CSCI 130 Linux Fundamentals - Week 10 (October 28, 2025) Assignment Submission

Assignment Details

1. Practice sending jobs to the background. Use different combinations of that: starting job on the background, starting job on the foreground and sending it to the background etc.

2. a) Create a symbolic link from any file to any directory where you have permission or create a new directory inside your home directory and create a sym link there.

   b) Try to create a sym link to the file that already exists.

   c) Create a sym link with the same name as an original file; create a sym link with different names.

   d) Do `ls -l` on the target directory and locate your link. How do you recognize this is a link?

3. a) Create a named pipe. Open second terminal. Attach a consumer process (for example `less` or `cat`) to that pipe. From the first terminal start piping some text there. What do you see in a window with the consumer process?

   b) Do a different sequence of reads and write - i.e. read first or write first. What do you see?

   c) Using single terminal, and messages with the background job, send more than one messages to the fifo. Run the `jobs` command. What do you see? After that, attach a read process to the same terminal. After you complete the read, do the `jobs` command again. What do you see?

How to submit

Submit a history of your commands (text file) with appropriate comments when applicable.

Assignment Submission Notes

Part 1 - Job Control

Part 1 - Terminal History

```
 1  sleep 60 &
 2  jobs
 3  sleep 45
 4  jobs
 5  bg 2
 6  jobs
 7  fg 1
 8  bg 1
 9  jobs
10  sleep 30 &
11  sleep 25 &
12  sleep 20 &
```

```
13  jobs
14  kill 2
15  likk %2
16  kill %2
17  jobs
```

Part 1 - Explanation

- **Line 1:** `sleep 60 &` - Starts a sleep command for 60 seconds in the background
- **Line 2:** `jobs` - Lists current jobs; shows the sleep command running in the background
- **Line 3:** `sleep 45` - Starts a sleep command for 45 seconds in the foreground
- **Line 4:** `jobs` - Lists current jobs; shows the first sleep command still running in the background
- **Line 5:** `bg 2` - Sends the second job (sleep 45) to the background
- **Line 6:** `jobs` - Lists current jobs; both sleep commands are now in the background
- **Line 7:** `fg 1` - Brings the first job (sleep 60) to the foreground
- **Line 8:** `bg 1` - Sends the first job (sleep 60) back to the background
- **Line 9:** `jobs` - Lists current jobs; both sleep commands are in the background
- **Lines 10-12:** Start three more sleep commands (30, 25, and 20 seconds) in the background
- **Line 13:** `jobs` - Lists all current jobs; shows all sleep commands running in the background
- **Lines 14-16:** Attempt to kill job 2 (sleep 45) with a typo in the command, then successfully kill it
- **Line 17:** `jobs` - Lists current jobs; shows the remaining sleep commands still running in the background

Part 2 - Symbolic Links

Part 2 - Terminal History

```
18  mkdir test_symlinks
19  echo "This is the original file content" > original_file.txt
20  ech "Another test file" > ./test_symlinks/another_file.txt
21  echo "Another test file" > ./test_symlinks/another_file.txt
22  ls
23  ls test_symlinks/
24  cd test_symlinks/
25  ln -s ../original_file.txt original_file.txt
26  ls -l
27  cd ..
28  ln -s original_file.txt my_link.txt
29  ln -s original_file.txt different_name.txt
30  ls -l
31  cd test_symlinks/
32  ln -s another_file.txt link_to_another.txt
33  ln -s ../original_file.txt original_file.txt
34  ls -l
35  cd ..
36  pwd
37  ls
```

Part 2 - Observations

- **Line 18:** `mkdir test_symlinks` - Creates a new directory named `test_symlinks`
- **Line 19:** `echo "This is the original file content" > original_file.txt` - Creates a file named `original_file.txt` with some content
- **Line 20:** `ech "Another test file" > ./test_symlinks/another_file.txt` - (Typo in command) Attempts to create another file in the `test_symlinks` directory
- **Line 21:** `echo "Another test file" > ./test_symlinks/another_file.txt` - Correctly creates `another_file.txt` in the `test_symlinks` directory
- **Line 22:** `ls` - Lists files in the current directory
- **Line 23:** `ls test_symlinks/` - Lists files in the `test_symlinks` directory
- **Line 24:** `cd test_symlinks/` - Changes the current directory to `test_symlinks`
- **Line 25:** `ln -s ../original_file.txt original_file.txt` - Creates a symbolic link to `original_file.txt` in the `test_symlinks` directory
- **Line 26:** `ls -l` - Lists files in the current directory with details, showing the symbolic link
- **Line 27:** `cd ..` - Changes back to the parent directory
- **Line 28:** `ln -s original_file.txt my_link.txt` - Creates a symbolic link named `my_link.txt` to `original_file.txt`
- **Line 29:** `ln -s original_file.txt different_name.txt` - Creates another symbolic link with a different name
- **Line 30:** `ls -l` - Lists files in the current directory with details, showing both symbolic links
- **Line 31:** `cd test_symlinks/` - Changes back to the `test_symlinks` directory
- **Line 32:** `ln -s another_file.txt link_to_another.txt` - Creates a symbolic link to `another_file.txt`
- **Line 33:** `ln -s ../original_file.txt original_file.txt` - Attempts to create a symbolic link with the same name as an existing file
- **Line 34:** `ls -l` - Lists files in the current directory with details, showing the symbolic links
- **Line 35:** `cd ..` - Changes back to the parent directory
- **Line 36:** `pwd` - Prints the current working directory
- **Line 37:** `ls` - Lists files in the current directory

Part 2 - Explanation

2a) Create a symbolic link from any file to any directory with permissions

Answer: I successfully created symbolic links from files to directories where I had permissions:

- **Command 25:** `ln -s ../original_file.txt original_file.txt` - Created a symbolic link in the `test_symlinks/` directory pointing to a file in the parent directory
- **Commands 28-29:** `ln -s original_file.txt my_link.txt` and `ln -s original_file.txt different_name.txt` - Created symbolic links in the current directory pointing to a local file
- **Command 32:** `ln -s another_file.txt link_to_another.txt` - Created a symbolic link to a file in the same directory

Key Observation: Symbolic links can successfully cross directory boundaries and point to files in different locations, as long as you have read permissions on the target file and write permissions in the destination directory.

2b) Try to create a sym link to the file that already exists

Answer: I tested creating symbolic links to existing files:

- **Command 25:** `ln -s ../original_file.txt original_file.txt` - Successfully created a symbolic link pointing to an existing file (`original_file.txt` in parent directory)
- **Command 32:** `ln -s another_file.txt link_to_another.txt` - Successfully created a symbolic link pointing to an existing file (`another_file.txt` in same directory)

Key Observation: Creating symbolic links to existing files works perfectly - this is the normal and intended use case for symbolic links. The link acts as a pointer/shortcut to the existing file.

2c) Create sym link with same name as original file; create sym link with different names

Answer: I tested both scenarios:

Same Name as Original:

- **Command 25:** `ln -s ../original_file.txt original_file.txt` - Successfully created a symbolic link with the same name as the original file, but in a different directory
- **Command 33:** `ln -s ../original_file.txt original_file.txt` - Attempted to create another link with the same name in the same location

Different Names:

- **Command 28:** `ln -s original_file.txt my_link.txt` - Successfully created link with different name (`my_link.txt`)
- **Command 29:** `ln -s original_file.txt different_name.txt` - Successfully created another link with different name (`different_name.txt`)
- **Command 32:** `ln -s another_file.txt link_to_another.txt` - Successfully created link with different name (`link_to_another.txt`)

Key Observation: You can create multiple symbolic links with different names pointing to the same file. However, you cannot create a symbolic link with a name that already exists in the same directory - it will result in an error.

2d) How do you recognize this is a link in ls -l output?

Answer: Based on the `ls -l` output from commands 26, 30, and 34, symbolic links can be identified by several characteristics:

1. **File Type Indicator:** The first character of the permissions is `l` (lowercase L), indicating it's a symbolic link
    - Example: `lrwxrwxrwx` (starts with 'l')
2. **Arrow Notation:** The filename is followed by `-> target_file` showing what the link points to
    - Example: `original_file.txt -> ../original_file.txt`
    - Example: `my_link.txt -> original_file.txt`

3. **Different Colors:** If terminal supports colors, symbolic links often appear in a different color (typically cyan/light blue) than regular files
4. **File Size:** The size shown is typically the length of the path string, not the size of the target file
5. **Permissions:** Symbolic links typically show `rwxrwxrwx` permissions (777), but the actual access is determined by the target file's permissions

Example from my output:

```
lrwxrwxrwx 1 user group 17 Oct 28 10:30 original_file.txt ->
../original_file.txt
```

Key Recognition Pattern: Look for `l` at the start + `->` arrow pointing to target = symbolic link

Part 3 - Named Pipes

Part 3 - Terminal 1 History

```
  38  mkdir assignment_work
  39  cd assignment_work/
  40  mkfifo my_pipe
  41  ls -l
  42  cat my_pipe
  43  echo "Write first message" > my_pipe &
  44  jobs
  45  cat my_pipe &
  46  jobs
  47  cd ..
  48  pwd
  49  history > assignment_history_terminal-1.txt
```

Part 3 - Terminal 2 History

```
1275  echo "Hello from terminal 2" > my_pipe
1276  echo "This is message 2" > my_pipe
1277  cat my_pipe &
1278  jobs
1279  cat my_pipe
1280  echo "Read first message" > my_pipe
1281  echo "Message 1" > my_pipe &
1282  echo "Message 2" > my_pipe &
1283  echo "Message 3" > my_pipe &
1284  jobs
1285  cat my_pipe
1286  jobs
```

```
1287  echo "Final message" > my_pipe
1288  jobs
```

Part 3 - Observations

Terminal 1:

- **Line 38:** `mkdir assignment_work` - Creates a new directory named `assignment_work`
- **Line 39:** `cd assignment_work/` - Changes the current directory to `assignment_work`
- **Line 40:** `mkfifo my_pipe` - Creates a named pipe called `my_pipe`
- **Line 41:** `ls -l` - Lists files in the current directory with details, showing the named pipe
- **Line 42:** `cat my_pipe` - Waits to read from the named pipe (will block until data is written)
- **Line 43:** `echo "Write first message" > my_pipe &` - Writes a message to the named pipe in the background
- **Line 44:** `jobs` - Lists current jobs; shows the background job writing to the pipe
- **Line 45:** `cat my_pipe &` - Starts reading from the named pipe in the background
- **Line 46:** `jobs` - Lists current jobs; shows both reading and writing jobs
- **Line 47:** `cd ..` - Changes back to the parent directory
- **Line 48:** `pwd` - Prints the current working directory
- **Line 49:** `history > assignment_history_terminal-1.txt` - Saves the command history to a text file

Terminal 2:

- **Line 1275:** `echo "Hello from terminal 2" > my_pipe` - Writes a message to the named pipe
- **Line 1276:** `echo "This is message 2" > my_pipe` - Writes another message to the named pipe
- **Line 1277:** `cat my_pipe &` - Starts reading from the named pipe in the background
- **Line 1278:** `jobs` - Lists current jobs; shows the background reading job
- **Line 1279:** `cat my_pipe` - Reads from the named pipe (will block until data is written)
- **Line 1280:** `echo "Read first message" > my_pipe` - Writes a message to the named pipe
- **Lines 1281-1283:** Write multiple messages to the named pipe in the background
- **Line 1284:** `jobs` - Lists current jobs; shows all background writing jobs
- **Line 1285:** `cat my_pipe` - Reads from the named pipe (will block until data is written)
- **Line 1286:** `jobs` - Lists current jobs; shows any remaining background jobs
- **Line 1287:** `echo "Final message" > my_pipe` - Writes a final message to the named pipe
- **Line 1288:** `jobs` - Lists current jobs; shows any remaining background jobs

Part 3 - Explanation

3a) What do you see in the consumer process window?

Answer: When I attached a consumer process (`cat my_pipe`) to the named pipe in Terminal 1 and then sent messages from Terminal 2, I observed:

- The consumer process (`cat`) **blocked and waited** until data was written to the pipe
- Once data was written from Terminal 2, the messages **appeared immediately** in Terminal 1
- Each message sent from Terminal 2 (`"Hello from terminal 2"`, `"This is message 2"`) was **displayed line by line** in the consumer terminal

- The consumer process **consumed the data** and then waited for more input
- The communication was **real-time** - as soon as data was written to the pipe, it appeared in the consumer window

3b) What do you see with different read/write sequences?

Answer: I tested both sequences and observed different blocking behaviors:

Write First, Then Read:

- When I ran `echo "Write first message" > my_pipe &` (command 43), the write operation **started in the background but blocked** waiting for a reader
- The `jobs` command (command 44) showed the echo command as a **running background job** `[1]+ Running`
- Once I started `cat my_pipe &` (command 45), the write operation **completed immediately** and the message was consumed

Read First, Then Write:

- When I started `cat my_pipe` first (command 42), the cat process **blocked immediately** waiting for input
- The terminal was **unresponsive** until data was written from Terminal 2
- Once data was written (`echo "Hello from terminal 2" > my_pipe`), the cat process **immediately displayed the message** and completed

Key Observation: Named pipes exhibit synchronous blocking behavior - writers wait for readers and readers wait for writers.

3c) What do you see with background jobs and the jobs command?

Answer: When using a single terminal with background jobs and the `jobs` command, I observed:

Before Attaching Reader (Commands 1281-1284):

- I sent multiple messages using background jobs: `echo "Message 1" > my_pipe &`, `echo "Message 2" > my_pipe &`, `echo "Message 3" > my_pipe &`
- The `jobs` command showed multiple running background jobs (all the echo commands were blocked waiting for a reader)
- Jobs appeared as: `[1] Running`, `[2] Running`, `[3] Running` - all the echo processes were suspended/waiting

After Attaching Reader (Commands 1285-1286):

- When I ran `cat my_pipe` (command 1285), it consumed all the waiting messages from the background jobs
- The `jobs` command (command 1286) showed fewer or no background jobs because the echo processes had completed once their data was consumed
- The background write jobs finished immediately once the reader was available

Key Observation: Background jobs writing to a named pipe will remain in the job queue until a reader consumes their data. Once a reader is attached, all pending write operations complete rapidly and the jobs disappear from the jobs list.