Exercise session 01

The UNIX shell.

Advanced Programming - SISSA, UniTS, 2024-2025

Giuseppe Alessio D'Inverno

01 Oct 2024

Exercise 1: basic Bash commands

Perform the following tasks in your command-line terminal.

1. Navigate to your home folder.
2. Create a folder named `test1`.
3. Navigate to `test1` and create a new directory `test2`.
4. Navigate to `test2` and go up one directory.
5. Create the following files: `f1.txt`, `f2.txt`, `f3.dat`, `f4.md`, `README.md`, `.hidden`.
6. List all files (including hidden ones).
7. List only `.txt` files.
8. Move `README.md` to folder `test2`.
9. Move all `.txt` files to `test2` in one command.
10. Remove `f3.dat`.
11. Remove all contents of `test1` and the folder itself in one command.

Exercise 2: dataset exploration

You can access an open dataset of logs collected from a high-performance computing cluster at the Los Alamos National Laboratories. The dataset is available on this webpage.

To download the dataset using `wget`, run the following command:

```
wget https://raw.githubusercontent.com/logpai/loghub/master/HPC/HPC_2k.log_structured.csv
```

After downloading the dataset, perform the following analyses using only Bash commands.

1. Find out how many unique node names are present in the dataset.
2. Export the list from the previous point to a file named `nodes.log`
3. Determine the number of times the "unavailable" event (E13) has been reported.
4. Identify the number of unique nodes that have reported either event E32 or event E33.
5. Calculate how many times the node "gige7" has reported a critical event (E15).
6. Find out how many times the "node-2" node has been reported in the logs.

Exercise 3: creating a backup script

In this exercise, you'll create a Bash script that automates the process of creating a backup of a specified directory. The script should accomplish the following tasks:

1. Receive the directory to backup as an input argument.
2. Create a timestamped backup folder inside a specified backup directory.
3. Copy all files and directories from the user-specified directory to the backup folder.
4. Compress the backup folder into a single archive file.

Note: You can use basic commands like `read`, `mkdir`, `cp`, `tar`, and `echo`.

Hint: Generate a timestamp in the format *YYYYMMDD_hhmmss* with `date +%Y%m%d_%H%M%S`.

Exercise 3: creating a backup script. Instructions

1. Create a new Bash script file named `backup.sh`.
2. Inside the script, use basic Bash commands to implement the following steps:
    i. Prompt the user to enter the directory they want to back up.
    ii. Create a timestamped backup folder (e.g., `backup_<timestamp>`) inside a specified backup directory (you can define this directory at the beginning of your script).
    iii. Copy all files and directories from the user-specified directory to the backup folder.
    iv. Compress the backup folder into a single archive file `backup_<timestamp>.tar.gz`.
    v. Display a message indicating the successful completion of the backup process.
3. Test your script by running it in your terminal. Ensure it performs all the specified tasks correctly.
4. (**Bonus**) Implement error handling in your script. For example, check if the specified input directory exists.

Exercise 4: hands on `git`. Collaborative file management (1/3)

1. Form groups of 2-3 members.
2. Designate one member to create a new repository (visit https://github.com/ and click the `+` button in the top right corner), and ensure everyone clones it.
3. In a sequential manner, each group member should create a file with a distinct name and push it to the online repository while the remaining members pull the changes.
4. Repeat step 3, but this time, each participant should modify a different file than the ones modified by the other members of the group.

Exercise 4: hands on `git`. Collaborative file management (2/3)

Now, let's work on the same file, `main.cpp`. Each person should create a hello world `main.cpp` that includes a personalized greeting with your name. To prevent conflicts, follow these steps:

1. Create a unique branch using the command: `git checkout -b [new_branch]`.
2. Develop your code and push your branch to the online repository.
3. Once everyone has finished their work, merge your branch into the `main` branch using the following commands:

```
git checkout main
git pull origin main
git merge [new_branch]
git push origin main
```

Exercise 4: hands on `git`. Collaborative file management (3/3)

How to deal with `git` conflicts

The first person to complete this process will experience no issues. However, subsequent participants may encounter merge conflicts.

Git will mark the conflicting sections in the file. You'll see these sections surrounded by `<<<<<<<`, `=======`, and `>>>>>>>` markers.

Carefully review the conflicting sections and decide which changes to keep. Remove the conflict markers (`<<<<<<<`, `=======`, `>>>>>>>`) and make the necessary adjustments to the code to integrate both sets of changes correctly.

After resolving the conflict, commit your changes and push your resolution to the repository.