# **Systems Programming (6CC514), 2024/25**

A logo on a black background

Description automatically generated

Blue text on a black background

Description automatically generated

**2024-25/100667771**

## **Files and directories**

1. Create the following structure of directories and files. Give Your name to the top directory. This directory will be your home directory.

A close-up of a diagram

Description automatically generated

Answer:

nickk@nickk-VirtualBox:~$ mkdir Nikolaos

nickk@nickk-VirtualBox:~$ cd Nikolaos

nickk@nickk-VirtualBox:~/Nikolaos$ touch vespa "vaz volvo" "ford daf" citroen

nickk@nickk-VirtualBox:~/Nikolaos$ mkdir Mashini

nickk@nickk-VirtualBox:~/Nikolaos$ cd Mashini

nickk@nickk-VirtualBox:~/Nikolaos/Mashini$ mkdir Razni

nickk@nickk-VirtualBox:~/Nikolaos/Mashini$ mkdir Francuz

nickk@nickk-VirtualBox:~/Nikolaos/Mashini$ cd ..

nickk@nickk-VirtualBox:~/Nikolaos$ cd ~

nickk@nickk-VirtualBox:~$ tree Nikolaos

//The script creates a directory structure with files and subdirectories to organize data systematically. Here's what it does:

* **mkdir**: Creates directories like Nikolaos, Mashini, Razni, and Francuz.
* **cd**: Navigates between directories to structure the hierarchy.
* **touch**: Creates files (vespa, vaz volvo, ford daf, citroen) in the Nikolaos directory.
* **tree**: Displays the resulting structure in a tree format.

1. Stay in Your home directory.

Answer:

nickk@nickk-VirtualBox:~$ cd Nikolaos

1. Move ford to directory Mashini.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ mv "ford daf" Mashini

nickk@nickk-VirtualBox:~/Nikolaos$ tree

**//mv "ford daf" Mashini**: Moves the file ford daf into the Mashini directory. Quotation marks are used to handle spaces in the file name.

1. Move all files, which names begin from “v” to directory Razni.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ mv v\* Mashini/Razni/

nickk@nickk-VirtualBox:~/Nikolaos$ tree

**//mv v\* Mashini/Razni/**: Moves all files in Nikolaos whose names start with the letter "v" (e.g., vespa, vaz volvo) into the Razni subdirectory inside Mashini.

1. Move vaz to directory Mashini.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ mv Mashini/Razni/"vaz volvo" Mashini/

nickk@nickk-VirtualBox:~/Nikolaos$ tree

1. Stay in directory Mashini.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ cd Mashini

nickk@nickk-VirtualBox:~/Nikolaos/Mshini$

1. Copy daf to this directory.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos/Mashini$ cp "ford daf" ..

nickk@nickk-VirtualBox:~/Nikolaos/Mashini$ tree

// **cp "ford daf" ..**: Copies the file ford daf from the current directory (Mashini) to its parent directory (Nikolaos). The .. specifies the parent directory.

1. Stay in directory Raznie.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos/Mashini$ cd Razni

nickk@nickk-VirtualBox:~/Nikolaos/Mashini/Razni$

1. Move vespa to directory Francuz and rename it into vespa2.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos/Mashini/Razni$ mv "vespa" ../Francuz/vespa2

nickk@nickk-VirtualBox:~/Nikolaos/Mashini/Razni$ cd ..

nickk@nickk-VirtualBox:~/Nikolaos/Mashini$ cd ..

nickk@nickk-VirtualBox:~/Nikolaos$ tree

1. Create three files with arbitrary names.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos/Mashini$ cd Razni

nickk@nickk-VirtualBox:~/Nikolaos/Mashini/Razni$ touch file1.txt file2.txt file3.txt

nickk@nickk-VirtualBox:~/Nikolaos/Mashini/Razni$ tree

1. Go to directory Mashini.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos/Mashini/Francuz$ cd ..

nickk@nickk-VirtualBox:~/Nikolaos/Mashini$

1. Copy two files from Raznie to Francuz using only one command.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos/Mashini$ cp Razni/file1.txt Razni/file2.txt Francuz/

nickk@nickk-VirtualBox:~/Nikolaos/Mashini tree

1. Stay in Your home directory.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos/Mashini$ cd ..

1. Show on the screen all levels under Your home directory.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ tree

1. **Protection of files**
2. Stay in Your home directory and execute the command ls -al > LIST which creates a file LIST showing initial protection of filers.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ ls -a > LIST

nickk@nickk-VirtualBox:~/Nikolaos$

1. Let files .profile and fruits and directory practice are situated in Your home directory. They will be used further.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ touch ~/fruits

nickk@nickk-VirtualBox:~/Nikolaos$ mkdir ~/practice

nickk@nickk-VirtualBox:~/Nikolaos$ ls -al ~

**// ls -al**: Lists all files (including hidden ones) in long format (detailed info).

**// ~**: Refers to the home directory.

1. Change protection of file .profile that no other category of users except you can read the file content. Check the result.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ ls -l ~/.profile

nickk@nickk-VirtualBox:~/Nikolaos$ chmod 600 ~/.profile

nickk@nickk-VirtualBox:~/Nikolaos$ ls -l ~/.profile

**// ls -l ~/.profile**: Displays the detailed information (permissions, owner, size, etc.) of the .profile file in your home directory.

**// chmod 600 ~/.profile**: Changes the file permissions of .profile so that only the owner has read and write permissions (no access for others).

**// ls -l ~/.profile**: Shows the updated details of the .profile file, confirming the new permissions.

1. Change protection of file fruits that only You or members of Your group can read file and write to it but all other users cannot do it.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ ls -l ~/fruits

nickk@nickk-VirtualBox:~/Nikolaos$ chmod 660 ~/fruits

nickk@nickk-VirtualBox:~/Nikolaos$ ls -l ~/fruits

 // **chmod 660**: Changes the file permissions for the file fruits.

* **6** (Owner): Read and write permissions.
* **6** (Group): Read and write permissions.
* **0** (Others): No permissions.

 // **~/fruits**: Specifies the file fruits in your home directory.

1. Check which group files of Your directory belongs to. Check which group you belong to. Change using a command the group of a file. Check in file /etc/group who belongs and who does not belong to Your group.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ ls -la ~

nickk@nickk-VirtualBox:~/Nikolaos$ groups

nickk@nickk-VirtualBox:~/Nikolaos$ cat /etc/group | grep nick

 // **ls -la ~**: Lists all files (including hidden ones) in the home directory, showing detailed information (permissions, owners, etc.).

 // **groups**: Displays the groups the current user belongs to.

 // **cat /etc/group | grep nick**: Searches and shows all the groups in the /etc/group file that contain the user "nick". This helps identify the groups to which the user "nick" belongs.

1. Cancel all access rights for all categories of users for directory practice. Execute command ls - al practice. Cancel Your own x-right. Execute command ls - al practice.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ chmod 000 ~/practice

nickk@nickk-VirtualBox:~/Nikolaos$ ls -al ~/practice

nickk@nickk-VirtualBox:~/Nikolaos$ chmod u-x ~/practice

nickk@nickk-VirtualBox:~/Nikolaos$ ls -al ~/practice

 // **chmod 000 ~/practice**: Removes all permissions (read, write, execute) for the file or directory practice. No user (owner, group, or others) can access it.

 // **ls -al ~/practice**: Shows the detailed information (permissions, owner, etc.) of practice after the permissions were changed to 000, which should reflect no access.

 // **chmod u-x ~/practice**: Removes the execute permission for the user (owner) of practice. This command might not change anything since practice already had no execute permission (due to the previous chmod 000).

 // **ls -al ~/practice**: Shows the updated details of practice after removing the execute permission for the user.

1. **Processes**
2. Determine who works in system and what processes are started by each user. For each process determine its state and priority. Find processes unattached to no one terminal.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ ps -eo user,pid,ppid,pri,state,tty,cmd

nickk@nickk-VirtualBox:~/Nikolaos$ ps -eo user,pid,ppid,pri,state,tty,cmd | grep “?”

// **user**: The user running the process.

// **pid**: The process ID.

//**ppid**: The parent process ID.

// **pri**: The priority of the process.

// **state**: The current state of the process (e.g., running, sleeping).

// **tty**: The terminal associated with the process.

// **cmd**: The command that started the process.

// **ps -eo user,pid,ppid,pri,state,tty,cmd | grep “?”**: Filters the output of the previous command to only show lines that contain ? (usually indicating processes that are not associated with any terminal).

1. Try to delete processes started by other users.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ kill 295

// **kill**: Sends a signal to a process. By default, the kill command sends the SIGTERM (signal 15), which requests a process to terminate gracefully.

// **295**: The process ID (PID) of the process you want to terminate.

3. Create a command dummy which incessantly prints into standard output stream its name. Start the command. Interrupt its running and delete the process.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ nano ~/dummy

nickk@nickk-VirtualBox:~/Nikolaos$ chmod +x ~/dummy

nickk@nickk-VirtualBox:~/Nikolaos$ ~/dummy

control + C (to stop)

the script in the background (Another terminal):

nickk@nickk-VirtualBox:~/Nikolaos$ ~/dummy &

In the main terminal: nickk@nickk-VirtualBox:~/Nikolaos$ ps -ef | grep dummy

nickk@nickk-VirtualBox:~/Nikolaos$ kill 3764

nickk@nickk-VirtualBox:~/Nikolaos$ ps -ef | grep dummy

// **nano ~/dummy**: Opens the dummy file in the nano editor where you can write your script (presumably a command or program that runs continuously, like an infinite loop).

// **chmod +x ~/dummy**: Makes the dummy script executable by adding execute permissions to the file.

// **~/dummy**: Runs the dummy script. If it's an infinite loop (e.g., while true; do echo "dummy"; done), it will keep running until manually stopped.

// **Control + C**: Interrupts the currently running process (in this case, it stops the dummy script).

// **~/dummy &**: Runs the dummy script in the background (denoted by the &), so the terminal is free for other commands while the script continues to run.

// **ps -ef | grep dummy**: Lists all running processes and filters for the ones related to dummy. This shows the process ID (PID) and other details of the running dummy script.

// **kill 3764**: Sends a termination signal to the process with PID 3764, which should be the background dummy script. This will stop the running script.

// **ps -ef | grep dummy**: Runs again to check if the dummy script is still running. It should no longer be listed since the process was killed.

1. Start process dummy in background mode. Redirect its output into a file. Find the corresponding process in the processes list. Determine its state. Delete it.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ ~dummy > dummy\_output.txt &

nickk@nickk-VirtualBox:~/Nikolaos$ ps -ef | grep dummy

nickk@nickk-VirtualBox:~/Nikolaos$ kill 3848

nickk@nickk-VirtualBox:~/Nikolaos$ ps -ef | grep dummy

// **~/dummy > dummy\_output.txt &**:

Runs the dummy script in the background (due to the & symbol).

The > operator redirects the output of the dummy script into a file named dummy\_output.txt. This means the script's output will be written to the file rather than being displayed in the terminal.

// **ps -ef | grep dummy**:

Lists all running processes and filters the output to show only those related to the dummy script.

If the dummy script is running, you should see the process listed along with its process ID (PID).

// **kill 3848**:

Sends a termination signal (SIGTERM) to the process with PID 3848, which is the running dummy script.

This should stop the script from running.

// **ps -ef | grep dummy**:

Checks again to see if the dummy script is still running.

After using the kill command, you should no longer see the process listed, as it was terminated.

1. Do the same operations decreasing the process priority using command nice. Change priority of running process using command renice.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ nice -n 10 ~/dummy > dummy\_output.txt &

nickk@nickk-VirtualBox:~/Nikolaos$ ps -o pid,nice,cmd -p 3880

nickk@nickk-VirtualBox:~/Nikolaos$ sudo renice -n 5 -p 3880

nickk@nickk-VirtualBox:~/Nikolaos$ ps -o pid,nice,cmd -p 3880

nickk@nickk-VirtualBox:~/Nikolaos$ kill 3880

nickk@nickk-VirtualBox:~/Nikolaos$ ps -ef | grep dummy

// **nice -n 10 ~/dummy > dummy\_output.txt &**:  
Runs the dummy script in the background with priority 10 and redirects output to dummy\_output.txt.

// **ps -o pid,nice,cmd -p 3880**:  
Displays the process information (PID, priority, and command) of process 3880.

// **sudo renice -n 5 -p 3880**:  
Changes the priority of the dummy process (PID 3880) to 5.

// **ps -o pid,nice,cmd -p 3880**:  
Shows the updated priority of the process after renice.

// **kill 3880**:  
Terminates the dummy process (PID 3880).

// **ps -ef | grep dummy**:  
Checks if the dummy process is still running. It should not be after termination.

4o mini

1. Start a few copies of the process dummy in background mode; use redirection of output into a dummy device. Find started processes in the processes list. Exit from the system. Enter the system. Check the processes list.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ ~/dummy > /dev/null &

nickk@nickk-VirtualBox:~/Nikolaos$ ~/dummy > /dev/null &

nickk@nickk-VirtualBox:~/Nikolaos$ ~/dummy > /dev/null &

nickk@nickk-VirtualBox:~/Nikolaos$ ps -ef | grep dummy

nickk@nickk-VirtualBox:~/Nikolaos$ exit

nickk@nickk-VirtualBox:~/Nikolaos$ ps -ef | grep dummy

// **~/dummy > /dev/null &** (repeated 3 times):  
Starts three instances of the dummy script in the background, discarding their output.

// **ps -ef | grep dummy**:  
Lists all running dummy processes.

// **exit**:  
Closes the terminal session, but background processes may continue running.

// **ps -ef | grep dummy** (after exit):  
Checks if dummy processes are still running after exiting the terminal.

1. Do the same operations starting processes in uninterrupted mode nohup. Compare process states after repeated entering into the system. Delete started processes.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ nohup ~/dummy > /dev/null 2>&1 &

nickk@nickk-VirtualBox:~/Nikolaos$ nohup ~/dummy > /dev/null 2>&1 &

nickk@nickk-VirtualBox:~/Nikolaos$ nohup ~/dummy > /dev/null 2>&1 &

nickk@nickk-VirtualBox:~/Nikolaos$ ps -ef | grep dummy

nickk@nickk-VirtualBox:~/Nikolaos$ ps -o pid,stat,cmd -p 2942

nickk@nickk-VirtualBox:~/Nikolaos$ kill 2942 3055 3073

nickk@nickk-VirtualBox:~/Nikolaos$ ps -ef | grep dummy

// **nohup ~/dummy > /dev/null 2>&1 &** (repeated 3 times):  
Starts three instances of the dummy script in the background, with output and errors redirected to /dev/null (discarded).

// **ps -ef | grep dummy**:  
Lists all running processes with "dummy" in their name.

// **ps -o pid,stat,cmd -p 2942**:  
Displays the status of the process with PID 2942 (process state, PID, command).

// **kill 2942 3055 3073**:  
Terminates the dummy processes with the specified PIDs.

// **ps -ef | grep dummy**:  
Verifies that the dummy processes have been terminated.

1. Enter commands of starting three dummy processes in various instances of time (for example, after 3, 5, and 7 minutes). Watch starting the processes then delete them. Do processes start after a user exits the system?

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ nohup ~/dummy > /dev/null 2>&1 &

nickk@nickk-VirtualBox:~/Nikolaos$ sleep 180 && nohup ~/dummy > /dev/null 2>&1 &

nickk@nickk-VirtualBox:~/Nikolaos$ sleep 300 && nohup ~/dummy > /dev/null 2>&1 &

nickk@nickk-VirtualBox:~/Nikolaos$ ps -ef | grep dummy

nickk@nickk-VirtualBox:~/Nikolaos$ kill 3127 3136

nickk@nickk-VirtualBox:~/Nikolaos$ ps -ef | grep dummy

// **nohup ~/dummy > /dev/null 2>&1 &**:  
Starts the dummy script in the background, with output and errors discarded.

// **sleep 180 && nohup ~/dummy > /dev/null 2>&1 &**:  
Waits for 180 seconds, then starts another dummy process in the background.

// **sleep 300 && nohup ~/dummy > /dev/null 2>&1 &**:  
Waits for 300 seconds, then starts yet another dummy process in the background.

// **ps -ef | grep dummy**:  
Lists all running dummy processes.

// **kill 3127 3136**:  
Terminates the dummy processes with PIDs 3127 and 3136.

// **ps -ef | grep dummy**:  
Verifies that the dummy processes with the specified PIDs are no longer running.

1. **Filters: sort**
2. Create a file named family that contains surnames, names and birth dates of a few persons. Each line has the following form: smith john 3/1/50 Separate name and birth date with a tabulation ().

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ nano family

inside the nano we write the names

nickk@nickk-VirtualBox:~/Nikolaos$ cat family

1. Use command sort to sort the file in alphabetic order.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ sort family

1. Use an option (key) of command sort to sort the file in reverse order.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ sort -r family

1. Create another file named work which contains surnames, names and birth dates of a few colleagues (group mates) having the same format as file family. Include Your personal data into the file.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ nano work

Inside the nano we write surnames, names and birth dates of a few colleagues

nickk@nickk-VirtualBox:~/Nikolaos$ cat work

1. Sort both files simultaneously by command sort. Pay your attention that your name is printed twice. Use an option (key) of command sort to print your name only once.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ sort family work

nickk@nickk-VirtualBox:~/Nikolaos$ sort -u family work

// **sort**: Sorts the content of the specified files (in this case, family and work).

// **-u**: Ensures that only unique lines are printed in the output, meaning duplicate lines between the two files will be removed.

1. Create sorted files from files work and family. Use the key “–o” or redirection arrow “>”. If You use “>” please choose a new file name for sorted file.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ sort -o sorted\_family family

nickk@nickk-VirtualBox:~/Nikolaos$ sort -o sorted\_work work

nickk@nickk-VirtualBox:~/Nikolaos$ sort family > sorted\_family

nickk@nickk-VirtualBox:~/Nikolaos$ sort work > sorted\_work

nickk@nickk-VirtualBox:~/Nikolaos$ cat sorted\_family

nickk@nickk-VirtualBox:~/Nikolaos$ cat sorted\_work

// The **sort** command to alphabetically sort the family and work files. The **command sort -o sorted\_family** family writes the sorted output into a new file **sorted\_family**. Similarly, **sort work > sorted\_work** writes the sorted content of work into **sorted\_work**. The command **cat sorted\_family** and cat **sorted\_work** should display the sorted content of the files.

1. Try using the key “-m” of command sort to sort source files work and family. Then try to sort with the ket “-m” already sorted files. Pay attention that this key does not sort files but merges sorted files.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ sort -m sorted\_family sorted\_work

nickk@nickk-VirtualBox:~/Nikolaos$ sort -m sorted\_family sorted\_work > merged\_file

nickk@nickk-VirtualBox:~/Nikolaos$ cat merged\_file

// The command **sort -m sorted\_family sorted\_work** merges the already sorted files **sorted\_family** and **sorted\_work** without re-sorting them, ensuring they are combined in order. The command **sort -m sorted\_family sorted\_work > merged\_file** saves this merged output into a new file named merged\_file. Running **cat merged\_file** displays the content of the merged file.

1. Sort Your files by the alphabetic order of name.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ sort -k 2 family

nickk@nickk-VirtualBox:~/Nikolaos$ sort -k 2 work

// The command **sort -k 2 family** sorts the file family based on the second column (the names of individuals). Similarly, **sort -k 2 work** sorts the file work by the second column (also names). The -k 2 option specifies sorting by the second field or column in the file.

1. If a few persons have the same surname the program do not sort them automatically according to names. To overcome it, insert the key “-2” which enlarges sorting field till the column number 3.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ sort -k 1,2 family

nickk@nickk-VirtualBox:~/Nikolaos$ sort -k 1,2 work

nickk@nickk-VirtualBox:~/Nikolaos$ sort -k 1,2 family > sorted\_family\_by\_surname\_name

nickk@nickk-VirtualBox:~/Nikolaos$ sort -k 1,2 work > sorted\_work\_by\_surname\_name

nickk@nickk-VirtualBox:~/Nikolaos$ sort -k 1,2 family

nickk@nickk-VirtualBox:~/Nikolaos$ sort -k 1,2 work

// The command **sort -k 1,2** sorts files by the first and second columns (surname and name). This is used to sort the family and work files. The results are saved in **sorted\_family\_by\_surname\_name** and **sorted\_work\_by\_surname\_name** using **>** for redirection.

1. Sort files in the order of the birth month. Try it with the key “+2”. Look how it has been done for people born in October, November, December. Try doing it with the key “+2n”.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ sort -k 3.3,3.4n -k 3.1,3.2n family

nickk@nickk-VirtualBox:~/Nikolaos$ sort -k 3.3,3.4n -k 3.1,3.2n work

nickk@nickk-VirtualBox:~/Nikolaos$ sort +2 family

nickk@nickk-VirtualBox:~/Nikolaos$ sort +2 work

nickk@nickk-VirtualBox:~/Nikolaos$ sort +2n family

nickk@nickk-VirtualBox:~/Nikolaos$ sort +2n work

// The command **sort -k 3.3,3.4n -k 3.1,3.2n** sorts files by birth month (column 3) and day (subcolumns 3.3 and 3.4) in ascending numeric order. **sort +2** sorts files starting from the third column, while **sort +2n** sorts by the same column but numerically. These commands are applied to both **family** and **work** files.

1. **Filters: grep**
2. Create a file named price.veg with the following data Lettuce .89 tomatoes 0.89 broccoli .79 cauliflower 0.89 parsely 2.14 avocado 1.19 carrots 0.69 ctltry 0.59 string-beans 1.29 onions 0.29 asparagus 1.39 corn 0.85 cabbage 0.99 lima-beans 1.29 etllow-beans 1.17 mushrooms 1.09 peppers 1.39 potatoes 0.39 artichokes 1.59 black-beans 1.89 blacheyed-beans 1.49

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ nano price.veg

Inside the nano we include all the data

1. Sort the file twice: once in alphabetic order and twice according to prices. Write results into two new files.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ sort price.veg > sorted\_alphabetically.veg

nickk@nickk-VirtualBox:~/Nikolaos$ sort -k 2n price.veg > sorted\_by\_price.veg

nickk@nickk-VirtualBox:~/Nikolaos$ cat sorted\_alphabetically.veg

nickk@nickk-VirtualBox:~/Nikolaos$ cat sorted\_by\_price.veg

1. Separate all beans in a file.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ grep 'beans' price.veg > beans.veg

nickk@nickk-VirtualBox:~/Nikolaos$ cat beans.veg

1. Find vegetables which names start with letter “s”.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ grep '^s' price.veg

// The command **grep '^s' price.veg** searches for lines in the file price.veg that start with the letter "s". It filters the content and displays only those lines where the vegetable name begins with "s".

1. Find vegetables which names start with a vowel.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ grep '^[aeiouAEIOU]' price.veg

// The command **grep '^[aeiouAEIOU]' price.veg** searches for lines in the file price.veg where the vegetable name starts with any vowel (both uppercase and lowercase vowels: A, E, I, O, U). It filters and displays only those lines where the vegetable name begins with a vowel.

1. Find vegetables which price ends with digit “9”.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ grep '9$' price.veg

1. Find vegetables which price ends with digits “09”, “39”, or “79”.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ grep '\(09\|39\|79\)$' price.veg

// The **command grep '\(09\|39\|79\)$' price.veg** searches through the price.veg file for lines where the price ends with "09", "39", or "79". The **\( and \)** are used to group the patterns, while the **\|** is used to separate the different options. The **$** at the end ensures that the match occurs only at the end of the line, specifically targeting prices that end with one of these digits.

1. Find vegetables which price does not end with digit “9”.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ grep -v '9$' price.veg

// The command **grep -v '9$' price.veg** filters out lines in the price.veg file where the price ends with "9". The **-v** option inverts the match

1. Find vegetables which names do not start with letters from “a” to “m”.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ grep -v '^[a-mA-M]' price.veg

// The command **grep -v '^[a-mA-M]' price.veg** filters out vegetables from the price.veg file whose names start with letters from "a" to "m". The **-v** option inverts the match, so only vegetables with names starting with letters "n" to "z" will be displayed.

1. Find vegetables which price ends with digit “9” but does not end with “39”, “59”, “79”, and “89”.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ grep '9$' price.veg | grep -v '\(39\|59\|79\|89\)$'

**4. Composing pipes (W4)**

7) kn – delete all processes running a specified command.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ nano ~/.bashrc

nano script: kn() {

if [[ -z "$1" ]]; then

echo "Usage: kn <command\_name>";

return 1;

fi

pids=$(ps -eo pid,comm | grep -w "$1" | awk '{print $1}');

if [[ -z "$pids" ]]; then

echo "No processes found for command: $1";

else

echo "$pids" | xargs kill -9;

echo "Processes running '$1' have been terminated.";

fi

}

nickk@nickk-VirtualBox:~/Nikolaos$ source ~/.bashrc

nickk@nickk-VirtualBox:~/Nikolaos$ ./dummy.sh &

nickk@nickk-VirtualBox:~/Nikolaos$ kn dummy.sh

nickk@nickk-VirtualBox:~/Nikolaos$ ps -eo pid,comm | grep dummy.sh

// The provided kn function is a shell script that deletes all processes running a specified command.

**.bashrc**: The script starts by editing the .bashrcfile to define the functionkn`.

**Function Definition**:

* The function checks if a command name is provided as an argument ($1). If not, it displays a usage message.
* It uses ps to list all processes and grep to filter those running the specified command.
* If matching processes are found, their process IDs (PIDs) are extracted using awk.
* The kill -9 command is used to forcefully terminate the processes by sending the kill -9 signal.

1. **Testing the Script**:

* A dummy script dummy.sh is run in the background.
* The kn function is then used to terminate all processes running dummy.sh.
* The command ps -eo pid,comm | grep dummy.sh checks if any dummy.sh processes are still running.

**5. Composing bash scripts (W5)**

1. Track changing access rights in in the specified sub-tree of directories.

Answer:

nickk@nickk-VirtualBox:~/Nikolaos$ nano monitor\_permissions.sh

nano script:

# Check if the directory argument is passed

if [ -z "$1" ]; then

echo "Usage: $0 <directory\_to\_monitor>"

exit 1

fi

# Directory to monitor

DIR="$1"

# Log file for tracking changes

LOGFILE="permissions\_change.log"

# Function to log permission changes

log\_permission\_change() {

FILE="$1"

ACTION="$2"

PERMISSIONS=$(stat -c "%A" "$FILE")

TIMESTAMP=$(date "+%Y-%m-%d %H:%M:%S")

echo "$TIMESTAMP | $ACTION | $FILE | $PERMISSIONS" >> "$LOGFILE"

}

# Start monitoring the directory and its subdirectories

echo "Monitoring directory: $DIR for permission changes..."

# Use inotifywait to watch for CHMOD (permission changes) in the directory

inotifywait -m -r -e chmod --format '%w%f' "$DIR" | while read FILE

do

log\_permission\_change "$FILE" "Permission Changed"

done

nickk@nickk-VirtualBox:~/Nikolaos$ chmod +x monitor\_permissions.sh

nickk@nickk-VirtualBox:~/Nikolaos$ ./monitor\_permissions.sh /home/nickk

nickk@nickk-VirtualBox:~/Nikolaos$ chmod 755 /home/nickk/file.txt

nickk@nickk-VirtualBox:~/Nikolaos$ cat permissions\_change.log

Script Explanation

* Input Validation: The script checks if a directory is provided as an argument. If not, it shows an error message and exits.
* inotifywait: This command listens for events in the specified directory and its subdirectories (-r for recursive). It waits for chmod events, which are triggered when file permissions change.
* stat: After detecting a permission change, stat -c "%A" is used to retrieve the current permissions of the file.
* Logging: The script logs the timestamp, action, file path, and the new permissions into a log file (permissions\_change.log).

1. **Make the script executable**:

* chmod +x monitor\_permissions.sh allows the script to run.

1. **Run the script**:

* ./monitor\_permissions.sh /home/nickk starts monitoring file permission changes in the /home/nickk directory.

1. **Change file permissions**:

* chmod 755 /home/nickk/file.txt modifies the permissions of file.txt to 755.

1. **View the log**:

* cat permissions\_change.log displays any logged changes.

This script tracks and logs changes to file permissions in the specified directory.