|  |  |
| --- | --- |
| **DOCUMENT RULES:** | |
| **Task Number / Name:** | **Git** |
| **Task name & column name should be written:** | **Bold (CTRL+B)** |
| **Commands should be written in the after # sign:** | *Italic (CTRL+I) #hostname* |
| **Output photo should be cropped or compressed:**  **Photo could be more than one:**  **If you need extra lines, add the line next after it:** | ***Description photo should be with title bar (CTRL + I + B)*** |
| **All other text should be written:** | Standard |
| **Font name and text size:** | Calibri and 9 |
| **Group name:** | Dev\_ops\_ |
| **Student name and surname:** | Murad Abbaszade |
| **E-mail:** | [muradabbaszade6@gmail.com](mailto:muradabbaszade6@gmail.com) |
| **WhatsApp number:** | **+994703664205** |

|  |  |
| --- | --- |
| Install Git on Ubuntu |  |
| Configure Git |  |
| hooks directory contains all custom hooks. These are small (usually) scripts which have to be executed before commit, or after, before push, etc.  branches - this is deprecated. Don't think about it anymore.  HEAD - pointer to the current branch and its latest commit.  config - configuration file for the repository.  info - the place where you stage the files using git add.  refs - the current state of the whole repo.  objects - commits, trees and blobs are stored here. May be very big.  logs - quite self explanatory.  description - description of the repository. |  |
|  |  |
| Create a directory  mkdir test-repo  and navigate there  cd test-repo  The directory is obviously empty  ls -al  Now it is time to initialize repository  git init  Now you should be able to execute status command  git status  And also the .git directory  ls -al |  |
| git status |  |
| git add newfile -- Add file to git repository |  |
| tree |  |
| Remove file from commit  cd test  git status  Let's imagine the situation, you want to remove all staged files from index for some reason. Honestly speaking, if you need to do so, please, revise your approach to work :)  Anyway, we have 4 files in stage. Let's remove testfile-01  git rm --cached testfile-01  git status |  |
| git log |  |
| Visual Commit |  |
| **Git revert**  cd test  git status  git log  Let's move ourselves back by one commit.  git revert --no-edit HEAD  We succesfully moved back by creating new commit.  git log  git revert --no-edit HEAD |  |
| Check the differences  cd test  git status  git log  Another command, git diff, allows to check the differences between HEAD and current working directory. Another words, what was changed during our recent work.  git diff  clear && git diff HEAD~1  clear && git diff HEAD~1 testfile-01  Now we see information about testfile-02 only in comparision of current working directory and one commit before HEAD. |  |
| Detailed information about previous commits |  |
| Gitignore  cd test && ls -al  git status  Let's create gitignore file  touch .gitignore  echo neveringit >> .gitignore  echo seconddirectory >> .gitignore  cat .gitignore  git status  Ok, we are ready to run  git add .  and  git status  And commit  git commit -a -m "first commit" |  |
| Git tags |  |
| git adog - this is obvious by now, I hope.  grep 'testfile-06 - this will select entries with message where this filename occurs (not the best way, but in our case it is more than enough).  awk '{print $2}' - with awk we are 'cutting' the output and print only the third (counted from 0) element, where separator (default one) is a space.  head -n1 - on the end we are printing ony the first element (if there is more records with the same name). git log shows commits by descending through date, so this works for us. |  |
| git checkout master  And now we delete v1.0 tag.  git tag -d v1.0  git tag  git adog  Yep, tag is deleted. |  |
| As we finished work on branch, we can delete it  git branch -d newbranch && git branch |  |
| System creates two repos. One in merge directory, where all process was completed (final merge). Let's see how our history looks.  cd merge  git adog  (do you remember our "a dog" config setting from git config module?) |  |