



CS 251 Outlab2 - Git and Bash

Bash (20 points)

Q1. File Handling [10 points]

Zomato food delivery needs a persistent storage which can store the information of their customers along with their order details. They need to store Customer's first name and Order ID. Along with this information they also need to keep track of time at which this data is stored in persistent storage. You need to create such storage using files.

Constraints: $1 \leq \text{OrderID} \leq 10^6$

Make a script **kv.sh** which implements the following functions:

1. storeInfo:

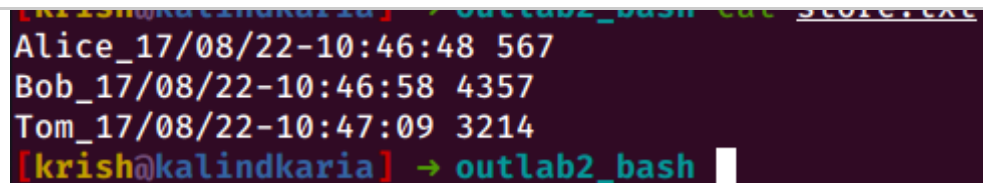


Published using Google Docs

[Learn More](#)[Report Abuse](#)

CS 251 Outlab2 - Git and Bash

Updated automatically every 5 minutes



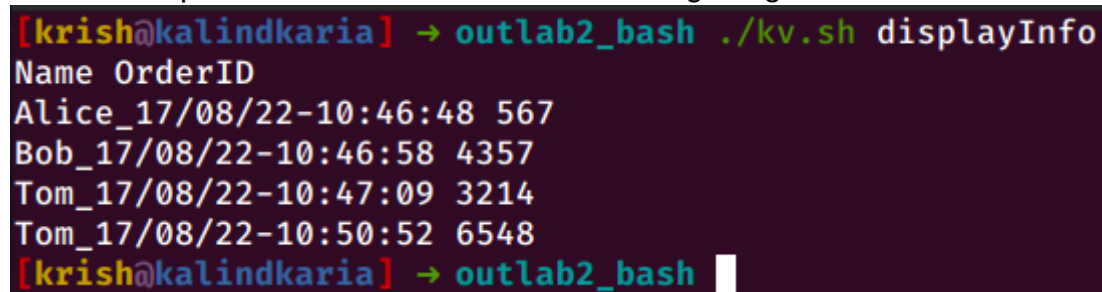
```
[krish@kalindkaria] -> outlab2_bash cat score.txt
Alice_17/08/22-10:46:48 567
Bob_17/08/22-10:46:58 4357
Tom_17/08/22-10:47:09 3214
[krish@kalindkaria] -> outlab2_bash
```

2. displayInfo:

Display information of Customer Name and Order ID on terminal.

E.g `./kv.sh displayInfo`

Show output on terminal as shown in following image



```
[krish@kalindkaria] -> outlab2_bash ./kv.sh displayInfo
Name OrderID
Alice_17/08/22-10:46:48 567
Bob_17/08/22-10:46:58 4357
Tom_17/08/22-10:47:09 3214
Tom_17/08/22-10:50:52 6548
[krish@kalindkaria] -> outlab2_bash
```

3. getOrderID:

Display the Order ID associated with given Customer Name

E.g `./kv.sh getOrderID Alice`



CS 251 Outlab2 - Git and Bash

Updated automatically every 5 minutes

```
OrderID's found are:
567
Alice ordered 1 times
[krish@kalindkaria] → outlab2_bash
```

Remember you need to store the <CustomerName,Order ID> pair in following format

CustomerName_DD/MM/YY-HH:MM:SS Order_ID

Note: Even if CustomerName is the same your this field (**CustomerName_DD/MM/YY-HH:MM:SS**) will always be unique because of timestamp. So duplicate CustomerName entries can be stored as well.

Q2. [10 points]

You are appointed as a top secret agent of R.A.W. team, India. The Head of R.A.W. team has decided to test your skills.

R.A.W. team has recently found some secret messages shared between two organizations that are a threat to our country.

These messages are intercepted and saved in a textual format and there is a suspect of few **recurring patterns and words**. Finding these patterns and words can help decoding



CS 251 Outlab2 - Git and BashUpdated automatically every 5 minutes

Interestingly, it has been found that all such messages have one thing common, i.e. the *very first pattern* of **#@\$** and the number of times these **patterns** and **words** are repeated is known beforehand.

Findings from the messages:

- All recurring **patterns** are of **length 3** and consists of **only special characters**, i.e., **!, @, #, \$, %, ^, & or ***.
- The very first pattern is common in all these messages, which is **#@\$**.
- The recurring **words** can be of any length and consist of small as well as capital letters.
- Number of times these patterns and words are repeated is known beforehand.
- Given that the number of occurrence = **3**, the syntax of their occurrence in a given text will be as follows:



CS 251 Outlab2 - Git and Bash

Updated automatically every 5 minutes

2. `<some_text> <first_recurring_word>
 <second_recurring_pattern>
 <second_recurring_word> <some_text>`

3. `<some_text> <second_recurring_word>
 <third_recurring_pattern>
 <third_recurring_word> OR <last_word>
 <some_text>`

4. `<some_text> <third_recurring_word> OR
 <last_word> <last_pattern> <some_text>`

Write a **Shell (.sh)** script to iterate over the given text using the syntax above and find out the ***last word*** and the ***last pattern*** found in these secret messages.

References:

- Regexper Example: [https://regexper.com/#\[A-Z\]\[a-z\]%2B\[0-9\]](https://regexper.com/#[A-Z][a-z]%2B[0-9])
- Regexper Documentation: <https://regexper.com/documentation.html>
- RegExr: <https://regexr.com>

Given

Two files are provided to solve this assignment.



CS 251 Outlab2 - Git and Bash

Updated automatically every 5 minutes

Procedure

- Open the skeleton program file, **decode.sh**.
- You will notice pre-written comments included in the skeleton program for your assistance to solve the assignment.
- To run and debug your solution, type the below command in Terminal:

`./decode.sh q2-sample.txt 3`

where **3** is the number of occurrences or times these patterns and words are repeated

- Refer to the **Expected Output** section below and debug your code to get the correct output.

Expected Output

- The provided sample TXT file, **q2-sample.txt** consists of some random text.
- The expected output of program **decode.sh** i.e., ***last_word*** and ***last_pattern*** found is shown below:



CS 251 Outlab2 - Git and Bash

Updated automatically every 5 minutes

We explore some advanced features of git in this lab

Q3. Git Hooks [10 points]

Git hooks allow developers to run custom actions upon / before / after execution of certain git commands like commit and push. This allows **automation** and prevents unwanted things from happening.

Read through <https://www.atlassian.com/git/tutorials/git-hooks> to get a detailed overview of git hooks. Look at <https://project-awesome.org/compscilauren/awesome-git-hooks#commit-msg> to find some good implementations of git hooks.

Use **bash** for coding the hooks and submit all the git hooks modified (without *.sample* extension) in folder **q3/**

In this task you have to implement git hooks that enforce the following functionality :

- **Reject** commits with commit message's character count > 40. [2 points]
- If a new branch is being pushed for the first time to the remote repo, make sure that the first commit message (commit next to base) where the branch is originating is "**Create <branch_name>**". [4 points]
- If any of the files being committed has the first line "**DO NOT COMMIT YET**" automatically **unstage** those files and proceed with committing other files.



CS 251 Outlab2 - Git and Bash

Updated automatically every 5 minutes

Q4. Rewriting history [15 points]

Merging is inelegant as it makes the commit history **non-linear** and adds an extra merge commit on top of that. **Rebasing**, on the other hand, maintains a linear history and also gives us more control over managing commits.

In this question we explore the technique of rebasing along with how to alter past commits.

Some references:

- <https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase>
- <https://www.atlassian.com/git/tutorials/rewriting-history>
- <https://www.atlassian.com/git/tutorials/rewriting-history/git-reflog>

git/ folder in the resources is the starter repository for this task. Analyze it using **git log --graph --all**. The repo is similar to what we developed in inlab with some minor differences.

Do the following:

1. Checkout to **hash_function** branch. Update the 3rd line of README.md to “<roll_no> is enjoying this lab.” (have no trailing whitespaces or newlines after this) under the last commit only with the commit message changed to “**Add README.md**”. See usage of **git commit --amend**.
2. Hermione accidentally committed **hash.o** file in the “**Update testing script**” commit which was added in **.gitignore** later. You have to remove the **hash.o** file from this and every following commit. See how you can use **git filter-branch** to achieve this. Make sure to



CS 251 Outlab2 - Git and Bash

Updated automatically every 5 minutes

interactive rebase for this purpose (**git rebase -i HEAD~n** with a suitable n).

Copy the repo and rename it to **q4_1** and place it in the submission folder (do it even if you have not attempted all the three parts).

4. After cleaning up the feature branch we would like to rebase it with master. Use **git rebase master**, resolve the conflicts and complete the rebase. Checkout to master and merge it with hash_function (fast-forward). You might want to use **git log --graph --all** to see what is going on.

You have to complete this part to get any credits for the following parts.

5. Now in master, we would like to split the first commit into two : “**Add hash declaration**” and “**Add main**” where just **hashing.hpp** and **main.cpp** are committed respectively. Make sure that the first commit is still under Harry but the second commit is under (**<roll_no>**, **<roll_no>**[@iitb.ac.in](mailto:iitb.ac.in)).
6. Change all of Hermione's commits (in master) under you (**<roll_no>**, **<roll_no>**[@iitb.ac.in](mailto:iitb.ac.in)). Can you do it in one command ?
7. Revert back to the state just before the merge with the rebased **hash_function** branch happened. See the usage of **git reflog**. Note that doing this should add just one extra reflog entry.

Copy the repo and rename it to **q4_2** and place it in the submission folder (do it even if you have not attempted the four parts)



CS 251 Outlab2 - Git and Bash

Updated automatically every 5 minutes

The following is what your submission directory must look like. Please follow the submission format strictly.

The submission directory must be named **<roll_no>_outlab2**

Compress it into a tarball using the following command

tar -cvzf <roll_no>_outlab2.tar.gz <roll_no>_outlab2

```
<roll_no>_outlab2
├── kv.sh
├── decode.sh
├── q3/
├── q4_1/ ( as per hash_function branch)
│   ├── .git/
│   ├── .gitignore
│   ├── hashing.cpp
│   ├── hashing.hpp
│   ├── main.cpp
│   ├── README.md
│   └── test.sh
├── q4_2/
│   ├── .git/
│   ├── .gitignore
│   ├── hashing.cpp
│   ├── hashing.hpp
│   └── main.cpp
└── .
```



Published using Google Docs

[Learn More](#)

[Report Abuse](#)

CS 251 Outlab2 - Git and Bash

Updated automatically every 5 minutes
