**CSC 209 UNIX Tools**

|  |  |  |
| --- | --- | --- |
|  | **Assignment 6** |  |

**Objectives: Shell Features and Process Control**

**Note**: **Read** the entire assignment carefully and carry out the following tasks one by one. For some steps, I may provide the required UNIX command. For others, identify the UNIX commands you need. **UNIX is case sensitive**. Create folders and files exactly as indicated below, paying special attention to case. Complete the report as indicated below and submit the same. I do not want to see copy-and-paste of your computer output in your report. By requiring you to write I am hoping that you will remember the UNIX commands used and the results observed.

1. Login to courses.server using your Webmail Net-ID and password.

Take a note of how the command prompt placed by the shell appears and report below:

Shell prompt: [51] [etay1@courses2016:~]$

Type an appropriate command to print the working directory.

Command to print the working directory: pwd

Your login directory:

/home/etay1

Use an appropriate command to switch to the **csc209** folder which you created in Assignment # 1.

Command to switch from login directory to **csc209**: cd csc209

1. Complete the following steps:

Determine the current working directory.

Command to print the working directory: pwd

Your current working directory: /home/etay1/csc209

You should currently be located in **csc209**. Create a new folder named **asgn06** under **csc209**. Set the access permissions for **asgn06** as **rwx --- ---** by typing a command:

**chmod 700 asgn06**

Command to create the **asgn06** folder: mkdir asgn06

Command to set the access permissions: chmod 700 asgn06

Move to **asgn06** and determine the current working directory.

Command need to switch from **csc209** directory to **asgn06**: cd asgn06

Command to print the working directory: pwd

Your current working directory: /home/etay1/csc209/asgn06

Shell prompt: [58] [etay1@courses2016:~/csc209/asgn06]$

**From now on, unless explicitly permitted, you should not move away from this asgn06 directory.**

**If you happen to log out in the middle of the exercise, be sure to use appropriate command(s)**

**and move to asgn06 before you continue.**

2. Without changing directory, using absolute path, execute one **ls** command with appropriate options to display one line showing a long listing of the access permissions and other characteristics of my directory **~nyu/csc209/asgn06**.

Command to list my **asgn06** directory: ls -ld ~nyu/csc209/asgn06

Output: drwx--x--x 4 nyu domain users 4096 Oct 29 2018 /home/nyu/csc209/asgn06

Execute an **ls** command with appropriate options for a long listing of the non-hidden contents of the **~nyu/csc209/asgn06 directory**. The command will fail.

Command to list contents of my **asgn06** directory ls -l ~nyu/csc209/asgn06

Error reported: ls: cannot open directory /home/nyu/csc209/asgn06: Permission denied

Explain why it failed. Specifically provide the access permission setting that would have allowed you to successfully execute the failed **ls** command.

Reason for error: need read permissions

Access permission setting that would have allowed successful execution:

(make minimal change from existing) : chmod 741

3. Without changing directory, with one command, copy the executable file **ghost** in my directory **~nyu/csc209/asgn06** to your directory **asgn06** as a hidden file with the name **.ghost**.

Command to copy: cp ~nyu/csc209/asgn06/ghost .ghost

Verify the success of copying with **ls** **-l** command for that specific file alone.

Command to verify success: ls -l .ghost

Output: -rwx------ 1 etay1 domain users 5198 Nov 26 21:06 .ghost

Determine the access permissions set on your file. Is it executable? If not, change permissions to make **.ghost** file in your directory executable.

Is the file executable? File is not executable

Command to fix permissions, even if not needed: chmod 700 .ghost

4. Use **pico** editor to create a file named **brandNew**. The file should contain one line in the format: This file was created on Whatday March 32, 2027 at 3:15 AM. (use day, date, and time appropriate for your case).

Command to create file: pico brandNew

Perform **ls –l** for this file alone to verify successful creation.

Command to verify successful creation: ls -l

Output: -rw------- 1 etay1 domain users 64 Nov 26 21:09 brandNew

(**ls –l** reports time of **last** **modification**. Since the file has just been created, the time of last modification happens to be the same as the time of creation. UNIX does **not** keep track of time of creation of a file or a directory.)

5. With one command, create two new directories named **test01** and **test02** under **asgn06**.

Command to create directories: mkdir test01 test02

Without changing directory, use **pico** editor to create two files, one by one, named **data01** and **data02** under **test01**. Each file contains one line of the following form: Data file number xx, where **xx** is replaced by 01 and 02, respectively.

Command to create **data01**: pico test01/data01

Command to create **data02**: pico test01/data02

Again use **pico** editor to create two more files named **data03** and **data04** under **test02**. Each file contains one line of the following form: Data file number xx, where **xx** is replaced by 03 and 04, respectively.

Command to create **data03**: pico test02/data03

Command to create **data04**: pico test02/data04

6. In UNIX, there is data structure named **inode**, where all the properties such as access permissions, ownership, size, etc., for all the files and directories are recorded. Each file or directory has a unique **inode** number. The **inode** number associated with a file can be determined by performing an **ls** command with **–i** option. For example, **ls –i brandNew** will determine the **inode** number for the file **brandNew**.

Command to determine **inode** number for **brandNew**: ls -i brandNew

Output: 6696436 brandNew

Without changing directory, with just one command, determine the **inode** numbers for all four files **data01**, **data02**, **data03**, and **data04**, using appropriate shell wildcard.

Command to determine **inode** number of all four files:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**inode** number for **data01**: 6696439 test01/data01

**inode** number for **data02**: 6696441 test01/data02

**inode** number for **data03**: 6696440 test02/data03

**inode** number for **data04**: 6696442 test02/data04

7. Move the file **data01** from directory **test01** to **test02** with the name of the file changed to **data05**.

Command to move: mv test01/data01 test02/data05

Determine **inode** number of **data05** under **test02**

Command: ls -i test02/data05

Output: 6696439 test02/data05

This **inode** number for **data05** will be the same as that of **data01**, because you did not create a new file; you just moved the file from one directory to another and renamed it.

Copy the file **data05** under **test02** as **data01** in **test01**.

Command to copy: cp test02/data05 test01/data01

Determine the **inode** number of **data01** under **test01**

Command: ls -i test01/data01

Output: 6696443 test01/data01

This **inode** number for this **data01** will be different from that of the original **data01**, because you did create a new file; the content may be the same as before, but this a new copy.

8. Perform the **cat** command to display the contents of the file **brandNew**.

Command to display the file: cat brandNew

Output: This file was created on Saturday November 26, 2022 at 9:09 PM.

Execute the command **sleep 60** to delay your next step by sixty seconds. [Necessary for right

answers]

Command to sleep for 60 seconds: sleep 60

Use **pico** to edit the file **brandNew** to include a second line in the format: This file was modified on Someday October 06, 2020 at 13:48 JM. (use day, date, and time appropriate for your case).

Command to edit the file: pico brandNew

Execute the command **sleep 60** to delay your next step by sixty seconds. [Necessary again]

Command to sleep for 60 seconds: sleep 60

Change the access permission of **brandNew** to make it **read only** for you alone, with no permissions for group members or others.

Command to change access permissions: chmod 400 brandNew

9. For each file or directory, UNIX maintains three different time fields in the **inode** data structure:

* Time of last access (**atime**), i.e., use, without modification, of file contents
* Time of last modification (**mtime**) of file contents
* Time of last change (**ctime**) of some property (file size, access permission, owner, group, and number of links) maintained in the **inode**

Perform an **ls** command on the file **brandNew** with the following options, one by one, and observe the date and time field:

Perform **ls –lu brandNew** to display the time of last use of the file, without modifying it.

Output: -r-------- 1 etay1 domain users 129 Nov 26 21:23 brandNew

Perform **ls –l brandNew** to display the time of last modification of the file **brandNew**

Output: -r-------- 1 etay1 domain users 129 Nov 26 21:23 brandNew

Perform **ls –lc brandNew** to display the time of last change of some property (maintained in the **inode**) of the file

Output: -r-------- 1 etay1 domain users 129 Nov 28 05:32 brandNew

You could also use **stat brandNew** command to determine the inode number, all three time fields and other information.

Command: stat brandNew

Output:

File: ‘brandNew’

Size: 129 Blocks: 8 IO Block: 4096 regular file

Device: 811h/2065d Inode: 6696436 Links: 1

Access: (0400/-r--------) Uid: (1398716188/ etay1) Gid: (1398600513/domain users)

Access: 2022-11-26 21:23:32.291303249 -0500

Modify: 2022-11-26 21:23:32.291303249 -0500

Change: 2022-11-28 05:32:43.632877080 -0500

Birth: -

When you read a file (say with **cat** or **cp** commands), only **atime** will change. When a file is modified, the size will change. As a result, both **mtime** and **ctime** will change. When you change access permission, only **ctime** will change.

10. Until now, we have used the **touch** command only to create empty files (files of size 0 bytes). What if we touch an already existing file? Type the command: **touch brandNew**

Use **stat** command to determine all three time fields.

Command: stat brandNew

Describe how the time fields have changed:

The access modify and fields changed to current time

All three time fields will change to the current time, i.e., the time at which the file is touched.

We can also selectively change **atime** or **mtime** to specific values. Type the command: **touch –a –t 201407010000.00 brandNew**

Use **stat** command to determine all three time fields.

Has **atime** changed from that determined above? no

If yes, state its new value? Access: 2014-07-01 00:00:00.000000000 -0400

Has **mtime** changed from that determined above? Modify: 2022-11-28 05:36:07.638549141 -0500

If yes, state its new value?

Has **ctime** changed from that determined above? Change: 2022-11-28 05:36:28.154714847 -0500

If yes, state its new value? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Option **–a** allows us to change **atime**. Likewise, option **–m** allows changing **mtime**. There is no option to set **ctime**. Also, when the **atime** and **mtime** fields are set using the **touch** command, **ctime** will reflect the time at which the **touch** command is executed. The date and time set could be in the future as well. Type the command: **touch –m –t 202010010000.00 brandNew**

Use the **stat** command to see which time fields have changed.

Has **atime** changed from that determined above? Yes

If yes, state its new value? below

Has **mtime** changed from that determined above? Yes

If yes, state its new value? below

Has **ctime** changed from that determined above? Yes

If yes, state its new value? Below

When I set that command it set all the times to current time?

[64] [etay1@courses2016:~/csc209/asgn06]$ stat brandNew

File: ‘brandNew’

Size: 129 Blocks: 8 IO Block: 4096 regular file

Device: 811h/2065d Inode: 6696436 Links: 1

Access: (0400/-r--------) Uid: (1398716188/ etay1) Gid: (1398600513/domain users)

Access: 2022-11-28 05:40:09.790513248 -0500

Modify: 2022-11-28 05:40:09.790513248 -0500

Change: 2022-11-28 05:40:09.790513248 -0500

Birth: -

11. Make sure your current working directory is **asgn06**.

Type each of the following six commands, observe the output and report:

|  |  |
| --- | --- |
| **Command** | **Output** |
| **echo** | blank |
| **echo \\*** | \* |
| **echo "\*"** | \* |
| **echo '\*'** | \* |
| **echo \*** | 201407010000.00 202010010000.00 –a brandNew –m –t test01 test02 |
| **echo \*/\*** | /bin/ /boot/ /dev/ /etc/ /export/ /home/ /lib/ /lib64/ /lost+found/ /media/ /mnt/ /opt/ /proc/ /restored/ /root/ /run/ /sbin/ /scripts/ /srv/ /sys/ /tmp/ /usr/ /var/ |

12. Determine the commands for which an alias has been set by simply typing the command: **alias**

Command to determine current aliases: alias

Current aliases: If you have grep aliases, skip them. List the first 7, if 7:

alias cls='clear'

alias egrep='egrep --color=auto'

alias fgrep='fgrep --color=auto'

alias grep='grep --color=auto'

alias h='history'

alias l.='ls -d .\* --color=auto'

alias ll='ls -l --color=auto'

alias lo='logout'

alias ls='ls --color=auto'

alias vi='vim +"set number"'

alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'

13. Execute the **ls** **–a** command on the current working directory (which should be **asgn06**).

Command: ls -a

Items listed: ­­­­­­­­­­­

. 201407010000.00 –a .ghost –t test02

.. 202010010000.00 brandNew –m test01

Create an alias named **ls** which stands for **ls –F**

Command: alias ls=’ls -F’

Re-execute the **ls –a** command on the current working directory. Because of the alias created, we are actually executing **ls –aF** on the current working directory.

Items listed:

./ 201407010000.00 –a .ghost\* –t test02/

../ 202010010000.00 brandNew –m test01/

The option **–F** adds a flag at the end of the file or directory name: directories are flagged with a **/** and executable files are flagged with an **\***

Type the command **ll –a** on the current working directory**.**

Describe the color coding: blue for directories, green for executables, white for regular files

Remove the alias newly created.

Command: unalias ls=’ls -F’

Type **ls** **–a** command on the current working directory.

Command: ls -a

Items listed:

./ 201407010000.00 –a .ghost\* –t test02/

../ 202010010000.00 brandNew –m test01/

14. Type the command **history** to see a long history of commands that you had typed recently.

Command to inspect saved history: history

Without changing directory, display the **.bashrc** file, which is in your login directory. Use **absolute** path for the **.bashrc** file.

Command to view **.bashrc** file: cat ~/.bashrc

Determine how many commands are preserved in each session and how many commands are saved at the time of logout.

Number of commands preserved in each session: 60

Which variable controls the above? histsize

Number of commands saved at logout: 50

Which variable controls the above? histfilesize

15. Without retyping the whole command, i.e., by using history substitution, we wish to re-execute the previouscommand, but with **.bashrc** replaced by **.bash\_profile**. Determine the command, execute it and report if any aliases are set in **.bash\_profile** file.

Command to inspect **.bash\_profile** file: cat ~/.bash\_profile

Aliases set in **.bash\_profile** file, if any: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

16. The UNIX **ping** command is used to verify if a server on a network is up and running. For example, to check if the web server at Cornell is working, type: **ping www.cornell.edu**

The ping command sends a sequence of data packets and if the server responds, keeps track of the round trip time for packets. But some servers are configured not to respond to ping. If the output has several lines, be sure to terminate its execution (by typing control-C), after say 20 lines. The command will print some statistics regarding round trip time (rtt): min, average, max, and deviation.

Check if [**www.cornell.edu**](http://www.cornell.edu) is working.

Command: ping www.cornell.edu

Was it necessary to terminate output? no

Ping Statistics:

PING ucomm-gw1.cornell.media3.us (20.42.25.107) 56(84) bytes of data.

Re-execute the previous command to check if [**www.brockport.edu**](http://www.brockport.edu) is up and running. That is, use history substitution, with **cornell** substituted by **brockport**

Command: ping www.brockport.edu

Was it necessary to terminate output? yes

In this case, you will get multiple responses from **www.brockport.edu**

Ping Statistics:

64 bytes from www.brockport.edu (146.20.54.132): icmp\_seq=20 ttl=47 time=20.2 ms

Re-execute the previous command to check if **www.cnn.com** is up and running. That is, use history substitution, with **brockport.edu** substituted by **cnn.com**

Command: ping www.cnn.com

Was it necessary to terminate output? yes

In this case, you will get multiple responses from **www.cnn.com**

Ping Statistics:

64 bytes from 151.101.211.5 (151.101.211.5): icmp\_seq=1 ttl=57 time=11.3 ms

17. Using the **echo** command, display the value of the shell variable **PATH**

Command: echo “$PATH”

The output will specify a sequence of directories separated by colon (**:**) symbol. It is used as explained below.

Output:

**/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/puppetlabs/bin:/home/etay1/.local/bin:/home/etay1/bin**

The Bash shell by itself can execute only a few very simple commands. Most of the time, it invokes an executable program file to get the job done. So when you execute a command by typing the name of the executable, the shell will have to look for the executable file. It will look for the executable file with the given name in the first directory listed in the **PATH** variable. If not found, it will look for it in the second directory, and so on. Note **.** is in the **PATH**. So eventually, it will look for the executable in the current working directory. That is, the **PATH** variable spells out a sequence of directories in which the shell will look for the executable, exactly in the order specified in **PATH**.

18. Use the pico editor, and create a file named **ping** in your current working directory (**asgn06**) with the following two lines:

**#!/bin/bash**

**echo "Hello, Have a Good Day!"**

Command to edit a new executable: pico ping

Make the file executable, and then execute it.

Command to make the file executable: chmod u+x ping

Command to execute: ping

If you simply typed **ping**, you will find that it did not execute your shell program. It executed the UNIX command **ping** and since you did not provide any server address, it reported correct usage. This problem can be easily detected by using **which** command.

Type **which ping** to see which program will be executed and exactly in which directory it is located.

Command to determine the location of UNIX **ping**: which ping

Directory in which UNIX **ping** is located: /usr/bin/ping

Having determined the directory in which the UNIX **ping** command is located, explain the reason for the failure to execute our executable **ping**, taking into account the **PATH** you just determined.

Reason for the failure to execute our **ping**: ping is not under out current directory

Hence, if we create an executable program with the same name as an existing shell command, the shell is not likely to execute our program, unless we provide the absolute or relative path to our executable file.

Command to run our executable ping: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Output: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

19. In operating systems terminology, a process means a program in execution. It is possible to have more than one instance of the same program in execution concurrently. Each process has a unique process id (PID). Type the command: **ps** Report processes that are running (at the time of reporting) and their corresponding PIDs.The **ps** process itself is short-lived and will terminate right after reporting.

Command typed: ps

Processes listed (name and PID):

PID TTY TIME CMD

73712 pts/0 00:00:00 bash

79398 pts/0 00:00:00 ps

Re-run the **ps** command again. Has the PID changed for the **ps** process?

Processes listed in the second run (name and PID):

PID TTY TIME CMD

73712 pts/0 00:00:00 bash

79438 pts/0 00:00:00 ps

Did the PID change for **ps**? yes

20. With the current login on courses active, use SSH (or PuTTY) and login into courses again. Move to asgn06 under csc209. If you wish, tile the two SSH windows vertically side by side on your screen. (To do so, minimize all the windows other than the two PuTTY windows. Right mouse click on an empty space in the task bar and choose Tile Windows Vertically.) Type the **ps** command in the new terminal window. Report processes that are running and their corresponding PIDs.

Command in new window: ps

Processes listed (name and PID):

PID TTY TIME CMD

79505 pts/1 00:00:00 bash

79582 pts/1 00:00:00 ps

Now type the command in the form **ps –u jdoe1** (with jdoe1 replaced by your Net-ID.). You should see all the processes that are currently run by you, no matter in which window. You as the owner can control all these processes. Report processes that are running and their corresponding PIDs.

Command in new window to list all processes in both windows: ps -u etay1

Processes listed (name and PID):

PID TTY TIME CMD

73711 ? 00:00:00 sshd

73712 pts/0 00:00:00 bash

79504 ? 00:00:00 sshd

79505 pts/1 00:00:00 bash

79608 pts/1 00:00:00 ps

21. Return to the original window. It is possible to write programs that cannot be terminated by typing Control-C. Execute the program **.ghost** in your **asgn06** directory, by simply typing the command name. Try to terminate it by typing Control-C. It will not work. Report what you observe.

Command in original window to run the program: ./ .ghost

Output observed when run:

This program is protected from Control-C

Try, typing it

Output on attempting Control-C: Ha, Ha, Ha! You can not terminate me!

Move to the new terminal window. Determine the PID of the ghost process. Issue the command of the form: **kill PID**, using the appropriate PID for the ghost process. See what you observe on the window on which ghost was running.

Command in new window to determine PID for .ghost: ps -u

Command in new window to terminate .ghost: kill 81152

Changes in the original window: Terminated

22. Return to the original window. Run the sleep command for 15 seconds: **sleep 15** After 15 seconds, you should see the command prompt.

Command in original window to sleep: sleep 15

Run the sleep command for 600 seconds. Move to the new terminal window, determine the process id for sleep and kill it before it is done.

Command in original window to sleep longer: sleep 600

Command in new window to determine PID for sleep: ps -u

Command in new window to terminate sleep: kill 81263

Changes in the original window: Terminated

23. Return to the original window. Run the sleep command for 1200 seconds, but place it in the background by typing: **sleep 1200 &** You will see a command prompt right away as you are running the process in the background. The system will report a job number and a process ID.

Command in original window: sleep 1200 &

Output displayed: [1] 81858

24. Run the sleep command for 600 seconds, in the background again.

Command in original window: : sleep 600 &

Output displayed: [2] 81899

Then run another sleep process in the foreground for 10 seconds.

Command in original window: sleep 10

At this stage, you have three jobs, one in the foreground and two in the background. Each job involves only one process, but it is possible to have multiple processes for each job as well. When the prompt appears after the **sleep 10** command completes, run **ps** command and report all PIDs.

Command in original window: ps

Output displayed:

PID TTY TIME CMD

81047 pts/0 00:00:00 bash

81858 pts/0 00:00:00 sleep

81899 pts/0 00:00:00 sleep

81935 pts/0 00:00:00 ps

25. Bring the first job to the foreground with the command: **fg %1** A foreground job can be killed by Control-C. Terminate this process that way.

How did you terminate? Control C

Use the kill command, but with job number to terminate Job 2: **kill %2** Then run **ps** command and report what you see on job 2. (Since there is only one process associated with Job 2, the process could have been terminated using its PID as well.)

Output displayed:

PID TTY TIME CMD

81047 pts/0 00:00:00 bash

82006 pts/0 00:00:00 ps

[2]+ Terminated sleep 600