Introduction to GNU/Linux systems

Exercises

Archil Elizbarashvili

1. Let's say you are standing in the /home/user/Documents directory. Go to  the /usr/share/common-licenses directory. Use both, the absolute  and the relative path.

**1. cd../../usr/share/common-licenses/   
 cd /usr/share/common-licenses**

2. Suppose you are standing in the /usr/local/bin directory. Go to the  /usr/share/common-licenses directory. Use both, the absolute and  the relative path.

**2. cd../../../usr/share/common-licenses/   
 cd /usr/share/common-licenses**

3. Suppose you are standing in some directory. Go to your home directory using  both, absolute and relative path.

**3.**  **cd  ~  
 cd /home/$user  
 cd $HOME**

4. Create files with the following names: "name", "name surname", "name \ surname", "'name" surname' "," '".", "' / \"

**4.** **touch name**

**touch "name surname"**

**touch "name \ surname"**

**touch "'name\" surname'"**

**touch "'"**

**touch "' / \\"**

5. Create all possible two-character filenames with A and B symbols.

**5.** **touch {A,B}{A,B}**

6. Create all possible four-character filenames with A, B and C symbols.

**6. touch {A,B,C}{A,B,C}{A,B,C}{A,B,C}**

**touch {A..C}{A..C}{A..C}{A..C}**

7. In the directory “test”, create all possible three-character with \, $, @ and ' symbols.

**7. mkdir -p test && touch test/{\\,\$,@,\'}{\\,\$,@,\'}{\\,\$,@,\'}**

**mkdir -p test && cd test && touch {\\,\$,@,\'}{\\,\$,@,\'}{\\,\$,@,\'}**

8. Create 10000 html extension files in the HTML directory. Do the operation in  the Shell with one command!

**8. mkdir HTML && cd HTML && touch {1..10000}.html**

9. Create filenames "name1", "name2" and "name3" with and without the  "txt" file extension. Do the operation in the Shell with one command!

**9. touch name{1..3}{,.txt}**

* **{,.txt} represents a brace expansion that expands to an empty string and ".txt". This adds the option to create files without the ".txt" extension and with the ".txt" extension.**

10. Create filenames "name1", "name2" and "name3" with the "txt" file  extension. The file extension should be given in all character case variants - uppercase, lowercase and mixed: txt, txT,… TXT. Do the operation in the  Shell with one command!

**10.** **touch name{1..3}.{t,T}{x,X}{t,T}**

11. Create three-character directory names where the first character is from a to z,  the second one is from A to Z, and the third one is a digit. Do the operation in  the Shell with one command!

**11. mkdir {a..z}{A..Z}{0..9}**

12. Create directories of all possible two-character names with the hexadecimal  characters. Do the operation in the Shell with one command!

**12. mkdir -p {{0..9},{A..F}}{{0..9},{A..F}}**

13. Create all possible three-character directory name variants with digits. In each  directory create one-character, two-character, three-character and four character filenames using the following symbols: A, B and C. Do the operation  in the Shell with one command!

Then display the following files:

A) 01234 ̅̅̅̅̅̅̅̅̅∀56789 ̅̅̅̅̅̅̅̅̅/∀CB̅̅̅̅

B) All two-character filename located in directories whose name does not  include the symbol "5"

C) All one and three-character filenames not containing "B" and located in  the directory name including the only symbols "1", "3" and "9"

13. **mkdir -p {0..9}{0..9}{0..9} && for dir in {0..9}{0..9}{0..9}; do (cd "$dir" && touch {A,B,C} {A,B,C}{A,B,C} {A,B,C}{A,B,C}{A,B,C} {A,B,C}{A,B,C}{A,B,C}{A,B,C}); done**

**mkdir {000..999} && touch {000..999}/{A..C}{,{A..C}}{,{A..C}{A..C}}**

**B) find . -type d -name '[^5][^5][^5]' -exec find {} -type f -name '??' \;**

**C)find {1,3,9}{1,3,9}{1,3,9}/ -type f -name '?' -o -name '???' | grep -E '^[^B]+$'**

14. Suppose you have a lot of image files with jpg, bmp and png extensions in  the "Photos/" directory. Select the jpg and png files containing at least one  digit in the filename and whose first letter is between D to P. Do the operation  in the Shell with one command!

**14. ls Photos/{D..P}\*{[0-9]\*,\*}.{jpg,png}**

**ls [D-P]\*[1-9]\*.{jpg,png}**

**find Photos/{D..P}\*{[0-9]\*,\*}.{jpg,png} -type f**

15. Suppose you have a lot of image files with jpg, bmp and png extensions in  the "Photos/" directory. Select the jpg and png files (file extension can be  given in uppercase, lowercase, or mixed cases) containing at least one digit in  the filename and whose first letter is between D to P. Do the operation in the  Shell with one command!

15. **find Photos/ -type f -iname "[D-P]\*[0-9]\*.[jp][pn]g"**

**ls Photos/"[D-P]\*[0-9]\*.{[jJ][pP][gG],[pP][nN][gG]}”**

**find Photos/ -type f -iname "[D-P]\*[0-9]\*.[jpPnN][pnNPgG]"**

**ls Photos/[D-P]\*[0-9]\*.{[jJ][pP][gG],[pP][nN][gG]}**

**ls [D-P]\*[1-9]\*.{{J,j}{p,P}{g,G},{p,P}{n,N}{g,G}}**

16. Suppose you have a lot of video files with flv, vob and ogg extensions in  the "Videos/" directory. Select the flv and ogg files (file extension can be  given in uppercase, lowercase, or mixed cases) containing at least one digit in

the filename and whose last letter is between K to Q. Do the operation in the  Shell with one command!

16. **ls Videos/\*[0-9]\*{K..Q}.{[fF][lL][vV],[oO][gG][gG]}**

**find Videos/ -type f \( -iname "\*[0-9]\*[K-Q].flv" -o -iname "\*[0-9]\*[K-Q].ogg" \)**

17. Suppose you have a lot of documents with pdf and txt extensions in the  "Documents/" directory. Select the seven-character (excluding the  extension) .pdf files and the .txt files containing at least 2 digits and whose  first letter is between C to R. Do the operation in the Shell with one  command!

17. **find Documents/ -type f ( -name "[C-R]\*[0-9]\*[0-9]\*.txt" -o -name "[C-Rc-r][0-9]\*[0-9]\*.pdf" )**

18. Describe graphically the following tree structure: /usr/{local/{bin,lib},lib/{gimp/?.?\*,emacs/[0-9]\*.[1- 9]}}Describe graphically the following tree structure:  /usr/{share/{common-licenses/{,L}GPL-

?,doc/HOWTO},{local/{bin,lib},lib/{gimp/?.?\* , emacs /  [0-9] \*. [1-9] ac}}

**18. /usr**

**├── local**

**│   ├── bin**

**│   └── lib**

**├── lib**

**│   ├── gimp**

**│   │   ├── ?.?\***

**│   └── emacs**

**│       └── [0-9]\*.[1-9]**

**/usr**

**├── share**

**│   ├── common-licenses**

**│   │   ├── (empty)**

**│   │   └── LGPL-?**

**│   └── doc**

**│       └── HOWTO**

**├── local**

**│   ├── bin**

**│   └── lib**

**└── lib**

**├── gimp**

**│   ├── ?.?\***

**└── emacs**

**└── [0-9]\*.[1-9]ac**

19. Suppose you have a filename toto containing another filename. Display the  content of the latter file on the screen. Do the operation in the shell with and  without a pipe.

**19.  cat $(cat toto)**

**cat toto | xargs cat**

20. Suppose you have a filename toto containing another filename that contains  another filename as well. Display the size of the latter file on the screen. Do  the operation in the shell with and without a pipe.

**20. stat -c %s $(cat $(cat toto))**

**cat toto | xargs cat | xargs stat -c %s**

**du -sh $(cat $(cat toto)) (Kilobytes)**

**du -sb $(cat $(cat toto))**

21. Suppose you have a filename foo in the bar/ directory containing the full  path to another filename. Display the size of the latter file on the screen. Do  the operation in the shell with and without a pipe.

**21. du -sh $(cat bar/foo)**

**/home/alex/exam/filedestination**

**cat bar/foo | xargs echo | xargs du -sh**

**/home/alex/exam/filedestination**

22. Create a filename toto if it does not exist.

**22. touch toto**

**[ -e "toto" ] || touch "toto"**

**[ ! -e "titi" ] && touch "toto"**

23. Create a filename toto if no titi filename exists. Do not display an error  message on the screen.

**23. [ -e "titi" ] || touch "toto" 2>/dev/null**

**ls titi 2>/dev/null || touch toto**

**[ ! -e "titi" ] && touch "toto" 2>/dev/null**

24. Create a filename toto if the filename foobar exists, otherwise create a  filename titi. Do not display an error message on the screen.

**24. test -e "foobar" && touch "toto" || touch "titi" 2>/dev/null**

25. Run the "ls -l" command on one terminal and display the output on the  other one.

**25. mkfifo mypipe (First Terminal)**

**ls -l | tee mypipe (First Terminal)**

**cat < mypipe (Second Terminal)**

**This is the FIFO method.**

**ls -l > /dev/pts/1**

26. Run a file content viewer command on one terminal and display the output  on the other one.

26. **mkfifo mypipe2 (First Terminal)**

**cat foo | tee mypipe2 (First Terminal)**

**cat < mypipe2 (Second Terminal)**

**cat foo > /dev/pts/1**

**(Launch PS command to figure out the tty code)**

**This is the FIFO method.**

27. Copy the toto file content to the titi file. Do not display an error message  on the screen if toto doesn’t exist.

**27. cat toto >> titi 2>/dev/null**

**cp toto titi 2>/dev/null**

28. Append the list of .conf files from the /etc directory to the  list\_conf.txt file of your home directory.

**28. find /etc -type f -iname "\*.conf" >> ~/list\_conf.txt**

**find /etc -type f -iname "\*.conf" >> $HOME/list\_conf.txt**

29. Append the list of .conf files from the /etc directory to the  list\_conf.txt file of your home directory and display the list on the  screen as well. Do the operation in the Shell with one command!

**29. find /etc -type f -name "\*.conf" | tee -a ~/list\_conf.txt**

30. Append the list .txt files of your HOME directory to the /tmp/list.txt file and display the list on the screen as well. Do the operation in the Shell  with one command!

**30. find ~/ -maxdepth 1 -type f -iname “\*.txt” | tee -a /tmp/list.txt**

**find $HOME/ -maxdepth 1 -type f -iname “\*.txt” | tee -a /tmp/list.txt**

31. Redirect the list of all files of /etc directory to the list\_etc.txt file and  list of .conf files of /etc directory to the list\_etc\_conf.txt file. Do  the operation in the Shell with one command!

**31. ls /etc > list\_etc.txt && ls /etc/\*.conf > list\_etc\_conf.txt**

32. Suppose you have 3 files: n1.txt, n2.txt, n3.txt each containing a  random number on one line. Calculate the following arithmetic expressions (n1+(n2\*n3))/n2 where n1, n2 and n3 are the file contents.

**32. result=$(($(cat n1.txt) + ($(cat n2.txt) \* $(cat n3.txt)))) && echo "scale=2; $result / $(cat n2.txt)" | bc**

33. Calculate the π value with the first 100 decimals.

**33. echo "scale=100; 4\*a(1)" | bc -l**

34. Create the variable named File and assign the value of an arithmetic  expression (for example: 12+3-5). Create a file with the same name as the  value of the arithmetic expression (10, 10=12+3-5). Do the operation in  the Shell with one command!

**34. File=$((12+3-5)) && touch $File**

35. Create the variable named File and assign the value of an arithmetic  expression (for example: 12+3-5). Edit a file with the same name as the value  of the arithmetic expression (10, 10=12+3-5). After inserting some  content into the file copy it to the new filename with the same name  expression plus 1 (e.g. 11=10+1). Do the operation in the Shell with one  command!

**35. File=$((12+3-5)) && echo "I am awesome" > $File && cp $File $((File+1))**

36. Color the shell invitation.

**36. PS1="\[\033[38;5;208m\]\u@\h:\w\$\[\033[0m\] "**

37. Change Shell invitation to: "I'm in… (current directory name),  Now it is… (current time), this is… (command number)  the command"

**37. PS1='I'"'"'m in \w, Now it is \A, this is \# the command: '**

**PS1='I'"'"'m in $(pwd), Now it is $(date "+%H:%M"), this is \# the command: '**

38. Create the following type of colored shell invitation. «My processes  number is - N $».

**38. PS1="My processes number is - \# \$ "**

39. Which of the following statements best describes what happens in the output  of the “$ cat <ls> cat” command?

A) Redirect the "ls" file content to the standard output of the "cat"  command

B) Concatenates the "<ls>" and the "cat" files on the screen

**C) Overwrites the "cat" file with the content of the "ls" file**

D) Incorrect command

E) Transfers the "ls" command output to the "cat" file

40. Which of the following statements best describes what happens in the output  of the ”$ cat <cat> ls” command?

A) Redirects the "ls" file content to the standard input of the "cat"  command

B) Concatenates the "<cat>" and the "ls" files on the screen

**C) Overwrites the "ls" file with the content of the "cat" file**

D) Incorrect command

E) Redirect the "cat" command output to the "ls" file

41. Which of the following statements best describes what happens in the output  of the “$ wc file.txt | tee wc” command?

**A) Prints the newline, word and byte counts for the file.txt and  copies them to the filename ”wc”**

B) Opens the ”file.txt” and the “wc” files in the text editor named “wc”

C) Prints the new line, word and byte counts for the file.txt and opens  the file “tee” with the command “wc”

D) Incorrect command

E) Prints the newline, word and byte counts for the file.txt and  redirects them to the standard input of the “tee” and the “wc”  commands

42. Analyze the following two commands. Are they doing the same thing?

A) $ cmd 2>&1 1>file

B) $ cmd 1>file 2>&1

**42. No, they are not doing the same thing. The order of the redirections are different.**

43. Open a terminal and run the following commands:

A) $ VAR1 = "Hello"

B) $ VAR2 = "Kiel”

C) $ export VAR1

D) $ bash

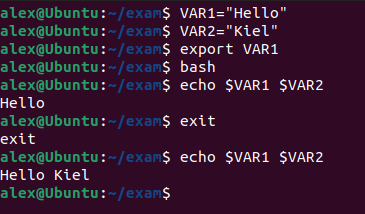
E) $ echo $ VAR1 $ VAR2

F) $ exit

G) $ echo $ VAR1 $ VAR2

Will the ”echo” command print the same result in both cases? Explain the result.

**43. The echo command will not print the same result in both cases due to the use of the export command and the creation of a new shell with bash. In the first case, the export VAR1 command makes the VAR1 variable available to child processes of the current shell. However, VAR2 is not exported, so it is not available to child processes. When you run bash to start a new shell (a child process), and then echo $VAR1 $VAR2, only VAR1 will be printed because VAR2 is not available in the new shell. After exiting the new shell with exit and running echo $VAR1 $VAR2 again in the original shell, both VAR1 and VAR2 will be printed because both variables are available in the original shell**

****

44. Open a terminal and run the following commands:

A) $ VAR1 = "Hello"

B) $ VAR2 = "Kiel"

C) $ export VAR1

D) $ echo $VAR1 $VAR2

E) $ unset VAR1

F) $ echo $VAR1 $VAR2

Will the ”echo” command print the same result in both cases? Explain the result.

**44. In the first case, VAR1 and VAR2 are both printed as they are both defined and available in the current shell. However, after calling unset VAR1, the VAR1 variable is removed, so when echo $VAR1 $VAR2 is run again, only VAR2 is printed**

45. Open a terminal and run the following commands:

A) $ VAR1 = "Hello"

B) $ VAR2 = "Kiel"

C) $ export VAR1 VAR2

D) $ echo $VAR1 $VAR2

E) $ export -n VAR1

F) $ bash

G) $ echo $VAR1 $VAR2

H) $ exit

I) $echo $VAR1 $VAR2

Will the echo command print the same result in all three cases? Explain the result.

**45. In the first case, VAR1 and VAR2 are both printed as they are both defined and exported to child processes. However, after calling export -n VAR1, the VAR1 variable is no longer exported to child processes. When a new shell is started with bash and echo $VAR1 $VAR2 is run, only VAR2 is printed because VAR1 is not available in the new shell. After exiting the new shell with exit and running echo $VAR1 $VAR2 again in the original shell, both VAR1 and VAR2 are printed because both variables are available in the original shell**

46. Open a terminal and create the following variables with the given values:

1. $ VAR1 = "Hello"

2. $ VAR2 = "Kiel"

Write answers for each command:

A) echo $VAR1$VAR2

B) echo $VAR1VAR2

C) echo ${VAR1}${VAR2}

D) echo $VAR13$VAR24

E) echo ${VAR1}${VAR2}4

F) echo $VAR1${{VAR2}

G) echo $VAR1{VAR2}

* **echo $VAR1$VAR2 will print the concatenation of the values of VAR1 and VAR2 because the variables are directly adjacent.**
* **echo $VAR1VAR2 will print nothing or an error message because it tries to print the value of a non-existent variable named VAR1VAR2.**
* **echo ${VAR1}${VAR2} will print the concatenation of the values of VAR1 and VAR2 because the braces {} separate the variable names.**
* **echo $VAR13$VAR24 will print nothing or an error message because it tries to print the values of non-existent variables named VAR13 and VAR24.**
* **echo ${VAR1}${VAR2}4 will print the concatenation of the values of VAR1, `VAR2**

47. Copy the “ls” command from the /bin directory to $HOME/bin and run it.

**47. cp /bin/ls $HOME/bin # Copy the "ls" binary to $HOME/bin**

**$HOME/bin/ls # Run the copied "ls" command**

48. exchange the “ps” and the “cat” commands effect (note that you do not have  administrator privileges!)

**48. nano ~/.bashrc**

**function ps() {**

**command cat "$@"**

**}**

**function cat() {**

**command ps "$@"**

**}**

**source ~/.bashrc**

**cd linux-training**

**cp /bin/cp ./ps**

**cp /bin/mv ./cat**

**PATH=$PWD:$PATH**

49. Remove all files and directories except those name start with “r” in the  directory $HOME/tmp

**49. rm $HOME/tmp/[^r]\***

50. Empty the $HOME/tmp directory. Delete hidden files as well. (Don’t’ try do  delete the . and .. directories). ~~rm -rf /tmp/dir{.[^.],.??\*,\*}~~

50.

1. **rm -rf /tmp/dir{.[^.],.??\*,\*}: This command uses brace expansion in Bash to generate three patterns: .[^.], .??\*, and \*. The .[^.] pattern matches hidden files that have one character after the dot, the .??\* pattern matches hidden files that have two or more characters after the dot, and the \* pattern matches all non-hidden files. The -rf option tells rm to remove files and directories recursively and without asking for confirmation.**
2. **find "$HOME/tmp/" -mindepth 1 ! -name '.' ! -name '..' -delete: This command uses the find utility to search for files in the $HOME/tmp directory. The -mindepth 1 option tells find to ignore the $HOME/tmp directory itself. The ! -name '.' ! -name '..' expression tells find to ignore files named . and ... The -delete option tells find to delete all files that match the search criteria**
3. **find $HOME/tmp -mindepth 1 -maxdepth 1 ! -name '.' ! -name '..' | xargs rm -rf**
4. **sudo rm -rf \* .[!.]\***

51. Which of the following commands empty the $HOME/tmpfile file  (multiple choice):

**A) cat /dev/null > $HOME/tmpfile**

B) cp /dev/null > $HOME/tmpfile

C)echo "" > $HOME/tmpfile

D) echo > $HOME/tmpfile

E) echo -n > $HOME/tmpfile

**F) true > $HOME/tmpfile**

**G) > $HOME/tmpfile**

**H) truncate -s0 tmpfile**

52. Find all .html files in your HOME directory and redirect the list to the  fichier.html file.

**52. find $HOME -maxdepth 1 -type f -name "\*.html" > fichier.html**

**ls $HOME/\*.html | xargs -I {} basename {} > fichier.html**

53. Find all files whose contents were last modified 2 days ago.

**53. find $HOME -type f -mtime 2**

54. Find all files whose contents were last modified 2 days ago and whose size are  between 23 B and 145 B. Display the list.

**54. find -type f -mtime -2 -size +23c -size -145c**

55. Find all files whose contents were last modified 1 day ago.

**55. find -type f -mtime 1**

56. Find all files whose contents were last modified 45 day ago and more recently  than the file toto.

**56. find -type f -mtime -45 -newer toto**

**find $HOME -type f -newer $HOME/toto -mtime -45**

57. Find all files whose contents were last modified 45 minutes ago.

**57. find $HOME -type f -mmin -45**

**find ~ /tmp -type f -mmin -45**

58. Find all files whose contents were last accessed 15 days ago.

**58 find $HOME -type f -atime 15**

* *-mtime +n* = finds the files and directories modified more than *n* days ago
* *-mtime -n* = finds the files and directories modified less than *n* days ago
* *-mtime n = finds the files and directories modified exactly n days ago*

59. Find all empty regular type files.

**59. find $HOME -type f -empty**

**60. Find all .txt files (extension can be in lowercase, uppercase or mixed)**

**60. find $HOME -type f -iname "\*.TXT"**

**ls $HOME/\*.{t,T}{x,X}{t,T} 2>/dev/null**

61. Find all directory type of files whose names start with the “a" or all not  directory type files whose names end with "k".

**61. find / -type d -iname “a\*” |  find / -type f -iname “\*k”**

62. Show the difference: find. -size between n, -n, + n.

**62.**

1. **find . -size n: This command will find files whose size is exactly n blocks. A block is 512 bytes.**
2. **find . -size -n: This command will find files whose size is less than n blocks.**
3. **find . -size +n: This command will find files whose size is more than n blocks.**

63. Find all files that have exactly the following permissions set: read for the  owner as well as for the group, and read/modify for other users.

**63. find . -type f -perm 664**

64. Find all files that have at least the following permissions set: read for the  owner as well as for the group, and read/modify for other users.

**64. find . -type f -perm -664**

65. Find all files that have the following permissions set: read for the owner as  well as for the group, and read/modify for other users.

**65. find /path/to/search -type f -perm /644**

66. Files all directories with sticky bit access: 777

**66. find /path/to/search -type d -perm -777**

67. Find all files in your home directory that you do not have access to.

**67. find $HOME -type f ! -readable**

**find $HOME -type f -not -readable**

68. Find all files that have an active SUID bit.

**68. find / -perm /4000**

69. Find all the regular type files in your home directory with 777 permissions  and convert them to 644

**69.  find $HOME -type f -perm 777 -exec chmod 644 {} \;**

**find $HOME -type f -perm 777 | xargs chmod 644**

70. Find all .mp4 or .avi files (extension can be in lowercase, uppercase or  mixed) in your home directory and delete them.

**70. find $HOME -type f -iname "\*.avi" -o -iname "\*.mp4" | xargs rm -fr**

71. Find all .mp4 or .avi files (extension can be in lowercase, uppercase or  mixed) in your home directory larger than 100 MB and delete them.

71. **find $HOME -type f \( -iname "\*.avi" -o -iname "\*.mp4" \) -size +100M -exec rm -f {} +**

**find $HOME -type f \( -iname "\*.avi" -o -iname "\*.mp4" \) -size +100M | xargs rm -fr**

72. Delete all empty directories in your home directory.

**72. find $HOME -type d -empty -exec rmdir {} \;**

**find $HOME -type d -empty | xargs rm -d**

**find $HOME -type d -empty -delete**

73. Find all files that have been read in the last two hours.

**73. find / -type f -amin -120**

**find /path/to/directory -type f -amin -120**

74. Find all configuration files (.conf) in /etc in directories, not in  subdirectories.

**74. find /etc -maxdepth 1 -type f -name “\*.conf”**

75. List running processes owned by the current user.

**75. ps -u $USER**

**ps -u $(whoami)**

**ps -p $(pgrep -u $USER)**

**ps -u $USER -o pid,ppid,cmd,%cpu,%mem**

**top -n 10 | grep $USER**

**htop --user $USER**

76. Findthe syslog process PID.

**76. pgrep rsyslog**

**ps aux | grep rsyslog**

77. Compare the number of processes owned by $USER and root.

**77. ps -e -o user= | grep $USER | wc -l**

**ps -e -o user= | grep root | wc -l**

**bc <<< "$(ps -e -o user= | grep root | wc -l) - $(ps -e -o user= | grep $USER | wc -l)"**

**echo $(ps -u root|wc -l)-$(ps -u|wc -l) | bc**

78. Sort all processes by CPU usage.

**78. ps aux --sort -%cpu**

79. Sort all processes by memory usage.

**79. ps aux –sort -%mem**

80. "ps -aux" is different from "ps aux"?

**80. The commands ps -aux and ps aux are effectively the same. The ps command is used to report a snapshot of the current processes. When used with the aux option, it displays detailed information about all processes. The difference in syntax is due to historical reasons: POSIX systems prefer options with a dash (like -aux), whereas BSD systems do not use a dash (like aux)**

81. Display a list of processes in a user-defined format: namely - PID number,  owner name and command name.

**81. ps -eo pid,user,comm**

82. Run the command top (in batch mode) and save the first 10 processes in  10p.txt files for further processing.

**82. top -b -n 1 | head -17 | tail -10 > 10p.txt**

**top -b -n 1 | head -n 17 > 10p.txt (Includes more information and headers)**

83. Run the top command (in batch mode) and save the list of the first 5 iterative  processes in the 5p.txt file for further processing.

**83. top -b -n 5 | tail -n +8 | head -5 > 5p.txt**

**In this command, we use tail -n +8 to skip the first 7 lines of top's output (which contain header information) and then use head -5 to select the first 5 lines of the remaining process list. The output is redirected to the 5p.txt file.**

84. Run the top command (in batch mode) to output only the working processes  and save the list of processes extracted from the first 10 iterations in a  10p.txt file for further processing.

**84. top -b -n 10 | tail +8 | head -10 > 10p.txt**

85. Run the command tee filename in the background. Bring the process to  the foreground, then re-move and run this command in the background  again.

**85.**

**# Start the command in the background**

**tee filename &**

**# Bring the background process to the foreground**

**fg**

**# Press Ctrl+Z to suspend the process**

**# Use the bg command to resume it in the background**

**bg**

86. Show the firefox process PID if it is running and kill it. if not, do not  display error on screen.

**86. pgrep firefox && pkill -9 firefox 2>/dev/null**

87. Display the sum of current time seconds, minutes, and hours.

**87. echo $(($(date +%H)\*3600 + $(date +%M)\*60 + $(date +%S)))**

88. Which day of the week was 01/01/2001.

**88. date -d "2001-01-01" "+%A"**

89. Display the day of the week, which will be in 2 years, 5 months and 6 days.

**89. date -d "$(date -d '+2 years 5 months 6 days' +'%Y-%m-%d')" "+%A"**

90. Display the day of the week which was 2 years, 5 months and 6 days ago.

**90. date -d "$(date -d '+2 years 5 months 6 days' +'%Y-%m-%d')" "+%A"**

91. Display the day of the week which will be in 2 years, 5 months and 6 days  ago.

91. **date -d "$(date -d '-2 years 5 months 6 days' +'%Y-%m-%d')" "+%A"**

92. Display the day of the week, which will be 2 years, 5 months ago and 6 days  after.

92. **date -d "$(date -d '+2 years 5 months 6 days' +'%Y-%m-%d')" "+%A"**

93. Display the number of seconds that have elapsed since the creation of UNIX.

**93. echo $(($(date +%s) - $(date -d '1970-01-01 00:00:00' +%s)))**

94. Display 5-10 lines of 2-5 columns of the file toto and redirect the output to the following filename: toto\_YYMMDDHHMM (where, YYMMDDHHMM is the current year, month, day, hour and minute)

**94. cut -f 2-5 toto | sed -n '5,10p' > "toto\_$(date +'%y%m%d%H%M')"**

95. Print the 3-5 columns of the toto file, where the columns are separated by  symbol “:”.

**95. awk -F: '{print $3, $4, $5}' toto**

**cat toto | cut -d : -f 3-5**

96. Find the duplicate lines in the toto file and display the count.

**96. uniq -c -d file.txt**

**sort toto | uniq -d | wc -l**

97. Remove all regular type files in the $HOME/tmp directory those name start  with a, m and r.

**97. find $HOME/tmp/ -type f \( -name "a\*" -o -name "m\*" -o -name "r\*" \) -exec rm -f {} \;**

**find $HOME/tmp/ -type f -name “[amf]\*” | xargs rm -f**

98. Count files in your home directory.

**98 find $HOME -type f | wc -l**

99. Suppose you have 4 files: surname.txt (contains student’s last names per  line), name.txt (contains students' names per line), brdate.txt (contains

student’s birthdays per line), place.txt (contains students' birth places per  line). Collect all this information in the students.txt filename containing  each student's last name, first name, date and place of birth per line.

**99. paste -d ' ' surname.txt name.txt brdate.txt place.txt > students.txt**

100.Suppose you have the directory $HOME/student containing multiple files.  Each file contains the students' name, surname, date of birth and place per  line. Compile all this information into one file student.txt which will  contain all student's name, surname, date of birth and place per line.

**100. cat list1.txt list2.txt list3.txt > students.txt**

101.Join two file: f1.txt and f2.txt. the first field of f1.txt and the third  field of f2.txt are the same.

**101. cat f1.txt f2.txt | sort | uniq > merged.txt**

102.Show all .jpg and .png file sizes in your home directory.

**102. ls $HOME/** **\*.{jpg,png}**

**du -h $HOME/\*.{jpg,png}**

103.Show all .jpg and .png files total size in your home.

**103.** **find $HOME -type f -iname "\*.jpg" -o -iname "\*.png" -exec du -csh {} + | grep total$**

104.List files in one column.

**104. ls -1**

105.Create the archive named archive\_YYMMDD.tar containing all files from  your home directory.

**105. tar -cvf archive\_$(date +%y%m%d).tar $HOME/\***

106.Create the compressed (gzip) archive of your personal directory named  archive\_YYMMDD.tgz

**106. tar -czvf archive\_$(date +%y%m%d).tgz $HOME/\***

107.Create the compressed (bzip2) archive of your personal directory named  archive\_YYMMDD.tbz2

**107. tar -cjvf archive\_$(date +%y%m%d).tbz2 $HOME/\***

108.Create the archive file archive\_YYMMDD.tar that contains all the files  from the $HOME/Documents directory. Later, in 5 hours and 15 minutes, add the $HOME/tmp directory to the archive.

**108. tar -cvf archive\_$(date +%y%m%d).tar $HOME/Documents/\***

**echo "tar -rvf archive\_$(date +%y%m%d).tar $HOME/tmp/\*" | at now + 5 hours 15 minutes**

**sleep 5h 15m && tar --append -f archive\_$(date +\%Y\%m\%d).tar -C $HOME/tmp .**

109.Create the archive archive.tar, which will include all the files from your  home directory. The next day, at 21:00, delete the directory $HOME/tmp from the archive and compress the archive with the Lempel-Ziv algorithm.

**109.** **tar -cvf archive.tar $HOME/\***

**echo "tar --delete -f archive.tar $HOME/tmp && gzip archive.tar" | at 21:00 tomorrow**

110.Copy the /etc/passwd file every day at 20:30 into $HOME/tmp/passwd,  except weekends.Count the number of processes run by an XYZ user every 10  minutes every Saturday-night in January (00: 00-06: 00) and collect this data  in the /tmp/jan filename.

111.From December 25 to January 2, every 15 minutes, every day, count the total  number of processes running in the system and collect this data in the  /tmp/hollydays file.

112.The first day of each month at 15:48 make a backup of your home directory in  /backups and name this archive after the following sample:  YYMMDDHHMM.tar (where, YYMMDDHHMM is the current year, month,  day, hour and minute)

113.From December 25 to January 2, every 3 hours every day, calculate the size of  the XYZ User Directory and save this data in the /tmp/XZYsize file.

114.Create the file toto in $HOME/tmp that you can only read.

**114. touch $HOME/tmp/toto**

**chmod 400 $HOME/tmp/toto**

115.Create the file in toto in $HOME/tmp that you can read and modify but  cannot delete.

**115. mkdir $HOME/tmpdir**

**chmod 1777 $HOME/tmpdir**

**touch $HOME/tmpdir/toto**

**chmod 600 $HOME/tmpdir/toto**

116.Create the file toto in $HOME/tmp that you can read and delete but not  modify.

**116. touch $HOME/tmp/toto**

**chmod 400 $HOME/tmp/toto**

117.Create the file toto in $HOME/titi in such a way that it is not possible for  you to visualize the titi content, but you can visualize the toto content.

**117. mkdir $HOME/titi**

**chmod 700 $HOME/titi**

**touch $HOME/titi/toto**

**chmod 444 $HOME/titi/toto**

118.Allow other users to delete your own files from $HOME/tmp directory.

**118. chmod 777 $HOME/tmp**

119.Allow other users to delete their own files from $HOME/tmp directory.

**119. chmod 1755 $HOME/tmp**

120.Open in the text editor vim a file that was last modified in your home directory.

**120. vim $(ls -t ~ | head -1)**

**vim $(ls -t ~ | head -n 1)**

121.Count hidden files in your home directory.

**121. find $HOME/ -maxdepth 1 -type f -iname ".\*" | wc -l**

122.Count hidden directories in your home directory.

122. **find $HOME/ -maxdepth 1 -type d -iname ".?\*" | wc -l**

123.Count hidden files and directories in your home directory.

**123. ls -d $HOME/.[!.]?\* | wc -l**

**ls -l -d $HOME/.[!.]?\* | wc -l**

**find $HOME/ -maxdepth 1 -iname ".?\*" | wc -l**

**ls -l -d $HOME/.?\* | wc -l**

124.Display all hidden filenames in your home directory and append this list to  the f\_ca.txt file.

**124. find $HOME/ -maxdepth 1 -iname ".\*" > f\_ca.txt**

**ls -l -d $HOME/.\* > f\_ca.txt**

125.Open the toto file with vim text editor in read-only mode.

**125. vim -R toto**

126.Open the toto with vim text editor by moving the cursor to the tenth row.

**126. vim +10 toto**

127.List .conf files from the /etc directory and copy these files to the USB key  (/media/usb).

**127. find /etc -maxdepth 1 -iname "\*.conf" -exec cp {} /media/usb \;**

128.Display your home directory size (in bytes).

**128. du -sb ~**

**du -sb $HOME**

129.Display the disk free information (in bytes).

**129. df --block-size=1**

130.Copy all your script files ($HOME/bin/\*) to the USB key (/media/usb)  preserving the following attributes: mode, ownership, timestamps.

**130. cp -rp $HOME/bin/\*.sh /media/usb/**

**cp -rp $HOME/bin/\* /media/usb/**

Exercises for using the grep command:

131.Display all lines of the toto file containing the expression Linux.   
**131. grep "Linux" toto**

132.Display all lines of the toto and titi files containing the expression  Linux.  
**132. grep "Linux" toto titi**

133.Display all lines of the toto file containing the expression Linux with  uppercase and/or lowercase characters.   
**133. grep "linux" -i toto**

134.Display all lines of the toto file containing the expression Linux or linux.   
**134. grep 'Linux' toto | grep 'linux' toto**

135.Display all lines of the toto file containing the expression Linux and  linux.   
**135. grep -E 'Linux|linux' toto**

136.How many lines of the toto file contain the expression linux in uppercase,  lowercase or mixt characters?

136. **grep "linux" -i toto | wc -l**

137.Display lines of the toto file containing Linux or linux and display 2 lines  before them as well.

**137. grep -B 2 -E 'Linux|linux' toto**

138.Display lines of the toto file containing Linux or linux and display 2 lines  after them as well.

138. **grep -A 2 -E 'Linux|linux' toto**

139.Display the numbered lines of the toto file containing the expression Linux and display N lines before and after as well where N is a value of the local  variable n.   
139. **grep -n -B "$n" -A "$n" 'Linux' toto**

140.Display the numbered lines of the toto file containing the expression Linux or linux and display N lines before and M lines after as well where N and M are the values of the local variables n and m.

140. **grep -n -B "$n" -A "$m" 'Linux' toto**

141.Display list of files of your home directory containing the expression Linux or linux.  
141. **grep -E 'Linux|linux' $HOME/\***

142.Display list of files of your home directory including subdirectories containing  the expression Linux or linux.

142. **grep -Er 'Linux|linux' $HOME/\***

143.Display all lines of the toto file containing the word Linux or linux.

**143. grep -e 'Linux' -e 'linux' toto**

**grep -E 'Linux|linux' toto**

144.Display all lines of the toto file containing the expression: «.\*».

**144. grep -E "\.\\*" toto**

145.Display all lines of the toto file containing the expression: «.\*» or  «.|».

**145. grep -E '(\.\\*|\.\|)' toto**

145.5. Display all rows of the toto file beginning with the expression Kiel.

**145.5 grep "^Kiel" toto**

146.Display all rows of the toto file ending with the expression Kiel.

146. **grep "Kiel$" toto**

147.Display all rows of the toto file beginning with the word Kiel.

**147. grep -w "^Kiel" toto**

148.Display the lines of the toto file containing only numbers.

**148. grep -E '^[0-9]+$' toto  
 grep -E ‘[[:digits:]]’ toto**

**^: Asserts the start of the line.**

**[0-9]: Matches any digit (0 to 9).**

**+: Specifies that there must be one or more occurrences of the preceding digit pattern**

149.Display the lines of a toto file containing only N characters.

**149. grep -E "^.{$N}$" toto**

**-{$N} - რაოდენობა. N ცვლადია.**

150.Display the lines of the toto file lines containing the following IP addresses:  192.168.2.105, 217.147.231.59.

**150. grep -E -w "192.168.2.105|217.147.231.59" toto**

**-w ან ერთი მისამართი ან მეორე სრულად.**

151.Display all files of the toto file containing the mobile phone numbers in Germany (+49xxxxxxxx).

**151. grep -E '\+49[0-9]{9}' toto**

**{9} - ცხრა ცალი**

**[0-9] - ყველა ციფრი**

**\+49 - როგორც არის ისე აღიქვას**

152.Display lines of the .bashrc and the .profile files containing comments  (skipping leading spaces)  
**152. grep -E '^\s\*#' .bashrc -E .profile**

153.Display the list of all usernames whose names end with one or more number.

153. **cat /etc/passwd | cut -d: -f1 | grep -E '.\*[0-9]+$'**

154.Display non-empty files of the toto file.   
**154. grep -v '^$' toto**

**-v - ვაკეთებთ მის შემდეგ ჩაწერილი არგუმენტის ინვერსიას  
 ^$ - ნიშნავს ცარიელ ხაზებს**

155.Display non-empty files of the toto file including lines with tabulations or  spaces.  
**155. grep -v '^[[:space:]]\*$' toto**

156.How many pipe type of file do you have in your home directory?

**156. find ~ -type p | wc -l**

157.How many regular type of files and directories do you have in your home directory?

**157. find ~ -type f | wc -l**

**find ~ -type p | grep -c '^'**

158.How many hidden files do you have in your home directory?

**158. find ~ -type f -name “.\*” | wc -l**

**find ~ -type f | grep -c '^'**

159.How many hidden directories do you have in your home directory?

**159. find ~ -type d -name “.\*” | wc -l**

**find ~ -type d -name ".\*" | grep -c '^'**

160.How many hidden files and directories do you have in your home directory?

**160. find ~ -type f -name ".\*" | wc -l && find ~ -type d -name ".\*" | wc -l**

**(find ~ -type f -name ".\*" && find ~ -type d -name ".\*") | grep -c '^'**

161.display all hidden filenames from your home directory and add this list to the  hidden\_file.txt file.

**161. ls -a ~ | grep '^\.' | tee hidden\_files.txt**

162.Display the lines of the toto file lines containing the occurrence of “FH”  repeated at least three time.

**162. grep 'FH.\*FH.\*FH' toto (არ არის სავალდებულო რომ ერთმანეთის მიყოლებით იყოს)**

**grep -E '(FH){3}' toto (ერთმანეთის მიყოლებით FHFHFH)**

163.Display the lines containing the words "le" or "la" (with uppercase or lowercase characters) and highlight them with color.

**163. grep -i --color=always -E 'le|la' toto ( -I case insensitive)**

164.Display the lines containing only the real numbers.

**164. grep -E '^[+-]?[0-9]\*\.?[0-9]+$' filename**

**Explanation of the regular expression:**

**^: Anchors the pattern to the beginning of the line.**

**[+-]?: Allows an optional plus or minus sign.**

**[0-9]\*: Matches zero or more digits.**

**\.?: Allows an optional decimal point.**

**[0-9]+: Matches one or more digits after the decimal point.**

**$: Anchors the pattern to the end of the line.**

165.Display all lines of the file toto containing the real IP addresses.

**165. grep -Eo '([0-9]{1,3}\.){3}[0-9]{1,3}' toto**

Exercises for using the sed command:

166.Substitute all the occurrences of /bin/sh with /bin/bash in the  /etc/passwd file.

**166. cat /etc/passwd | sed -I ‘.s./bin/sh.bin/bash.’**

167.Display 5 to 9 rows of the /etc/passwd file.

**167. sudo sed -n '5,9p' /etc/passwd**

168.Delete rows 5 through 9 of the /etc/passwd file.

**168. sed -i '5,9d' /etc/passwd**

169.Delete the first 5 rows of the /etc/passwd file.

**169. sed -i '1,5d' /etc/passwd**

170.Display the /etc/passwd file content replacing the second «:» with «!».

**170. sed 's/:/!/'2 /etc/passwd**

171.Display the /etc/passwd file content replacing all occurrence of «:» with  «!».

**171. sed 's/:/!/g' /etc/passwd**

172.Display only the first column of the /etc/passwd file.

**172. sed 's/:.\*//' /etc/passwd**

**sed 's/\([^:]\*\).\*/\1/' /etc/passwd**

173.Display the content of the /etc/passwd file without the username field.

**173. sed 's/^[^:]\*://' /etc/passwd**

174.Display the content of the /etc/passwd file without the GID field.   
**174. sed 's/^\([^:]\*:[^:]\*:[^:]\*\):[^:]\*:\(.\*\)/\1:\2/' /etc/passwd**

175.Add the word "START:" to the beginning of each line of the file.

**175. sed 's/^/START:/' /etc/passwd**

176.Display the username and their UID fields from the /etc/passwd file.   
**176. sed -n 's/^\([^:]\*\):[^:]\*:\([^:]\*\).\*$/\1:\2/p' /etc/passwd**

177.Display those usernames whose interpreter is /bin/bash

**177. sed -n '/\/bin\/bash$/s/:.\*//p' /etc/passwd**

183.Delete all lines containing numbers in the toto file.

**183. sed ‘/[0-9]/d’ toto**

184.Delete all numbers in the toto file.

**184. sed ‘s/[0-9]//g’ toto**

185.Replace the word Windows with Linux in the toto file in rows 3-7.

**185. sed '3,7s/Windows/Linux/g' toto**

186.Replace the word Windows with Linux in the toto file on all lines except  3-7.   
 **186. sed '1,2s/Windows/Linux/g; 8,$s/Windows/Linux/g' toto**

187.Delete the first line of the toto file.  
**187. sed -i ‘1d’ toto**

188.Delete the third line of the toto file.   
188. **sed -i ‘3d’ toto**

189.Delete the last line of the toto file.  
189. **sed -i ‘$d’ toto**

190.Delete the line 5-9 of the toto file.

**190. sed -i '5,9d' toto**

191.Delete all lines except the fifth of the toto file.   
**191. sed -i ‘1,4d;6,$d’ toto**

192.Delete all line except 5 to 9 of the toto file.

**192. sed -i ‘1,4d;10,$d’ toto**

193.Delete the first and last lines of the toto file.

**193. sed -i '1d;$d' toto**

194.Delete the first N character on every line of the toto file.

**194. sed -i 's/^N//' toto**   
195.Delete the last N character on every line of the toto file.

**195. sed 's/N$//' emzo**  
196.Delete the Nth character on all lines of the toto file.

**196. N=5**

**sed -i "s/^\(.\{$((N-1))\}\).\{1\}/\1/" toto  
 აქ ვიყენებთ “ “ ბრჭყალებს რადგან გვინდა ჩავანაცვლოთ N ცვლადი რიცხვით. ამ ცვლადში შენახულია რიცხვი shell ში.**

197.Delete the first and last characters in every line of the toto file.

**197 sed -i 's/^.//';s/.$//' toto**

198.Count .conf files in the /etc directory.

199.Count all blank lines in the toto file.

**199. sed -n '/^$/p' toto | wc -l**

200.Delete all lines in the toto file that are empty or contain spaces.   
**200. sed -i '/^ \*$/d' toto**

201.Delete lines of the toto file containing only the uppercase characters.

**201. sed -i '/^[A-Z]\*$/d' toto**

202.Delete lines of the toto file not containing the uppercase characters

**202. sed -i '/[A-Z]/!d' toto**

203.Delete lines containing the expression Linux in the toto file.

**203. sed -i '/Linux/d' toto**

204.Delete lines not containing the expression Linux in the toto file.

**204. sed -i '/Linux/!d' toto**

205.Delete lines not containing the expression Linux or linux in the toto file.

**205. sed -i '/Linux\|linux/!d' toto**

206. Delete lines not containing the expression Linux, linux, Unix or unix in the toto file.

206. sed -i '/Linux\|linux\|Unix\|unix/!d' toto

207.Delete lines of the file toto from first to that line containing the expression  Windows.

**207. sed -i '1,/Windows/d' toto**

208.Delete lines from the toto file starting with the line containing the  expression Windows to the end.

**208. sed -i '/Windows/,$d' toto**

209.Delete the last line of the toto file if it contains the expression Windows or  windows.

**209. sed -i '${/Windows\|windows/d;}' toto**

210.Delete all even lines in the toto file.

**210. sed -i 'n;d' toto**

211.Delete all odd lines in the toto file.

**211. sed -i '1~2d' toto**

212.Count lines of the toto file.

**212. wc -l < toto**

213.Convert all Latin characters into uppercase in the toto file.

**213. tr '[:lower:]' '[:upper:]' < toto > temp && mv temp toto**

214.Convert uppercase characters into lowercase and vice versa in the toto file.

**214. tr '[:lower:][:upper:]' '[:upper:][:lower:]' < toto > temp && mv temp toto**

215.Insert a blank character after each character in the toto file.

**215. sed -i 's/./& /g' toto**

216.Insert two blank line next to each line in the toto file.

**216. sed -i 'G;G' toto**

217.Insert one blank character before each line in the toto file.

**217. sed -i 's/^/ /' toto**

218.Insert the contents of the toto file at the bottom of the file titi.

**218. cat toto >> titi**

219.Enclose all uppercase characters in parentheses in the toto file.

**219. sed -i 's/\([A-Z]\)/(\1)/g' toto**

220.Enclose all lines in the double quotes in the toto file.

**220. sed -i 's/.\*/"&"/' toto**

221.Replace each occurrence of space at the beginning of each line with a "+".   
**221. sed -i 's/^ /+/' filename**

Shell scripts

222.Write a shell script that takes a sequence of files as arguments and moves  them to the $HOME/files/ directory.

222. #!/bin/bash

# Check if the target directory exists, if not, create it

if [ ! -d "$HOME/files" ]; then

  mkdir -p "$HOME/files"

fi

# Loop through all arguments

for file in "$@"; do

  # Check if the file exists

  if [ -f "$file" ]; then

    # Move the file to the target directory

    mv "$file" "$HOME/files/"

  else

    echo "File $file does not exist."

  fi

done

223.Write a shell script that copies a file given as the first argument to the file  given as the second argument.

223. #!/bin/bash

# Check if the correct number of arguments are provided

if [ $# -ne 2 ]; then

  echo "Usage: $0 source\_file target\_file"

  exit 1

fi

# Check if the source file exists

if [ ! -f "$1" ]; then

  echo "Source file $1 does not exist."

  exit 1

fi

# Copy the source file to the target file

cp "$1" "$2"

224.Write a shell script that generates a menu with 3 arguments. if the first  argument is selected - the script should show the disk free space size in MB. If  second - the script should display $USER home directory size in KB. If third – exit the script.

224. #!/bin/bash

while true; do

    echo "1. Show disk free space in MB"

    echo "2. Show $USER home directory size in KB"

    echo "3. Exit"

    echo -n "Please enter your choice: "

    read choice

    case $choice in

        1) df -m ;;

        2) du -sk /home/$USER ;;

        3) exit 0 ;;

        \*) echo "Invalid choice, please select 1, 2 or 3." ;;

    esac

done

225.Write a script that generates the following menu:

A) Disk free space (only for physical file systems, not virtual file systems) B) $HOME Directory Size (This is the size of HOME directories) C) Number of .conf files in the /etc

D) The largest file size in $HOME (only the regular file type)

If selecting an option do the appropriate action.

225. #!/bin/bash

echo "1. Show disk free space for physical file systems"

echo "2. Show $HOME directory size"

echo "3. Show number of .conf files in the /etc directory"

echo "4. Show The largest file size in $HOME directory"

echo -n "Please enter your choice: "

read choice

case $choice in

    1) df -x tmpfs -x devtmpfs -h ;;

    2) du -sh $HOME;;

    3) find /etc -name "\*.conf" | wc -l ;;

    4) find $HOME -type f -exec ls -lh {} \; | sort -rh -k5,5 | head -n1 ;;

esac

226.Show How many color is supported in your terminal and display all color  code combinations with the appropriate colors (font color, background color,  format).

226.

#!/bin/bash

# Check how many colors the terminal supports

tput colors

# Display all color code combinations

for attr in 0 1 4 5 7 ; do

    echo "--------------------------------------------------------------------------------"

    printf "ESC[%s: Foreground; Background\n" "$attr"

    for fore in {30..37} ; do

        for back in {40..47} ; do

            # The -n option with echo command omits trailing newline

            echo -en "\033[$attr;${fore};${back}m ^[${attr};${fore};${back}m \033[0m"

        done

        echo

    done

    echo

done

227.Write a script that calculates the values of the functions f(x)=x2+15x-2 for  all x=[-10,10] and display the sum of their values.

227. #!/bin/bash

# Initialize sum

sum=0

# Loop through the range [-10, 10]

for ((x=-10; x<=10; x++)); do

    # Calculate the function value for the current x

    fx=$((x\*x + 15\*x - 2))

    # Display the result for the current x

    echo "f($x) = $fx"

    # Add the result to the sum

    sum=$((sum + fx))

done

# Display the sum of all function values

echo "Sum of all function values: $sum"

228.Write a script that shows the sum of all odd numbers from 1 to 50,000.

228. #!/bin/bash

# Initialize the sum

sum=0

# Loop through the range [1, 50000]

for ((x=1; x<=50000; x++))

do

    # Check if x is odd

    if (( x % 2 == 1 ))

    then

        # Add x to the sum

        ((sum += x))

    fi

done

# Display the sum

echo "The sum is equal to: $sum"

229.Write a script that shows the sum of all third numbers from 1 to 2023.

229. #!/bin/bash

# Initialize the sum

sum=0

# Loop through every third number in the range [1, 2023]

for ((x=1; x<=2023; x+=3))

do

    # Add x to the sum

    ((sum += x))

done

# Display the sum

echo "The sum is equal to: $sum"

230.Write a script that takes a sequence of numbers as an argument and displays  the sum.

230. #!/bin/bash

# Check if at least two arguments are provided

if (( $# < 2 )); then

    echo "Enter at least two arguments"

    exit 1

fi

# Initialize the sum

sum=0

# Loop through all arguments

for num in "$@"

do

    # Add the current number to the sum

    ((sum += num))

done

# Display the sum

echo "The sum is: $sum"

231.Write a script that takes a sequence of numbers as an argument and selects the  largest of them.

231. #!/bash/bin

# Check if at least one argument is provided

if (( $# < 1 ))

then

    echo "Enter at least one argument"

    exit 1

fi

# Initialize the largest number as the first argument

largest=$1

# Loop through all arguments

for num in "$@"

do

    # If the current number is greater than the largest found so far

    if (( num > largest ))

    then

        # Update the largest number

        largest=$num

    fi

done

# Display the largest number

echo "The largest number is: $largest"

232.Write a script that takes a number as an argument and display its factorial.

232. #!/bin/bash

# Check if an argument is provided

if (( $# != 1 )); then

    echo "Enter a non-negative integer"

    exit 1

fi

# Check if the argument is a non-negative integer

if ! [[ $1 =~ ^[0-9]+$ ]]; then

    echo "Please enter a non-negative integer."

    exit 1

fi

# Calculate and display the factorial

num=$1

factorial=1

for ((i=2; i<=num; i++)); do ((factorial \*= i)); done

233.Write a script that takes a number as an argument and displays the same  number of Fibonacci numbers. Make sure the argument is unique and integer.

233. #!/bin/bash

# Check if exactly one argument is provided and if it's a non-negative integer

if (( $# != 1 )) || ! [[ $1 =~ ^[0-9]+$ ]]; then

    echo "Enter a non-negative integer”

    exit 1

fi

# Get the number of Fibonacci numbers to generate

num=$1

# Initialize the first two Fibonacci numbers

a=0

b=1

# Display the Fibonacci numbers

for ((i=0; i<num; i++)); do

    echo -n "$a "

    fn=$((a + b))

    a=$b

    b=$fn

done

echo

234.Delete all your files from /tmp directory.

**234. rm -rf /tmp/\***

235.Delete all backup files in $HOME.

**235. find $HOME -name "\*~" -type f -delete**

236.Compare the number of processes running by root and $USER.

**236. echo "Root processes: $(ps -U root -u root u | wc -l)" && echo "User processes: $(ps -U $USER -u $USER u | wc -l)"**

237.Compare two files passed as arguments to the script by their last modification  time.

237. #!/bin/bash

# Check if exactly two arguments are provided

if (( $# != 2 )); then

    echo "Usage: $0 file1 file2"

    exit 1

fi

# Get the last modification times of the files

time1=$(stat -c %Y "$1")

time2=$(stat -c %Y "$2")

# Compare the modification times and display the result

if (( time1 > time2 )); then

    echo "$1 is newer than $2"

elif (( time1 < time2 )); then

    echo "$2 is newer than $1"

else

    echo "$1 and $2 have the same modification time"

fi

238.Compare two files passed as arguments to the script by their last access time.

238. #!/bin/bash

# Check if exactly two arguments are provided

if (( $# != 2 )); then

    echo "Enter two filenames as arguments for comparison"

    exit 1

fi

# Get the last modification times of the files

time1=$(stat -c %Y "$1")

time2=$(stat -c %Y "$2")

# Display the modification times in a human-readable format

echo "Modification time of $1: $(date -d @$time1)"

echo "Modification time of $2: $(date -d @$time2)"

# Compare the modification times and display the result

if (( time1 > time2 )); then

    echo "$1 is newer than $2"

elif (( time1 < time2 )); then

    echo "$2 is newer than $1"

else

    echo "$1 and $2 have the same modification time"

fi

239.Check if two files passed as arguments to the script share the same inode.

239. #!/bin/bash

# Check if exactly two arguments are provided

if (( $# != 2 )); then

    echo "Enter exactly two filenames as arguments"

    exit 1

fi

# Get the inodes of the files

inode1=$(stat -c %i "$1")

inode2=$(stat -c %i "$2")

# Compare the inodes and display the result

if (( inode1 == inode2 )); then

    echo "$1 and $2 share the same inode"

else

    echo "$1 and $2 do not share the same inode"

fi

240.Check if you are the owner of the file passed as an argument to the script. If  yes, check if you are belonging to its group.

240. #!/bin/bash

# Check if exactly one argument is provided

if (( $# != 1 )); then

    echo "Usage: $0 file"

    exit 1

fi

# Get the owner and group of the file

owner=$(stat -c %U "$1")

group=$(stat -c %G "$1")

# Check if you are the owner of the file

if [[ $owner == $USER ]]; then

    echo "You are the owner of the file"

    # Check if you belong to the group of the file

    if id -nG "$USER" | grep -qw "$group"; then

        echo "You belong to the group of the file"

    else

        echo "You do not belong to the group of the file"

    fi

else

    echo "You are not the owner of the file"

fi

241.Suppose the script takes a number as an argument - N. Print n, n-1, n-2… 2, 1,  BOOM !!!

241. #!/bin/bash

# Check if exactly one argument is provided and if it's a non-negative integer

if (( $# != 1 )) || ! [[ $1 =~ ^[0-9]+$ ]]; then

    echo "Enter a non-negative integer number”

    exit 1

fi

# Get the number to count down from

num=$1

# Count down from the number to 1

for ((i=num; i>=1; i--)); do

    echo $i

    sleep 1

done

# Print "BOOM !!!"

echo "BOOM !!!"

242.Show the day of the week that was in 2000-2023 on the current date.

242. #!/bin/bash

# Get the current month and day

month\_day=$(date +%m-%d)

# Loop over the years from 2000 to 2023

for year in {2000..2023}; do

    # Get the day of the week for the current date in the current year

    day\_of\_week=$(date -d "$year-$month\_day" +%A)

    # Display the year, date, and day of the week

    echo "On $year-$month\_day, the day of the week was $day\_of\_week"

done

243.Write a script that lists the names of all your files in $HOME/files and the  names of all your directories in $HOME/dirs. (Use the recursively)

243.  #!/bin/bash

# List the names of all files in $HOME/files

echo "Files in $HOME/files:"

find $HOME -type f -print

# List the names of all directories in $HOME/dirs

echo "Directories in $HOME/dirs:"

find $HOME -type d -print

244.Find the last modified file in your personal directory.

**244. find ~ -type f -printf '%T+ %p\n' | sort -r | head -n 1**

245.Find the last read file in your personal directory.

**245. find ~ -type f -printf '%A+ %p\n' | sort -r | head -n 1**

246.Write a script that copies all shell script names and contents to  $HOME/scripts, as follows:

Script Name 1:

File Contents 1

…

File Name N:

File Contents N

245. #!/bin/bash

# Create the scripts directory if it doesn't exist

mkdir -p $HOME/scripts

# Initialize the output file

output=$HOME/scripts/all\_scripts.txt

echo "" > $output

# Find all shell scripts in the current directory

for file in $(find . -name "\*.sh"); do

    # Append the script name to the output file

    echo "Script Name: $file" >> $output

    # Append the script contents to the output file

    echo "File Contents:" >> $output

    cat $file >> $output

    echo "" >> $output

done

247.Write a script that guesses which character was typed on the keyboard: a  number, a lowercase character, an uppercase character or other character.

247. #!/bin/bash

# Prompt the user to enter a character

read -p "Enter a character: " -n 1 char

echo

# Check if the character is a number

if [[ $char =~ [0-9] ]]; then

    echo "You entered a number."

# Check if the character is a lowercase letter

elif [[ $char =~ [a-z] ]]; then

    echo "You entered a lowercase letter."

# Check if the character is an uppercase letter

elif [[ $char =~ [A-Z] ]]; then

    echo "You entered an uppercase letter."

# If the character is not a number or a letter, it's an other character

else

    echo "You entered an other character."

fi

248.Write a script that saves full paths of .png files in the  $HOME/files\_png.txt file, .jpg files in the $HOME/files\_jpg.txt file and .bmp files in the $HOME/files\_bmp.txt file. After the script is  done, display how long it took.

248. #!/bin/bash

# Start the timer

start=$(date +%s%3N)

# Find images and write their paths in the respective files

find . -type f -name "\*.png" -exec realpath {} \; > $HOME/files\_png.txt

find . -type f -name "\*.jpg" -exec realpath {} \; > $HOME/files\_jpg.txt

find . -type f -name "\*.bmp" -exec realpath {} \; > $HOME/files\_bmp.txt

# Stop the timer

end=$(date +%s%3N)

# Calculate the duration in milliseconds

duration=$((end - start))

# Display how long the script took

echo "The script took $duration milliseconds."

249.Show the following extension file sizes of your home directory: tar.gz,  conf, html, sh.

249. #!/bin/bash

# Array of file extensions

extensions=("tar.gz" "conf" "html" "sh")

# Loop over the file extensions

for ext in "${extensions[@]}"; do

    # Get the sizes of the files with the current extension and sum them up

    sum=$(find $HOME -type f -name "\*.${ext}" -exec du -sk {} \; | cut -f1 | paste -sd+ - | bc)

    # Display the total size

    echo "The total size of files ending with .${ext} format is: ${sum}K"

done

250.Write a script that converts all regular type filenames to uppercase in  $HOME/tmp.

250. #!/bin/bash

# Start the timer

SECONDS=0

# Set the target directory

dir=$HOME/tmp

# Change to the target directory

cd $dir

# Loop over all regular files in the current directory

for file in \*; do

    # Check if it's a regular file (not a directory or a link)

    if [ -f "$file" ]; then

        # Convert the file name to uppercase and rename the file

        mv "$file" "$(echo $file | tr 'a-z' 'A-Z')"

    fi

done

# Display the final message and the elapsed time

echo "All of the filenames in $dir have been converted into uppercase. The operation took $SECONDS seconds."

251.Write a script that converts the contents of all regular files to uppercase I  $HOME/tmp.

251. #!/bin/bash

# Set the target directory

dir=$HOME/tmp

# Change to the target directory

cd $dir

# Loop over all regular files in the current directory

for file in \*; do

    # Check if it's a regular file (not a directory or a link)

    if [ -f "$file" ]; then

        # Convert the contents of the file to uppercase and overwrite the file

        tr 'a-z' 'A-Z' < "$file" > temp && mv temp "$file"

    fi

done

echo "All of the contents of the files in $dir have been converted into uppercase."

252.Check if the contents of two file passed as an argument to the script are  identical.

252. #!/bin/bash

# Check if two arguments are provided

if [ $# -ne 2 ]; then

    echo "Usage: $0 file1 file2"

    exit 1

fi

# Check if the files exist

if [ ! -f $1 ] || [ ! -f $2 ]; then

    echo "Both arguments must be files"

    exit 1

fi

# Compare the files

cmp -s "$1" "$2"

exitcode=$?

# Check the exit code

if [ $exitcode -eq 0 ]; then

    echo "The files are identical"

elif [ $exitcode -eq 1 ]; then

    echo "The files are different"

else

    echo "An error occurred while comparing the files"

fi

253.Check if the filename given as an argument to the script is a script.

253. #!/bin/bash

# Check if an argument is provided

if [ $# -ne 1 ]; then

    echo "Usage: $0 filename"

    exit 1

fi

# Check if the file exists

if [ ! -f $1 ]; then

    echo "The argument must be an existing file"

    exit 1

fi

# Check the file extension

ext="${1##\*.}"

script\_extensions=("sh" "py" "pl" "rb" "php" "js" "ps1" "tcl" "lua" "groovy" "R")

if [[ " ${script\_extensions[@]} " =~ " ${ext} " ]]; then

    echo "The file is a script (based on the file extension)"

    exit 0

fi

# Check the shebang or the interpreter line

shebang=$(head -n 1 $1)

if [[ $shebang == \#\!\* ]]; then

    echo "The file is a script (based on the shebang line)"

    exit 0

fi

echo "The file is most likely not a script"

254.Display the names of those command whose PID are given as arguments to  the script.

254. #!/bin/bash

# Check if at least one argument is provided

if [ $# -eq 0 ]; then

    echo "Usage: $0 pid1 pid2 ..."

    exit 1

fi

# Loop over all arguments

for pid in "$@"; do

    # Check if the PID exists

    if ps -p $pid > /dev/null; then

        # Display the name of the command

        cmd=$(ps -p $pid -o comm=)

        echo "PID $pid: $cmd"

    else

        echo "PID $pid does not exist"

    fi

done

255.Write a script that prints only the PID of the command given as an argument.

255. #!/bin/bash

# Check if an argument is provided

if [ $# -ne 1 ]; then

    echo "Usage: $0 command"

    exit 1

fi

# Get the PID of the command

pid=$(pgrep -f "$1")

# Check if the command is running

if [ -z "$pid" ]; then

    echo "The command is not running"

else

    echo "PID of the command: $pid"

fi

256.Write a script that inverts a string passed as an argument. (one => eno)

256. #!/bin/bash

# Check if an argument is provided

if [ $# -ne 1 ]; then

    echo "Enter a string”

    exit 1

fi

# Invert the string

inverted=$(echo $1 | rev)

# Print the inverted string

echo $inverted

257.Write a script that counts the number of vowels in a string passed as an  argument.

257. #!/bin/bash

# Check if an argument is provided

if [ $# -ne 1 ]; then

    echo "Enter a string"

    exit 1

fi

# Count the number of vowels

vowels=$(echo $1 | grep -o -i "[aeiou]" | wc -l)

# Print the number of vowels

echo $vowels

258.Write a script that does the same thing as the command basename.

258. #!/bin/bash

# Check if an argument is provided

if [ $# -ne 1 ]; then

    echo "Enter a path to a file"

    exit 1

fi

# Get the basename

basename=$(echo $1 | awk -F/ '{print $NF}')

# Print the basename

echo $basename

259.Write a script that does the same thing as the command dirname.

259. #!/bin/bash

# Check if an argument is provided

if [ $# -ne 1 ]; then

    echo "Usage: $0 path"

    exit 1

fi

# Get the dirname

dirname=$(echo $1 | rev | cut -d/ -f2- | rev)

# Print the dirname

echo $dirname

260.Remove the last directory in the variable PATH.

260. export PATH=$(echo $PATH | rev | cut -d: -f2- | rev)

261.In your home directory for each file with the suffix .bak (backup file) ask the  user to rename the .bak file (To get rid of the suffix), weather the  corresponding file (without the .bak suffix) is missing, or the corresponding  file is older than the .bak file. If the answer starts at y or Y, rename the file.

261. #!/bin/bash

for file in $(pwd)/\*.bak; do

    # Check if the file without the .bak extension exists

    if [ ! -e "${file%.bak}" ] || [ "$file" -nt "${file%.bak}" ]; then

        echo "Do you want to rename $file to ${file%.bak}? (y/n)"

        read answer

        # Check if the answer starts with y or Y

        if [[ $answer = y\* ]] || [[ $answer = Y\* ]]; then

            mv "$file" "${file%.bak}"

            echo "$(basename $file) has been renamed to $(basename ${file%.bak})"

        elif [[ $answer = n\* ]] || [[ $answer = N\* ]]; then

            echo "Exiting the program"

            exit

        fi

    fi

done

262.Write a script that prints the M line starting from the Nth row of the filename  passed as the 3rd argument. N and M are the first and second arguments.

262. #!/bin/bash

N=$1

M=$2

FILE=$3

# Calculate the end line

END=$((N+M-1))

# Use sed to print the lines from N to END

sed -n "${N},${END}p" $FILE

263.Write a script which prints the following motifs.

A) b) c) d) e) f) g) h)

264.Write a script that prints Pascal numbers.

264. #!/bin/bash

echo "Enter the number of rows: "

read rows

coef=1

for((i=0;i<rows;i++))

do

 for((space=1;space<=rows-i; space++))

 do

   echo -n " "

 done

 for((j=0;j<=i;j++))

 do

   if [ $j -eq 0 -o $i -eq 0 ]

   then

       coef=1

   else

       coef=$((coef\*(i-j+1)/j))

   fi

   echo -n "$coef   "

 done

 echo

done

265.Write a script that prints Pascal numbers so that you can replace pairs of  numbers with a space, odd with asterisk. What pattern will you get?

265. #!/bin/bash

# Function to calculate factorial

factorial() {

    result=1

    for (( i=2; i<=$1; i++ )); do

        result=$(( result \* i ))

    done

    echo $result

}

# Function to calculate binomial coefficient

binomial() {

    echo $(( $(factorial $1) / ( $(factorial $2) \* $(factorial $(( $1 - $2 ))) ) ))

}

# Function to print Pascal's triangle

pascal() {

    for (( i=0; i<$1; i++ )); do

        for (( j=0; j<=i; j++ )); do

            num=$(binomial $i $j)

            if (( num % 2 == 0 )); then

                printf " "

            else

                printf "\*"

            fi

        done

        echo

    done

}

# Print Pascal's triangle

pascal 10

266.There are N doors in a row (let’s say 100), all doors are initially closed. A  person walks through all doors multiple times and toggle (if open then close,  if close then open) them in the following way: In the first walk, the person  toggles every door. In the second walk, the person toggles every second door,  i.e., 2nd, 4th, 6th, 8th, … In the third walk, the person toggles every third  door, i.e. 3rd, 6th, 9th, … In the 100th walk, the person toggles the 100th  door. Which doors are open in the end? Display visually the doors state for each iteration.

266. #!/bin/bash

# Initialize the doors

doors=( $(printf "0 %.0s" {1..100}) )

# Walk through the doors

for ((i=1; i<=100; i++)); do

    for ((j=i-1; j<100; j+=i)); do

        # Toggle the door

        doors[j]=$((1-doors[j]))

    done

    # Print the state of the doors after each walk

    echo "After walk $i: ${doors[@]}"

done

# Print the numbers of the open doors at the end

echo "Open doors: $(for ((i=0; i<100; i++)); do ((doors[i])) && echo -n "$((i+1)) "; done)"

267.Write a shell script that generates 15-character random password containing numbers, uppercase and lowercase characters, punctuation marks and other  special characters.

267. #!/bin/bash

# Generate a random password

password=$(cat /dev/urandom | tr -dc '[:alnum:][:punct:]' | fold -w 15 | head -n 1)

# Print the password

echo "Random Password: $password"

1. რას ნიშნავს ფასილიტატორის როლი და რატომ არის ეფექტური ფასილიტაცის მნიშვნელოლვანი ეჯაილ გუნდის ცხოვრებაში

**1. Agile მეთოდოლოგიაში ფასილიტატორი, როგორც წესი, არის როლი, რომელსაც ასრულებს Scrum Master ან Agile Coach. ფასილიტატორის მთავარი პასუხისმგებლობაა უზრუნველყოს, რომ გუნდი მიჰყვება Agile-ს პრინციპებს, პრაქტიკებს და იყოს მათი გზამკვლევი დაბრკოლებებთან გამკლავების დროს, და დაეხმაროს გუნდს გამოწვევების გადალახვაში რომელიც შეიძლება წარმოიშვას განვითარების პროცესში.  
  
ეფექტური ფასილიტაცია მნიშვნელოვანია Agile გუნდისთვის, რადგან ის:**

**1. უზრუნველყოფს მკაფიო კომუნიკაციას და აგვარებს კონფლიქტებს  
2. ხელს უწყობს ფოკუსის შენარჩუნებას და მხარს უჭერს უწყვეტ გაუმჯობესებას**

1. გამოყავით ყველაზე ხშირად დაშვებული შეცდომები დეილი სქრამის ივენთზე და მათი რისკები

**2. 1) 15-წუთიან ვადის გადაჭარბება**

**2) სპრინტის მიზანზე ფოკუსირების ნაკლებობა**

**3) გუნდის წევრები ერთმანეთის ნაცვლად ესაუბრებიან Scrum Master-ს**

**4) გადაჭარბებული აქცენტი მიკრო მენეჯმენტზე**

**Daily Scrum, თუ სათანადოდ არ იმართება, შეიძლება გახდეს ხანგრძლივი, დამაბნეველი შეხვედრა, რომელიც არღვევს სამუშაო ნაკადს, შორდება სპრინტის მიზანს, აფერხებს გუნდის თვითორგანიზებას და პოტენციურად გადაიქცევა დემოტივირებად მიკრო-მენეჯმენტის სესიად, რაც უარყოფითად აისახება გუნდის პროდუქტიულობასა და კრეატიულობაზე.**

1. რას ნიშნავს რელაციური შეფასება სთორი ფოინთებში და რა უპირატესობები აქვს ამ პრატქიკას კომპლექსური პროდუქტების/პროექტების მართვის დროს?

**3. Agile მეთოდოლოგიაში, სთორი ფოინთი არის საზომი ერთეული, რომელიც გამოიყენება Agile პროექტის მენეჯმენტში, რათა შეფასდეს ძალისხმევის საჭიროება პროდუქტის ბექლოგის ელემენტის ან ნებისმიერი სხვა სამუშაოს შესასრულებლად. ისინი ინიშნება სამუშაოს სირთულის, მოცულობისა და წარუმატებლობის რისკის მიხედვით.**

**მისი უპირატესობებია: გაუმჯობესებული შეფასება, სიზუსტე, მოქნილობა, და პროგნოზირებადობა. ეს ყოველივე ხელს უწყობს ღირებულ დისკუსიას და აქტიურ მონაწილეობას გუნდის წევრებს შორის.**

1. აღწერეთ პროექტის რისკების მართვის პროცესი, რა საფეხურების გავლა არის აუციელებელი და რა კრიტექრიუმებია მნიშვნელოვანი გადაწყვეტილების მიღების პროცესში

**4. პროექტის რისკის მენეჯმენტი გულისხმობს პოტენციური საფრთხეების იდენტიფიცირებას, რომლებმაც შეიძლება გავლენა მოახდინონ პროექტის მიზნებზე და ნაბიჯების გადადგმას მათ შესამცირებლად. ძირითადი რისკები მოიცავს მომხმარებლის საჭიროებების არასწორ ინტერპრეტაციას და მომხმარებლის სპეციფიკური საჭიროებების ნაკლებობას. ამ რისკების სამართავად, ჩვენ უნდა გამოვიყენოთ შემდეგი კრიტერიუმები:**

**ეფექტური კომუნიკაცია და მომხმარებლის ჩართულობა განვითარებაში**

**მოქნილობა რესურსების შეფასებისა და მოთხოვნების რეგულირებისას**

**ეფექტური ინსტრუმენტებისა და ტექნოლოგიების გამოყენება**

1. რომელ ივენთებს იცნობთ სქრამში და რომელია მათ შორის ყველაზე მნიშვნელოვანი

**5. Scrum-ში მე ვიცნობ Sprint Planning, Daily Scrum, Sprint Review და Sprint Retrospective ღონისძიებებს. ყველა მათგანი მნიშვნელოვანია, რადგან თითოეული მათგანი ემსახურება უნიკალურ მიზნებს Scrum პროცესში. თუმცა ვიტყოდი რომ Retrospective არის ის კრიტიკული ნაწილი, რის მეშვეობითაც ყველაზე ეფექტურად ვპოულობთ გაუმჯობესების გზებს.**

1. რა არის სქრამ რეტროსპექტივის მიზანი და შედეგი?

**6. Scrum Retrospective-ის მიზანია წარსული სპრინტის შედეგის განხილვა და გაუმჯობესების შესაძლებლობების იდენტიფიცირება. შედეგი კი, ყალიბდება მომავალი სპრინტის გაუმჯობესების განხორციელების გეგმა.**

1. რა არის სქრამ პლენინგის მიზანი და შედეგი?

**7. Scrum Planning-ის მიზანია განსაზღვროს ბიჯები და სტრატეგია მომავალი სპრინტისთვის. შედეგი არის სპრინტის ბექლოგი, რომელიც არის დავალებების სია, რომელსაც გუნდი ასრულებს სპრინტში.**

1. რა არის სქრამ რევიუს მიზანი და შედეგი?

**8. Scrum Review-ის მიზანია სპრინტის შედეგის შემოწმება და მომავალი ადაპტაციების განსაზღვრა. შედეგი არის შესწორებული პროდუქტის ბექლოგი, რომელიც განსაზღვრავს სავარაუდო ჩამორჩენილ ელემენტებს შემდეგი სპრინტისთვის.**

1. ახსენით რას ნიშნავს პროექტის სტეიქჰოლდერი და რა პროცესია საჭირო სტექიჰოლდერების მართვისთვის?

**9. Stakeholder არის ის, ვინც დაინტერესებულია პროექტით ან გავლენას ახდენს მის შედეგებზე. დაინტერესებული მხარეების მართვა გულისხმობს მათ იდენტიფიცირებას, მათი საჭიროებებისა და მოლოდინების გააზრებას, მათთან რეგულარულ კომუნიკაციას და პროექტში მათი ჩართვას საჭიროებისამებრ. მიზანია უზრუნველყონ მათი მოთხოვნილებების დაკმაყოფილება და მათი გავლენის პოზიტიურად გამოყენება პროექტის წარმატებისთვის.**

1. მოიყვანეთ მაგალითი ისეთი პროდუქტისა სადაც ეჯაილ მიდგომა არ იმუშავებს და რატომ?

**10. Agile შეიძლება კარგად არ იმუშაოს ფართომასშტაბიანი ინფრასტრუქტურული პროექტისთვის, როგორიცაა კაშხლის ან გზატკეცილის მშენებლობა. ეს პროექტები მოითხოვს ფართო წინასწარ დაგეგმვას და ცვლილებები შუა გზაზე შეიძლება იყოს ძალიან ძვირი და სარისკო. ასეთ კონტექსტში უფრო ტრადიციული, Waterfall მეთოდი შეიძლება იყოს უფრო შესაფერისი.**

პრაქტიკული დავალებები, საიდანაც 4 შეგხვდებათ გამოცდაზე:

1. შექმენით 3 კითხვიანი კითხვარი, რომლითაც შეაფასებთ რამდენად ასრულებს სქრამ გუნდის პროდაქტ ოუნერი საკუთარ ანგარიშვალებულებას.

**1. 1. შეგიძლიათ აღწეროთ სიტუაცია, როდესაც რთული გადაწყვეტილების მიღება მოგიწიათ პროდუქტის ბექლოგის შესახებ?  
2. როგორ უმკლავდებით კონფლიქტებს სტეიქჰოლდერების მოლოდინებსა და დეველოპმენტის გუნდის შესაძლებლობებს შორის?  
3. შეგიძლიათ გაგვიზიაროთ შემთხვევა, როდესაც მოგიწიათ პასუხისმგებლობის აღება პროდუქტის გადაწყვეტილებაზე, რომელმაც არ გამოიღო მოსალოდნელი შედეგი?**

1. შექმენით 3 კითხვიანი კითხვარი, რომლითაც შეაფასებთ რამდენად ასრულებს სქრამ გუნდის სქრამ მასტერი საკუთარ ანგარიშვალებულებას.

**2. 1. შეგიძლიათ აღწეროთ სიტუაცია, როდესაც კონფლიქტის მოგვარება მოგიწიათ გუნდში და დარჩა თუ არა გუნდი სპრინტის მიზნებზე ორიენტირებული?  
2. როგორ მეთოდებით უჭერთ მხარს პროდუქტის მფლობელს და განვითარების გუნდს Scrum-ის პრაქტიკის გაგებაში და ამოქმედებაში?   
3. გვიამბეთ იმ დაბრკოლებების შესახებ, რომლებიც აფერხებდა გუნდის წინსვლას და რა ნაბიჯები გადადგით მათ მოსაგვარებლად?**

1. შექმენით პროექტის კრიტიკული ბილიკის დიაგრამა (Critical Path Diagram) თქვენს მიერ დამუშავებული პროდუქტის მაგალითზე

**3. [პრე-რეკვიზიტების მოგროვება (2 კვირა)] --> [აპლიკაციის დიზაინი (3 კვირა)] --> [აპლიკაციის დეველოპმენტი (5 კვირა)] --> [აპლიკაციის ტესტირება (3 კვირა)] --> [აპლიკაციის გაშვება ლაივ რეჟიმში (1 კვირა)]**

1. შექმენით OKR ერთი მიზნისთვის თქვენს მიერ დამუშავებული პროდუქტის მაგალითზე.

**4. ძირითადი შედეგი 1: ჩვენ უნდა შევიმუშავოთ აპლიკაციის პროტოტიპი 2 თვის განმავლობაში.**

**ძირითადი შედეგი 2: ჩვენ უნდა მივაღწიოთ 90% მოწონების რეიტინგს ბეტა ტესტის მომხმარებლებისგან.**

**ძირითადი შედეგი 3: ჩვენ უნდა გავუშვათ აპლიკაცია და გვყავდეს 1000 აქტიური მომხმარებელი პირველი 3 თვის განმავლობაში.**

1. შექმენით სტეიქჰოლდერების მატრიცა თქვენს მიერ დამუშავებული პროდუქტის მაგალითზე სადაც მინიმუმ 3 სტეიქჰოლდერი იქნება მაგალითად მოყვანილი

**5.**

| **Stakeholder** | **Interest** | **Influence** | **Impact** |
| --- | --- | --- | --- |
| **Government** | **High** | **Medium** | **High** |
| **Citizens** | **High** | **Low** | **High** |
| **Local Businesses** | **Medium** | **High** | **Medium** |

**ეს მატრიცა მიუთითებს იმაზე, რომ მთავრობას და მოქალაქეებს აქვთ ძლიერი ინტერესი და დიდი გავლენა პროექტზე, ხოლო ადგილობრივ ბიზნესს აქვს ზომიერი ინტერესი და საშუალო გავლენა.**

1. შექმენით რისკების მატრიცა თქვენს მიერ დამუშავებული პროდუქტის მაგალითზე, სადაც 3 რისკი იქნება მაგალითად მოყვანილი

**6.**

| **Risk** | **Impact** | **Likelihood** | **Mitigation** |
| --- | --- | --- | --- |
| **Technical Failure** | **High** | **Medium** | **Regular maintenance, backup systems** |
| **Legal Issues** | **High** | **Low** | **Consult legal experts** |
| **User Acceptance** | **Medium** | **High** | **Conduct usability tests** |

**ეს მატრიცა მიუთითებს, რომ technical faliure და user acceptance პრობლემის გაჩენის უფრო მაღალი შანსია, ასევე ორივეს მნიშვნელოვანი გავლენა აქვს აპლიკაციის წარმატებაზე. სამართლებრივი საკითხები ნაკლებად სავარაუდოა, მაგრამ მაინც დიდი გავლენა აქვს.**