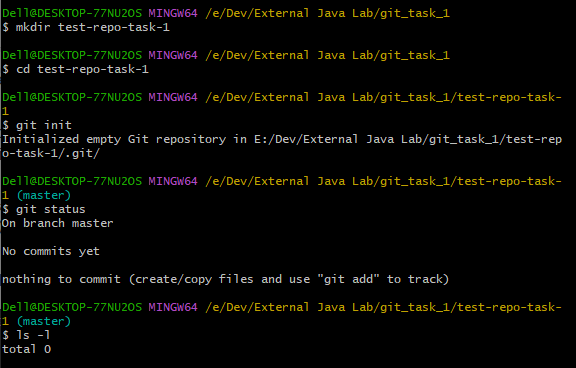
### Task 1

We are going to set up simple git branching strategy.

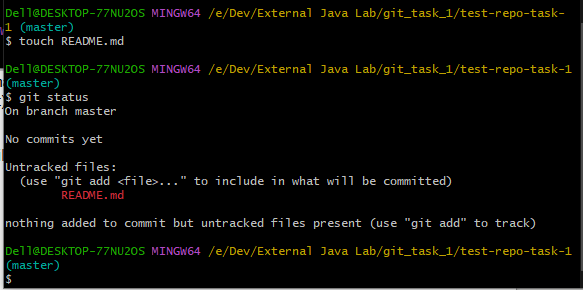
1. Create empty git repository with  git init  command. Also read up about --bare option and bare repositories themselves.



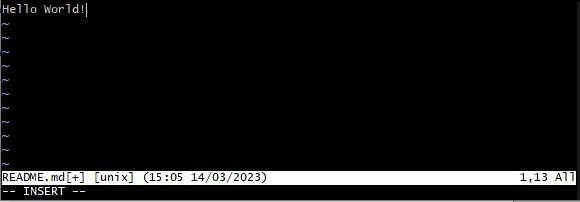
*git-ls-files — Show information about files in the index and the working tree*

*(-l use a long listing format)*

1. Let us create new file with  touch README.md  command.

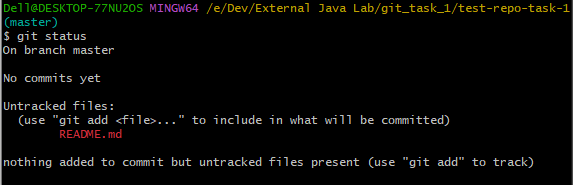


1. To modify the existing file, we will use VIM text editor. This editor is widely used for rebasing and other git activities so every developer should get used to it. Type vim README.md command to open vim text editor. Press letter  i  to enter the insert mode. Type in “Hello World” or something you like. Press  escape  on the keyboard to exit from insert mode and go to the command mode. In the command mode press  shift + ZZ  to save current file and exit.

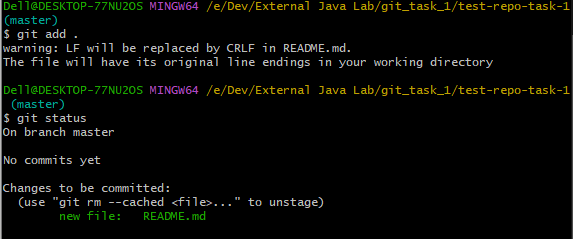


In the bottom left corner of the above screenshot you can see that currently the editor is in insert mode.

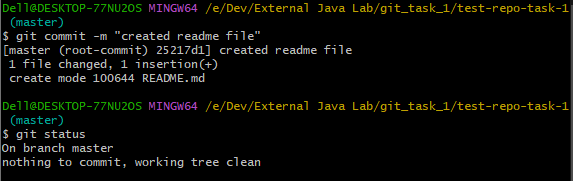
4. To check the progress type  git status  command. You should see that readme file is untracked now. You should check for a status as often as it is possible. If you always read status carefully enough, you will never end up in strange unpredictable situations.



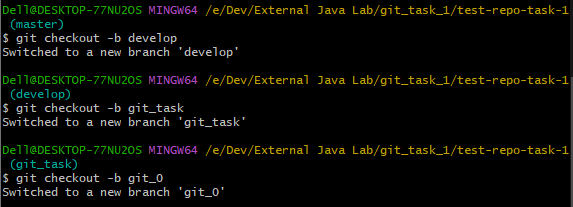
5. To track readme you can use  git add .  command. This will track and stage all the files in the current directory. Now you should see that your readme file is green which means it is also staged. Take a moment to read about tracking and staging and how they are different with each other.

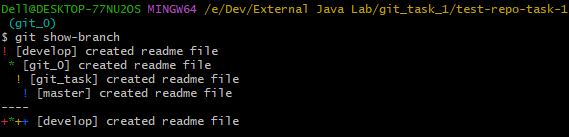


6. It is time to commit changes. To create commit use  git commit -m “created readme”  command which will create commit with corresponding message. Your working tree should be clean now.

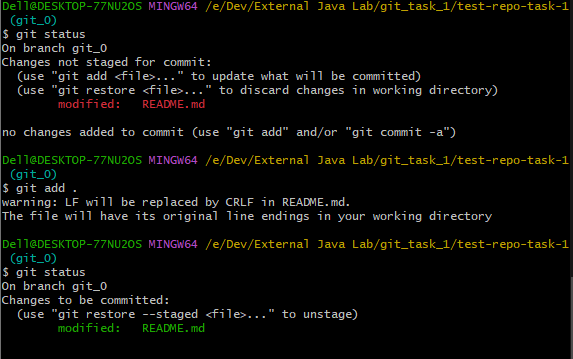
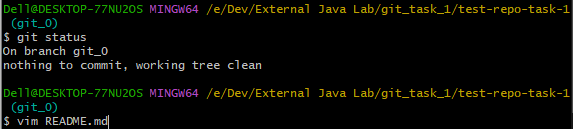


7. We are going to create new branch called develop, which will contain all the functionality you are currently working on. To create a new branch, execute git checkout -b develop command. This will create new branch from the current (master). Create  **git\_task**  branch from  **develop**  and  **git\_0**  branch from  **git\_task**.

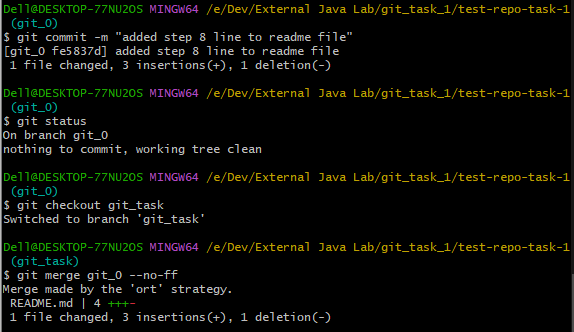
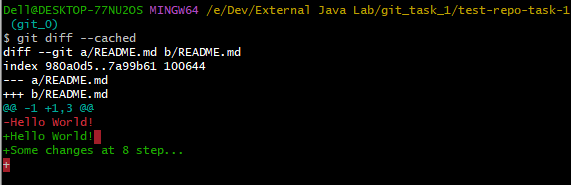


You can use git show-branch command to view all branches at once. The branch you are currently working with is marked with \***\*** symbol.

8. Now we will add some changes to README.md, with  **vim**, check status, stage changes with  git add README.md  command, commit changes, check the result (status) then checkout to  **git\_task**  branch (with  git checkout git\_task  command) and  **merge**  changes from  **git\_0**  branch with  git merge git\_0 --no-ff  command. Take some time to read about merging and  **no-ff**  option.  This is usually the default pull request merging behavior so you will work with it on you daily basis.



Also note that you can view staged changes with  git diff --cached  command. For example there we can see that new line “changes from git\_0” was added to the file.

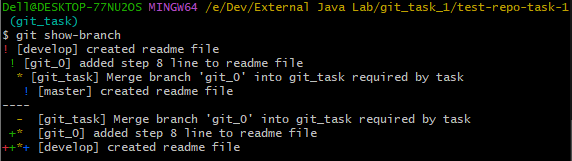
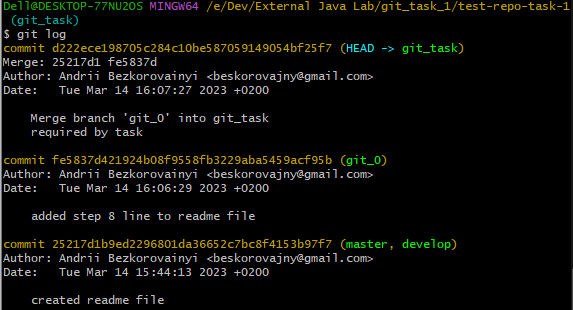


*The  --no-ff   flag prevents  git merge  from executing a "fast-forward" if it detects that your current  HEAD  is an ancestor of the commit you're trying to merge. A fast-forward is when, instead of constructing a merge commit, git just moves your branch pointer to point at the incoming commit. This commonly occurs when doing a git pull without any local changes.*

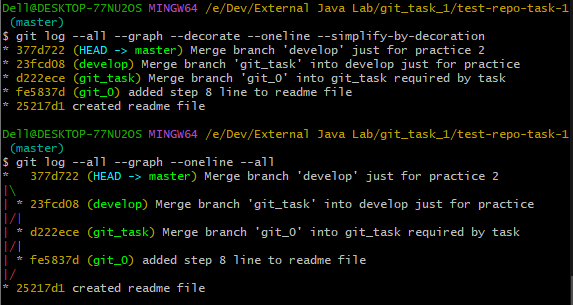
Also note that you can view your latest changes with  gitkcommand (which should invoke gitk tool). If this command does nothing or returns error, this usually means that you do not have gitk installed by default (e.g. working from mac). This tool is not necessary and is the matter of choice.



You can also use git show-branch command (or even  git log  command) to quickly look through your recent work.



9. Now repeat merging process to get your commit to  **master**  (through  **develop**  of course) branch. When you finish, you can use something this git log --all --graph --decorate --oneline --simplify-by-decoration  or this  git log --graph --oneline –all   command. These commands were found on the stackoverflow as an answer to a question of viewing git history as a tree. This is not the best solution but it is more than enough for our needs. You output should look similar to this:



From this picture you can see that the HEAD is currently looking at the master’s latest commit. We have created 5 commits: 2 with actual edits and 3 merge commits.

**The idea here is simple.** You should always develop you features in custom branches made from  **develop**  branch. When you feature is ready it goes to the  **develop**  branch (usually with pull request instead of merge). When the release time has come, all the merging to the  **develop**  branch should stop (or you may create custom **pre\_release** branches with locked functionality).  **develop**  branch (or  **pre\_release**) should be tested and stabilized. As soon as the branch is stable, it goes to the  **master**  branch (or you may have different  **release**  branches and even don’t have a **master**, or don’t use  **master**  branch at all). This procedure might differ from project to project and always should be designed with the respect to the concrete project and goals. All this helps to keep the repository clean and always have some stable versions for the production.