Unix Scripting

Week11-Session1

Agenda

- Named Pipes

Pipes in Unix

- Pipes
  - `ls  | grep x`
    - Bash and other shells run both commands, connecting the output of the first to the input of the second.
  - The above is an example of an "**unnamed pipe**".
    - The pipe exists only inside the kernel and cannot be accessed by processes that created it, in this case, the bash shell.
  - The other sort of pipe is a "**named**" pipe
- In computing, a **named pipe** is an extension to the traditional pipe concept on Unix and Unix-like systems, and is one of the methods of **inter-process communication**.

Named Pipes

- a **named pipe** is a special file that can be used even over multiple shell sessions.
- It is a special file that follows the FIFO (first in, first out) mechanism.
  - It can be used just like a normal file; i.e., you can write to it, read from it, and open or close it.
- To create a named pipe, the command is:

```
mkfifo pipe-name
```

How name-pipe works

- Create a named pipe

```
mkfifo mypipe
```

- Run a command and redirect the output to the pipe

```
echo “hello” > mypipe &
```

- Read from the pipe and display it

```
cat mypipe
```

- What happen?

Working with named pipe

- Once the pipe has been read or "drained," it's empty, though it still will be visible as an empty file ready to be used again.
- Named pipe content resides in memory rather than being written to disk.
  - It is passed only when both ends of the pipe have been opened.
  - And you can write to a pipe multiple times before it is opened at the other end and read.
- By using named pipes, you can establish a process in which one process writes to a pipe and another reads from a pipe

Check the named pipe

- Use `ls -l pipe_name`

```
[shahdad.shariatmadar@mtrx-node06pd ~]$ ls -l mypipe
prw-rw-rw- 1 shahdad.shariatmadar users 0 Jul 28 08:28 mypipe
```

- Notice the size of the named pipe is zero and it has a designation of **"p"**.
- You can remove pipe using “**rm**” command!

mkfifo or mknod

- Named pipes are created via **mkfifo** or **mknod**
  - **`mkfifo`** `/tmp/testpipe`
  - **`mknod`** `/tmp/testpipe p`

When to use named pipe

- Named-Pipe is a special file in file-system in which multiple processes can access this special file for reading and writing like any ordinary file.
- Pipes allow separate processes to communicate without having been designed explicitly to work together.
- If we only need a name as a reference, with content that comes directly from another process, and we don’t want to store data on disk, then named-pipe is our best choice!

An IPC use case

- Using **named pipes** as a message queue, where a “writer” process sends messages into a named pipe, which are taken off at the other side by a “reader” process asynchronously.

An IPC use case

- Using a named pipe, you can start the backup and the shutdown **cron jobs** at the same time and have the shutdown just wait till the backup writes to the named pipe.
  - Cron is a job scheduling utility present in Unix like systems. Cron jobs help us automate our routine tasks, whether they're hourly, daily, monthly, or yearly.
- When the shutdown job reads something from the pipe, it then pauses for a few minutes so the **cron e-mail** can go out, and then it shuts down the system.

Named pipe advantages

- you don't have to start the reading/writing processes at the same time
- you can have multiple readers/writers which *do not need common ancestry*
- as a file you can control ownership and permissions

Process Substitution

- Piping the stdout of a command into the stdin of another is a powerful technique. But, what if you need to pipe the stdout of multiple commands?
  - This is where **process substitution** comes in.
- **Process substitution** feeds the output of a process (or processes) into the stdin of another process.
  - Here is how to use it:
    - >(command_list)
    - <(command_list)
      - Note: There is *no* space between the the "<" or ">" and the parentheses. Space there would give an error message.

Activity

- Consider the following example in BB:

```
cat cars | tee >(awk '/ford/
{total+=$5} END {print "Total
fords: " total}') \
>(awk '/chevy/ {total+=$5} END
{print "Total chevys: " total}')
```

  - Demonstrate how it works
  - Using your own work, explain your observation on how this code works
    - Role of tee?, role of “process substitution”?

More fun with named pipe

- Some interesting links:
  - http://hassansin.github.io/fun-with-unix-named-pipes
  - https://www.youtube.com/watch?v=6lik_f1Vp54