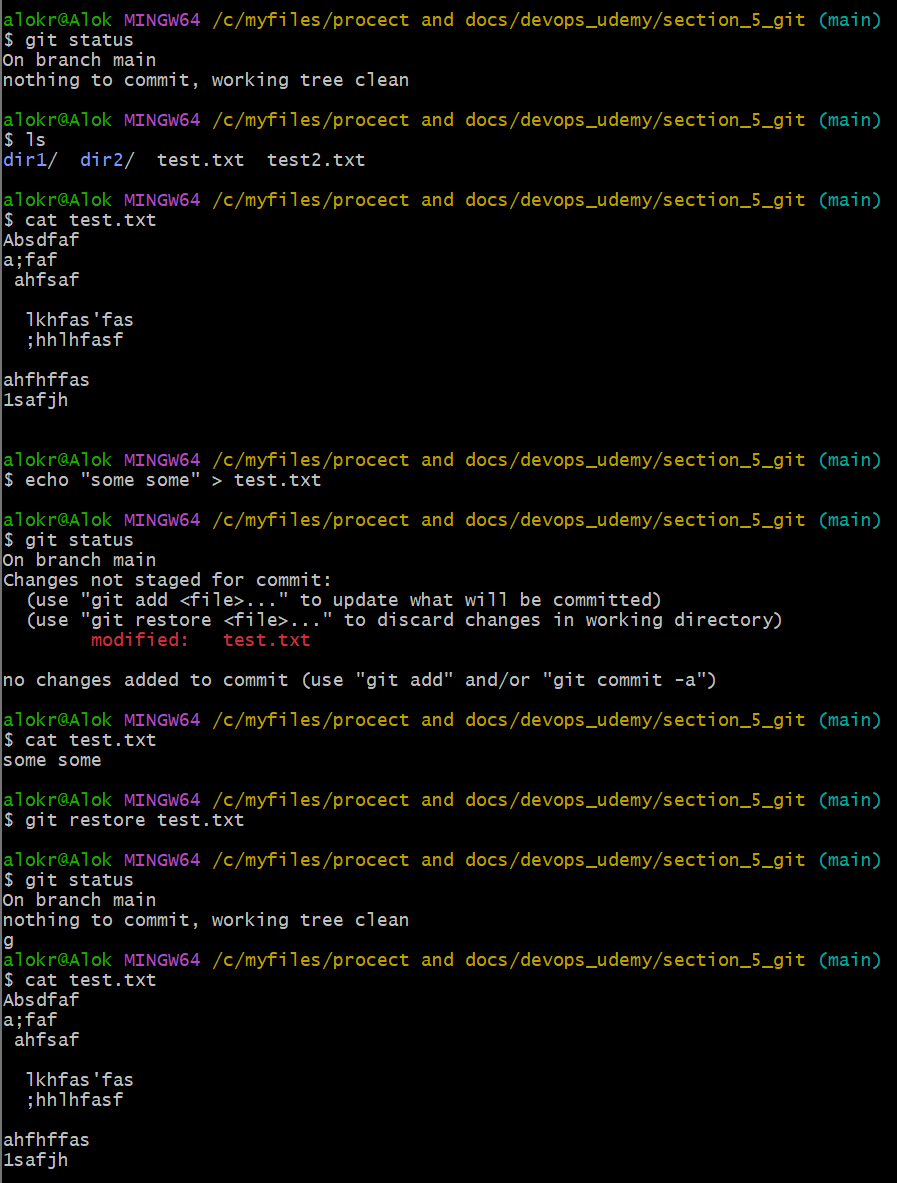
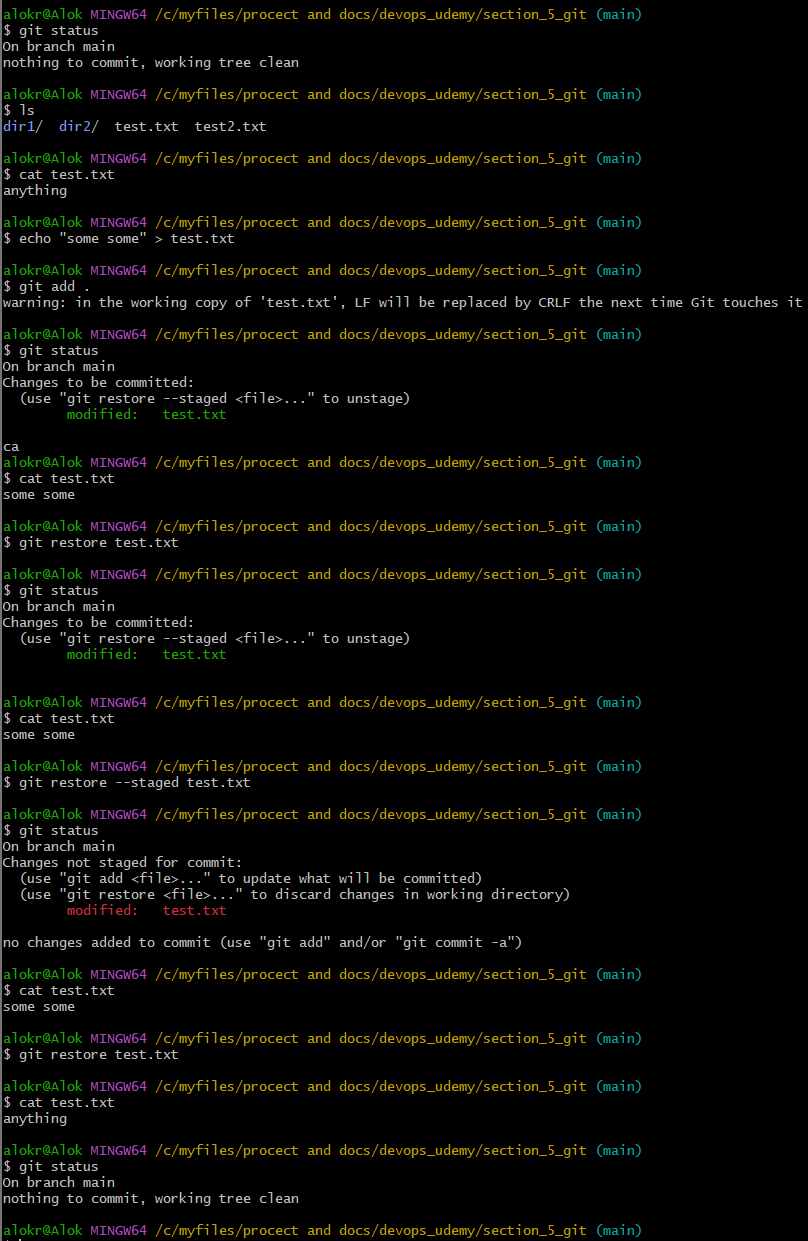
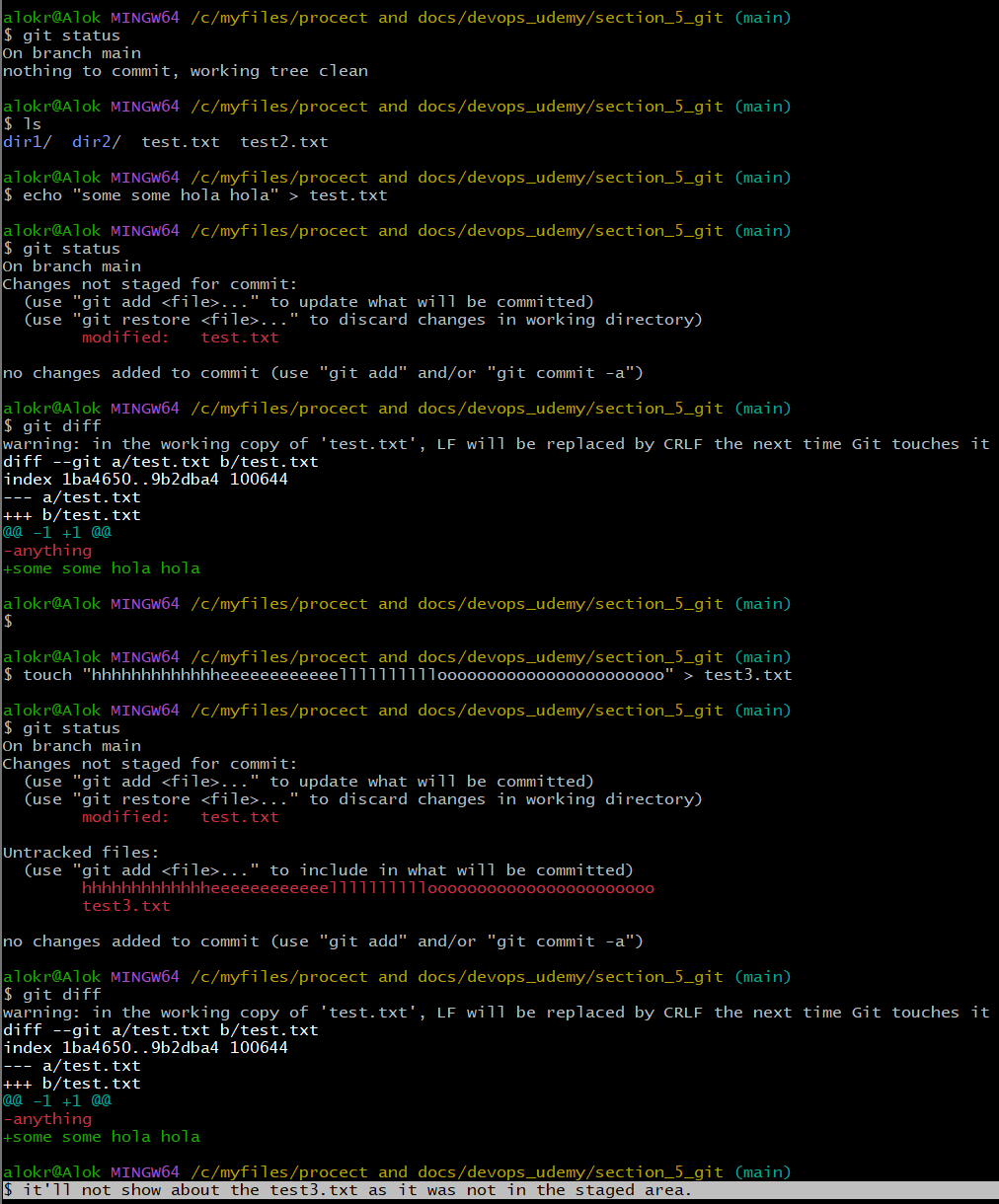
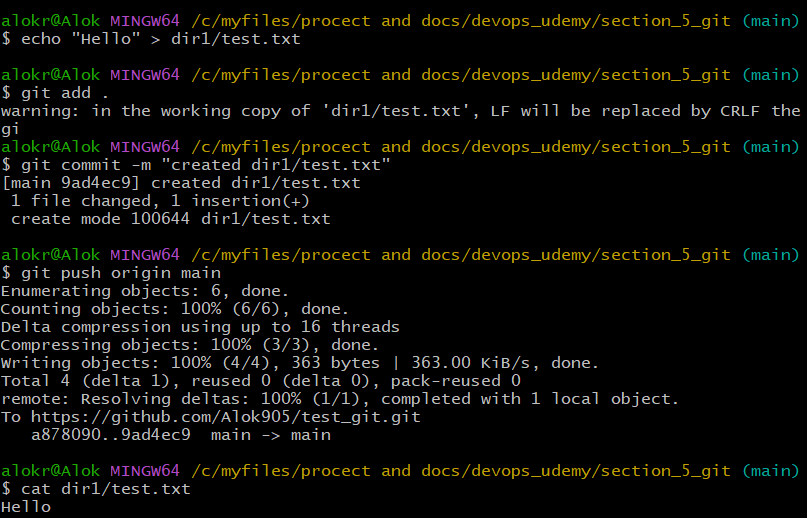
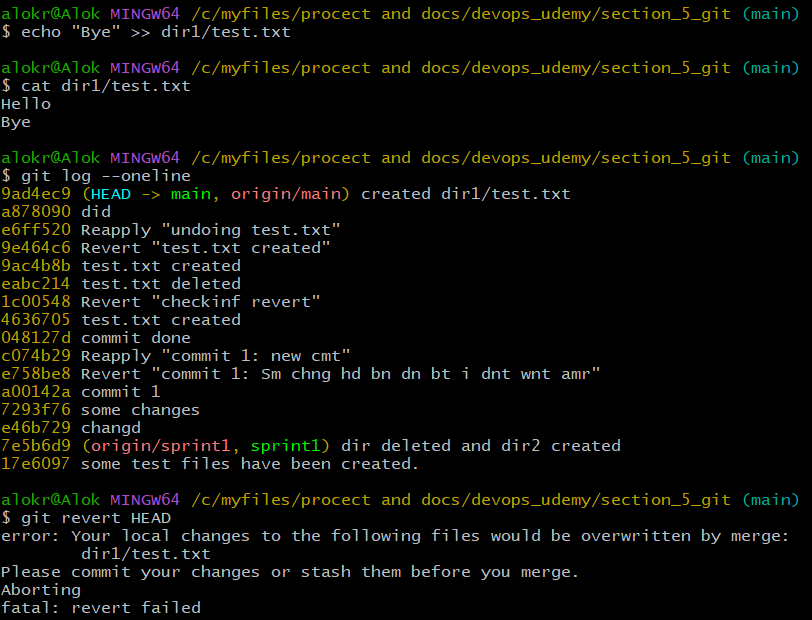
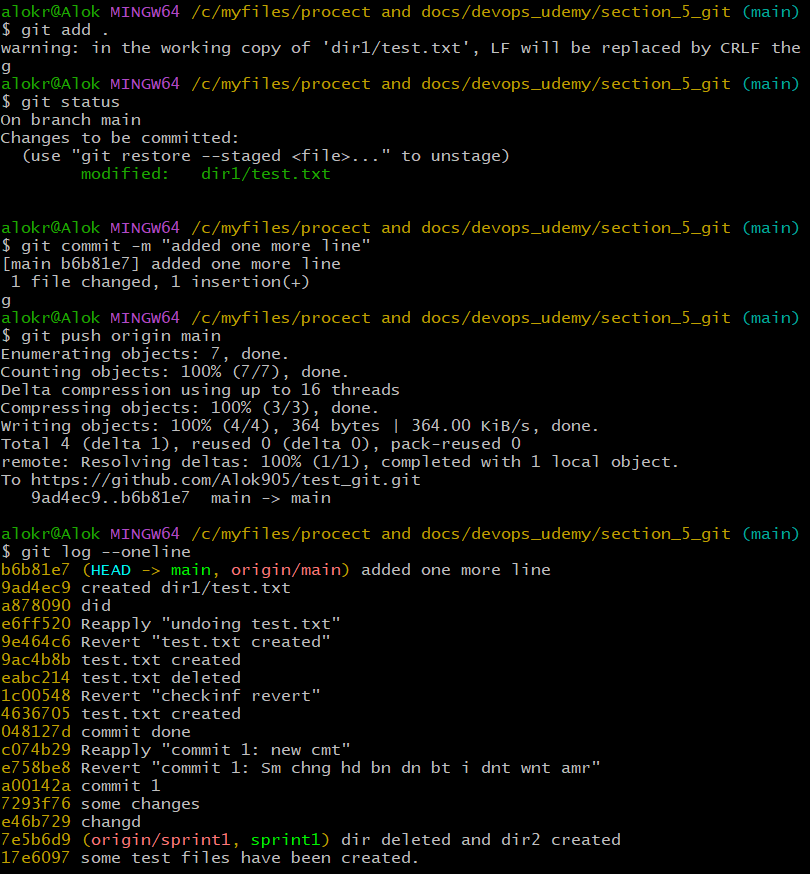
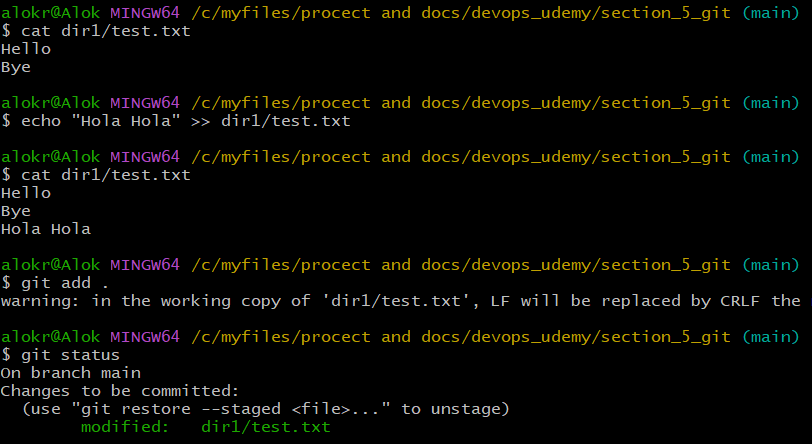
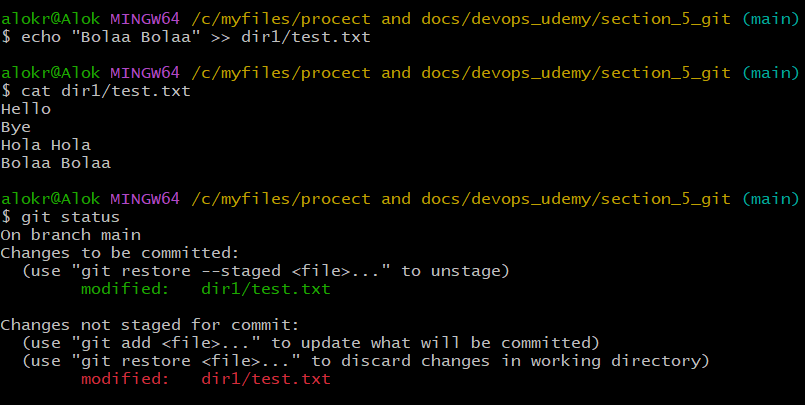
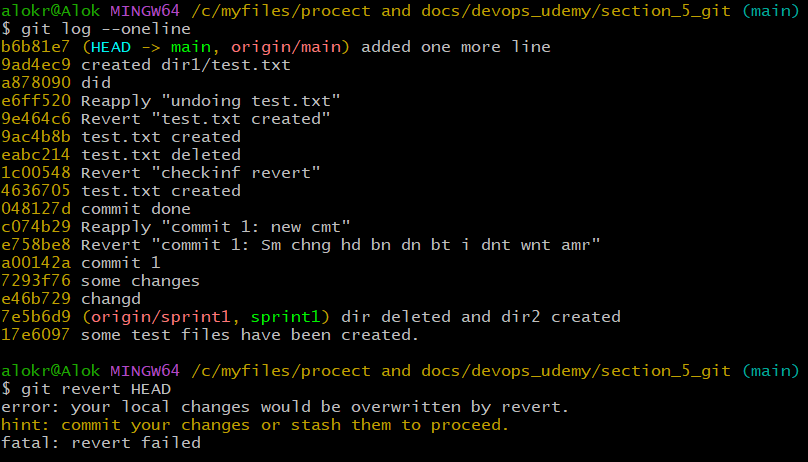
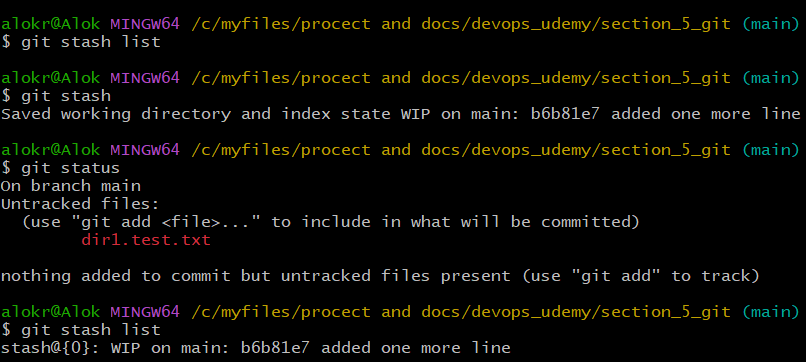
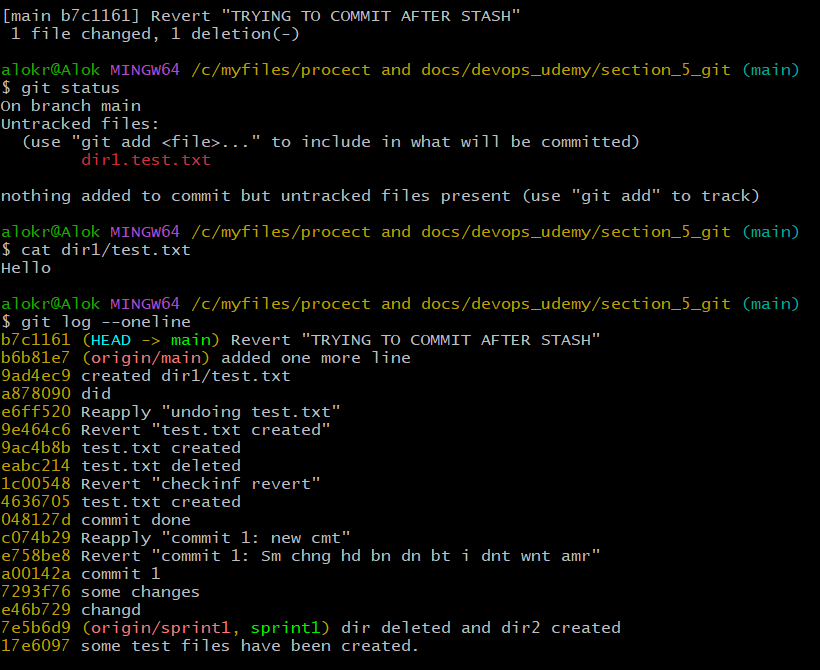
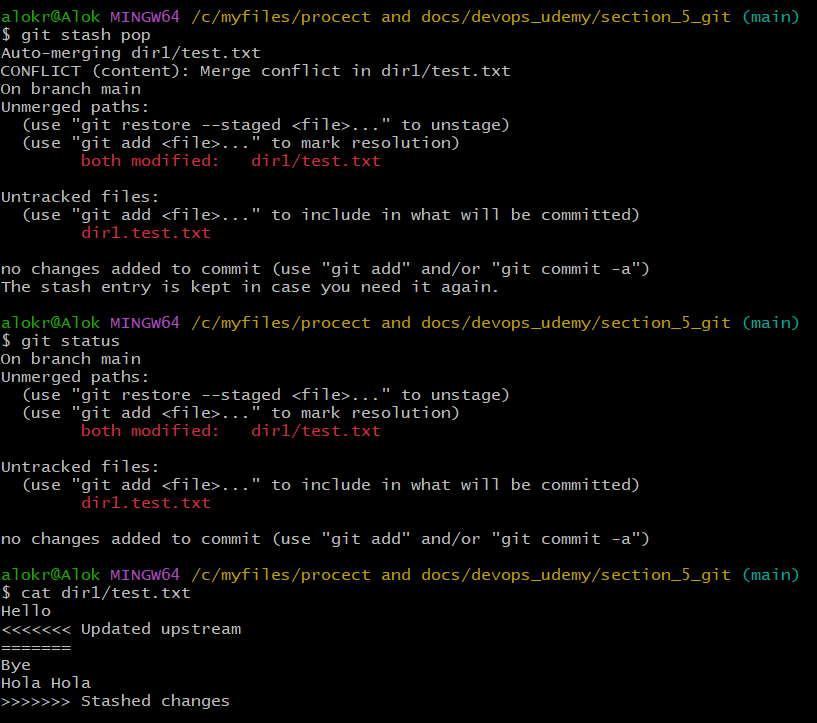
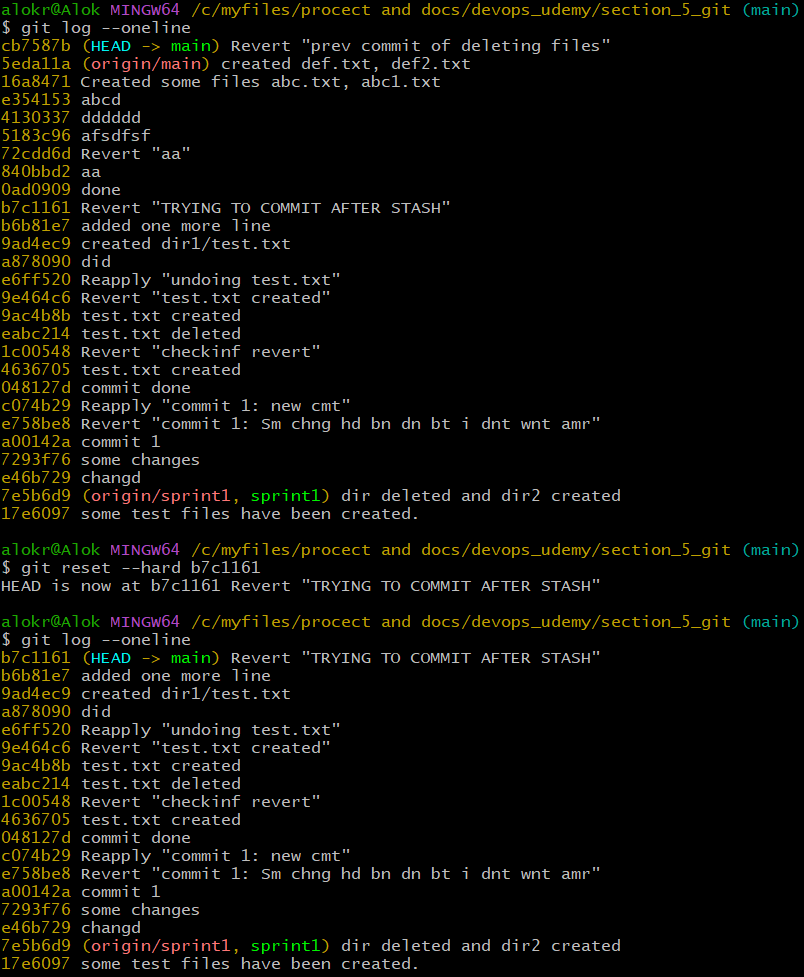
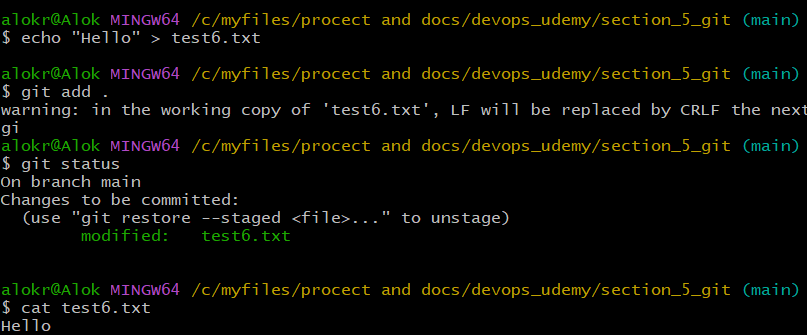
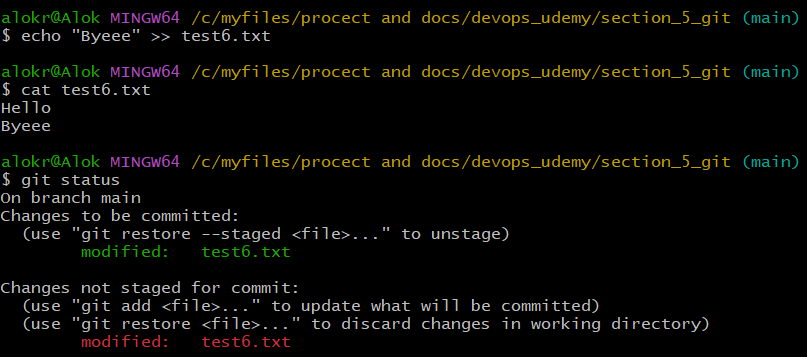
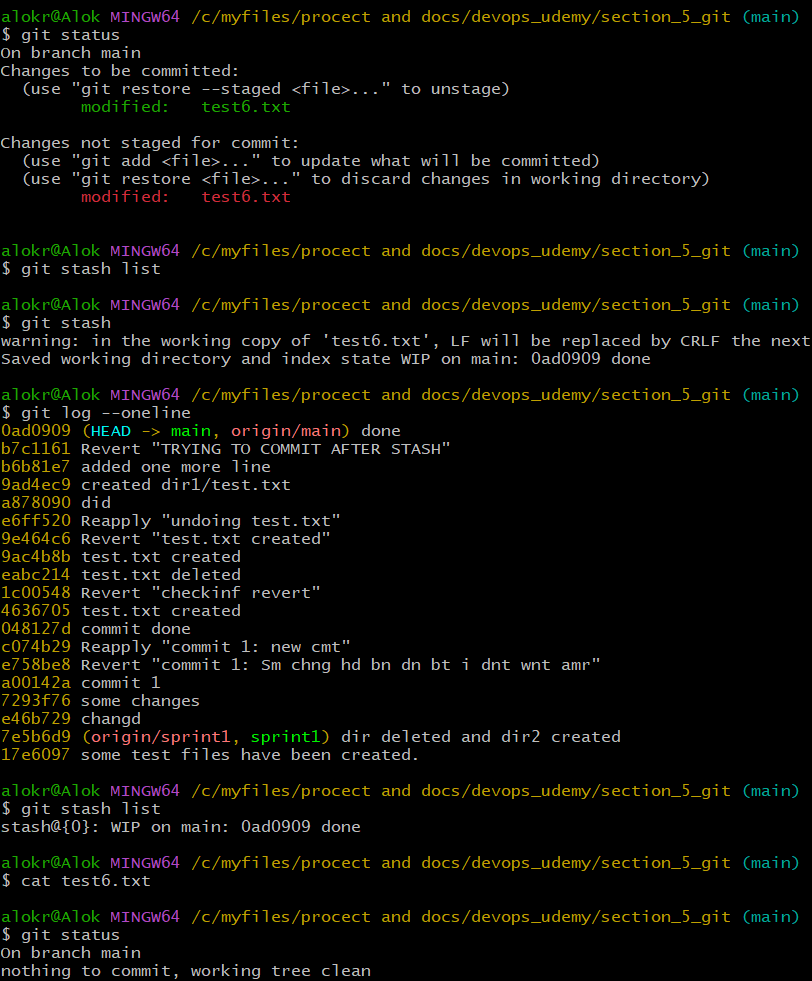
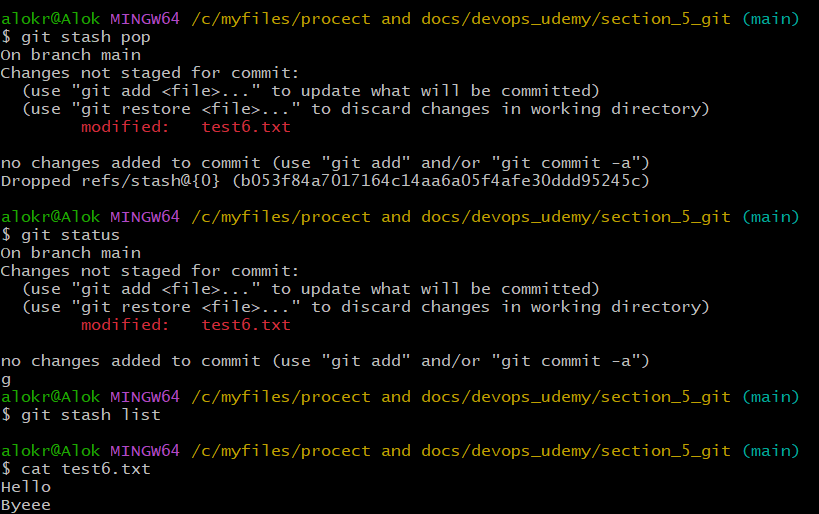
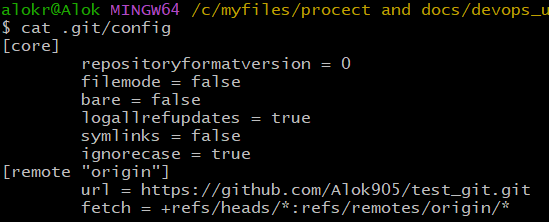
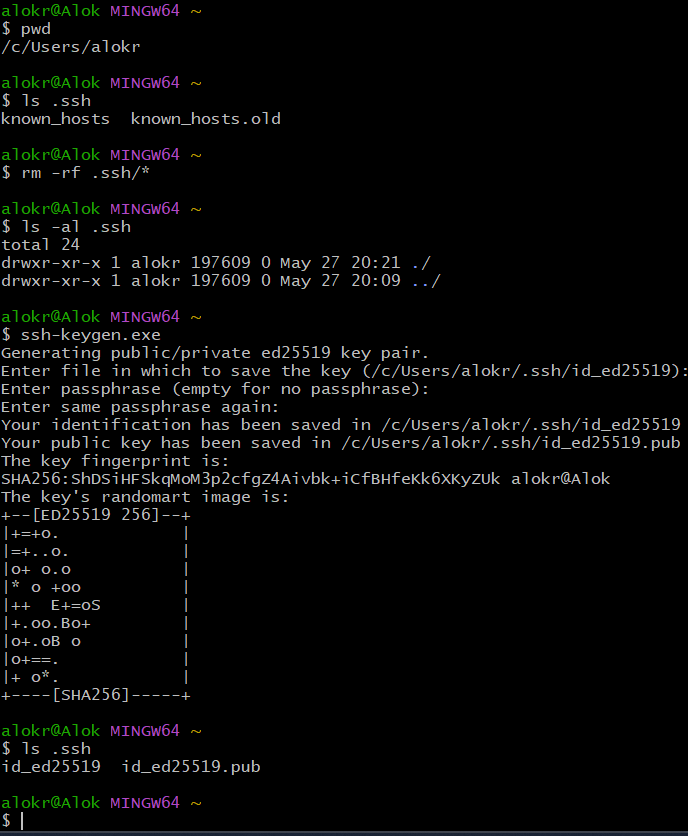
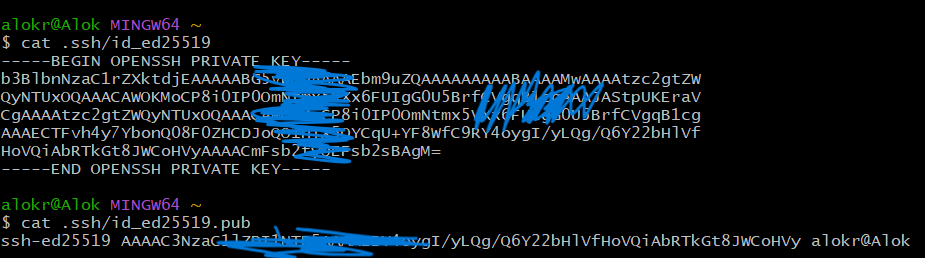
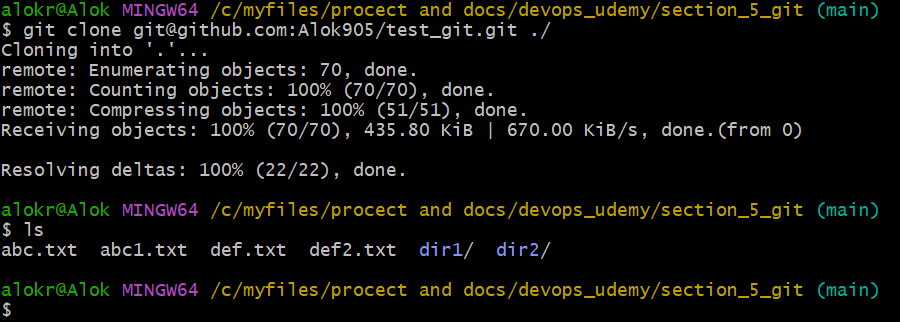
* **git push --all origin**
  + Push all the branches to the remote repository.
* **git checkout <filename>**
  + If you had changed one file and have not commited yet, then running this command will erase everything you updated and make it same as the last commited version.
  + It doesn’t effect any untracked file. (as git doesn’t know about this file)
  + If you have alreadyy staged the file (using git add) then you are executing this command, then it doesn’t effect this as the changes are already staged.
  + *Hence, if the file is not untracked & the changes are unstaged, then only the* ***git checkout <filename>*** *will reset this file to the last commit version.*
* **git restore <filename>**
  + Same as *git checkout <filename>.*
  + But it is recommended to use in stead of *git checkout <filename>.*
  + git checkout should be used for switching among the branches only.
  + It has some additional features like you can rollback to a specific commit in the past. I.e.
    - ***--source=<commit>***
  + 
  + ***git restore --staged <filename>*** : It unstaged the file from staging area.
    - Let the file is already committed, now you have changed (added or deleted) some lines in side the file, and staged it (git add command). Now you are using this command *git restore --staged <filename>*, it’ll only unstage the current updation (means the added/deleted lines) as it the file was pushed to tracking stage(git add) in the previous commit.
    - Ex:
      * The file is committed.
      * Now changed come lines. And executed *git add .*
      * Now executed *git restore --staged <filename>*
      * Now the changed in side the fill will be there, but unstaged.
      * Now execute *git restore <filename>*
      * Now the changes done in this file will be no more.
    - 
* **git diff**
  + It shows the differences between *working directory* and *staging area*.
  + 
* **git diff --cached**
  + It shows the difference between *staging area* and *last commit (HEAD).*
* **git revert** and **git stash:**
  + Let you created one file test.txt, insert some texts i.e.“Hello”, staged it, committed it and pushed it to main branch.
    - 
  + Now let you appended one more line i.e. “bye” inside that test.txt file. (now content is: *Hello \nBye*)
  + Now you executed the command: *git revert HEAD. It’ll give some error saying..* 
    - 
    - 
      * I’ll undo this commit afterwards (bcs If I would undo the previous commit then the file would have been deleted)
    - I appended a new line to the file and staged it.
      * 
    - I appended one more line to the file “Bola Bola” but didn’t stage it.
      * 
  + Now If I do git revert HEAD it’ll give error.
    - 
  + You need to ***stash*** it now.
    - Stash means the changes that have been done (only unstaged) will be saved in a stash stack so that the working directory will be clean.
    - 
  + Now you can **revert** it.
    - 
    - 
    - 
    - The last commit was previously ***b6b81e7***, which was for added the line “Bye”. Now the latst commit is ***b7c1161*** which deleted that line “Bye”. Only “Hello” is there.
  + Now, to get the ***stashed*** changes, you need to execute ***git stash pop***
    - 
* ***git reset***
  + Unlike *git stash* this command i.e. ***git reset*** doesn’t create another commit.
  + It means, let you have 4 commits: c1, c2, c3, c4 and you want to go back to commit c1.
  + Then it doesn’t create any other commit c5 and roll back to c1, it’ll delete c2, c3, c4 and go to c1.
  + 
* **NOTE: *git revert* will only undoes the perticular commit and doesn’t effect later commits but *git reset* will remove all the later commits.**
* there is a already committed file: test.txt
  + - I added one line "First Line" and committed. (C1)
    - I added one more line "Second Line" and committed. (C2)
    - I added one more line "Third Line" and committed. (C3)
    - I added one more line "Fourth Line" and committed. (C4)
    - I added one more line "Fifth Line" and committed. (C5)
  + *Now I did git revert C2. Will it give any error?*
  + It usually won't cause an error unless:
    - The line added in C2 ("Second Line") was modified in later commits (C3–C5).
    - For example, if C3 edited "Second Line" instead of just appending "Third Line", Git may throw a merge conflict during the revert.
* Working of ***git stash***
  + 
  + 
  + 
  + 
  + Steps:
    - I added one line in a already created file “Hello”.
    - Sta*ge*d it
    - Added one more line “Bye”
    - Then ***stashed*** it
    - Now the staged and unstaged changes were stashed and working tree became clean.
    - Then *unsta****sh****ed*
    - Got everything in side the file.
    - ***Note: The staged modification will be now in unstaged state.*** As git can’t quarentee that the unstaged changes are still valid or not.
* ***Github ssh login:***
  + 
  + Here the authentication is based on http so you need the password. There is a chance of loosing or leaking of the password.
  + 
  + 2 keys are there: *id\_ed25519 (private key), id\_ed25519.pub (public key)*
  + 
  + Add the public key in your github account (not github repo). *Settings > SSH and GPG keys > add ssh key*
  + 
* **Git Tags, Semantic Versioning** 
  + 