



COLLEGE OF ENGINEERING, CONSTRUCTION, AND LIVING SCIENCES

BACHELOR OF INFORMATION TECHNOLOGY (BIT)

CHECKPOINT-1, SEMESTER 2, 2025

AUTOMATION OF USER, PROCESS, AND PERMISSION MANAGEMENT IN A COMPUTING
SYSTEM

ID616001 OPERATING SYSTEMS CONCEPTS

LEVEL 6, CREDITS 15

VERSION 1.0

NAME : Cameron Yeoman

USER : yeomecj1

DATE: 04/08/2025

VM Public IP: 4.147.186.144

Repository: [ChubbyLobsters/checkpoint1-files-YEOMCJ1: Files for Checkpoint 1 OPS](#)

Rubric Marking Sheet

Criteria	Max Marks	Marks Awarded
Task 1: VM Connection and User Setup [15 Marks]		
Retrieve VM Public IP Address	2	
Connect to VM using PuTTY	5	
Change the account username	5	
Change the password	3	
Task 2: User Onboarding Automation [40 Marks]		
Parse <code>users.csv</code> file correctly	5	
User account management (exist/create/update)	5	
Group creation and membership validation	5	
Home directory setup with permissions	5	
Project directory creation and permission setup	5	
Log significant actions with timestamps	5	
Error handling and input validation	5	
Inline comments and script readability	5	
Task 3: Critical Log Analysis [20 Marks]		
Filter critical logs using keywords	5	
Tokenize filtered logs into individual words	5	
Count frequency and sort top 10 tokens	5	
Output written to <code>top10_critical.txt</code>	5	
Code Defense Presentation [25 Marks]		
Clarity of explanation and walkthrough	5	
Justification of implementation choices	5	
Understanding of logic and flow	5	
Awareness of error handling/validation	5	
Responses to questions (completeness/confidence)	5	
Total	100	

Assessor's Comments:

Assessor Name:

Signature:

Date:

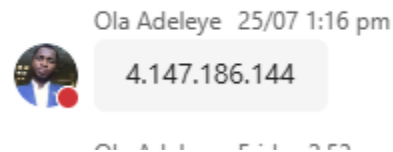
Section #1

Connect to Azure VM using PuTTY

1. Retrieve Your VM Public IP Address

My VM Public IP address was provided by the lecturer directly, as login to the Azure portal was not available.

: 4.147.186.144

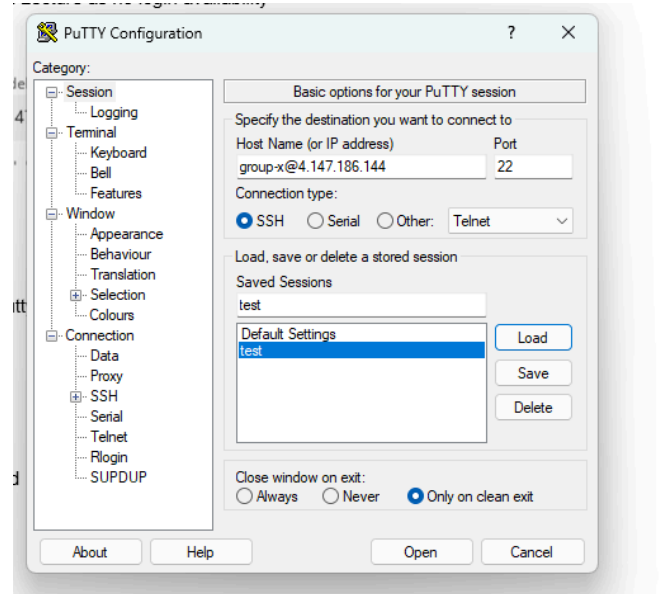


A screenshot of the Microsoft Azure portal. The top navigation bar shows "Home" and "labvm-1 Virtual machine". Below the navigation bar, there is a search bar and a list of tabs: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Resource visualizer, Connect, Networking, Settings, Availability + scale, Security, Backup + disaster recovery, Operations, Monitoring, Automation, and Help. The main content area shows the details of the virtual machine "labvm-1". The "Overview" tab is selected, displaying a table of properties: Operating system (Linux (ubuntu 22.04)), Size (Standard B2s (2 vcpus, 4 GiB memo...)), Public IP address (4.147.186.144), Virtual network/subnet (labvm-vnet/labvm-Subnet), DNS name (Not configured), Health state (-), Time created (7/14/2025, 1:55 AM UTC), and Tags (edit). Below this table, there are tabs for Properties, Monitoring, Capabilities (7), and Recommendations (5). The "Networking" tab is selected and circled in red, showing details for the network interface "labvm-nic-1": Public IP address (4.147.186.144), Public IP address (IPv6) (-), Private IP address (-), Private IP address (IPv6) (-), and Virtual network/subnet (labvm-vnet/labvm-Subnet).

Section #1

2. Connect Using PuTTY

```
group-x@labvm-1: ~  
Using username "group-x".  
group-x@4.147.186.144's password:  
Access denied  
group-x@4.147.186.144's password:  
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-1030-azure x86_64)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:        https://ubuntu.com/pro  
  
System information as of Fri Aug 1 01:07:32 UTC 2025  
  
System load:  0.0      Processes:      119  
Usage of /:   8.5% of 28.89GB  Users logged in:  1  
Memory usage: 11%      IPv4 address for eth0: 10.2.0.4  
Swap usage:   0%  
  
* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s  
just raised the bar for easy, resilient and secure K8s cluster deployment.  
  
https://ubuntu.com/engage/secure-kubernetes-at-the-edge  
Expanded Security Maintenance for Applications is not enabled.
```



I accessed the virtual machine using PuTTY by entering the provided IP address and logging in with the assigned Group-X username and password.

Screenshot: Successful SSH login via PuTTY.

Section #1

3. Change the Account Username

```
*** System restart required ***
Last login: Fri Aug  1 01:03:17 2025 from 202.49.0.118
group-x@labvm-1:~$ whoami
group-x
group-x@labvm-1:~$ sudo adduser tempadmin
Adding user `tempadmin' ...
Adding new group `tempadmin' (1001) ...
Adding new user `tempadmin' (1001) with group `tempadmin' ...
Creating home directory `/home/tempadmin' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
Sorry, passwords do not match.
passwd: Authentication token manipulation error
passwd: password unchanged
Try again? [y/N] y
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for tempadmin
Enter the new value, or press ENTER for the default
  Full Name []:
  Room Number []:
  Work Phone []:
  Home Phone []:
  Other []:
Is the information correct? [Y/n] y
group-x@labvm-1:~$ sudo usermod -aG sudo tempadmin
group-x@labvm-1:~$ sudo usermod -l yeomcjl group-x
usermod: user group-x is currently used by process 237946
group-x@labvm-1:~$ ^C
group-x@labvm-1:~$ █
```

Used SSH in the VM to create a new administrative user named tempadmin with permissions to manage user accounts.

Commands Used:

```
sudo adduser tempadmin
sudo usermod -aG sudo tempadmin
```

Screenshot: Confirmation of successful creation and sudo access for tempadmin.

Section #1

3. Change the Account Username (2)

```
Using username "tempadmin".
tempadmin@4.147.186.144's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-1030-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

System information as of Fri Aug 1 01:17:41 UTC 2025

System load:  0.0          Processes:      118
Usage of /:   8.5% of 28.89GB    Users logged in: 0
Memory usage: 11%          IPv4 address for eth0: 10.2.0.4
Swap usage:   0%

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

Updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '24.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

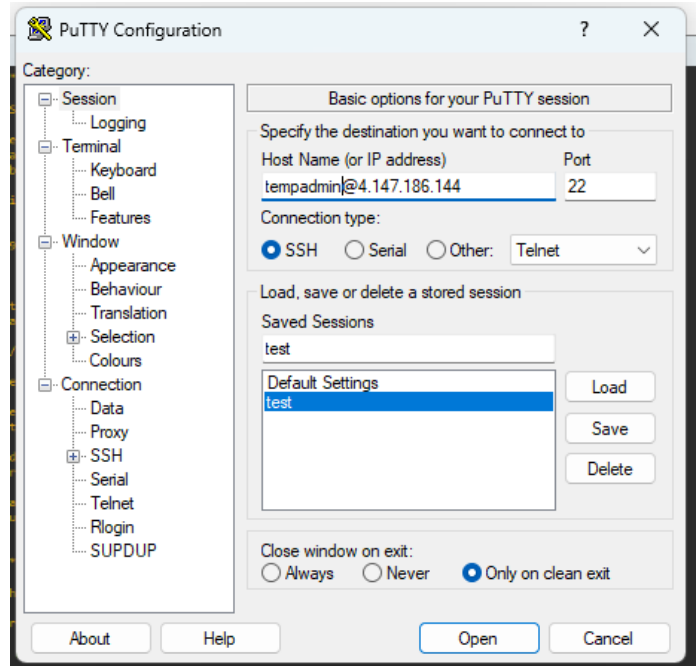
** System restart required **

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

tempadmin@labvm-1:~$ sudo usermod -l yeomcj1 group-x
[sudo] password for tempadmin:
tempadmin@labvm-1:~$ sudo groupmod -n yeomcj1 group-x
tempadmin@labvm-1:~$ sudo usermod -d /home/yeomcj1 -m yeomcj1
tempadmin@labvm-1:~$ sudo passwd yeomcj1
New password:
Retype new password:
passwd: password updated successfully
tempadmin@labvm-1:~$
```



Logged into the VM via PuTTY using the new **tempadmin** user credentials to perform user account modifications.

Screenshot: Successful SSH login as **tempadmin**.

Task Performed – Renaming User Account:

Renamed the default **group-x** user to **yeomcj1** and updated associated group and home directory:

Commands Used:

<code>sudo usermod -l yeomcj1 group-x</code>	<code># Rename the user</code>
<code>sudo groupmod -n yeomcj1 group-x</code>	<code># Rename the group</code>
<code>sudo usermod -d /home/yeomcj1 -m yeomcj1</code>	<code># Move and rename home directory</code>

Section #1

4. Change the Account Password

```
*** System restart required ***

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

empadmin@labvm-1:~$ sudo usermod -l yeomcj1 group-x
[sudo] password for tempadmin:
empadmin@labvm-1:~$ sudo groupmod -n yeomcj1 group-x
empadmin@labvm-1:~$ sudo usermod -d /home/yeomcj1 -m yeomcj1
empadmin@labvm-1:~$ sudo passwd yeomcj1
New password:
Retype new password:
passwd: password updated successfully
empadmin@labvm-1:~$
```

```
Usage of /: 8.5% of 28.89GB  Users logged in: 0
Memory usage: 11%          IPv4 address for eth0: 10.2.0.4
Swap usage: 0%

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

3 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '24.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

*** System restart required ***
Last login: Fri Aug 1 01:07:33 2025 from 202.49.0.118
yeomcj1@labvm-1:~$
```

Changed Password for user account yeomcj1:

Set a new secure password for the renamed user account using the command:
sudo passwd yeomcj1

Logged in using updated credentials:

Reconnected to the VM via PuTTY using the new username **yeomcj1** and the newly set password:
ssh yeomcj1@4.147.186.144 -p 22

Screenshot shows successful SSH login with the new credentials.

Section #2

User Onboarding Automation

The screenshot shows a GitHub repository interface for 'checkpoint1-files-YEOMCJ1'. The repository is public and has 1 branch and 0 tags. The commit history shows an initial commit by 'Ubuntu' 3 days ago, which included several files. The README file is also visible, titled 'checkpoint1-files-YEOMCJ1' and containing the text 'Files for Checkpoint 1 OPS'.

File	Commit Message	Time
Onboard_users.sh	Initial commit of checkpoint1 files including onboarding scri...	3 days ago
README.md	Initial commit	3 days ago
critical_log_analysis.sh	Initial commit of checkpoint1 files including onboarding scri...	3 days ago
filtered_logs.txt	Initial commit of checkpoint1 files including onboarding scri...	3 days ago
sys_log.txt	Initial commit	3 days ago
tokens.txt	Initial commit of checkpoint1 files including onboarding scri...	3 days ago
top10_critical.txt	Initial commit of checkpoint1 files including onboarding scri...	3 days ago
users.csv	Initial commit of checkpoint1 files including onboarding scri...	3 days ago

README

checkpoint1-files-YEOMCJ1

Files for Checkpoint 1 OPS

Script name: `onboard_users.sh`

Input file: `users.csv`

The script processes each user from the input file to:

- Add the user and their group
- Update the user's shell if needed
- Create the user's home directory
- Create a `/opt/projects/<username>` directory
- Log all actions to `/var/log/user_onboarding_audit.log`
- Validate input and handle errors properly

Section #2

Setup & Organization Via Github

```
/home/yeomcj1/Checkpoint
yeomcj1@labvm-1:~/Checkpoint$ ls
checkpoint1-files
yeomcj1@labvm-1:~/Checkpoint$ cd checkpoint1-files/
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ls
sys_log.txt  users.csv
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ touch Onboard_users.sh
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$

yeomcj1@labvm-1:~$ ped
Command 'ped' not found, but there are 16 similar ones.
yeomcj1@labvm-1:~$ pwd
/home/yeomcj1
yeomcj1@labvm-1:~$ mkdir Checkpoint
yeomcj1@labvm-1:~$ cd
yeomcj1@labvm-1:~$
yeomcj1@labvm-1:~$ pwd
/home/yeomcj1
yeomcj1@labvm-1:~$ cd Checkpoint/
yeomcj1@labvm-1:~/Checkpoint$ git clone https://github.com/OlayinkaOP/checkpoint1-files.git
Cloning into 'checkpoint1-files'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (4/4), done.
yeomcj1@labvm-1:~/Checkpoint$ pwd
/home/yeomcj1/Checkpoint
yeomcj1@labvm-1:~/Checkpoint$ ls
checkpoint1-files
yeomcj1@labvm-1:~/Checkpoint$
```

Write a shell script named `onboard_users.sh` to automate the user onboarding process using data from a CSV file (`users.csv`). The script must:

- Create user accounts and assign groups
- Set up home and project directories
- Log all actions for auditing
- Include proper input validation and error handling
- Be well documented within the script

Development process:

I used Git Bash to clone the required repository and navigate (`cd`) to the project directory. From there, I started developing the `onboard_users.sh` script according to the specifications.

Section #2

1. Read and parse users.csv

```
point1- yeomcj1@labvm-1:~/Checkpoint$ pwd
terminal t /home/yeomcj1/Checkpoint
.es.git yeomcj1@labvm-1:~/Checkpoint$ ls
checkpoint1-files
yeomcj1@labvm-1:~/Checkpoint$ cd checkpoint1-files/
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ls
sys_log.txt  users.csv
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ touch Onboard_users.sh
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ nano Onboard_users.sh
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ vim Onboard_users.sh
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ chmod +x Onboard_users.sh
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ./Onboard_users.sh
Username: rob, Group: admin/lecturer, Shell: /bin/bash
Username: ola, Group: lecturer, Shell: /bin/bash
Username: andy, Group: student, Shell: /bin/bash
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$
```

I began by creating the script file using `touch onboard_users.sh` and then edited it in Vim, entering insert mode to write the initial code. After writing the start of the script, I made it executable with `chmod +x onboard_users.sh`. I then tested the script in the console, confirming it processed the CSV correctly and achieved the expected output.

Key parts of the script include reading and processing the `users.csv` file while skipping the header line using:

```
tail -n +2 users.csv | while IFS=',' read -r username groupname shell
```

I trimmed fields and replaced slashes in group names with underscores for compatibility with:

```
groupname="${groupname// /_}"
```

For readability and verification, I used `echo` to print parsed fields:

```
echo "Username: $username, Group: $groupname, Shell: $shell"
```

The sample input from `users.csv` was:

:

```
username,groupname,shell
```

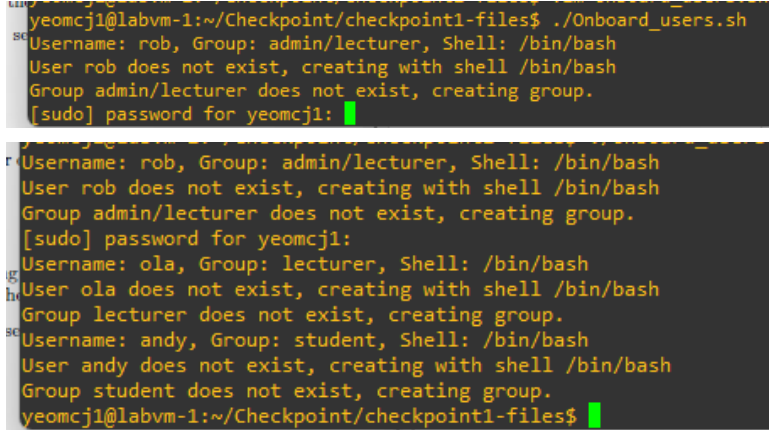
```
rob,admin/lecturer,/bin/bash
```

```
ola,lecturer,/bin/bash
```

```
andy,student,/bin/bash
```

Section #2

2. User Account Management



```
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ./Onboard_users.sh
Username: rob, Group: admin/lecturer, Shell: /bin/bash
User rob does not exist, creating with shell /bin/bash
Group admin/lecturer does not exist, creating group.
[sudo] password for yeomcj1:

Username: rob, Group: admin/lecturer, Shell: /bin/bash
User rob does not exist, creating with shell /bin/bash
Group admin/lecturer does not exist, creating group.
[sudo] password for yeomcj1:
Username: ola, Group: lecturer, Shell: /bin/bash
User ola does not exist, creating with shell /bin/bash
Group lecturer does not exist, creating group.
Username: andy, Group: student, Shell: /bin/bash
User andy does not exist, creating with shell /bin/bash
Group student does not exist, creating group.
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$
```

I then wrote the code to handle user creation on Linux using a loop to process each user from the CSV.

First, the script checks if a user already exists with:

```
if id "$username" &>/dev/null; then
```

If the user exists, it updates the user's shell and outputs a message:

```
echo "User $username exists, updating shell to $shell"
```

```
sudo usermod -s "$shell" "$username"
```

If the user does not exist, it outputs a creation message and proceeds to check the user's group:

```
echo "User $username does not exist, creating with shell $shell"
```

The script then verifies if the group exists using:

```
if ! getent group "$groupname" > /dev/null; then
```

```
echo "Group $groupname does not exist, creating group."
```

```
sudo groupadd "$groupname"
```

```
fi
```

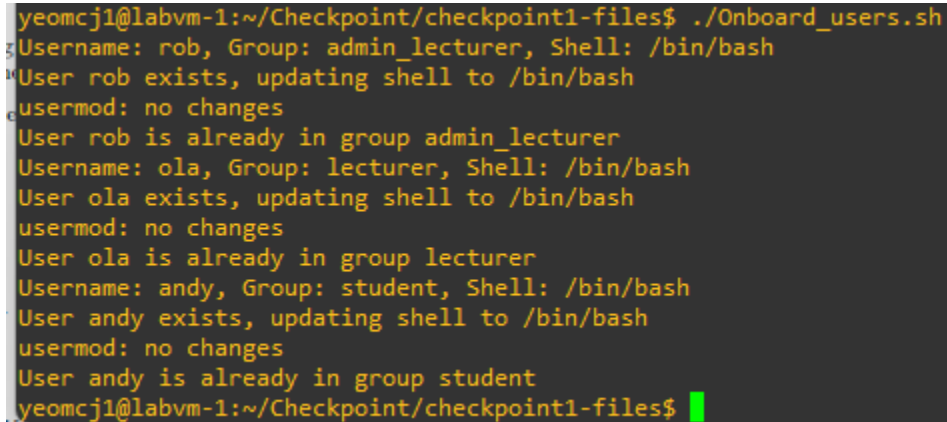
Finally, if the group exists or after creating it, the script creates the user with:

```
sudo useradd -m -g "$groupname" -s "$shell" "$username"
```

Screenshots demonstrate the script running, including user creation and password confirmation prompts.

Section #2

3. Group Creation and Membership Validation



```
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ./Onboard_users.sh
Username: rob, Group: admin_lecturer, Shell: /bin/bash
User rob exists, updating shell to /bin/bash
usermod: no changes
User rob is already in group admin_lecturer
Username: ola, Group: lecturer, Shell: /bin/bash
User ola exists, updating shell to /bin/bash
usermod: no changes
User ola is already in group lecturer
Username: andy, Group: student, Shell: /bin/bash
User andy exists, updating shell to /bin/bash
usermod: no changes
User andy is already in group student
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$
```

When it came to group management, I referred to open book resources and carefully reviewed the `users.csv` file. I recognized that verifying and ensuring users belong to the correct groups was crucial. To improve the onboarding process, I implemented group verification that:

- Confirms users are assigned to the specified groups
- Avoids creating duplicate groups
- Prevents users from losing their group memberships
- Simplifies adding users to the correct groups whether they are new or existing

To achieve this, the script checks if a user is already a member of the group using:

```
if id -nG "$username" | grep -qw "$groupname"; then
    echo "User $username is already in group $groupname"
else
    echo "Adding user $username to group $groupname"
    sudo usermod -aG "$groupname" "$username"
fi
```

This ensures users are only added to groups when necessary, maintaining clean group memberships and preventing redundant changes.

Screenshot shows the updated script running

Section #2

4. Home Directory Setup

```
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$ ./Onboard_users.sh
Username: rob, Group: admin_lecturer, Shell: /bin/bash
User rob exists, updating shell to /bin/bash
usermod: no changes
User rob is already in group admin_lecturer
Home directory /home/rob exists.
Setting permissions of /home/rob to 700
Username: ola, Group: lecturer, Shell: /bin/bash
User ola exists, updating shell to /bin/bash
usermod: no changes
User ola is already in group lecturer
Home directory /home/ola exists.
Setting permissions of /home/ola to 700
Username: andy, Group: student, Shell: /bin/bash
User andy exists, updating shell to /bin/bash
usermod: no changes
User andy is already in group student
Home directory /home/andy exists.
Setting permissions of /home/andy to 700
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$
```

I ensured that each user's home directory is created with the correct permissions and ownership as follows:

Defined the home directory path for the current user:

```
homedir="/home/$username"
```

Checked if the home directory already exists:

```
if [ -d "$homedir" ]; then
```

```
    # directory exists, proceed to verify permissions and ownership
```

```
else
```

```
    # create the directory and set ownership and permissions
```

```
    sudo mkdir "$homedir"
```

```
    sudo chown "$username":"$groupname" "$homedir"
```

```
    sudo chmod 700 "$homedir"
```

```
Fi
```

Verified the directory permissions are set to **700** (read, write, execute for owner only):

```
perm=$(stat -c "%a" "$homedir")
```

```
if [ "$perm" != "700" ]; then
```

```
    sudo chmod 700 "$homedir"
```

```
fi
```

Verified the directory ownership is set correctly to the user and group, and corrected it if necessary:

```
owner=$(stat -c "%U" "$homedir")
```

```
if [ "$owner" != "$username" ]; then
```

```
    sudo chown "$username":"$groupname" "$homedir"
```

```
fi
```

This approach ensures the home directory is secure and properly owned, whether it is newly created or pre-existing.

Screenshot shows the updated script running displaying the new dir

Example : directatory/home/ola

Section #2

5. Create a project directory for the user

```
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$ ./Onboard_users.sh
Username: rob, Group: admin_lecturer, Shell: /bin/bash
User rob exists, updating shell to /bin/bash
usermod: no changes
User rob is already in group admin_lecturer
Home directory /home/rob exists.
Creating project directory /opt/projects/rob
Username: ola, Group: lecturer, Shell: /bin/bash
User ola exists, updating shell to /bin/bash
usermod: no changes
User ola is already in group lecturer
Home directory /home/ola exists.
Creating project directory /opt/projects/ola
Username: andy, Group: student, Shell: /bin/bash
User andy exists, updating shell to /bin/bash
usermod: no changes
User andy is already in group student
Home directory /home/andy exists.
Creating project directory /opt/projects/andy
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$
```

To meet the requirements for setting up a project directory, I added logic to the script that automatically creates a user-specific directory under `/opt/projects`, ensuring proper ownership and permissions.

Set the path for the user's project directory:

```
project_dir="/opt/projects/$username"
```

For example, if the username is `rob`, the directory becomes `/opt/projects/rob`.

Checked if the directory already exists to prevent overwriting existing data:

```
if [ -d "$project_dir" ]; then
    echo "Project directory $project_dir already exists."
else
    echo "Creating project directory $project_dir"
    sudo mkdir -p "$project_dir"
    sudo chown "$username":"$groupname" "$project_dir"
    sudo chmod 750 "$project_dir"
fi
```

Ownership is set to `<username> : <groupname>` to ensure correct access control.

Permissions are set to `750`, meaning:

User has full access (`7`)

Group has read and execute access (`5`)

Others have no access

This ensures each user has their own secure project workspace, while still allowing their group to collaborate if needed.

Screenshot shows this section of the script running successfully in the terminal.

Section #2

6 Logging with Timestamps

```
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$ vim Onboard_users.sh
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$ ./Onboard_users.sh
Username: rob, Group: admin_lecturer, Shell: /bin/bash
User rob exists, updating shell to /bin/bash
usermod: no changes
User rob is already in group admin_lecturer
Home directory /home/rob exists.
Project directory /opt/projects/rob already exists.
Username: ola, Group: lecturer, Shell: /bin/bash
User ola exists, updating shell to /bin/bash
usermod: no changes
User ola is already in group lecturer
Home directory /home/ola exists.
Project directory /opt/projects/ola already exists.
Username: andy, Group: student, Shell: /bin/bash
User andy exists, updating shell to /bin/bash
usermod: no changes
User andy is already in group student
Home directory /home/andy exists.
Project directory /opt/projects/andy already exists.
```

To enhance traceability and auditing, I added a logging function that records all key actions taken by the script. Each log entry includes:

A **timestamp**

A **custom message** describing the action performed

The logs are written to:

`/var/log/user_onboarding_audit.log`

This makes it easy to review actions later or monitor the script in real time.

Logging setup:

Defined a constant for the log file location:

`LOGFILE="/var/log/user_onboarding_audit.log"`

Created a reusable logging function:

```
log_action() {  
    local message="$1"  
    local timestamp  
    timestamp=$(date '+%Y-%m-%d %H:%M:%S')  
    echo "$timestamp - $message" | sudo tee -a "$LOGFILE" > /dev/null  
    # Uncomment the next line if you want to print to console too:  
    # echo "$message"  
}
```

Instead of using `echo` throughout the script, I replaced it with `log_action` to ensure every major step is recorded consistently.

Example usage:

```
log_action "Created home directory for $username"  
log_action "Added $username to group $groupname"
```

You can monitor activity in real time by running:

```
sudo tail -f /var/log/user_onboarding_audit.log
```

This provides live output of the script's operations, which is helpful for debugging or validation during onboarding.

Section #2

6. Logging with Timestamps Command Example

```
~ sudo tail -f /var/log/user_onboarding_audit.log
```

```
rding_audit.log
2025-08-01 02:08:46 - User ola shell updated to /bin/bash
2025-08-01 02:08:46 - User ola is already in group lecturer
2025-08-01 02:08:46 - Home directory /home/ola exists.
2025-08-01 02:08:46 - Project directory /opt/projects/ola already exists.
2025-08-01 02:08:46 - Processing user: andy, group: student, shell: /bin/bash
2025-08-01 02:08:46 - User andy exists, updating shell to /bin/bash
2025-08-01 02:08:46 - User andy shell updated to /bin/bash
2025-08-01 02:08:46 - User andy is already in group student
2025-08-01 02:08:46 - Home directory /home/andy exists.
2025-08-01 02:08:46 - Project directory /opt/projects/andy already exists.
```

```
usermod: no changes
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$ sudo tail -f /var/log/user_onboa
rding_audit.log
2025-08-01 02:20:18 - Shell updated for ola.
2025-08-01 02:20:18 - User ola already in group lecturer.
2025-08-01 02:20:19 - Home directory /home/ola exists. Checking permissions.
2025-08-01 02:20:19 - Project directory /opt/projects/ola already exists.
2025-08-01 02:20:19 - Processing entry: username='andy', groupname='student', sh
ell='/bin/bash'
2025-08-01 02:20:19 - User andy exists. Updating shell to /bin/bash.
2025-08-01 02:20:19 - Shell updated for andy.
2025-08-01 02:20:19 - User andy already in group student.
2025-08-01 02:20:19 - Home directory /home/andy exists. Checking permissions.
2025-08-01 02:20:19 - Project directory /opt/projects/andy already exists.
```

Screenshots reflect the changes explained above providing the logs for the users/files. It provides timestamps and a message detailing the changes/events.

Example : 2025-08-01 02:20:18 - Shell updated for ola.

Section #2

7. Logging with Timestamps Command Example

```
usermod: no changes
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$ vim Onboard_users.sh
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$ ./Onboard_users.sh
usermod: no changes
usermod: no changes
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$
```

To meet the requirement of making the script resilient and reliable, I implemented error handling and input validation throughout the user onboarding process. This ensures the script continues processing even if individual entries fail, and that all input data is properly validated.

To prevent the script from crashing or stopping when it encounters an error, I used `continue` statements. This allows it to skip faulty entries and move on to the next line in the CSV:

```
# Example: Skip if required fields are missing
if [[ -z "$username" || -z "$groupname" || -z "$shell" ]]; then
    log_action "ERROR: Missing field(s) in CSV entry."
    continue
fi
```

This protects against bad CSV formatting (e.g., blank lines, incomplete rows), which could otherwise cause errors during user creation.

To ensure system compatibility and avoid injecting invalid usernames or groups (which could include spaces, slashes, or special characters), I used a regular expression:

```
VALID_NAME_REGEX="^[a-zA-Z0-9._-]+$"

if ! [[ "$username" =~ $VALID_NAME_REGEX ]]; then
    log_action "ERROR: Invalid characters in username: $username"
    continue
fi

if ! [[ "$groupname" =~ $VALID_NAME_REGEX ]]; then
    log_action "ERROR: Invalid characters in groupname: $groupname"
    continue
fi
```

Commands like `groupadd` and `useradd` are wrapped in checks to log errors and continue if they fail:

```
if sudo groupadd "$groupname"; then
    log_action "Group $groupname created"
else
    log_action "ERROR: Failed to create group $groupname"
    continue
fi

if sudo useradd -m -s "$shell" "$username"; then
    log_action "User $username created"
else
    log_action "ERROR: Failed to create user $username"
    continue
fi
```

This error-handling strategy ensures:

- Faulty entries don't break the whole process
- Logs explain exactly what went wrong and where
- The script finishes processing all valid users in the CSV

Screenshot does not indicate when error responses are displayed but shows changes in script format as it now does not print unless valid and informative data.

Section #2

8. Add inline comments to explain the logic

```
Inline Comments Made
:

# Define the log file path
# Function to log actions with timestamp
# Define a regex pattern for valid usernames and group names
# Read users.csv, skipping the header
# Trim whitespace from fields
# Log the user entry being processed
# Skip entry if any field is missing
# Validate username format
# Validate group name format
# Replace any slashes in groupname to make it valid
# Check if shell exists and is executable, else skip user
# Create group if it doesn't already exist
# If user exists, update shell only if different; otherwise, create the user
# Ensure user is added to the correct group
# Setup or correct user's home directory
# Set correct permissions if needed
# Correct ownership if needed
# Create and configure home directory if it doesn't exist
# Setup or validate project directory
```

```
# Create group if it doesn't already exist
if ! getent group "$groupname" > /dev/null; then
    log_action "Group $groupname does not exist, creating group."
    if sudo groupadd "$groupname"; then
        log_action "Group $groupname created successfully."
    else
        log_action "ERROR: Failed to create group $groupname. Skipping user $username."
        continue
    fi
fi

# If user exists, update shell only if different; otherwise, create the user
if id "$username" &>/dev/null; then
```

```
# Define the log file path
LOGFILE="/var/log/user_onboarding_audit.log"

# Function to log actions with timestamp
log_action() {
    local message="$1"
    local timestamp
    timestamp=$(date '+%Y-%m-%d %H:%M:%S')
    echo "$timestamp - $message" | sudo tee -a "$LOGFILE" > /dev/null
}
```

Section #3

Critical Log Analysis

```
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ls
Onboard_users.sh sys_log.txt users.csv
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ touch critical_log_analysis.sh
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ls
Onboard_users.sh critical_log_analysis.sh sys_log.txt users.csv
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$
```

1. Filter critical logs using keywords

```
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ls
Onboard_users.sh sys_log.txt users.csv
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ touch critical_log_analysis.sh
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ls
Onboard_users.sh critical_log_analysis.sh sys_log.txt users.csv
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ chmod +x critical_log_analysis.s
h
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ./critical_log_analysis.sh
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ./critical_log_analysis.sh
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ vim critical_log_analysis.sh
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ./critical_log_analysis.sh
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ./critical_log_analysis.sh
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$ ls
Onboard_users.sh filtered_logs.txt users.csv
critical_log_analysis.sh sys_log.txt
yeomcj1@labvm-1:~/Checkpoint/checkpoint1-files$
```

To identify high-priority issues in system logs, I created a script that filters for critical log entries based on specific keywords.

```
touch critical_log_analysis.sh
```

```
chmod +x critical_log_analysis.sh
```

The following command is used to extract only the most important log messages:

```
grep -iE 'ERROR|CRITICAL|FATAL' sys_log.txt > filtered_logs.txt
```

The screenshot shows both the script being created and successfully made executable, confirming setup and readiness for use.

Section #3

2. Tokenize Filtered Logs

To prepare the filtered log data for deeper analysis, I tokenized the text—converting it into individual words (tokens) that can be counted or analyzed further.

The command used:

```
tr -s '[:space:][:punct:]' '\n' < filtered_logs.txt | grep -v '^$' > tokens.txt
```

The file `tokens.txt` contains one word per line, cleaned of punctuation and spacing—ideal for counting, searching, or further log analysis.

This process ensures logs are properly broken down into structured tokens using standard Linux text utilities (`tr` and `grep`), as required.

Screenshots show the key tokens being captured and printed to console

```
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$ ./critical_log_analysis.sh
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$ cat tokens.txt
Jul
15
08
45
04
server1
kernel
kernel
763425
643212
EXT4
fs
error
device
sda1
ext4
find
entry
1453
inode
66732
comm
rsync
reading
directory
lblock
0
Jul
15
08
45
04
server1
```

```
kernel
763425
643214
EXT4
fs
sda1
previous
I
0
error
to
superblock
detected
Jul
15
09
15
13
server1
sshd
64321
error
maximum
authentication
attempts
exceeded
for
invalid
user
admin
from
10
0
0
```


Section #3

3. Count and Sort Tokens

Count frequency and output top 10 tokens

```
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$ vim critical_log_analysis.sh
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$ ./critical_log_analysis.sh
Top 10 critical tokens:
 8 15
 7 server1
 7 Jul
 5 kernel
 5 error
 5 09
 5 0
 4 45
 3 sshd
 2 user
yeomcjl@labvm-1:~/Checkpoint/checkpoint1-files$
```

To analyze the most common words in critical logs, I implemented a pipeline that counts how often each token appears and sorts them in descending order.

```
sort tokens.txt | uniq -c | sort -nr | head -n 10 > top10_critical.txt
```

Output:

The terminal displays the top 10 most frequent words, and the output is also stored in `top10_critical.txt`:

```
echo "Top 10 critical tokens:"
cat top10_critical.txt
```

This helps quickly identify recurring keywords in error logs—useful for diagnostics, alerts, or prioritization.

As required, the result of the frequency analysis is saved to:

```
top10_critical.txt
```

This text file contains the most frequent tokens (words) from the critical logs, along with their occurrence counts. It's easy to review later or use as part of a larger reporting or automation process.

Screenshot shows the top 10 stored/frequent tokens printing on script running

Section #3

3. Complete Workflow Recap:

Step 1: Filter critical logs

```
grep -iE 'ERROR|CRITICAL|FATAL' sys_log.txt > filtered_logs.txt
```

Step 2: Tokenize filtered logs into individual words

```
tr -s '[:space:][:punct:]' '\n' < filtered_logs.txt | grep -v '^$' > tokens.txt
```

Step 3: Count and sort tokens, output top 10

```
sort tokens.txt | uniq -c | sort -nr | head -n 10 > top10_critical.txt
```

Step 4: Display the top 10 tokens

```
echo "Top 10 critical tokens:"
```

```
cat top10_critical.txt
```

The screenshot displays a file explorer on the left and a code viewer on the right. The file explorer shows a directory structure with files like `Onboard_users.sh`, `README.md`, `critical_log_analysis.sh`, `filtered_logs.txt`, `sys_log.txt`, `tokens.txt`, `top10_critical.txt`, and `users.csv`. The code viewer shows the content of `top10_critical.txt`, which lists the top 10 critical tokens and their counts.

Line	Token	Count
1	server1	8
2	Jul	7
3	kernel	5
4	error	5
5	09	5
6	0	5
7	45	4
8	sshd	3
9	user	2
10		

Conclustion

Github Documentation

I uploaded my final script and related files to the GitHub repository I cloned earlier.

During the process, I ran into permissions issues, so I had to adjust file permissions and set up SSH authentication to push to GitHub securely.

After fixing this issue I pushed to my main Origin.

Commits

main	All users	All time
Commits on Aug 1, 2025		
Initial commit of checkpoint1 files including onboarding scripts Ubuntu committed 3 days ago	b966418	<>
Initial commit OlayinkaOP authored and Ubuntu committed 3 days ago	6757245	<>
Initial commit ChubbyLobsters authored 3 days ago	Verified 58c8f2f	<>

Two-factor authentication

Two-factor authentication adds an additional layer of security to your account by requiring more than just a password to sign in. [Learn more about two-factor authentication.](#)

Preferred 2FA method

Set your preferred method to use for two-factor authentication when signing into GitHub.

Authenticator app

