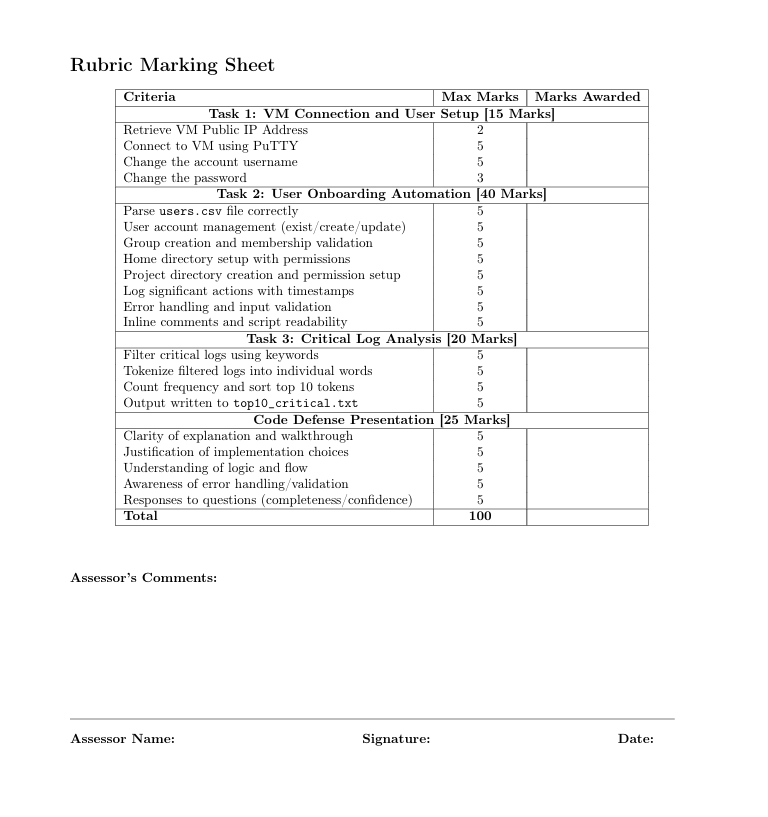


**NAME : Cameron Yeoman  
USER : yeomcj1  
DATE: 04/08/2025  
VM Public IP: 4.147.186.144  
Repository:** [**ChubbyLobsters/checkpoint1-files-YEOMCJ1: Files for Checkpoint 1 OPS**](https://github.com/ChubbyLobsters/checkpoint1-files-YEOMCJ1)



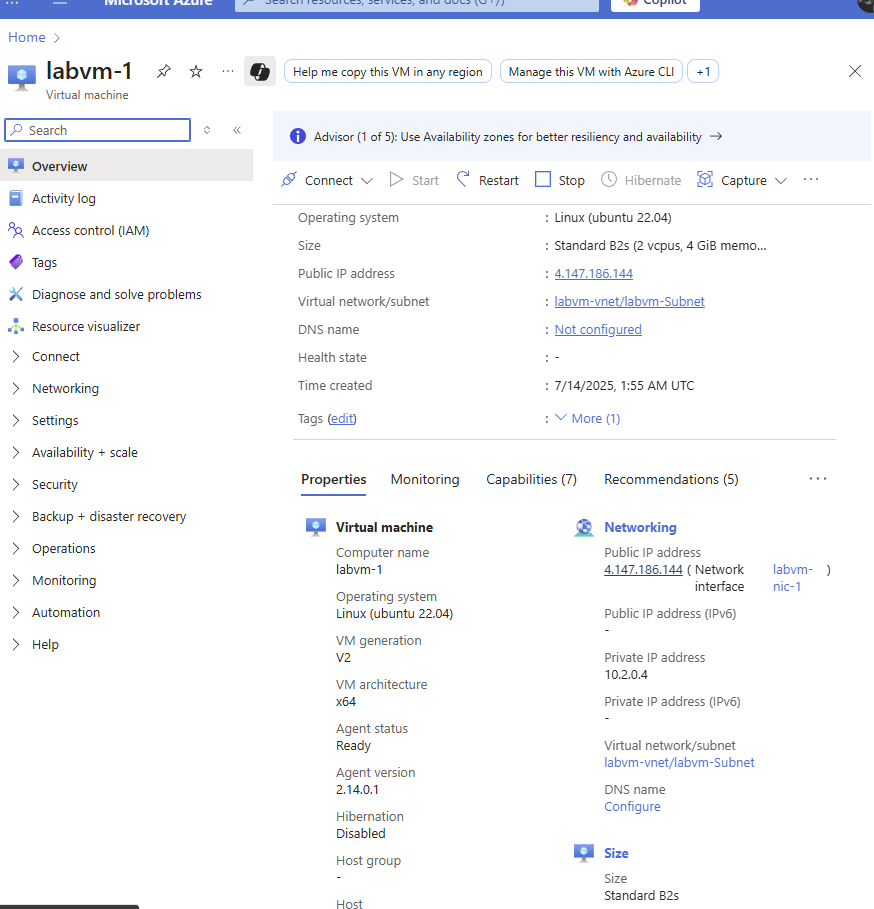
## **Section #1**

Connect to Azure VM using PuTTY

## 

### **1. Retrieve Your VM Public IP Address**

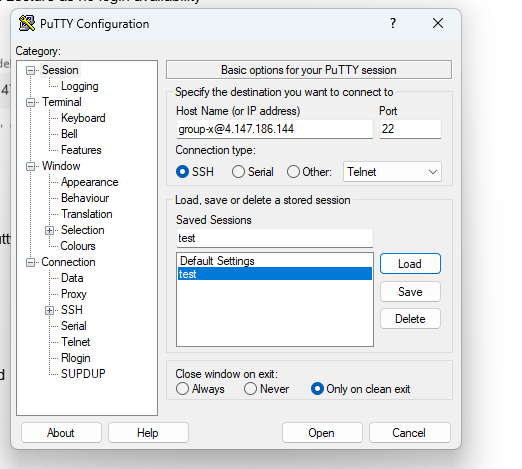
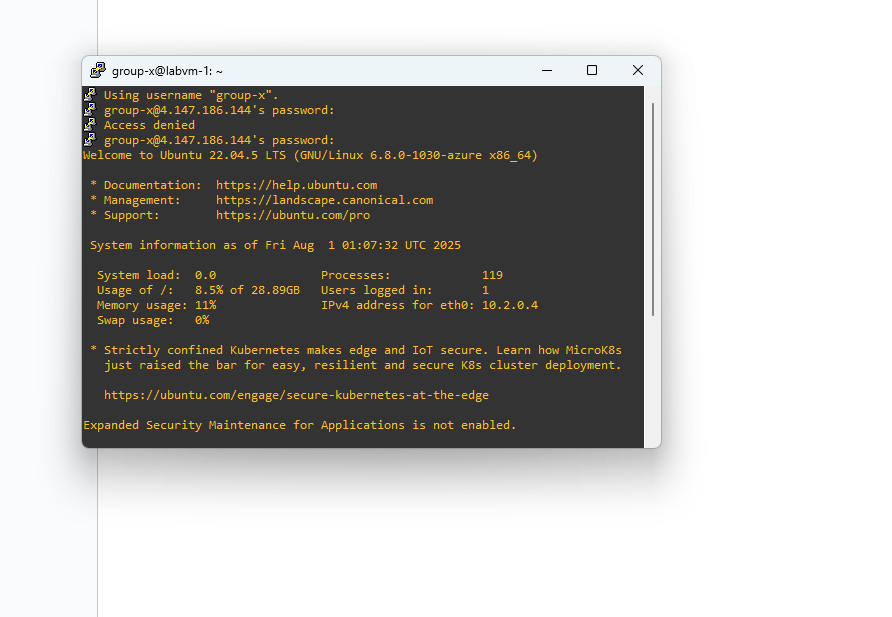
| **My VM Public IP address was provided by the lecturer directly, as login to the Azure portal was not available.**  **: 4.147.186.144** |
| --- |





## **Section #1**

### **2. Connect Using PuTTY**



| **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**

| **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.** |
| --- |

## 

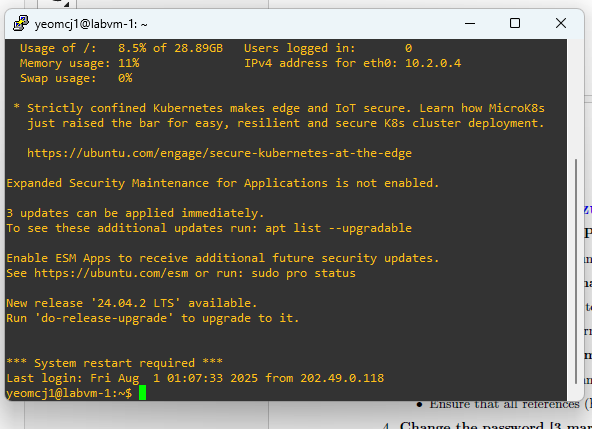
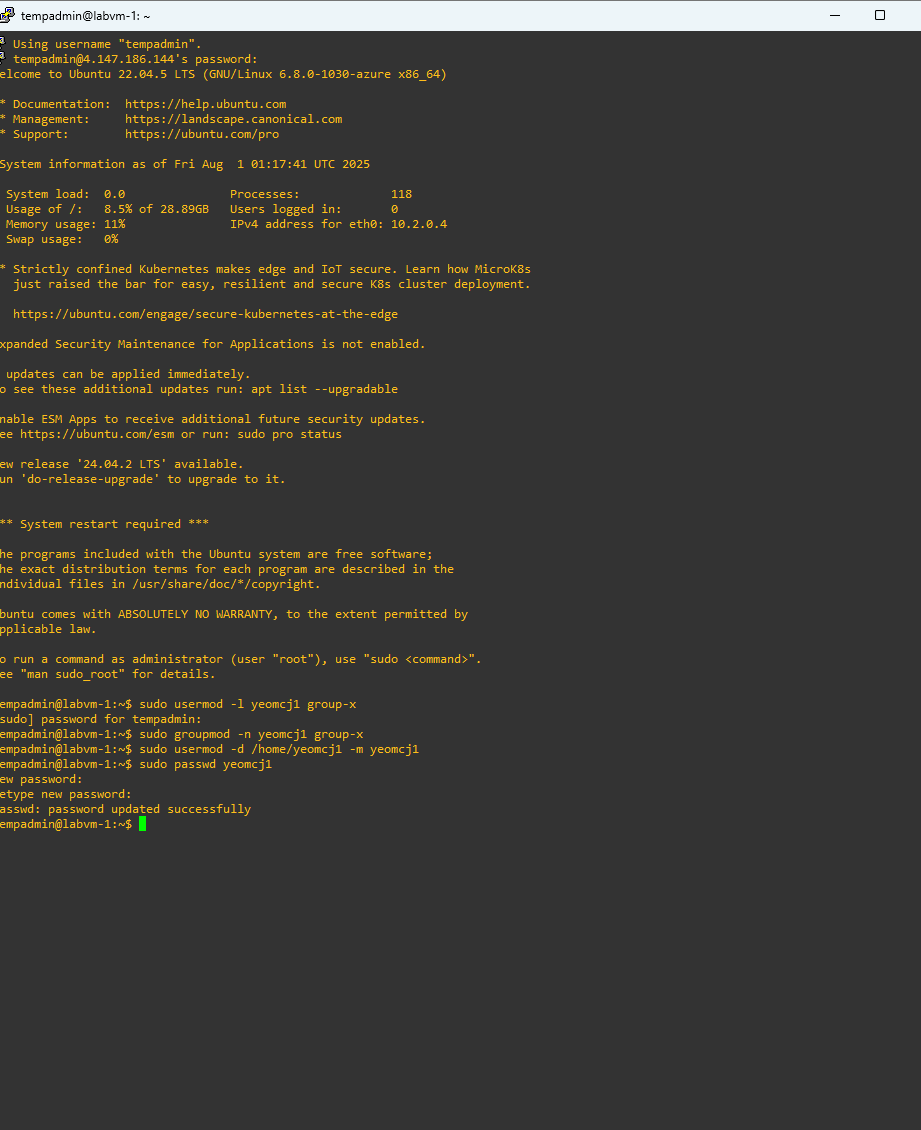
## **Section #1**

### **3. Change the Account Username (2)**

| 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:**  sudo usermod -l yeomcj1 group-x # Rename the user  sudo groupmod -n yeomcj1 group-x # Rename the group  sudo usermod -d /home/yeomcj1 -m yeomcj1 # Move and rename home directory |
| --- |

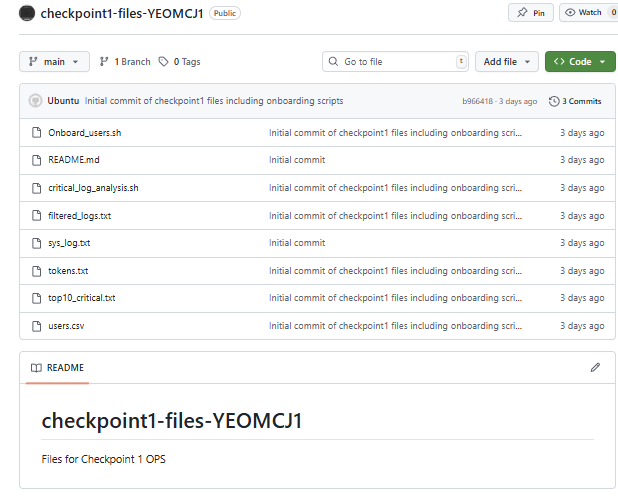
## **Section #1**

### **4. Change the Account Password**



| **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.** |
| --- |

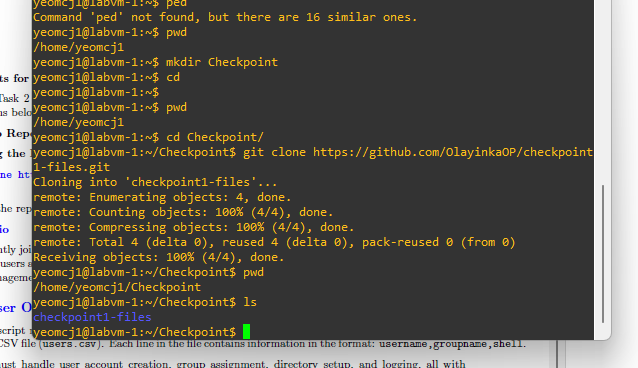
## **Section #2**

User Onboarding Automation

| **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**

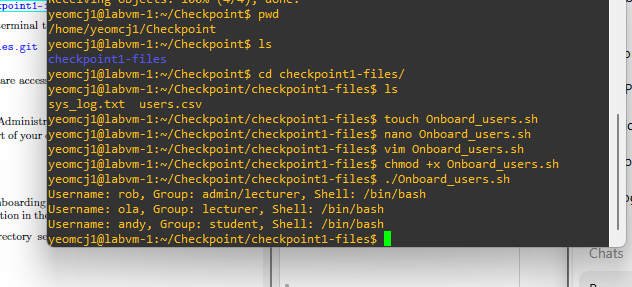
### **Setup & Organization Via Github**



| 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**

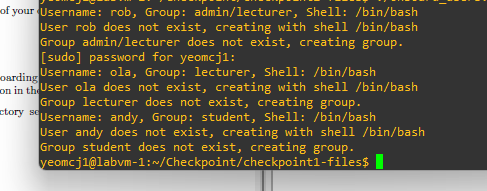
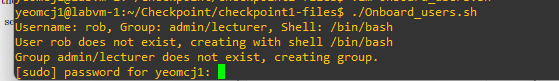
### **1. Read and parse users.csv**



| 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**

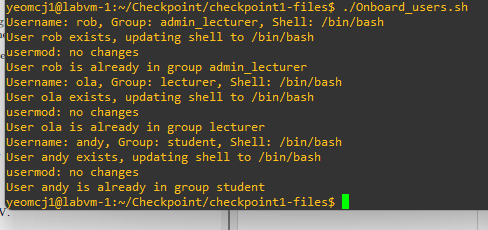
### **2. User Account Management**



| 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**



| 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**

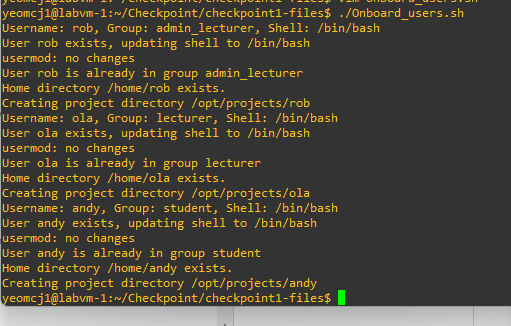
### 

### **4. Home Directory Setup**

| 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**

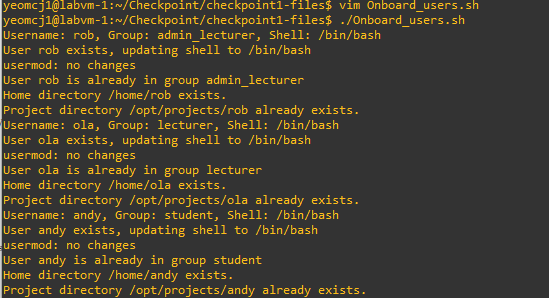
### **5. Create a project directory for the user**



| 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**

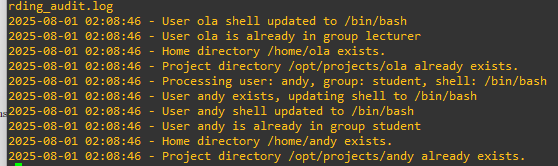


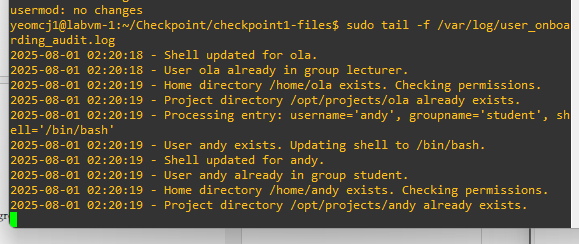
| To enhance traceability and auditing, I added a logging function that records all key actions taken by the script. Each log entry includes:  A **timestamp**  A **custom message** describing the action performed  The logs are written to:  /var/log/user\_onboarding\_audit.log  This makes it easy to review 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





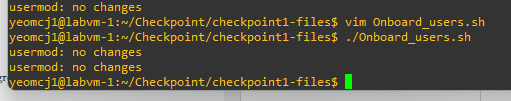
Screenshots reflect the changes explained above providing the logs for the users/files.

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**

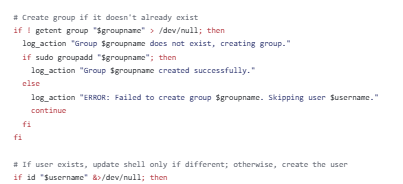
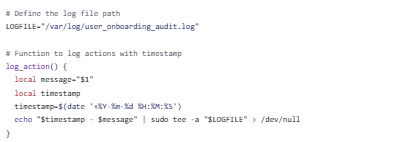


| To meet the requirement of making the script resilient and reliable, I implemented error handling and input validation throughout the user onboarding process. This ensures the script continues processing even if individual entries fail, and that all input data is properly validated.  To prevent the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

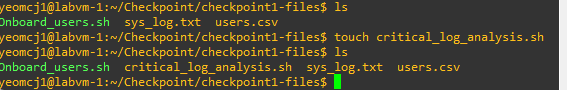
**8. Add inline comments to explain the logic**

| 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 |
| --- |

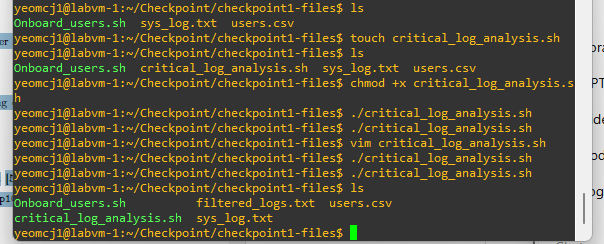


## **Section #3**

Critical Log Analysis



**1. Filter critical logs using keywords**

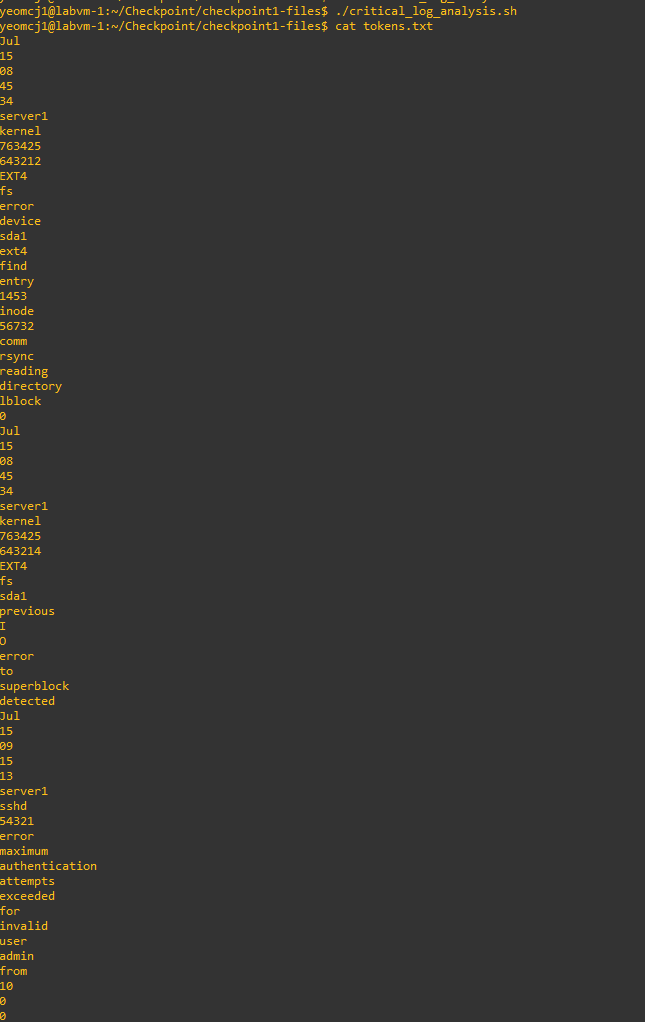
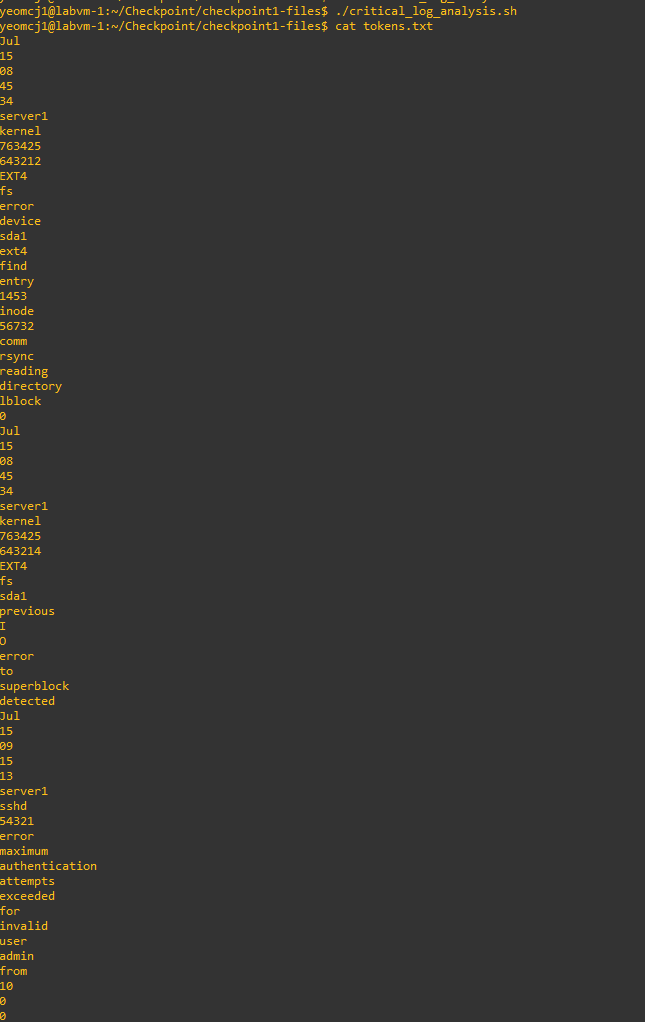


| To identify high-priority issues in system logs, I created a script that filters for critical log entries based on specific keywords.  touch critical\_log\_analysis.sh  chmod +x critical\_log\_analysis.sh  The following command is used to extract only the most important log messages:  grep -iE 'ERROR|CRITICAL|FATAL' 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**

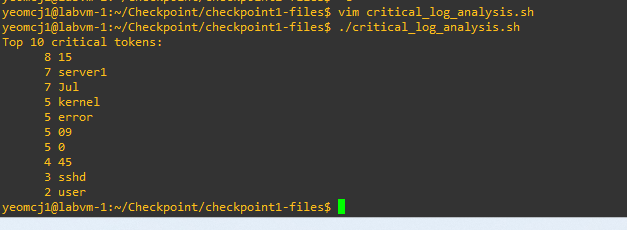
| To prepare the filtered log data for deeper analysis, I tokenized the text—converting it into individual words (tokens) that can be counted or analyzed further.  The command used:  tr -s '[:space:][:punct:]' '\n' < filtered\_logs.txt | grep -v '^$' > tokens.txt  The file tokens.txt contains one word 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 |
| --- |



## **Section #3**

**3. Count and Sort Tokens**

Count frequency and output top 10 tokens

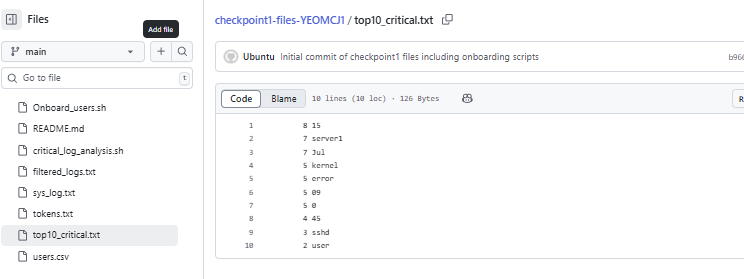


| To analyze the most common words in critical logs, I implemented a pipeline that counts how often each token appears and sorts them in descending order.  sort tokens.txt | uniq -c | sort -nr | head -n 10 > top10\_critical.txt **Output:** The terminal displays the top 10 most frequent words, and the 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 |
| --- |

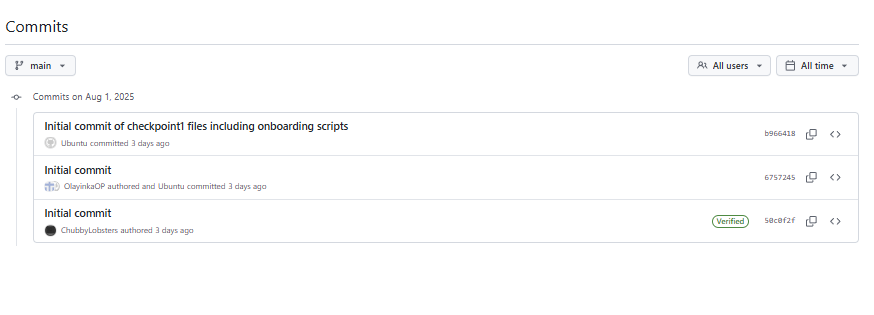


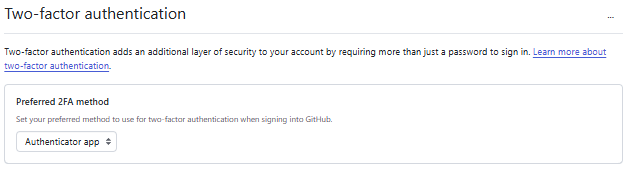
## **Conclustion**

## 

## **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. |
| --- |

****

****