GitLab4 Working With Git

GT-WU-A-04 - Working With Git

By the end of this lab you should be able to:

- Use the git log command to help you clarify if you are committing the correct version of a file
- Use the git status command to help you clarify the current state of your repository

Step 1 - Edit files within your Git repository and view using Git log/diff

- Ensure you are within your git repository
    - *cd c:/*git_repos/newrepo
- Create a new file in your repository named "diff.txt"
    - *echo diff > diff.txt*
- Use what you know to add and commit it.
    - *git add diff.txt*
    - *git commit -m "added diff.txt"*
- Make some changes to text files within your working directory and use git diff to view the effect of those changes.
    - Do NOT add or commit it yet
    - When the change is made, compare it with*:*
        - *git diff diff.txt*
    - use git diff HEAD to compare it to your repository
        - *git diff HEAD diff.txt*
    - The result of both these diffs should be the same as the staging area and HEAD revision have identical versions at present.
- Add the change to your staging area
    - *git add diff.txt*
- Use git diff --cached to compare your staging area to your repository.
    - *git diff --cached diff.txt*
    - The result should be the same as before, as the staging area now contains the changes from your working directory but the repository does not.

Step 2 - Using git status

- Use the command *git status --short* to view the current state of your working directory at any time.
- The following Git status's are possible:
- *M = modified*
- ?? = unknown
- *A = added*
- *D = deleted*
- *R = renamed*
    - Note - more are possible if you wish to try, you can view them with *git status --help*
- If you are using git bash, you may notice the changes in your working directory are colored red, while the changes in your index are green.

- Try to recreate all of the above scenarios. When you think you have managed to recreate one, run *'git status --short'* within the repository to see if you have successfully achieved the desired status. For further infromation, check appendix A at the end of this exercise.
- NOTE : **DO NOT COMMIT** these changes yet

Step 3 - Ignoring files

- Set git to ignore executable and log files by placing the following into a .gitignore file (at the root of your repository) :
    - `*.log`

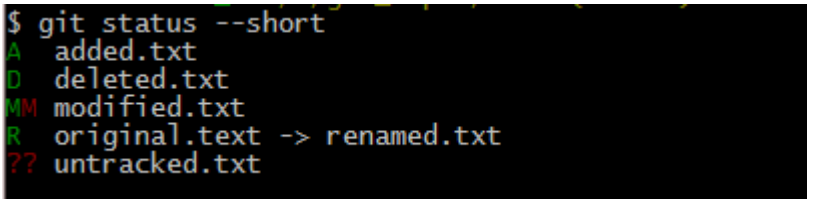
    - `*.exe`

- create some files which match the pattern, e.g. :
    - test.log
    - application.exe
- Run the git status command to see what output is provided for those files?
- Is there a way to ask git status to include output on ignored files? Where would you go to find out?

- *See appendix B for the answer*

Step 4 - Stashing changes

- You are now going to take the status that you have saved, and store it for re-use later as a stash.
  - use *git status* to make sure you have some staged, and some unstaged changes e.g.

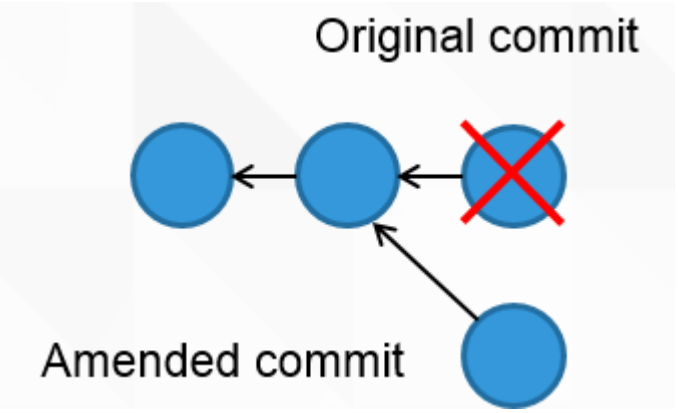


- You can then stash those changes for later use
  - *git stash save*
    - *Saved working directory and index state WIP on master : <hash> <commit msg>*
    - *Head is now at <hash> <commit msg>*
- You can verify the stash is there
  - *git stash list*
    - *stash@{0}: WIP on master: <hash> <commit msg>*
- Check that your working directory no longer contains those changes
  - *git status*
  - You will notice that any untracked files have not been included in the stash
- you can then re-apply your stash
  - *git stash apply*
- Check that your working directory is back to its previous state
  - *git status*
- You will see at this point that changes you previously staged through *git rm* or *git mv* are no longer staged, this is not the result we were after so lets blow away those changes and start again:
  - *git reset --hard*
    - *HEAD is now at <hash> <commit msg>*
- Now lets re-apply the stash, this time including the index
  - *git stash apply --index stash@{0}*
- Check that your working directory is back to its previous state
  - *git status*
- If you are happy with the results, you can safely delete the saved stash
  - git stash drop stash@{0}
    - *Dropped stash@{0} (<hash>)*
- Check the stash has been deleted
  - *git stash list*
- Finally, commit your changes
  - *git commit -m "status experimentation"*

Optional steps - going further

Step 1 - Fix a mistake

- At the end of the last step, you committed a set of files you created while creating some status output, it turns out, one of those files contained a modification you didnt intend to commit was a mistake.
- run the command:
  - *git log -2*
- take a note of the last two Hash ID's on your branch
- Firstly, you can recover the correct version of the file , in the command below, replace <file> with the name of a file you modified (it must be a modified file, **not** a new file)
  - *git checkout* HEAD^ <file>
  - This command will recover the version of <file> from the last commit before the current HEAD (HEAD is pointing at the commit we just created)
- Check your file no longer includes the previously committed modifications
- *git status* should show your file as modified and staged
- If it is not staged, stage it now
  - *git add <file>*
- The rest of the staging area looks exactly the same as the last commit.
- You can then overwrite your previous commit,
  - *git commit --amend -m "amended commit to correct <file>"*
- Now again run the command :
  - *git log -2*
- the last commit at the top of the branch should now have a different hash id



Step 2 - recover lost changes

- The original commit is now an unreachable object, but we have now lost the changes we originally made to <file>, we overwrote them with amend but forgot to keep a backup!
- We could avoid this in the future by simply copying the file outside of the working directory prior to the checkout, but we also know that the original commit includes this change.
- we could recover the entire commit but really, we only want the one file.
- Firstly, we need to figure out the ID of that commit
    - *git reflog*
- The commit in question should appear second on the resulting list
    - *<hash> HEAD@[1]: commit: status experimentation*
- The seven digit <hash> at the start of that line is the id we are looking for
- We can then checkout the file from that commit by specifying the hash
    - *git checkout <hash> <file>*
- *git status* will now show you have recovered your changes, and the modification is staged!
- Lets assume we dont want the file stages - we don't want to mistakenly commit it again
    - *git reset <file>*
- *git status* will now show the change as unstaged.

You could also recover the whole commit if you wanted

git checkout -b <branchname> <hash>

We will look at branches in more detail in the next module.

Appendix A

- *M = modified*
    - Red - change the contents of a file which is already under version control
    - Green - use *git add* on the modified file.
- ?? = unknown
    - Red - create a new file within your working directory but do not add it to version control.
- *A = added*
    - Green - create a new file within your working directory and stage it with *git add*
- *D = deleted*
    - Red - Remove a file from the working directory using OS commands (i.e right click on it)
    - Green - use *git rm* on a file already under version control.
- *R = renamed*
    - Green - rename or move a file with *git mv*
    - note - renaming by any other method will result in a delete and add rather than a rename.

Appendix B

- To find out the answer : *git status --help*
- To show ignored files with *git status - git status --ignored*

You have reached the end of this module and should now move onto module 5