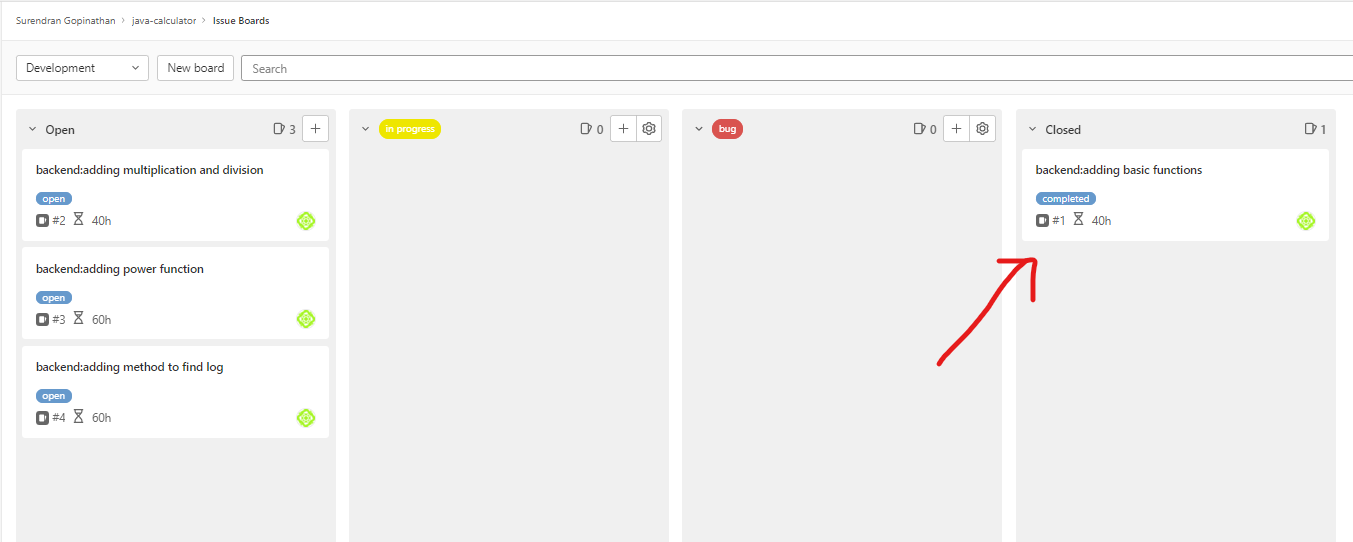
### Figure 18 – Reviewing changes in MR after pushing code



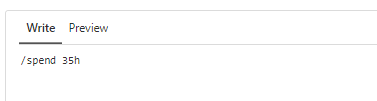
### Figure 19 – Merge request after merging code



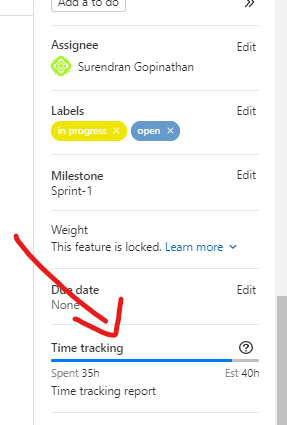
### Figure 20 – Issue board after closing first merge request and issue



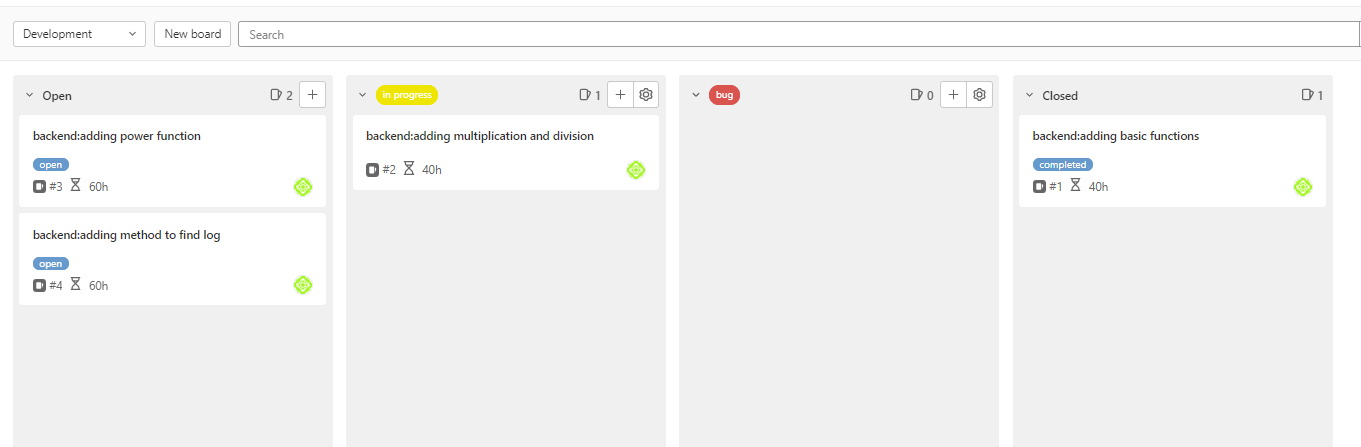
### Figure 21 – adding time spent on an issue



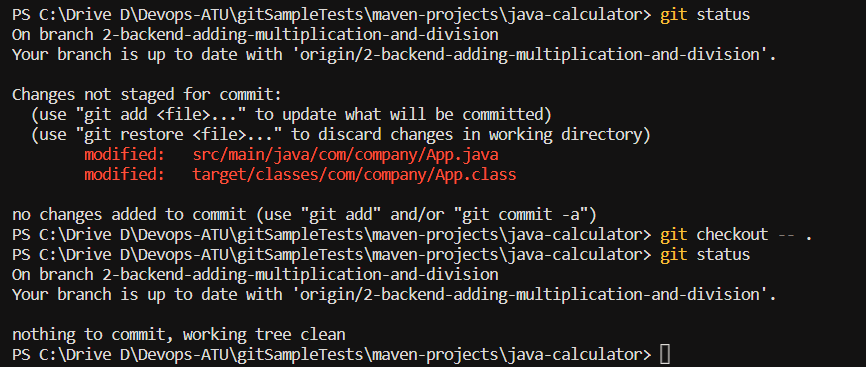
### Figure 22 – Comparing time spent with time estimate



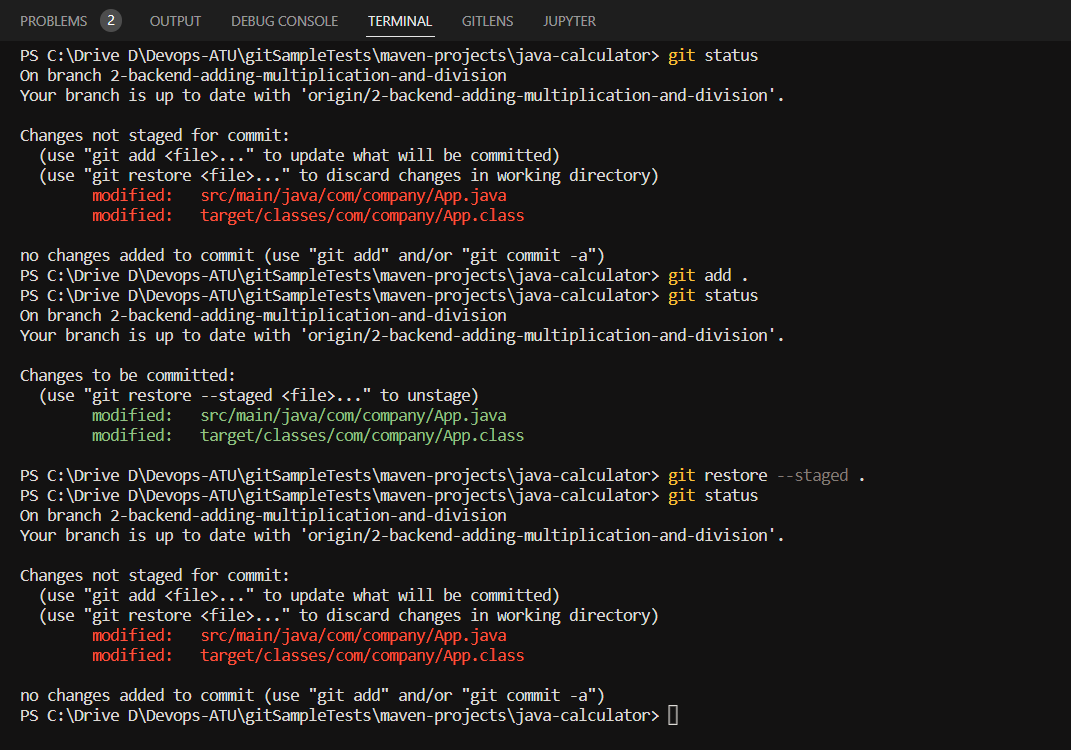
### Figure 23 – Moving next issue to in progress section and working on it



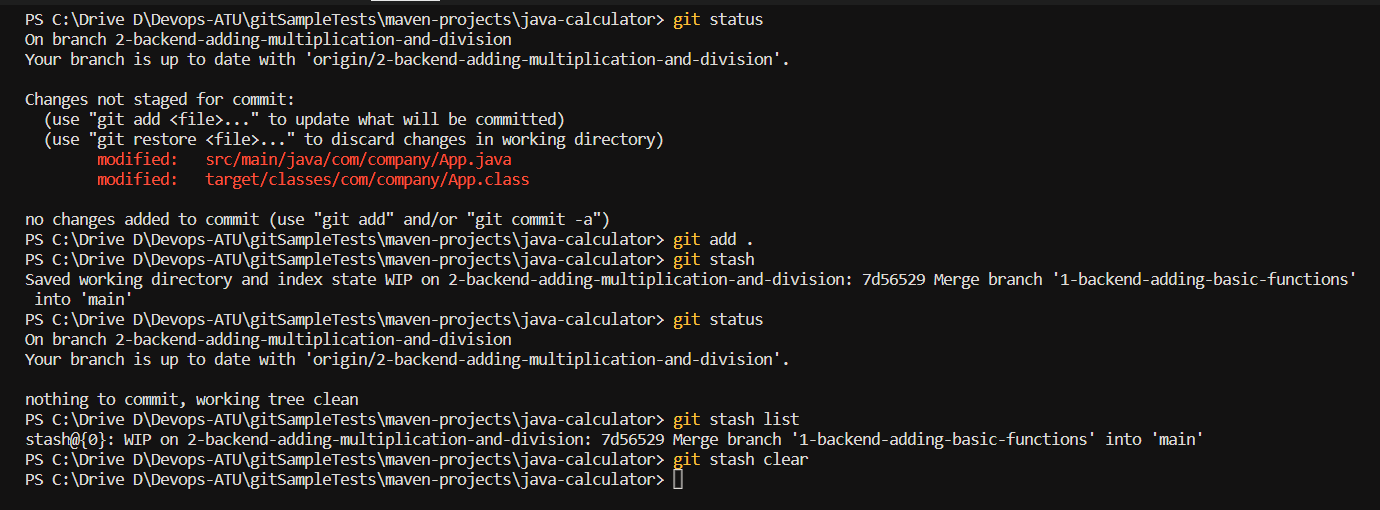
### Figure 24 – Using git checkout to remove unstaged changes

****

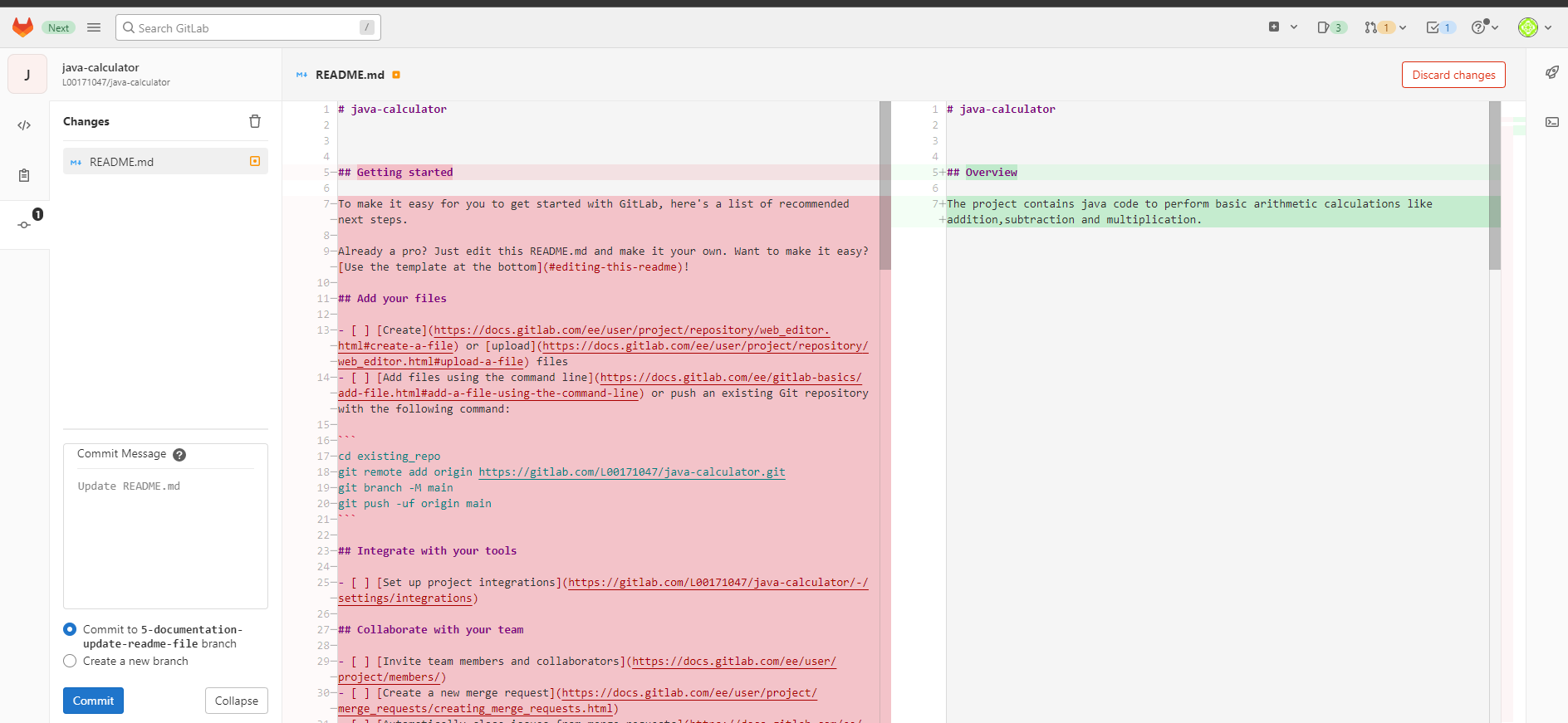
### Figure 25 – Using git restore to remove staged changes



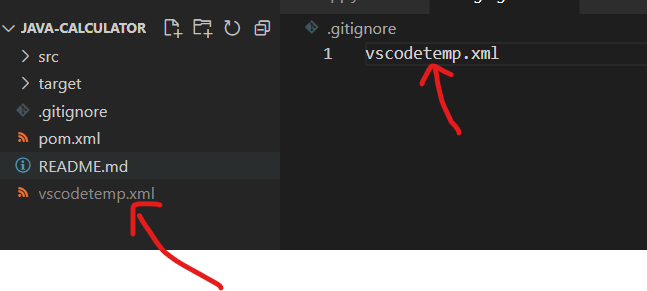
### Figure 26 – Using git stash to stash changes



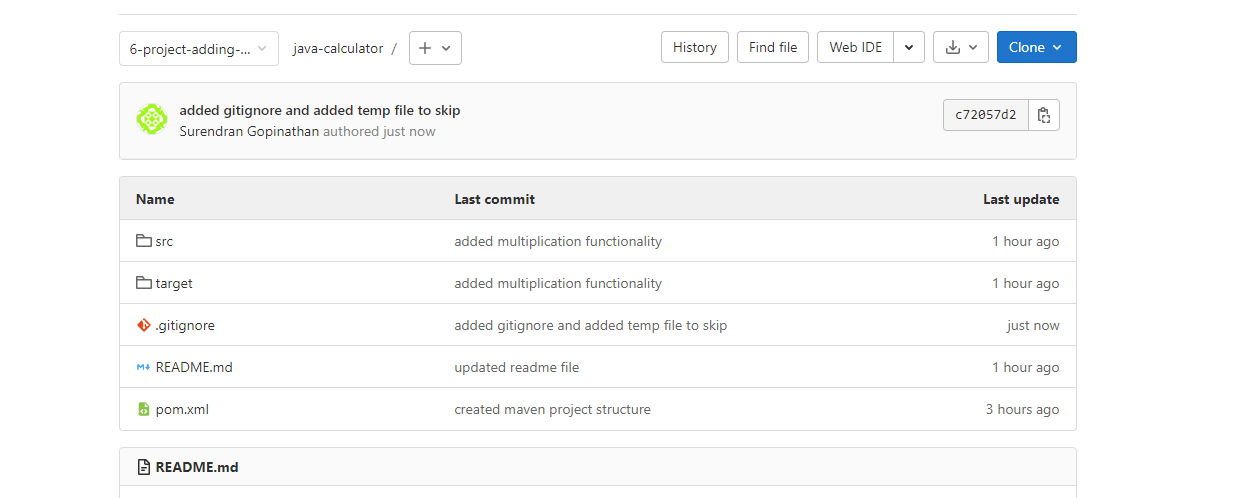
### Figure 27 – Using gitlab IDE



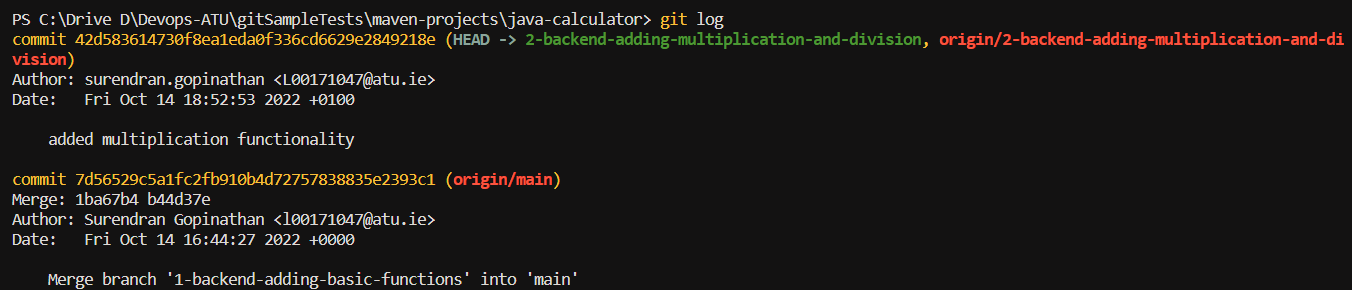
### Figure 28 – Creating .gitignore file in local repository



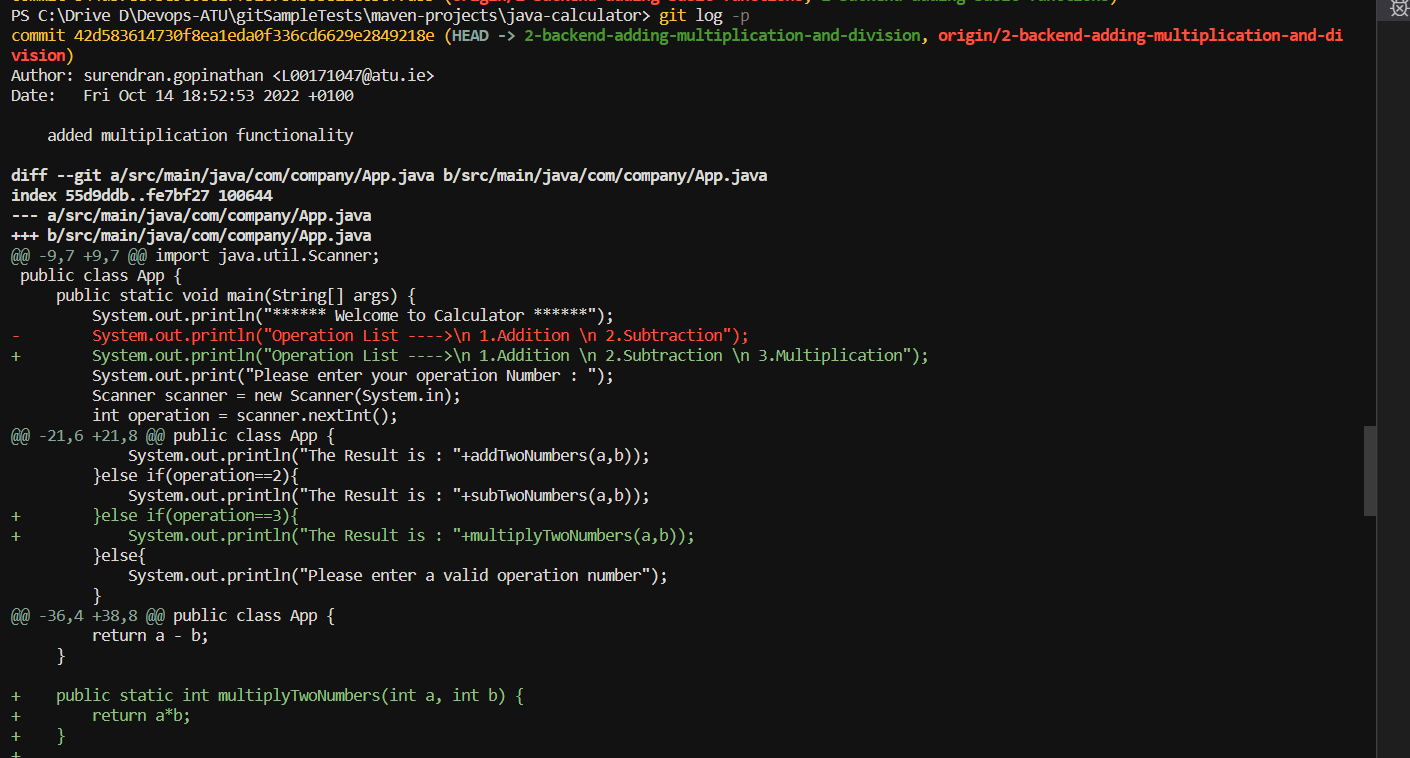
### Figure 29 – remote repository where vscodetemp file not pushed because of gitignore file



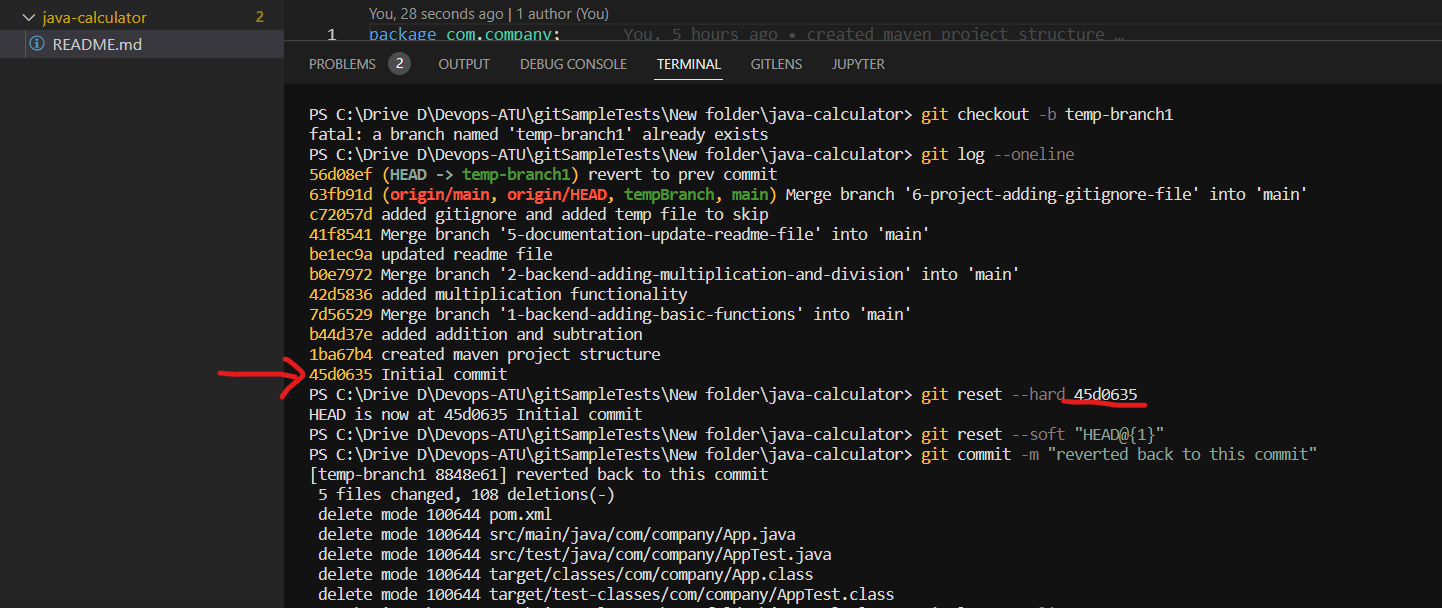
### Figure 30 – Using git log to view commits



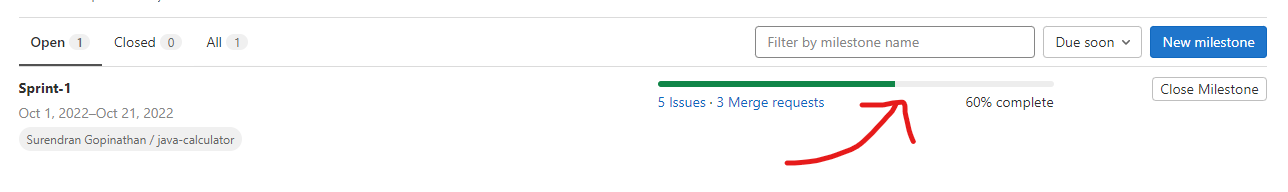
### Figure 31 – Using git log -p to view commits in details along with code changed



### Figure 32 – Using git reset to checkout to new commit and moving HEAD to the commit.



### Figure 33 – Status of milestone/sprint



### Figure 34 – Status of sprint in issue board

