[CS200]-STT II **Fall 2020-21**

Assignment: II

Name: Shubham Gupta RollNo: 11941140 Email: shubhamgupta@iitbhilai.ac.in

Collaborators Names: Aayush Deshmukh(11940010), Santaz Sahiti(11940230)

Question 1,
a) In Git, in a paricular repository, we can have multiple working directories by using the concept of linked working tree. You can construct multiple branches on the tree, where each branch will correspond to a working directory. On checking out of a branch, we can shift the working directory in the repository.
We will be using the worktree command here to demonstrate this working:
In the git repo, the command:
-> git worktree add <path>, will make a worktree in the repo.
This will allow multiple directories. Then,
-> git worktree add <path> -b branch1, will make a new branch (branch1) i.e. a new working directory.
Once, in this directory, we can do the work of our needed in this directory i.e branching, staging, committing etc. We can further in this worktree make more branches and directories and work with simultaneous directories in the requisite manner.

b) Four Git Commands are status, checkout, diff and add that will form a directed edge from the Staging Area(Index) to the current working directory.
We have used the four commands listed above in the following example of the git flow.
The sequence of commands is as follows:

We are currently in the master branch of the repo
Using the status to show where we are currently,

-> git status

Using checkout to make a new branch(-b) and move to that branch

-> git checkout -b firstbranch

Making a file, staging and committing to the branch

-> echo iitbhilai > 1.txt
-> git add 1.txt
-> git commit -m ”Commit in firstbranch is complete”

After the commit is done, use diff to find the changes/modification among the Index and working directory

-> git diff

In this manner, we are able to implement all the four commands and show the working across the Staging Area and Working Directory in the above git flow

c) To commit a file in parts in git we will use the the patch command,
For the file in the git repository, the command:
-> git add –patch <file>, will break the file into sensible parts called ”hunks”.
This ”hunk” is made during the staging area and the breaking of file is done. Once breaking is done, we can stage this ”hunk” for commit. We can allow git to commit the file, once it asks us to commit the file.
In this manner, as the ”hunk” is commited, we have essentially committed one part of a file.
Similarly, we can again distribute the file into ”hunks” and then commit the next part of the file as required. The sequence of commands:

-> git init Q1
-> cd Q1
-> touch first.txt
-> echo ”hello”>>first.txt

Breaking the files into hunks

-> git add –patch first.txt

Making few changes

-> echo ”google”>>first.txt

Putting the modified back in the repo

-> git add -N first.txt
-> git add –patch first.txt

On adding the required ”hunk” and then committing the file,
Obtaining the graph, we see the committed file is distributed in parts and gives multiple hashes:

-> git graph

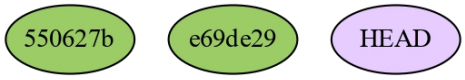

Figure 1.1: Single File in Parts

d) We have turned two differrent commits in the repository into a single commit by using the property of git rebase and squashing. Using squashing to actually turn the commits log to a single commit log. The rebase command is used to convert the commit to squash. The git flow is shown as follows:

Initialise the Git Repository

-> git init mergecommits
-> cd mergecommits

Making 5 different files and committing them individually

-> echo f1 > 1.txt
-> echo f2 > 2.txt
-> echo f3 > 3.txt
-> echo f4 > 4.txt
-> echo f5 > 5.txt
-> git add 1.txt
-> git commit -m ”1”
-> git add 2.txt
-> git commit -m ”2”
-> git add 3.txt
-> git commit -m ”3”
-> git add 4.txt
-> git commit -m ”4”
-> git add 5.txt
-> git commit -m ”5”

Checking the current state of the repo and obtaining the graph

-> git log
-> git graph

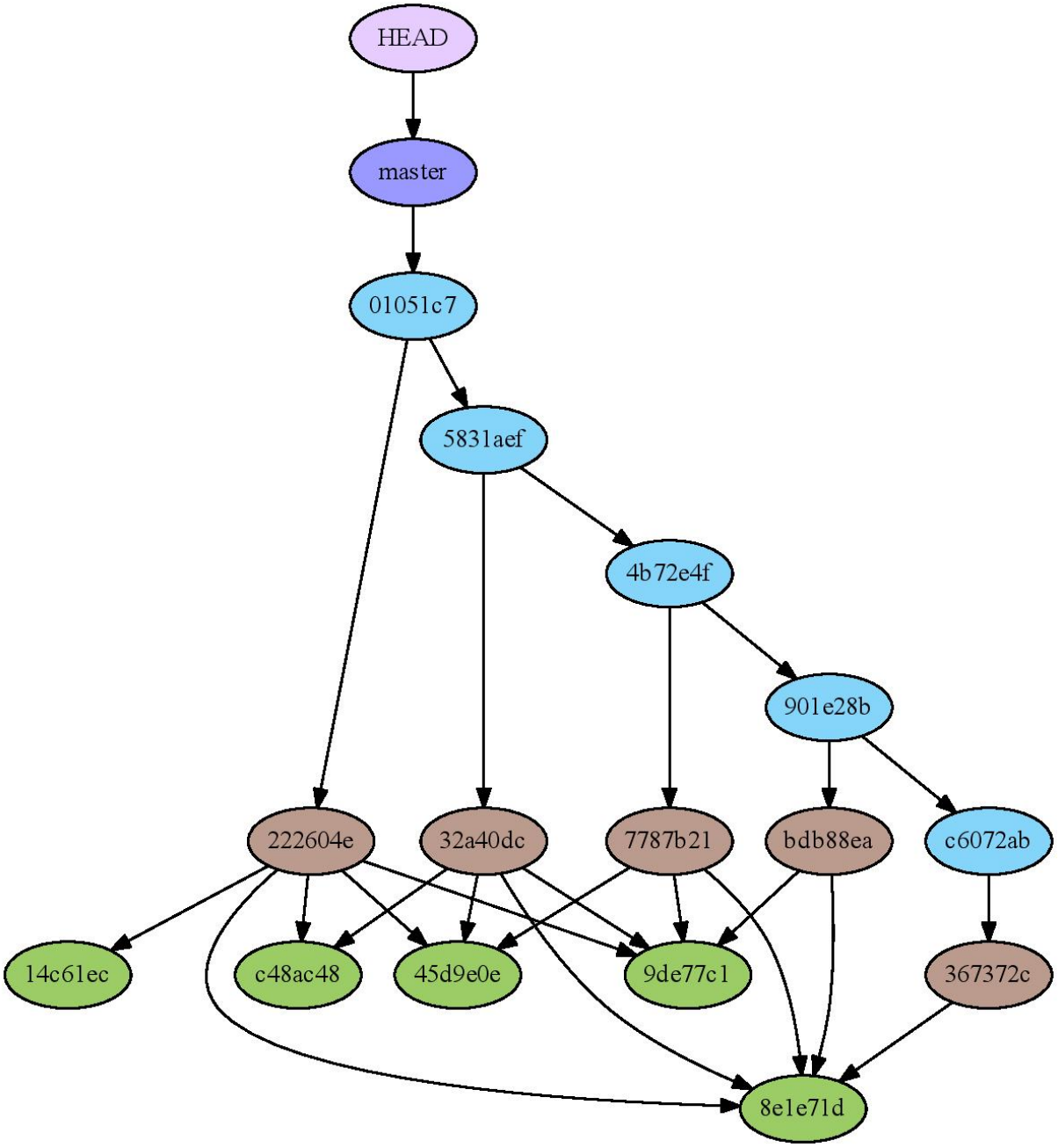

Figure 1.2: Git Graph after five commits

This git rebase command will help bring the last 2 commits(4 and 5) on an interactive console

-> git rebase –interactive HEAD~2 (~2 will indicate the last 2 commits)

Once, in the interactive session replace the pick of commit 5 with squash, this will allow the commit to be simultaneous(turn 2 commits as 1 in the log history)
Exit the interactive console by pressing Esc + Enter + :wq (twice) and notice the changes, you will see the two commits have actually turned into one on the screen.

Checking the current state of the repo

-> git log

Obtain the graph

-> git graph

You can verify the changes in the git graph samples attached below.

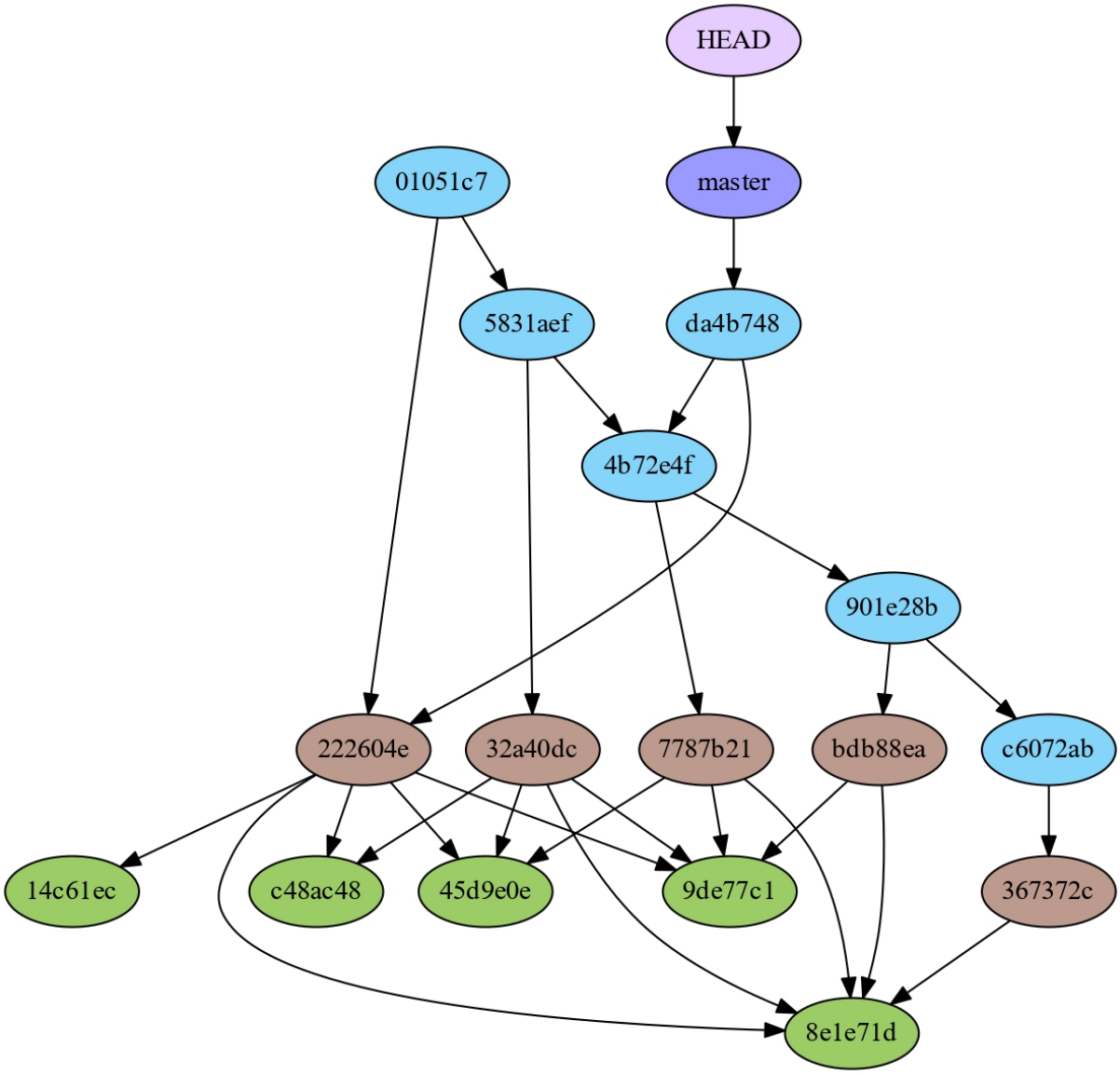

Figure 1.3: Git Graph after merging of commits