Section 1: Unix Fundamentals

Imagine you're working on a remote server where you only have access to the command line. No fancy graphical user interface, just you and the terminal. Let's learn some essential commands to navigate and manipulate files. You'll have to SSH into Igor for this one. Open a terminal and type ssh your_username@doc.gold.ac.uk then press return. It will prompt you for your password and then ask you about a fingerprint. Press 'Y' here and you're in your own personal Unix shell!

1.1 File Creation with Nano

- Open the terminal or navigate to Igor using SSH.
- Use `nano` to create a new file: `nano my_first_file.txt`
- Type some text into the file.
- Press `Ctrl + X` to exit.
- Press `Y` to save the changes.
- Press `Enter` to confirm the file name.

1.2 Essential Commands

- `ls`: List files and directories. Try `ls -l` for more detailed information.
- `cd`: Change directory. For example, `cd Documents` moves you to the Documents directory.
- `pwd`: Print the current working directory. If you get lost this is helpful.
- `mkdir`: Create a new directory. For example, `mkdir new_directory`.
- `rm`: Remove files or directories. **Be careful!** `rm -rf` is powerful and can delete entire directories.
- `cp`: Copy files or directories. For example, `cp file1.txt file2.txt`.
- `mv`: Move or rename files or directories. For example, `mv file.txt Documents/`.

1.3 Input/Output Redirection and Piping

- Create two files, `errors1.txt` and `errors2.txt`, and add a few lines with the word "error" in them.
- Use `cat` to concatenate the files and pipe the output to `grep` to count the occurrences of "error":

Bash
```
cat errors1.txt errors2.txt | grep "error" | wc -l
```
Use code with caution.

1.4 Finding Files

- Use `find` to locate all `.js` files modified in the last week:

Bash
```
find . -name "*.js" -mtime -7
```

Tip: In situations where you don't have version control like Git, commands like `find` become crucial for tracking down specific file versions.

1.5 Text Manipulation

- **grep:** Search for patterns in files. For example, `grep "function" code.js` finds lines containing "function" in the file `code.js`.
- **awk:** Extract data from files. For example, `awk '{print $1}' data.txt` prints the first column of `data.txt`.
- **sed:** Perform text transformations. For example, `sed 's/old/new/g' file.txt` replaces all occurrences of "old" with "new" in `file.txt`. Look familiar? If you've used regex it should!

1.6 Process Management

- `ps`: List running processes.
- `kill`: Terminate a process by its ID. For example, `kill 1234`.

Tip: Use `ps aux | grep "process_name"` to find the process ID of a specific process.

Section 2: Code Optimization and Quality

2.1 Code Formatting

- **Prettier (JavaScript):**
  - Install: `npm install -g prettier`
  - Format a file: `prettier --write my_code.js`
  - Npm can be a bit buggy on the lab machines. Several web-based alternatives exist e.g. https://prettier.io/playground/
- **Black (Python):**
  - Install: `pip install black`
  - Format a file: `black my_code.py`

Task: Take some unformatted JavaScript and Python code and use Prettier and Black to automatically format it. Observe the changes and how it improves readability.

2.2 Linting

- **ESLint (JavaScript):**
  - Install: `npm install -g eslint`
  - Lint a file: `eslint my_code.js`
- **Pylint (Python):**
  - Install: `pip install pylint`
  - Lint a file: `pylint my_code.py`

Task: Use ESLint and Pylint to analyze some JavaScript and Python code. Identify the issues they highlight and understand how to fix them. There are formal guidelines for languages such as these. For Javascript it follows the ECMAScript standard. For Python we have the PEP guidelines which quite cleverly state 'A Foolish Consistency is the Hobgoblin of Little Minds.' It's good to follow styling, but remember we're humans not computers. Sometimes common sense overrides robotic logic!

Section 3: Debugging with OnlineGDB

3.1 Online Code Formatting

- Go to https://www.onlinegdb.com/.
- Paste some code you wrote in a previous lab.
- Use the "Beauty" button to automatically format the code.

3.2 Debugging Tutorial

- Read the OnlineGDB debugger tutorial: https://www.onlinegdb.com/blog/brief-guide-on-how-to-use-onlinegdb-debugger/
- Experiment with setting breakpoints and stepping through the code using the "Step Over," "Step Into," and "Step Out" functions. This kind of tooling is very helpful for finding dodgy intervention points in your code.

Advanced Challenge (Open Ended)

You are given a large log file (`server.log`) from a web server. Your task is to:

1. Use `grep` to find all error messages.
2. Use `awk` to extract specific information from the error messages (e.g., timestamp, error type).
3. Use `sort` and `uniq` to identify the most frequent errors.
4. Write a shell script to automate this log analysis process.

This challenge will test your ability to combine different Unix commands to effectively analyze and troubleshoot a real-world scenario. Good luck! Here’s some sample log file that I have generated for you that conforms to rules we specified.

2024-12-05 10:00:00 INFO Server started successfully. 2024-12-05 10:01:15 WARN User 'guest' failed login attempt. 2024-12-05 10:02:30 INFO User 'admin' logged in. 2024-12-05 10:05:00 ERROR Database connection failed: Timeout error. 2024-12-05 10:06:45 INFO User 'user123' logged in. 2024-12-05 10:08:00 WARN Low disk space on partition /dev/sda1. 2024-12-05 10:10:10 ERROR Database connection failed: Timeout error. 2024-12-05 10:12:00 INFO Server statistics: CPU usage 20%, Memory usage 60%. 2024-12-05 10:15:35 ERROR Failed to process request from IP 192.168.1.100: Invalid request format. 2024-12-05 10:17:00 WARN User 'guest' failed login attempt. 2024-12-05 10:20:00 ERROR Database connection failed: Timeout error. 2024-12-05 10:22:15 INFO User 'admin' logged out. 2024-12-05 10:25:00 WARN High network traffic detected. 2024-12-05 10:30:00 INFO Server running smoothly.

More Tips:

- Use `grep "ERROR" server.log` to find all error messages.
- Use `awk '{print $1, $2, $5}' server.log` to extract the timestamp, log level, and error message.
- Use `sort` and `uniq -c` to count the occurrences of each unique error message.
- Consider using `grep`, `cut`, `sort`, and `uniq` in a pipeline to automate the analysis.