The bash script starts with the shell available on my system. Since the project is bash scripting, the shell should be /bin/bash. Following the shell, there are several echo that contain strings in it. Command echo is responsible for printing the text. The “Menu” is made up of echo in standard output.

The project takes number between 1 to 9, all of which does separate commands. Each command definition can be seen on the Menu. To logically put it, the user has to enter the related number of whatever function they need bash to perform. In order to exit the Menu, the user has to enter 9. Unless the user enters 9, the bash takes command numbers continuously. In the bash script that loop is written with “while true” loop. To perform the commands between 1 to 9, I used case conditional statement. When the user enters the number, the case statements matches that number, if there is no responding number in the case conditional statement, it outputs the message “Invalid number!”. In order to recognize the number the user entered, I used the read command. What read -p does is provide the prompt to the user asking to enter the number.

There are several echo “” commands throughout the whole bash script. I used these as a line separator to match the output given in the “OSS project 1” file.

1. The first command that gets the data of the movie identified by a specific movie id from file u.item, prompts the user to enter the movie id with “read -p … move\_id”. When the movie\_id is entered, it is being searched in the file u.item. The command awk is used for manipulating data, and -F ‘|’ is responsible for field separating (separates the line of data into sub-data using ‘|’). What -v movie\_id=”$movie\_id” does is that it assigns the movie\_id the user entered to shell variable movie\_id. Since the data in u.item is separated by ‘|’ the very first field should be the string before the first occurrence of ‘|’. Using the variable movie\_id, the command searches through the data of the first field that matches the variable move\_id. When the lines that match movie\_id are found, the whole data embedded in that line is printed to the terminal.
2. The second command prompts the user by asking “Do you want to get the data of 'action' genre movies from 'u.item'?(y/n)”. If the user inputs “y”, the script proceeds to find all data in u.item that contain the number 1 in their 7th field. Otherwise, the user has to choose from the Menu again. The data is being processed in a pre-organized field with the help of -F ‘|’ and only the first two fields of data which contain 1 in their 7th field are printed (movie id and movie title). Also, the command head -n 10 is used for displaying the first 10 data found in file u.item.
3. The third command asks for the user to enter the movie id 1 to 1682. When the variable movie\_id is entered, it is assigned to the shell variable movie\_id in the awk command -v option. There is no need to use field separator -F ‘|’, because the data in file u.data are already separated. All ratings of user-entered movie\_id, which are in the 3rd field, are being added altogether, along with the counting of how many ratings there are added one by one with the command $2==movie\_id {sum += $3; count++}. After this action is done, the action after the END, which is “{print sum/count}' u.data” starts to execute. The action after END prints the average rating of the movie, using the formula of dividing the sum of ratings by the count. The result of the awk command is assigned to the variable average. Therefore, the average rating is rounded to six decimals and assigned to the variable roundAverage by roundAverage=$(printf "%.6f" "$average"). And roundAverage is rounded up to five decimal places and printed out to the terminal using printf "average rating of $movie\_id: %.5f" "$roundAverage" command.
4. The fourth command asks the user “Do you want to delete the ‘IMDb URL’ from ‘u.item’?(y/n)”. The input is assigned to a variable res. If the res is equivalent to the character “y”, the command proceeds to print the expected output. If the res is not equal to y, the user is asked to choose another command from the Menu. The command sed stands for stream editor, we can use it to manipulate or substitute text files. In order to delete the URL, I used sed command substitution. In expression 's/|\*|http:[^|]\*|/|||/', s means substitution and the line |\*|http:[^|]\*| is the targeted stream to be substituted. The substituting stream, which in our case is |||, comes after the targeted stream. To put it logically, any data containing a URL(http:) starting and ending with ‘|’, should be substituted with ‘|||’ due to the command sed 's/|\*|http:[^|]\*|/|||/'. Lastly the command cat u.item | sed … | head -n 10 prints out the first 10 lines in u.item.
5. The fifth command is responsible for getting the data about users from u.user and manipulating the stream. The user is asked, "Do you want to get the data about users from ‘u.user’?(y/n)”. The input from the user is assigned to variable res. If the variable res is equal to y, the expected output will come. Otherwise, the user is asked to choose another command from the Menu again. Once the res is equivalent to “y”, the command sed 's/^/user /; s/|/ is /; s/|/ years old /; s/M/male /; s/F/female /; s/|//; s/|[0-9]\*$//' manipulates the stream. The first expression s/^/user /; puts the word “user” at the beginning of each line. After this, when the first occurrence of ‘|’ happens, it is substituted with “ is ”. As for the second occurrence of ‘|’, the second character ‘|’ is substituted with the string “ years old ”. According to the explanation output of command 5, the gender characters M and F should be converted to male and female respectively. In order to perform this manipulation, the occurrence of M is substituted with “male ” and the occurrence of F is substituted with “female ” with “s/M/male /; s/F/female /;”. After converting the gender characters to responding words, the third occurrence of ‘|’ should be deleted by the expression “s/|//;”, because the occupation should be printed out as it is. Lastly, the zip code along with the occurrence of ‘|’ at the end of the line should be deleted by s/|[0-9]\*$//. The command cat u.user | sed … | head -n 10 prints the first 10 lines only.
6. The sixth command modifies the format of the release date in u.item. The user is asked, "Do you want to Modify the format of ‘release data’ in ‘u.item’?(y/n)”. The answer of the user is assigned to variable res with read -p … res. If the variable res is y, the release date is modified and printed out. Otherwise, the user is asked to enter the number from the Menu again. Once the res is equal to “y”, the data in u.item are to be printed. In order to modify the format of the release date, the format dd-MMM-YYYY should be changed to yyyymmdd. The command sed 's/|\([0-9][0-9]\)-\([A-Za-z][A-Za-z][A-Za-z]\)-\([0-9][0-9][0-9][0-9]\)|/|\3\2\1|/; s/Jan/01/; s/Feb/02/; s/Mar/03/; s/Apr/04/; s/May/05/; s/Jun/06/; s/Jul/07/; s/Aug/08/; s/Sep/09/; s/Oct/10/; s/Nov/11/; s/Dec/12/;' captures each part with “\(...\)” then converts it using “|\3\2\1|” . The captured parts are then organized in the following order 3, 2, 1. In the initial format dd-MMM-YYYY, the day is at the beginning, and the captured part is referred to as 1. According to the sed command, the captured part 1 is placed at the end of the stream. Since the day is 2 digit number, the month is 3 character string, and the year is 4 digit number, \([0-9][0-9]\)-\([A-Za-z][A-Za-z][A-Za-z]\)-\([0-9][0-9][0-9][0-9]\)|/|\3\2\1|/ captures the parts and organize. The character ‘|’ at the beginning and at the end are supposed to be kept. The expression s/Jan/01/; s/Feb/02/; s/Mar/03/; s/Apr/04/; s/May/05/; s/Jun/06/; s/Jul/07/; s/Aug/08/; s/Sep/09/; s/Oct/10/; s/Nov/11/; s/Dec/12/; is responsible for modifying the month displayed as 3 character string to 2 digit number. For example, if the month is displayed as Jan, which stands for January, the command sed ‘s/Jan/01/’ changes Jan to 01.
7. The seventh command gets the data of movies rated by a specific user id from u.data. Unlike any other commands until now, this command output can be obtained using at least two files. The user is asked, "Please enter the ‘user id’(1~943) :". When the user id is entered, it is assigned to variable user\_id by the command read -p … user\_id. According to the prompt given, the movie id of all movies rated by the user\_id should be printed in ascending order first on the terminal. The command awk -v user\_id="$user\_id" '$1==user\_id {print $2}' u.data does assign the variable user\_id to shell variable user\_id using -v option. The awk command prints the second field of u.data if the first field matches with the shell variable user\_id. The command sort -n sorts in ascending order. And the command tr '\n' ' ' replaces the new line with space. To put it simply, the sorted field of u.data is in a column-ascending order, and tr command makes it in a row-ascending order. What the command sed 's/ /|/g; s/|$//' does is that all the spaces between movie id are replaced with the character ‘|’, and the last ‘|’ should be deleted (the last ‘|’ is denoted by “|$”). Consequently, the result is assigned to the variable movie\_id. Since the result is converted to a variable, echo “$movie\_id” is used to print out the result on the terminal. Thus, the command grep -E "^(($movie\_id))\|" u.item matches the pattern of u.item with ^(($movie\_id))\|. -E is an option of grep that enables extended expressions. The pattern to search for in u.item is "^(($movie\_id))\|". ^ matches the pattern with each line beginning and the data in movie\_id are enclosed in double brackets. The reason for using grep is that movie\_id contains several movie ids derived from the file u.data within itself, and it is not possible to match the variable movie\_id with data in u.item by using only movie\_id with awk. The command awk -F '|' '{print $1 "|" $2}' separates the field by ‘|’ and prints the matching data in u.item in “movie id|movie title” format. Lastly, head -n 10 prints out the first 10 movies rated by the user identified by the input user id.
8. The ninth command is the exit. When the user enters 9, the command prints out a string “Bye!” to the terminal with echo “Bye!”, and gets out of bash with exit 0.