Tutorial8: Links / Process Management

CDOT
SICT AR Meeting Area
People

get involved with CDOT

as a Student
as an Open Source Community Member
as a Company

courses

BTC640
BTH740
BTP300
DPI908
DPS901
DPS905
DPS909
DPS911
DPS914
DPS915
DPS924
DPS931
EAC234
ECL500
GAM531
GAM666
GAM670
GPU610
LUX Program
MAP524
OOP344
OPS235
OPS245
OPS335
OPS345
OPS435
OPS445
OPS535
OPS635
OSD600
OSD700
OSL640
OSL740
OSL840
Real World Mozilla
RHT524
SBR600

LINKING FILES / MANAGING PROCESSES

Main Objectives of this Practice Tutorial

- Define the term **i-node** as it relates to the Unix/Linux File System
- Issue the **ls -i** command to view **i-node** (index) numbers associated with Unix/Linux files
- Define the terms **Hard** and **Symbolic** Links
- Issue the **`ln`** command to create **hard** and **symbolic** links
- Define term **process** as it relates to the Unix/ Linux operating system
- **Run** and **terminate** processes in the foreground and background
- **Display** and **manipulate** background and foreground processes
- Use **`alias`** and **`history`** commands in Unix/Linux

Tutorial Reference Material

Course Notes

Slides:

- Week 8 Lecture 1 Notes: PDF | PPTX
- Week 8 Lecture 2 Notes:

Concepts / Commands

Links:

- Hard Links
- Symbolic Links

Managing Processes:

- inode

Linux Commands:

- ln
- ps
- top
- fg

YouTube Videos

Brauer Instructional Videos:

- Inodes and Links
- Processes and Jobs

SEC520
SPO600
SRT210
ULI101

course projects

Course Project List
Potential Course Projects
Contrib Opportunities

links

CDOT
Planet CDOT
FSOSS

Tools

What links here
Related changes
Special pages
Printable version
Permanent link
Page information

PDF | PPTX

- Manipulating Processes

- bg
- jobs
- kill
- sleep
- alias , unalias
- history

KEY CONCEPTS

i-node (index) ID Number of a File

An **i-node** is a **database** containing information (e.g. *file type*, *owner*, *permissions*, etc.) for all files that are created on the Unix/Linux filesystem.

```
[ murray.saul ] pwd
/home/murray.saul/link-demo1
[ murray.saul ] touch myfile.txt
[ murray.saul ] ln myfile.txt myfile1.hard.lnk
[ murray.saul ] ln myfile.txt myfile2.hard.lnk
[ murray.saul ] ln myfile.txt ~/myfile3.hard.lnk
[ murray.saul ]
[ murray.saul ] ls -li . ~/myfile3.hard.lnk
3261599590 -rw-r--r-- 4 murray.saul users  0 Feb  3 08:39 /home/murray.saul/myfile3.hard.lnk

.:
total 0
3261599590 -rw-r--r-- 4 murray.saul users 0 Feb  3 08:39 myfile.txt
3261599590 -rw-r--r-- 4 murray.saul users 0 Feb  3 08:39 myfile1.hard.lnk
3261599590 -rw-r--r-- 4 murray.saul users 0 Feb  3 08:39 myfile2.hard.lnk
```

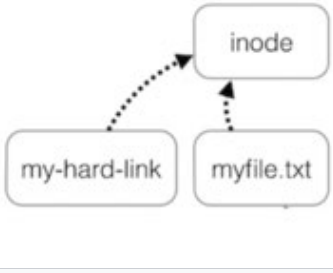
The **i-node number** is like a **finger-print**, and is considered to be **unique** for each file on the Unix / Linux file system.

The *i-node number* is like a **finger-print**, and is considered to be **unique** for each file on the Unix / Linux file system.

Referring to the diagram on the far right, issuing the `ls` command with the `-i` option displays the i-node number for each file. You can see that each file (whether it is a directory or regular file) has its own unique i-node number.

Hard Links

A **Hard link** is a **reference** to the physical data on a file system.
It does this by creating a file that shares the **same i-node number** with the original file.



A **Hard link** is a file which is created that shares the **same i-node number** with the original file
(Image licensed under cc)
Image manipulated by author

Advantages: If only one hard link remains (even if original file has been removed), the **data in that hard linked file is NOT lost**. The data in hard linked files are **automatically updated** when original file are updated.

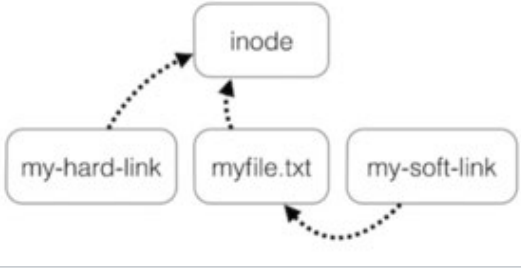
Disadvantages: Hard links **take-up extra space**, you **cannot hard link directories**, and you **cannot hard link files from other Unix/Linux servers** (since the inode number may already be used by the other Unix/Linux server).

Examples:

```
ln myfile.txt myfile1.hard.lnk
ln myfile.txt ~/backups/myfile.hard.lnk
```

Symbolic Links

A **Symbolic Link** is an indirect **pointer** to a file and are also known as **soft** link or **symlink**. The symbolic link file contains the **pathname** to the original file.



A **Symbolic Link** is an indirect **pointer** to a file and are also known as **soft** link or **symlink**. The symbolic link file contains the pathname to the original file. (Image licensed under cc)

Advantages: symbolic links are **shortcuts** to other files, where the symbolic link only contains the pathname to the original file, you **can create symbolic links on different Unix/Linux servers**, and that you **can create symbolic links for directories**.

Disadvantages: Symbolic links are NOT good for backup purposes since a symbolic link can point to a nonexistent file (referred to as a "broken link").

Examples:

```
ln -s otherfile.txt otherfile1.sym.lnk
ln -s otherfile.txt ~/backups/otherfile.sym.lnk
```

Managing Processes

All **commands/programs** (**tasks**) that are **running** on a Unix/Linux computer system are referred to as **processes**.

Characteristics of Processes:

- Each process has an **owner**
- Each process has a unique ID (**PID**)
- Processes keep their *PID* for their **entire life**.
- Usually a parent sleeps (*suspends*) when a child is running (the exception is when the child process is running in the background)
- UNIX / Linux processes are **hierarchical**. The process structure can have **child processes**, **great grandchild processes**, etc.

Users can **manage processes** to become more **productive** while working in the Unix / Linux Command-line environment.
Processes that run in the terminal are known as **foreground** processes. You can run or send processes currently running in the *foreground* to the **background** to free-up your terminal (e.g. issue other Linux commands).

Below are a listing of common **Linux commands** and **keyboard shortcuts** to manage foreground and background processes:

Linux Command	Purpose

/ Key Combination	
`ps`	Displays snapshot information about processes. *Examples:* `ps , ps -l , ps -ef , ps -u , ps aux`
`top`	The **top** command provides a realtime status of running processes. **NOTE:** You can press **ctrl-c** to exit
`ctrl-c`	**Terminates** a process running in the foreground
`ctrl-z`	Sends a process running in the foreground into the **background**.
`fg`	Moves a background job from the current environment into the foreground. *Example:* `fg %job-number`
`bg`	Runs (starts) the most recent process that was placed into the background. *Example:* `bg %job-number`

`jobs`	The **jobs** utility displays the **status** of jobs that were started in the current shell environment. *Example:*
	`jobs`
	`[1]+ Stopped vim a  <-- Job #1 (+ most recent process / background)`
	`[2]  Running sleep 200 &  <-- Job #2`
	`[3]  Running sleep 300 &  <-- Job #3`
	`[4]- Running sleep 400 &  <-- Job #4 (- second recent process / background)`

`kill`	The kill command sends the specified signal to the specified processes or process groups. If no signal is specified, the **TERM** signal is sent. The default action for this signal is to terminate the process.
	Examples:
	`kill PID , kill -9 PID , kill %job-number ,`
	`kill -9 %job-number`

Aliases / Command History

Aliases:

An **alias** is a **nickname** to an existing command or group of commands.

An alias existing in **system memory** and will be **lost** when your current Linux session ends, unless the alias is set in a **start-up file** (e.g. `~/.bashrc`. You will learn about using start-up files

later in this course.

Examples:

`alias` (Alias command without an argument will display all the aliases currently set)

```
alias dir=ls
alias ls='ls -al'
alias clearfile='cat /dev/null >'
```

`unalias alias-name` (removes alias from memory)

Command History:

The filename `~/.bash_history` stores recently executed command lines

Examples of commands that use command history:

`up arrow` or `down arrow`	move to **previous** command or **next** command within Bash shell prompt
`fc -l`	display last **16** commands
`history | more`	display all stored commands
`!num`	**re-execute** an issued command number by command number (determined from *history* command)
`!xxx`	**re-run** a most recent previously-issued command beginning with string "xxx"

INVESTIGATION 1: LINKING FILES

ATTENTION: This online tutorial will be required to be completed by **Friday in week 9 by midnight** to obtain a grade of **2%** towards this course

In this investigation, you will learn how to create **hard links** and **symbolic links** on your Matrix account,
and observe the advantages and limitations of using both types of links.

Perform the Following Steps:

1. **Login** to your matrix account.

2. Issue a Linux command to **confirm** you are located in your **home** directory.

   NOTE: You will remain in your **home** directory to get practice using pathnames.

3. Issue the following Linux command to create a directory called **~/links**:
   `mkdir ~/links`

4. Issue the **ls -ld** command to confirm that the directory **~/links** exists.

   ls -ld ~/links drwxr-xr-x 2 twwong9 users 6 Mar 17 13:06 /home/twwong9/links

5. Use a text editor to create a file called **`~/links/data-file.txt`**
   (i.e. without changing to the links directory).

6. Enter the following text displayed below:

   ```
   This is line 1
   This is line 2
   This is line 3
   ```

7. Save your editing session and exit your text editor.

Hard links share the same **i-node** with regular files on a Unix / Linux filesystem.

8. Issue the following Linux command:
   `ls -li ~/links/data-file.txt`

   View the **i-node** number for this file. What does this *i-node* number represent?
   1253582860 -rw-r--r-- 1 twwong9 users 45 Mar 17 13:08 /home/twwong9/links/data-file.txt
   We will now create a **hard link** file to demonstrate how creating hard links are useful for **back-ups**.

9. Issue the following Linux command to create the following **hard link** in the same directory:
   `ln ~/links/data-file.txt ~/links/data-file.hard.lnk`

10. Issue the following Linux command to display *i-node* ID numbers for both files:
    `ls -li ~/links/data-file.txt ~/links/data-file.hard.lnk`

    What do you notice about both of those file's *i-node* numbers?
1253582860 -rw-r--r-- 2 twwong9 users 45 Mar 17 13:08 /home/twwong9/links/data-file.hard.lnk
1253582860 -rw-r--r-- 2 twwong9 users 45 Mar 17 13:08 /home/twwong9/links/data-file.txt
11. Use a text editor to edit **`~/links/data-file.txt`**
    and **add some lines of text** to the bottom of that file.

12. Save your editing session and exit your text editor.

13. Issue the following Linux command:
    `cat ~/links/data-file.hard.lnk`

    ```
    This is line 1
    This is line 2
    This is line 3
    This is the adding line
    This is the final line
    ```

    You should notice that the hard linked file also contains the additional line(s) that you added to the original file.
    This is very useful for backing up your files without using the **cp** command!

14. Use a text editor to edit the hard-linked file **`~/links/data-file.hard.lnk`**
    and add some lines to the bottom of this file.

15. Save your editing session and exit your text editor.

16. Issue the following Linux command:

```
cat ~/links/data-file.txt
```

The original file also contain the additional lines I added to the hard-linked file

What happened to this **original** file?</u> file?

What does this mean in terms of creating hard-linked files for back-ups?

The content of hard-linked files will be sync

17. Issue the following Linux command to create a hard-linked file in your **home** directory:

```
ln ~/links/data-file.txt ~/data-file.hard.lnk
```

18. Issue the following Linux command to compare all file's *i-node* numbers:

```
ls -li ~/links/data-file.txt ~/links/data-file.hard.lnk ~/data-file.hard.lnk
```

```
1253582860 -rw-r--r-- 3 twwong9 users 101 Mar 17 13:16 /home/twwong9/data-file.hard.lnk
1253582860 -rw-r--r-- 3 twwong9 users 101 Mar 17 13:16 /home/twwong9/links/data-file.hard.lnk
1253582860 -rw-r--r-- 3 twwong9 users 101 Mar 17 13:16 /home/twwong9/links/data-file.txt
```

What do you notice about all of those file's *i-node* numbers?

They are all the same

19. Issue the following Linux command to check that you created those hard links:

```
~uli101/week8-check-1
```

If you encounter errors, then view the feedback to make corrections, and then re-run the checking script.
If you receive a congratulation message that there are no errors, then proceed with this tutorial.

20. Issue the following Linux command to remove the **~/links** directory and its contents:

```
rm -rf ~/links
```

21. Issue a Linux command to confirm that the **~/links** directory has been removed.

ls -ld ~/links ls: cannot access /home/twwong9/links: No such file or directory

22. Issue the following Linux command to view the contents of your linked file in your **home** directory:

```
cat ~/data-file.hard.lnk
```

What do you notice? What does this tell you about hard links?
The data-file.hard.lnk still exist, and the content still preserve.

We will now learn how to create **symbolic links**.

23. Issue the following Linux command to create a directory called **~/links2**:

```
mkdir ~/links2
```

NOTE: You will remain in your **home** directory to get practice using pathnames.

24. Issue the `ls -ld` command to confirm that the directory called **~/links2** exists.
drwxr-xr-x 2 twwong9 users 6 Mar 17 13:32 /home/twwong9/links2
25. Use a text editor to create a file called `~/links2/text-file.txt`
(i.e. without changing to the **links2** directory).

26. Enter the

```
[ murray.saul ] ln -s ~/links2/text-file.txt ~/links2/text-file.sym.lnk
[ murray.saul ]
[ murray.saul ] ls -li ~/links2/text-file.txt ~/links2/text-file.sym.lnk
133784071 lrwxrwxrwx 1 murray.saul users 38 Mar  6 11:49 /home/murray.saul/links2/text-file.sym.lnk -> /home/murray.saul/links2/text-file.txt
133784072 -rw-r--r-- 1 murray.saul users 54 Mar  6 11:49 /home/murray.saul/links2/text-file.txt
```

Symbolic links are **pointers** (i.e. pathnames) to **regular files** and **directories**.
They do **NOT** share the same **i-node**.

following text displayed below:

```
This is line one
This is line two
This is line three
```

27. Save your editing session and exit your text editor.

28. Issue the following Linux command to create the following **symbolic** link in the same directory:
    `ln -s ~/links2/text-file.txt ~/links2/text-file.sym.lnk`

29. Issue the following Linux command to display *i-node* numbers for both files:
    `ls -li ~/links2/text-file.txt ~/links2/text-file.sym.lnk`

    What do you notice about both of these file's *i-node* numbers?
    What do you notice about the size of the file `~/links2/text-file.sym.lnk`*?*
    What ***pathname*** *do you think this* symbolic-linked *file represents?*

2295276143 lrwxrwxrwx 1 twwong9 users 34 Mar 17 13:35 /home/twwong9/links2/text-file.sym.lnk -> /home/twwong9/links2/text-file.txt
2295276165 -rw-r--r-- 1 twwong9 users 53 Mar 17 13:34 /home/twwong9/links2/text-file.txt

the pathname points to /home/twwong9/links2/text-file.txt

30. Issue the following Linux command to create the following **symbolic link** in your **home** directory:
    `ln -s ~/links2/text-file.txt ~/text-file.sym.lnk`

31. Issue the following Linux command to display *i-node* numbers for all of those files:
    `ls -li ~/links2/text-file.txt ~/links2/text-file.sym.lnk ~/text-file.sym.lnk`

```
2295276143 lrwxrwxrwx 1 twwong9 users 34 Mar 17 13:35 /home/twwong9/links2/text-file.sym.lnk -> /home/twwong9/links2/text-file.txt
2295276165 -rw-r--r-- 1 twwong9 users 53 Mar 17 13:34 /home/twwong9/links2/text-file.txt
 124543719 lrwxrwxrwx 1 twwong9 users 34 Mar 17 13:42 /home/twwong9/text-file.sym.lnk -> /home/twwong9/links2/text-file.txt
```

    What do you notice about all of those file's i-node numbers? They are different
    What is the file size of `~/text-file.sym.lnk`?
    What **pathname** do you think this *symbolic-linked* file contains?

the pathname points to /home/twwong9/links2/text-file.txt

32. Use a text editor to edit the **symbolic** link file called `~/links2/text-file.sym.lnk` and add some lines to the bottom of that file.

33. Save your editing session and exit your text editor.

34. Issue the following Linux command to view the contents of the **original** file:
    `cat ~/links2/text-file.txt`

The content of original file also gets updated

```
This is line one
This is line two
This is line three
It is /links2/text-file.sym.lnk
```

    What did you notice? This happened because when you edited the symbolic-linked file, you were redirected (via *pathname*) to the original file.

35. Use a text editor to edit the **original** file called `~/links2/text-file.txt` and add some lines to the bottom of that file.

36. Save your editing session and exit your text editor.

37. Issue the following Linux command to view the contents of the **symbolic** linked file:

```
cat ~/links2/text-file.sym.lnk
```

What did you notice? Again, when you view the contents of the symbolic-linked file, you are redirected (via *pathname*) to the original file.

The content of symbolic-linked file also get updated

This is line one
This is line two
This is line three
It is /links2/text-file.sym.lnk
It is ~/links2/text-file.txt now

38. Issue the following Linux command to check that you created those symbolic links:
```
~uli101/week8-check-2
```

If you encounter errors, then view the feedback to make corrections, and then re-run the checking script.
If you receive a congratulation message that there are no errors, then proceed with this tutorial.

39. Issue the following Linux command to remove the **~/links2** directory:
```
rm -rf ~/links2
```

40. Issue a Linux command to confirm that the **~/links2** directory has been removed.

ls -ld ~/links2 ls: cannot access /home/twwong9/links2: No such file or directory

41. Issue the following Linux command to view the contents of the **original** file called **`~/links2/text-file.txt`**:
```
cat ~/text-file.sym.lnk
```

What happened? Why did does this happen?

```
cat: /home/twwong9/text-file.sym.lnk: No such file or directory
```

42. Issue the

```
[ murray.saul ] rm -rf ~/links2
[ murray.saul ]
[ murray.saul ] ls -l ~/text-file.sym.lnk
lrwxrwxrwx 1 murray.saul users 38 Mar  6 12:05 /home/murray.saul/text-file.sym.lnk -> /home/murray.saul/links2/text-file.txt
```

Example of a **broken link** when a symbolic link points to a **non-existent file**.

following Linux command:
```
ls -l ~/text-file.sym.lnk
```

This output indicates a "**broken link**" and indicates this is not an effective method of backing up files.

lrwxrwxrwx 1 twwong9 users 34 Mar 17 13:42 /home/twwong9/text-file.sym.lnk -> /home/twwong9/links2/text-file.txt

43. Issue a command to delete the **`~/text-file.sym.lnk`** file which is a *broken link*.

rm ~/text-file.sym.lnk

44. Issue the following Linux command:
```
ln -s ~jason.carman/example t8example
```

point to other Linux/Unix servers

45. Issue the

```
[ murray.saul ] ln -s ~murray.saul/scripts scripts
[ murray.saul ]
[ murray.saul ] ls -ld scripts
lrwxrwxrwx 1 murray.saul users 25 Mar  6 11:58 scripts -> /home/murray.saul/scripts
```

Symbolic links can be used to point to **directories** as well as regular files. Symbolic links can also point to files on **other** Unix/Linux filesystems.

following Linux command:
```
ls -ld t8example
```

What do you notice? Symbolic links are good for creating "short-cuts" to both **regular files** and **directories**.

lrwxrwxrwx 1 twwong9 users 26 Mar 17 14:08 t8example -> /home/jason.carman/example

In the next investigation, you will learn how to **manage processes** on your Matrix server.

INVESTIGATION 2: MANAGING PROCESSES

In this investigation, you will learn how to **manage processes** on a Unix / Linux server.

Perform the Following Steps:

1. Make certain that you are logged into your Matrix account.

2. Issue a Linux command to confirm that you are located in your **home** directory.

   The **sleep** command **pauses for a specified number of seconds** before returning to the shell prompt.
   In this tutorial, we will be using this command to **simulate** the management of "long-running" processes.

3. Issue the following Linux command: `sleep 700`

   Notice that this process will run for **700 seconds**, and is forcing the user to **wait** until this process finishes.
   A process that is **running in the terminal** is referred to as a **foreground processes**.
   發送先發製人的信號來管理這些流程
   The Unix/Linux system is designed to allow users to send **preemptive signals** to manage those processes.

4. Press the following **key combination** to **terminate** the command running on the terminal:
   `ctrl-c` `control + c, not command + c`

   You should notice that the process that was running in the foreground has been **interrupted** (i.e. terminated).
   NOTE: The **ctrl-c** key combination sends **SIGINT** (**Signal Interrupt** - which is signal **#2**) to *terminate* a process that is running on the terminal (i.e. a **foreground** process).

5. Reissue the Linux command: `sleep 700`

6. Press the **key combination**: `ctrl-z`

   `Sends a process running in the foreground into the background. Process is stopped (suspended in background and requires bg command to run in background.`

7. You should now see output similar to what is displayed below:
   `[1]+ Stopped sleep 700`

   NOTE: This indicates that this process has been placed into the **background**. This is useful in order to "**free-up**" the terminal to run other Linux commands.

```
[ murray.saul ] sleep 700
^Z
[1]+  Stopped                 sleep 700
[ murray.saul ]
[ murray.saul ] jobs
[1]+  Stopped                 sleep 700
```

Running a command in the terminal, pressing **ctrl-z** to place into the background, and issuing the **jobs** command to view processes in the background.

8. Issue the following Linux command: `jobs`

   You should see the following output similar that was displayed above:
   `[1]+ Stopped sleep 700`

   This display indicates that this process (that is now in the background) has **stopped**.
   In other words, the *sleep* command is NOT counting-down to zero to terminate.

   NOTE: You need to use the **bg** command to **run** that process that was sent into the **background**.

9. Issue the following Linux command:
   `bg`

   NOTE: You can use the bg command WITHOUT arguments to run recent in the background. From the **jobs** command, the process that has a plus sign "+" indicates the most recent process placed into the background.

```
[ murray.saul ] bg
[1]+ sleep 700 &
[ murray.saul ] jobs
[1]+  Running                 sleep 700 &
[ murray.saul ]
```

Using the **bg** command to **run recent process** that was placed into background from using **ctrl-z** keys.

10. Issue the following Linux command: `jobs`

    You should see the following output similar that was displayed above:
    `[1]+ sleep 700 &`

    This display indicates that this process in the background is **running in the background** (indicated by the ampersand character "**&**"). Now this command has resume pausing until **700 seconds**.

11. Issue the following Linux command:
    `fg`

    You should notice that the *sleep* command is now running in the **foreground**.

12. Press the **key combination** to **terminate** the process running in the **foreground**:
    `ctrl-c`

    You can issue Linux commands with ampersand "**&**" in your terminal to **run** processes automatically in the **background** without having to issue *ctrl-z* and *bg* short-cut keys.

```
[ murray.saul ] sleep 500 & sleep 600 & sleep 700 &
[1] 90875
[2] 90876
[3] 90877
[ murray.saul ]
[ murray.saul ] jobs
[1]   Running                 sleep 500 &
[2]-  Running                 sleep 600 &
[3]+  Running                 sleep 700 &
[ murray.saul ]
```

Using the **ampersand** character **&** to run a **series of processes** in the background.

13. Issue the following Linux commands:
    `sleep 500 & sleep 600 & sleep 700 &`

```
[1] 114191
[2] 114192
[3] 114193
```

14. Issue the **jobs** command. What do you notice?

```
[1]   Running          sleep 500 &
[2]-  Running          sleep 600 &          second recent
[3]+  Running          sleep 700 &          most recent
```

In the jobs command output, jobs that display a plus sign (**+**) indicates the **most recent** process
placed in to the background, and a minus sign (**-**) indicates the **second most recent** process
placed into the background.

The **kill** command issued to terminate processes that are running in the **foreground** or **background**.
Issuing the kill command without options would send the **SIGTERM** signal (eg. *signal terminate* - which is signal **#15**).

15. Issue the following Linux command to **terminate** the **first** job running in the background:
    ```
    kill %1
    ```
    NOTE: You can specify job number preceded by percent % with the **kill**, **bg**, and **fg** commands to specify the processes' job number.

```
[ murray.saul ] jobs
[1]   Running                 sleep 500 &
[2]-  Running                 sleep 600 &
[3]+  Running                 sleep 700 &
[ murray.saul ]
[ murray.saul ] kill %1
[ murray.saul ]
[1]   Terminated              sleep 500
[ murray.saul ]
[ murray.saul ] jobs
[2]-  Running                 sleep 600 &
[3]+  Running                 sleep 700 &
```
Using the **kill %1** command to **terminate** job #1.

16. Issue the **jobs** command. What do you notice?
    [1] Terminated sleep 500
    [2]- Running sleep 600 &
    [3]+ Running sleep 700 &

    After a few seconds:
    [2]- Running sleep 600 &
    [3]+ Running sleep 700 &
17. Issue the following Linux commands:
    ```
    kill %2
    kill %3
    ```
    ```
    [2]-  Terminated              sleep 600
    [3]+  Terminated              sleep 700
    ```
18. Issue the **jobs** command (you may have to issue the *jobs* command several times to get final result).
    What do you notice?
    nothing
19. Let's use **grouping** to run several commands in sequence within a single process.

```
[ murray.saul ] (sleep 400; sleep 500; sleep 600) &
[1] 91611
[ murray.saul ]
[ murray.saul ] jobs
[1]+  Running                 ( sleep 400; sleep 500; sleep 600 ) &
```
Using round brackets to **group** a series of commands to be run as **one process**.

20. Issue the following Linux command:
    ```
    (sleep 400; sleep 500; sleep 600) &
    ```
    ```
    [1] 115770
    ```
21. Issue the **jobs** command. What do you notice?
    You should notice all commands are run in a group as just one process.
    [1]+ Running (sleep 400; sleep 500; sleep 600) &
22. Issue the following Linux command to terminate the first job running in the **background**:
    ```
    kill %1
    ```

    NOTE: If issuing the kill command does not work, then you would need to send a STRONGER signal
    to "**kill**" (not "*SIGTERM* - which is signal *#15*") the process. The **SIGKILL** signal (signal **#9**)

would be required to do this by issuing the **kill** command with the option: **-9**.

23. Issue the **jobs** command and make certain there are no processes that are running in the **background**.
    [1]+ Terminated (sleep 400; sleep 500; sleep 600)
    You can also manipulate processes by their **PID** (**process ID**). Let's terminate our Matrix Bash shell process
    by using the **kill** command using that processes' **PID**.

24. Issue the following Linux command: **ps**

25. Note in the **ps** command output the PID of the process called **bash**.

    You will be using that PID when issuing the next Linux command.

26. Issue the following Linux command (using the bash processes' PID number instead of "PID"):
    kill PID
    kill 96881
    What did you notice?
    Connection to matrix.senecacollege.ca closed.
    FYI: If the command did NOT work, issue the following Linux command (using the bash processes' PID number instead of "PID"):
    kill -9 PID

In the next investigation, you will learn how to **create aliases** and **view command history** on your Matrix server.

INVESTIGATION 3: ALIASES / COMMAND HISTORY

In this investigation, you will learn how to **manage aliases** and **Linux command history** on your Matrix account.

Perform the Following Steps:

1. Make certain that you are logged into your Matrix account.

2. Issue a Linux command to confirm that you are located in your **home** directory.

3. Issue the following Linux command:
   alias | more
   Observe those existing aliases that have previously been declared. Take a few moments to run those aliases to see what happens.

```
[ murray.saul ] alias
alias ..='cd ..'
alias ...='cd ../..'
alias cd..='cd ..'
alias cls='clear'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias lal='ls -al'
alias lh='ls --human-readable --size -1 -S --classify'
```

Issuing the **alias** command (without arguments) will display a list of existing aliases on your Unix / Linux system.

```
alias ..='cd ..'
alias ...='cd ../..'
alias cd..='cd ..'
alias cls='clear'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias mc='. /usr/libexec/mc/mc-wrapper.sh'
alias rvm-restart='rvm_reload_flag=1 source '\''/usr/local/rvm/scripts/rvm'\'''
alias vi='vim'

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

4. Issue the following to create an alias: `alias lal='ls -al'`

5. Issue the following alias: `lal`

   What do you notice? issue lal just like issuing ls -al

6. Issue the following to create another alias (lowercase l and h):
   `alias lh='ls --human-readable --size -1 -S --classify'`

7. Issue the following command to confirm that this newly-created alias is stored in memory:

   ```
   alias | grep "lh"
   ```
   ```
   alias lh='ls --human-readable --size -1 -S --classify'
   ```

8. Issue the following alias: `lh` ls -h -s -l -S --classify
   -h, --human-readable with -l: print sizes in human readable format (e.g., 1K 234M 2G)
   -s, --size : print the allocated size of each file, in blocks -S: sort by file size
   What do you think this command does?

9. **Logout** of your Matrix account and then **login** to your Matrix account.

10. Reissue the **lal** alias. What happened?
    -bash: lal: command not found
11. Reissue the **lh** alias. What happened?
    -bash: lh: command not found
12. Issue the **alias | grep lh** command without any arguments to see if it is stored in memory.
    nothing
13. Reissue the command to create the **lh** alias in **step #6**.

14. Run the **lh** alias to confirm that it is properly set in memory.

15. Issue the following Linux command to edit your ~/.bashrc startup file:
    `nano ~/.bashrc`

16. Add the following line at the **bottom** of this file:
    `alias lh='ls --human-readable --size -1 -S --classify'`

17. Save your editing changes and exit your text editor.

18. **Logout** of your Matrix account, then **login** to your Matrix account.

19. Reissue the **lh** alias. What happened?
    lh alias still set in memory and work
20. Issue the following Linux command: `unalias lh`

21. Run the **lh** alias to see what happens.

    What happenned? -bash: lh: command not found

22. **Logout** of your Matrix account, then **login** to your Matrix account.

23. Reissue the **lh** alias. What happened? Why?

    lh alias still set in memory and work

24. Reissue the **lal** alias. Why **didn't** this alias work?

    It is because lal alias doesn't insert at the bottom of ~/.bashrc

    The checking script below is designed to act as a **filter** with a **pipeline command**.
    This will allow to check if your **lh** alias exists when it is checked in this program.

25. Issue the following Linux pipeline command:

    ```
    alias | ~uli101/week8-check-3
    ```

    If you encounter errors, then view the feedback to make corrections, and then re-run the checking script.
    If you receive a congratulation message that there are no errors, then proceed with this tutorial.

    We will complete this investigation by learning to execute p**reviously issued commands** by using **command history**.

26. Issue the following Linux command: **`history | grep "lh"`**

```
417  2023-03-17 14:52:37 alias lh='ls --human-readable --size -l -S --classify'
418  2023-03-17 14:52:48 alias | grep "lh"
420  2023-03-17 14:54:02 lh
428  2023-03-17 15:06:56 lh
429  2023-03-17 15:07:32 alias | grep lh
430  2023-03-17 15:08:02 alias | grep "lh"
431  2023-03-17 15:08:37 alias lh='ls --human-readable --size -1 -S --classify'
432  2023-03-17 15:08:48 lh
433  2023-03-17 15:09:16 alias lh='ls --human-readable --size -l -S --classify'
434  2023-03-17 15:09:19 lh
437  2023-03-17 15:12:16 lh
438  2023-03-17 15:13:12 unalias lh
439  2023-03-17 15:13:16 lh
441  2023-03-17 15:15:27 lh
444  2023-03-17 15:20:23 history | grep "lh"
```

    What do you notice?

27. Type an exclamation mark **!** followed by the number by one of those commands listed in the **history list** and press **`ENTER`**

    What happened? !441: command 441 be executed

28. Type the following: **`!unalias`** and press **`ENTER`**

    What happened? unalias lh

    -bash: lh: command not found

29. Issue the following Linux command: **`history | grep "lh"`**

```
417  2023-03-17 14:52:37 alias lh='ls --human-readable --size -l -S --classify'
418  2023-03-17 14:52:48 alias | grep "lh"
420  2023-03-17 14:54:02 lh
428  2023-03-17 15:06:56 lh
429  2023-03-17 15:07:32 alias | grep lh
430  2023-03-17 15:08:02 alias | grep "lh"
431  2023-03-17 15:08:37 alias lh='ls --human-readable --size -1 -S --classify'
432  2023-03-17 15:08:48 lh
433  2023-03-17 15:09:16 alias lh='ls --human-readable --size -l -S --classify'
434  2023-03-17 15:09:19 lh
437  2023-03-17 15:12:16 lh
438  2023-03-17 15:13:12 unalias lh
439  2023-03-17 15:13:16 lh
441  2023-03-17 15:15:27 lh
444  2023-03-17 15:20:23 history | grep "lh"
445  2023-03-17 15:22:18 lh
446  2023-03-17 15:23:36 unalias lh
447  2023-03-17 15:23:53 lh
448  2023-03-17 15:24:06 history | grep "lh"
```

    What happened?

LINUX PRACTICE QUESTIONS

The purpose of this section is to obtain **extra practice** to help with **quizzes**, your **midterm**, and your **final exam**.

Here is a link to the MS Word Document of ALL of the questions displayed below but with extra room to answer on the document to simulate a quiz:

https://github.com/ULI101/labs/raw/main/uli101_week8_practice.docx

Your instructor may take-up these questions during class. It is up to the student to attend classes in order to obtain the answers to the following questions. Your instructor will NOT provide these answers in any other form (eg. e-mail, etc).

Review Questions:

1. Hard Links:
    a. What is the purpose of creating a hard-link?
    b. What is a limitation of a hard link?
    c. Write a single Linux command to create a hard link called **~/backup/myfile.txt.lnk** for the existing file called **~/myfile.txt**
    d. Write a single Linux command to display the **i-node** number for both files. Are the **i-node** numbers identical?

2. Symbolic (Soft) Links:
    a. What is the purpose of creating a symbolic (soft) link?
    b. What is a limitation of a symbolic (soft) link?
    c. Write a single Linux command to create a symbolic link called **~/shortcuts/murray.saul.lnk** to the existing directory called **~murray.saul**
    d. Are the i-node numbers identical for both of those files?
    e. What data is contained in the file called **~/shortcuts/murray.saul.lnk**?

3. Background / Foreground Processes:
    a. Write a single Linux command to run the program called **~/clean.sh** in the **background**.
    b. Write a single Linux command to place the previously issued program in the **foreground**.
    c. Write a single Linux command to **confirm** that this program is running in the background.
    d. What **key-combination** would you issue to send that program again into the **background**?
    e. Write a single Linux command to have that process sent into the background to **continue running**?

4. Managing Background processes:
   Use the following diagram to answer the accompanying questions.
   Each of the following questions will use the diagram below and are treated as independent situations.

```
[1]  Stopped vim a
[2]- Stopped vim b
[3]+ Stopped vim c
```

    a. Write a single Linux command to bring the second-recently process placed in the background into the **foreground**.
    b. Write a single Linux command to **terminate job #3**.

5. Write a single Linux command to display running processes in “real-time”.
6. Write a single Linux command to terminate a process that has the following PID: **22384**

7. Aliases / History:
    a. Write a linux command to create an **alias** called **ld** that issues the command: **ls -ld**

   b. Write a linux command to unset the **alias** created in the previous question.
   c. Issue a Linux command to list **history** of commands that match the pattern called **touch**.

8. Create a **table** listing each Linux command, useful options and command purpose for the following Linux commands:
   ln , **ps** , **top** , **fg** , **bg** , **jobs** , **kill** , **alias** , **unalias** , **history**

Author: Murray Saul

Category: ULI101