**cloud computing**

**LAB 06**

**Name:** Abiha Nadeem

**Roll no:** 2023-BSE-001

**Submitted to:** Engr. Muhammad Shoaib

**Task 1 – Switch to root with su - and back to a normal user**

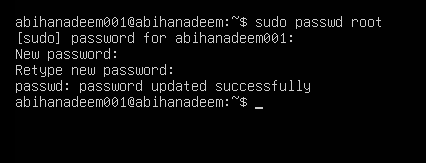
Goal: Demonstrate switching to the root account using su - and exiting back to your normal user.

1. Set a root password (Ubuntu root is disabled by default; this enables su - temporarily for the lab):

sudo passwd root

# Enter a temporary root password for the lab

* Save screenshot as: task1\_set\_root\_password.png



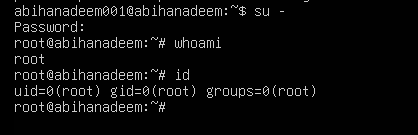
1. Switch to root and verify:

su -

whoami

id

* Save screenshot as: task1\_su\_root.png

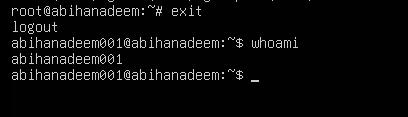


1. Switch back to your normal user:

exit

whoami

* Save screenshot as: task1\_exit\_to\_user.png



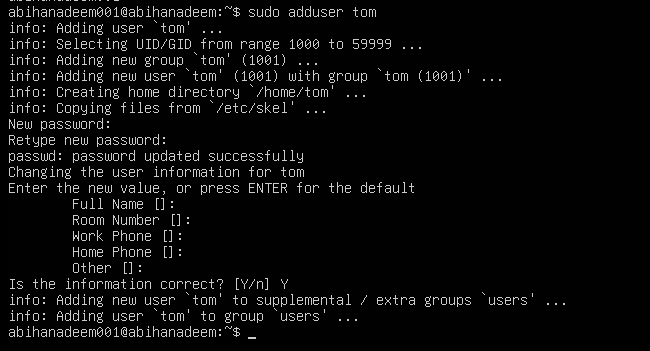
**Task 2 – Create user tom and verify in passwd/group/shadow**

Goal: Create a user named tom, then verify the account in system files.

1. Create user tom (interactive, sets password and home directory):

sudo adduser tom

* Save screenshot as: task2\_adduser\_tom.png



1. Verify tom in system files (view and visually confirm presence):

cat /etc/passwd

* Save screenshot as: task2\_verify\_passwd.png



cat /etc/group

* Save screenshot as: task2\_verify\_group.png



sudo cat /etc/shadow

* Save screenshot as: task2\_verify\_shadow.png

Notes: /etc/shadow stores password hashes (not plaintext). You must use sudo to read it.



**Task 3 – Create groups; change tom’s primary and secondary groups**

Goal: Create groups developer, devops, and designer. Change tom’s primary group and manage secondary groups.

1. Create groups and verify by viewing /etc/group (visually confirm entries exist):

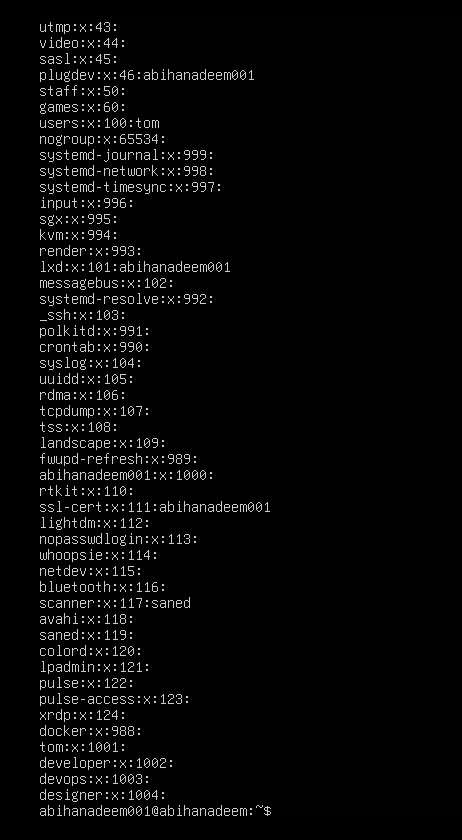
sudo groupadd developer

sudo groupadd devops

sudo groupadd designer

cat /etc/group

* Save screenshot as: task3\_groupadd.png

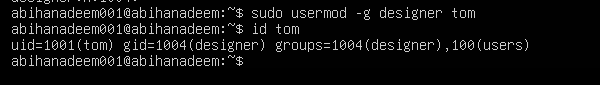


1. Change tom’s primary group to designer and verify:

sudo usermod -g designer tom

id tom

* Save screenshot as: task3\_change\_primary\_group.png



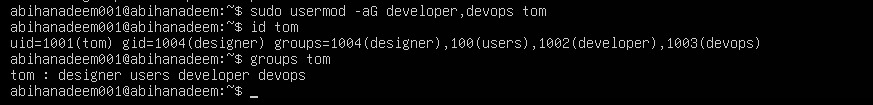
1. Add secondary groups developer and devops to tom and verify:

sudo usermod -aG developer,devops tom

id tom

groups tom

* Save screenshot as: task3\_add\_secondary\_groups.png



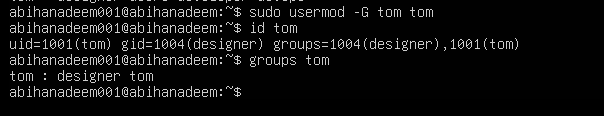
1. Replace all secondary groups so only tom (user’s own group) remains and verify:

sudo usermod -G tom tom

id tom

groups tom

* Save screenshot as: task3\_reset\_secondary\_groups.png



**Task 4 – Create/delete users (Jerry, Scooby) and groups (jolly, anime)**

Goal: Create users using both adduser and useradd, demonstrate login/password/home directory differences, then delete users/groups.

1. Create users:

sudo adduser Jerry

sudo useradd Scooby

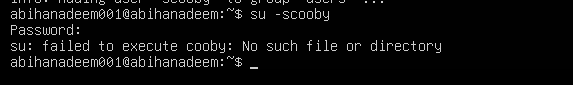
* Save screenshot as: task4\_add\_users.png



1. Try to log in as Scooby immediately (expected authentication failure because there is no password yet):

su - Scooby

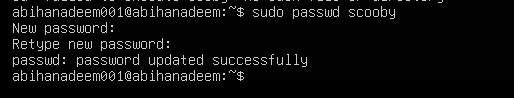
* Save screenshot as: task4\_scooby\_su\_auth\_failure.png



1. Set a password for Scooby:

sudo passwd Scooby

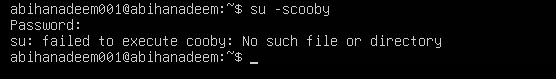
* Save screenshot as: task4\_set\_password\_scooby.png



1. Try logging in as Scooby again (home directory still missing; expect a message such as “No directory, logging in with HOME=/”):

su - Scooby

* Save screenshot as: task4\_scooby\_su\_no\_home.png



1. Show that Scooby’s home directory does not exist yet and what /etc/passwd says:

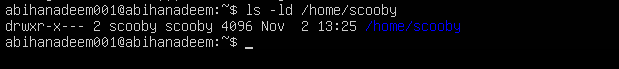
exit

cat /etc/passwd

ls -ld /home/Scooby

* Save screenshot as: task4\_scooby\_no\_home.png





1. Manually create Scooby’s home directory and set proper ownership and permissions:

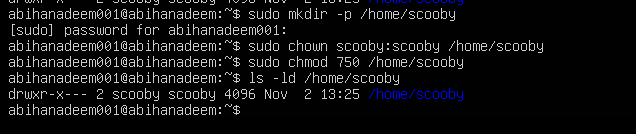
sudo mkdir -p /home/Scooby

sudo chown Scooby:Scooby /home/Scooby

sudo chmod 750 /home/Scooby

ls -ld /home/Scooby

* Save screenshot as: task4\_scooby\_create\_home.png



1. Log in as Scooby again and verify you land in the correct home directory:

su - Scooby

pwd

ls -la

* Save screenshot as: task4\_scooby\_login\_success.png



1. Verify users in system files and observe shell of Scooby:

exit

cat /etc/passwd

* Save screenshot as: task4\_verify\_users.png

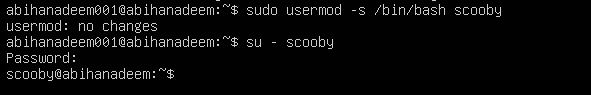


1. Change the shell from /bin/sh to /bin/bash

sudo usermod -s /bin/bash Scooby

su - Scooby

* Save screenshot as: task4\_shell\_switching.png

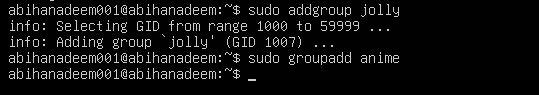


1. Create groups:

sudo addgroup jolly

sudo groupadd anime

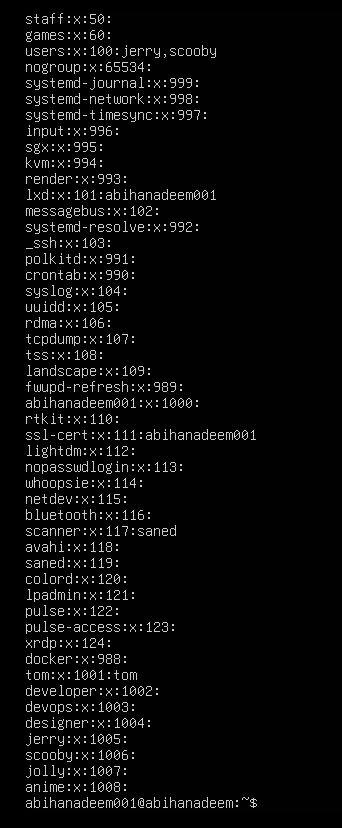
* Save screenshot as: task4\_add\_groups.png



1. Verify groups:

cat /etc/group

* Save screenshot as: task4\_verify\_groups.png

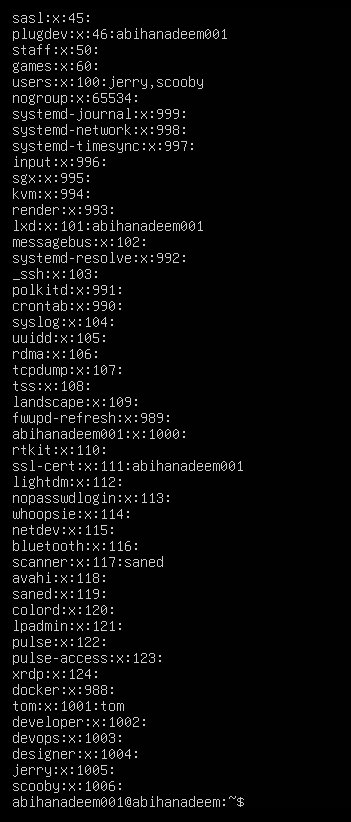


1. Delete groups and users:

sudo delgroup jolly

sudo groupdel anime

cat /etc/group



sudo deluser --remove-home Jerry

sudo userdel -r Scooby

cat /etc/passwd

* Save screenshots as: task4\_delete\_groups.png, task4\_delete\_users.png

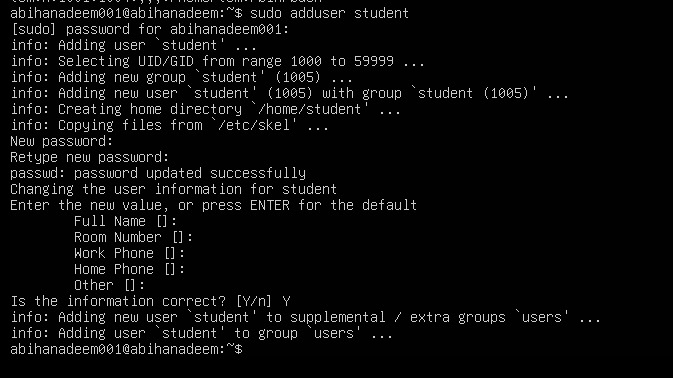


**Task 5 – Create user Student; create files; set owner/group; identify file types**

1. Create Student:

sudo adduser Student

* Save screenshot as: task5\_create\_student.png



1. Switch to Student and create files:

su - Student

touch file1

mkdir -p dir1

touch dir1/file2

ls -l

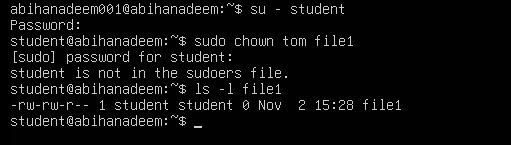
* Save screenshot as: task5\_create\_files.png



1. Change owner then group for file1 (separate commands):

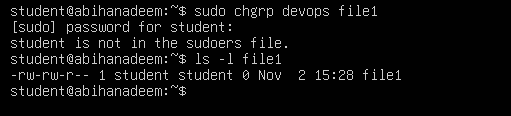
sudo chown tom file1

ls -l file1



sudo chgrp devops file1

ls -l file1



* Save screenshots as: task5\_chown\_file1.png, task5\_chgrp\_file1.png

1. Identify files/directories and show /dev/null:

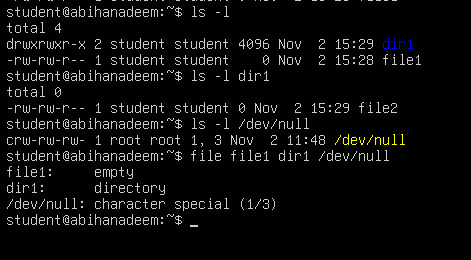
ls -l

ls -l dir1

ls -l /dev/null

file file1 dir1 /dev/null

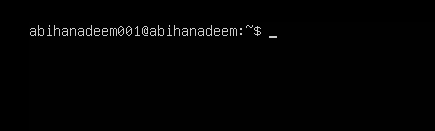
* Save screenshot as: task5\_file\_types.png



1. Exit Student:

exit

* Save screenshot as: task5\_exit\_student.png



**Task 6 – Change permissions using symbolic mode**

Target file: ~/file1 (run these as the Student user)

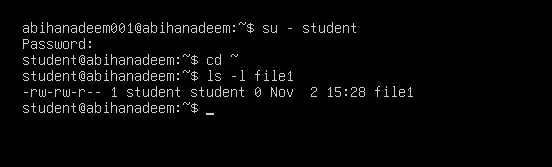
1. Ensure Student and file present:

su - Student

cd ~

ls -l file1

* Save screenshot as: task6\_su\_student.png

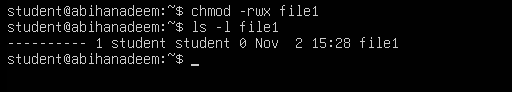


1. Remove all permissions:

chmod -rwx file1

ls -l file1

* Save screenshot as: task6\_chmod\_remove\_rwx.png

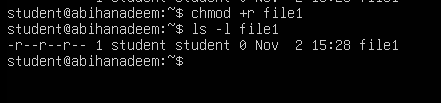


1. Add read to all:

chmod +r file1

ls -l file1

* Save screenshot as: task6\_chmod\_add\_r.png

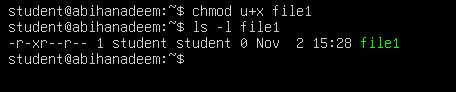


1. Add execute to user:

chmod u+x file1

ls -l file1

* Save screenshot as: task6\_chmod\_u\_plus\_x.png

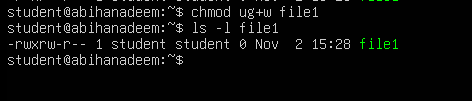


1. Add write to user and group:

chmod ug+w file1

ls -l file1

* Save screenshot as: task6\_chmod\_ug\_plus\_w.png

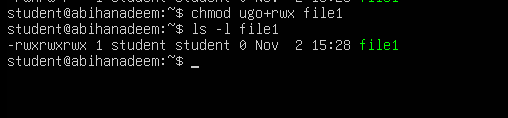


1. Remove all permissions (explicit):

chmod ugo-rwx file1

ls -l file1

* Save screenshot as: task6\_chmod\_ugo\_minus\_rwx.png



**Task 7 – Change permissions using “set” symbolic form (u= g= o=)**

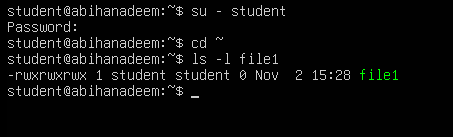
Ensure you are Student:

su - Student

cd ~

ls -l file1

* Save screenshot as: task7\_student\_context.png

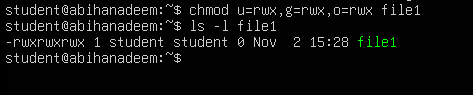


1. Set all to rwx:

chmod u=rwx,g=rwx,o=rwx file1

ls -l file1

* Save screenshot as: task7\_chmod\_set\_all\_rwx.png

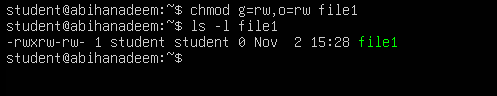


1. Remove execute from group and others:

chmod g=rw,o=rw file1

ls -l file1

* Save screenshot as: task7\_remove\_exec\_go.png

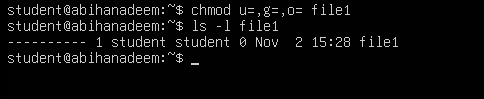


1. Remove all permissions:

chmod u=,g=,o= file1

ls -l file1

* Save screenshot as: task7\_remove\_all\_perms.png



**Task 8 – Change permissions using numeric (octal) mode**

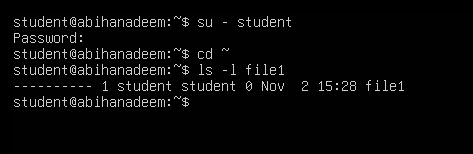
Ensure you are Student:

su - Student

cd ~

ls -l file1

* Save screenshot as: task8\_student\_context.png

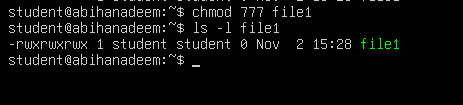


Run each command and capture screenshot after each ls:

chmod 777 file1

ls -l file1

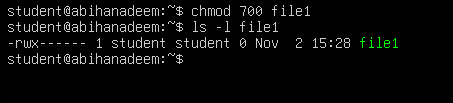
* task8\_chmod\_777.png



chmod 700 file1

ls -l file1

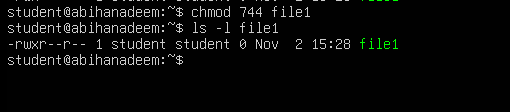
* task8\_chmod\_700.png



chmod 744 file1

ls -l file1

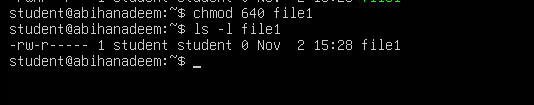
* task8\_chmod\_744.png



chmod 640 file1

ls -l file1

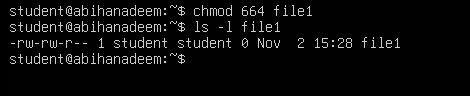
* task8\_chmod\_640.png



chmod 664 file1

ls -l file1

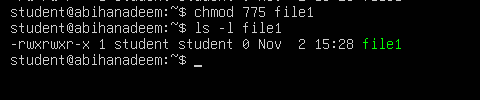
* task8\_chmod\_664.png



chmod 775 file1

ls -l file1

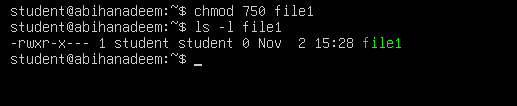
* task8\_chmod\_775.png



chmod 750 file1

ls -l file1

* task8\_chmod\_750.png



**Task 9 – Practice pipes, pagers, grep, and redirects with /var/log/syslog**

1. less:

sudo cat /var/log/syslog | less

# quit q

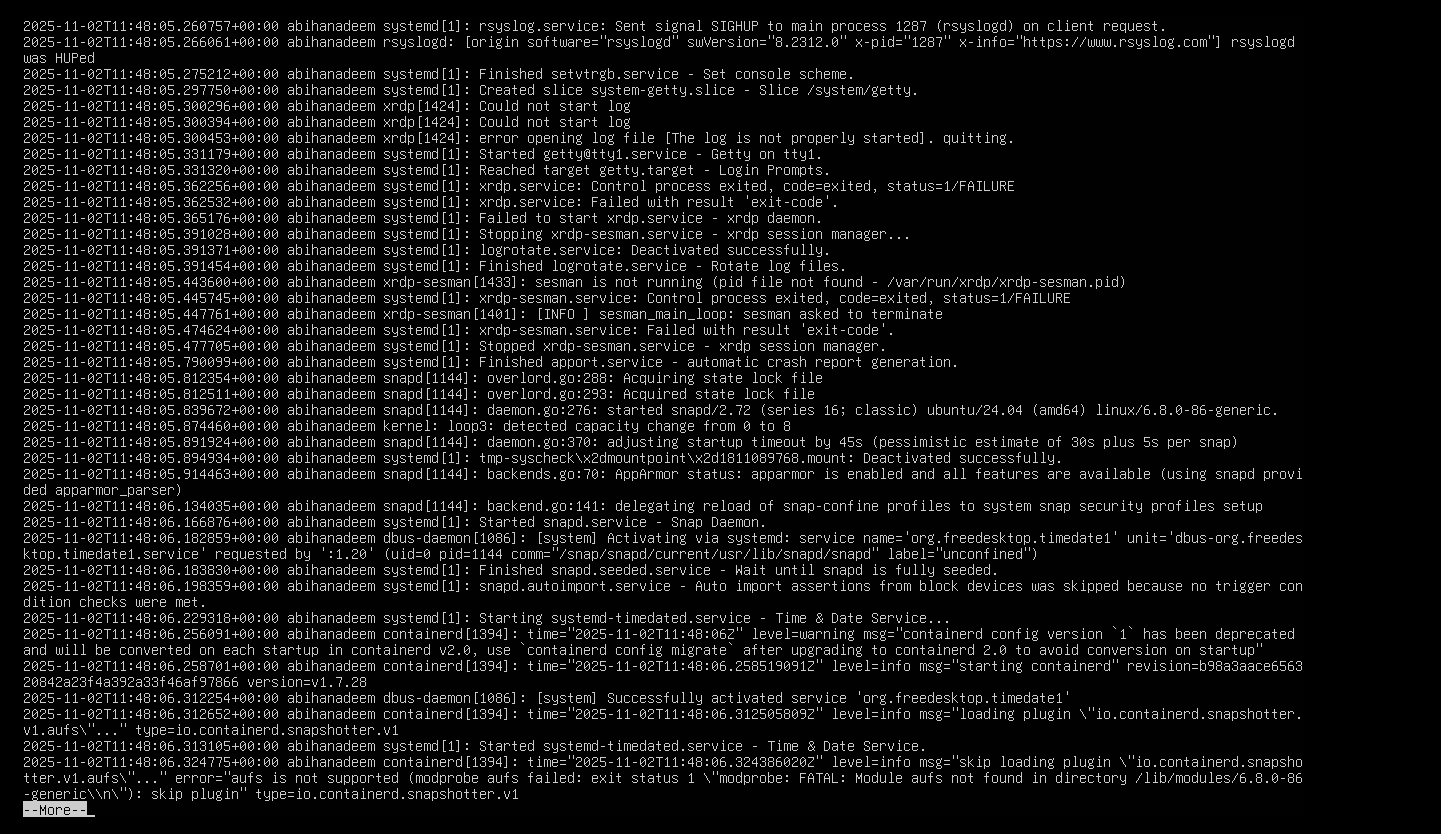
* task9\_grep\_less.png



1. more:

sudo cat /var/log/syslog | more

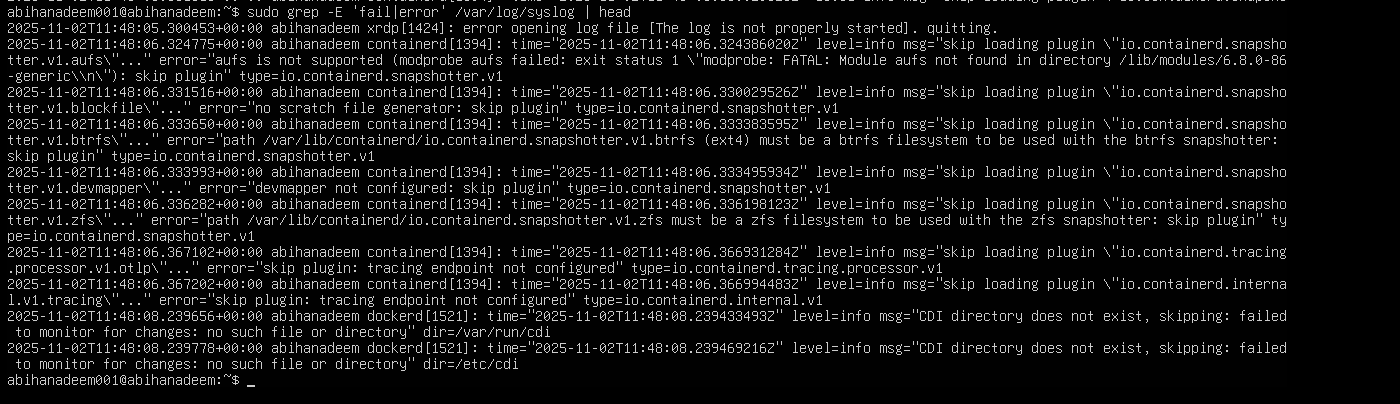
* task9\_grep\_more.png



1. grep failures/errors:

sudo grep -E 'fail|error' /var/log/syslog | head

* task9\_grep\_head.png



1. redirect:

sudo grep -i systemd /var/log/syslog > ~/syslog\_systemd.txt

* task9\_redirect\_overwrite.png

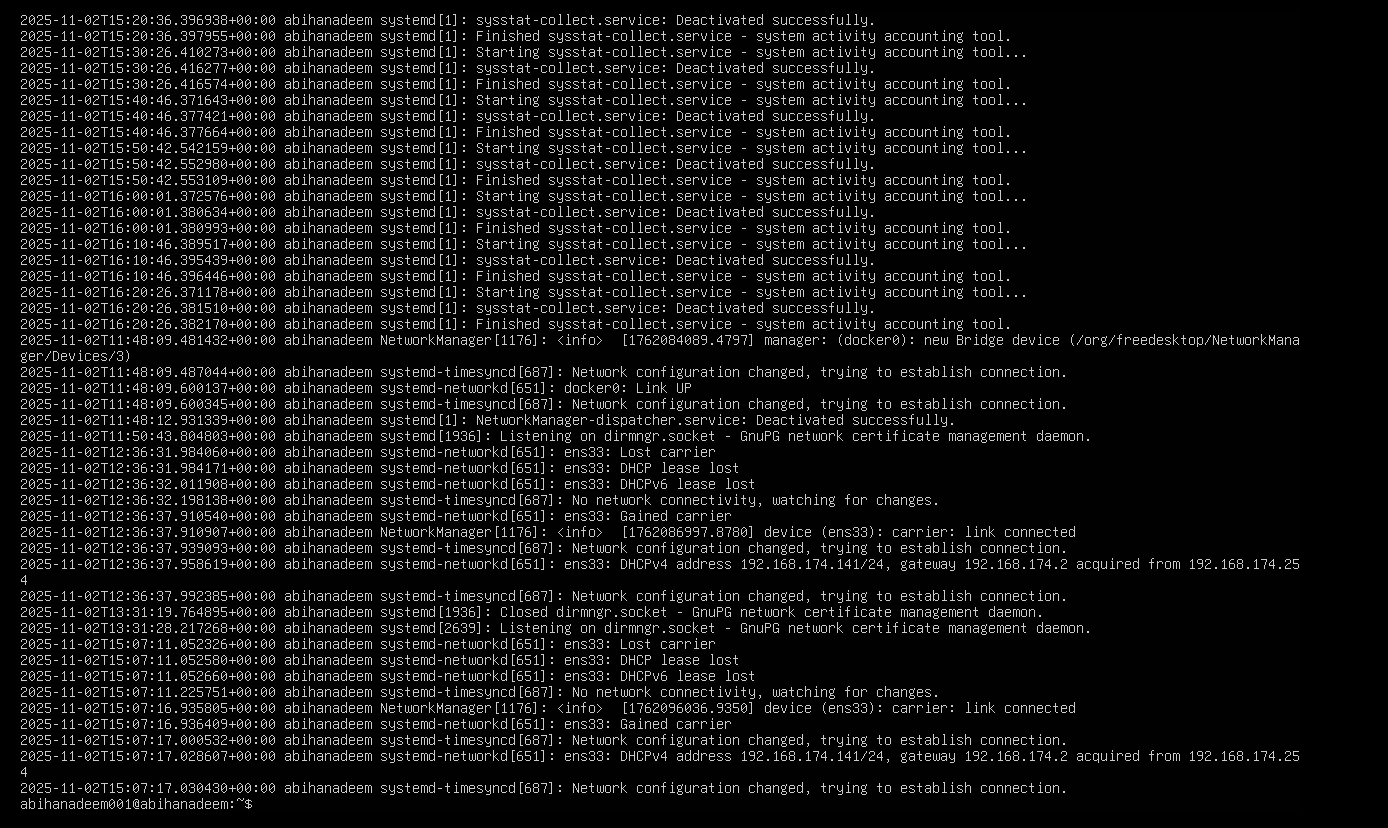


append:

sudo grep -i network /var/log/syslog >> ~/syslog\_systemd.txt

cat ~/syslog\_systemd.txt

* task9\_redirect\_append.png

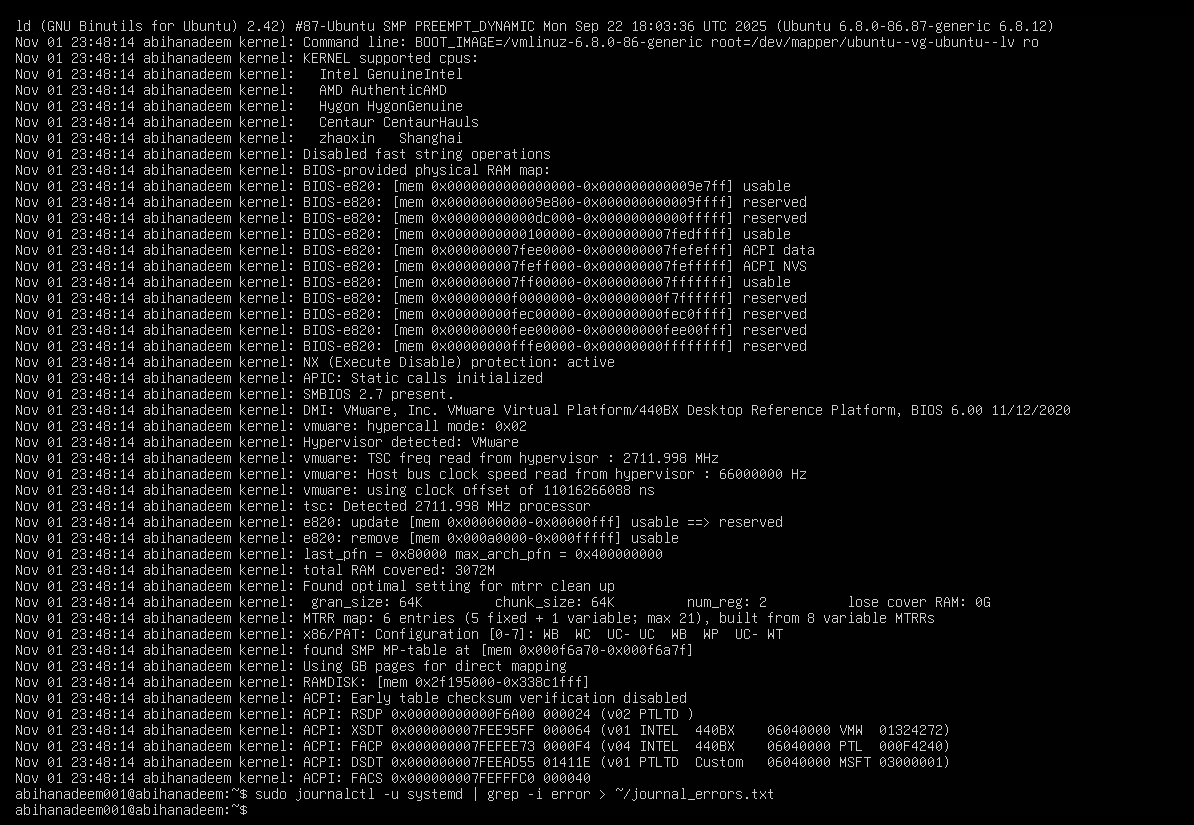


Alternative (journalctl) if needed:

sudo journalctl | less

sudo journalctl -u systemd | grep -i error > ~/journal\_errors.txt

* task9\_journalctl\_alternative.png



**Task 10 – Script setup.sh – variables, command substitution, file/dir checks, permissions (use vim)**

Goal: Using vim, write a script named setup.sh that implements each numbered step below. After writing the code for each step, run the script and capture screenshots showing the vim editor (script content) and the script output for that step. Students must add the code for each step into the same file setup.sh step-by-step (i.e., write 1., save, run and screenshot; then append 2., save, run and screenshot; and so on).

For each step you MUST:

* Open vim and edit setup.sh
* Insert only the code shown for that step (append to the existing file)
* Save and quit vim (:wq)
* Make the file executable if not already: chmod +x setup.sh
* Run the script: ./setup.sh
* Capture two screenshots:
  + One showing the vim editor with the script content after you added the step (use the vim screen before :wq)
  + One showing the terminal output after running the script (show the command and the output)

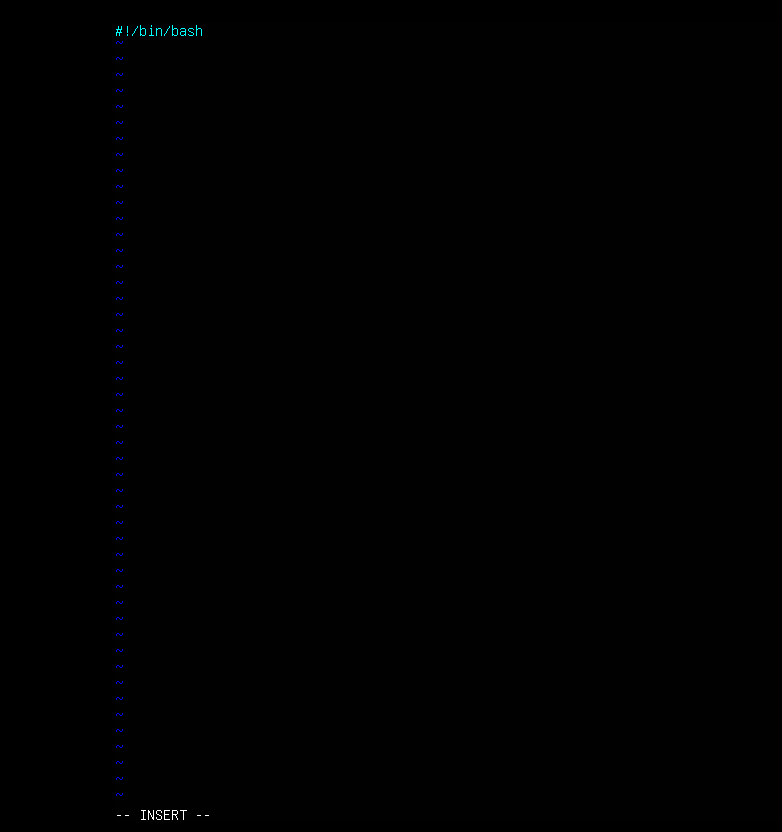
Start in your Student home directory (recommended).

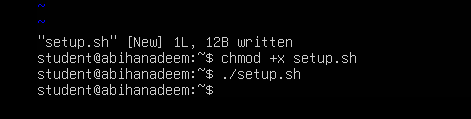
1. Include bash shebang

* Code to add (enter in vim as the first line of the file):

#!/bin/bash

* Steps:
  1. vim setup.sh → add the shebang line → save and quit
  2. chmod +x setup.sh
  3. ./setup.sh
* Screenshots:
  1. vim editor showing the shebang: task10\_b1\_vim.png
  2. script run output (likely no output but show ./setup.sh run): task10\_b1\_run.png





1. Define variable var1 and echo it

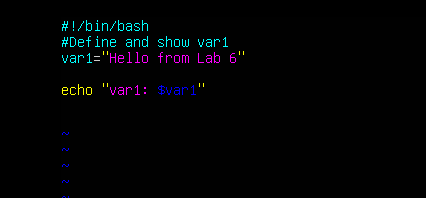
* Code to append:

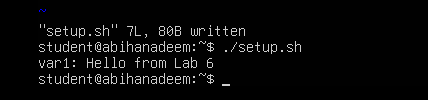
# Define and show var1

var1="Hello from Lab 6"

echo "var1: $var1"

* Steps:
  1. vim setup.sh → append the code above → save and quit
  2. ./setup.sh
* Screenshots:
  1. vim editor showing var1 code appended: task10\_b2\_vim.png
  2. script run output showing var1 printed: task10\_b2\_run.png





1. Save output of ls -l into variable allFiles and echo it

* Code to append:

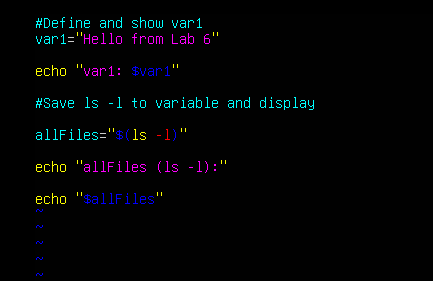
# Save ls -l to variable and display

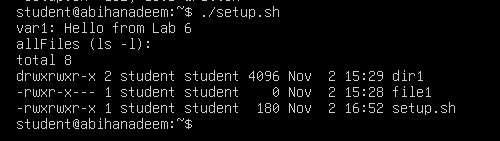
allFiles="$(ls -l)"

echo "allFiles (ls -l):"

echo "$allFiles"

* Steps:
  1. vim setup.sh → append the code above → save and quit
  2. ./setup.sh
* Screenshots:
  1. vim editor showing allFiles code appended: task10\_b3\_vim.png
  2. script run output showing the ls -l content echoed: task10\_b3\_run.png





1. If directory dir1 exists echo a message; else create it

* Code to append:

# Directory check

if [ -d "dir1" ]; then

echo "Directory dir1 exists."

else

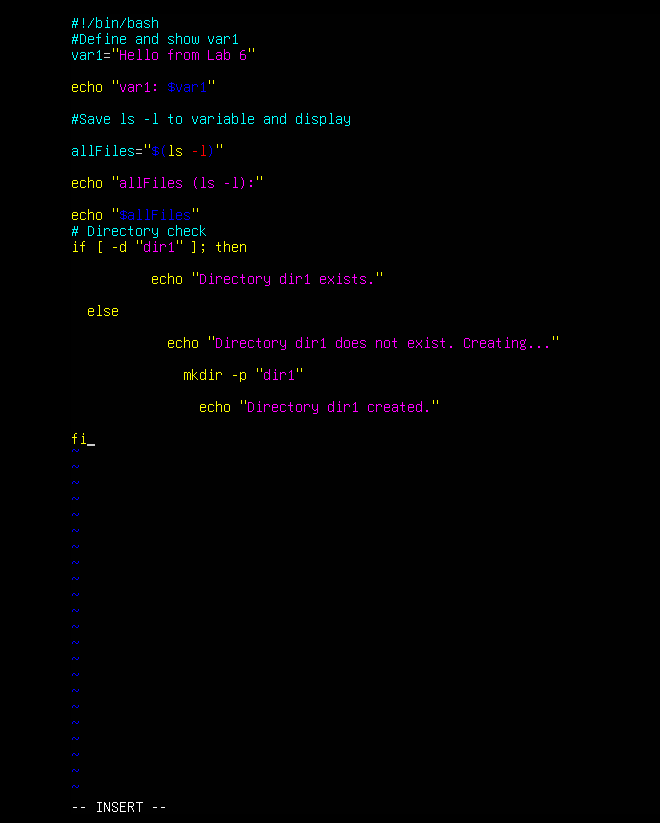
echo "Directory dir1 does not exist. Creating..."

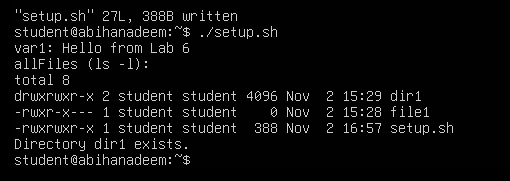
mkdir -p "dir1"

echo "Directory dir1 created."

fi

* Steps:
  1. vim setup.sh → append the code above → save and quit
  2. ./setup.sh
* Screenshots:
  1. vim editor showing dir1 check code: task10\_b4\_vim.png
  2. script run output showing directory message or creation: task10\_b4\_run.png





1. If file dir1/file2 does not exist, create it

* Code to append:

# File check

if [ -f "dir1/file2" ]; then

echo "file2 already exists."

else

echo "file2 does not exist. Creating..."

touch "dir1/file2"

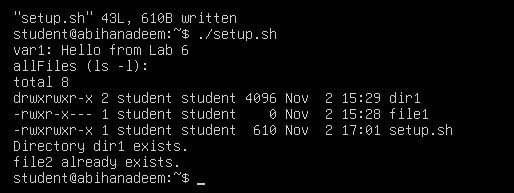
chmod a-rwx "dir1/file2"

echo "file2 created."

fi

* Steps:
  1. vim setup.sh → append the code above → save and quit
  2. ./setup.sh
* Screenshots:
  1. vim editor showing file2 check code: task10\_b5\_vim.png
  2. script run output showing file creation message or existence: task10\_b5\_run.png





1. Check read, write, execute permissions on dir1/file2; grant missing user perms and show final ls

* Code to append:

# Permission checks for dir1/file2 (user permissions)

f="dir1/file2"

if [ ! -r "$f" ]; then

echo "Read permission missing; granting to user..."

chmod u+r "$f"

fi

if [ ! -w "$f" ]; then

echo "Write permission missing; granting to user..."

chmod u+w "$f"

fi

if [ ! -x "$f" ]; then

echo "Execute permission missing; granting to user..."

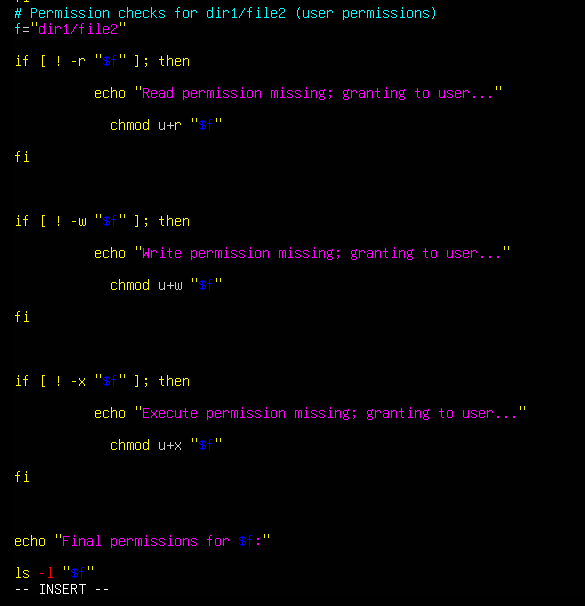
chmod u+x "$f"

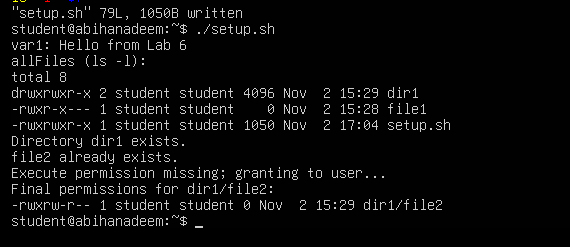
fi

echo "Final permissions for $f:"

ls -l "$f"

* Steps:
  1. vim setup.sh → append the code above → save and quit
  2. ./setup.sh
* Screenshots:
  1. vim editor showing permission-check code: task10\_b6\_vim.png
  2. script run output showing the permission grants and final ls -l dir1/file2: task10\_b6\_run.png





**Task 11 – Script setup.sh – argument comparisons (eq, ne, gt, lt, ge, le) and string checks**

Updated: replace the previous single-script approach with an incremental exercise. Students will overwrite setup.sh and then add each individual if-test one-by-one. After adding each if-test they must run the script with example arguments and capture screenshots. This teaches the individual comparison operators and makes each if statement a separate step.

Important overall instructions

* Start by overwriting setup.sh (vim setup.sh) and add only what the step asks (do not add all tests at once).
* After editing in vim, save (:wq), make executable (chmod +x setup.sh) if needed, then run the script with the example commands shown for each step.
* For each step capture two screenshots:
  + A vim screenshot showing the current file buffer with the newly added lines (before :wq) — name as specified for the step.
  + A terminal screenshot showing the commands you ran (chmod +x setup.sh if necessary) and the script outputs for the example invocations — name as specified for the step.
* For the numeric comparisons, set a variable num=$1 at the top of the file before adding the individual if-tests (this will be the initial step). For string checks, set str=$2 before adding the string if-tests.

1. create file with shebang and set num and str variables

* In vim create/overwrite setup.sh and insert:

#!/bin/bash

num=$1

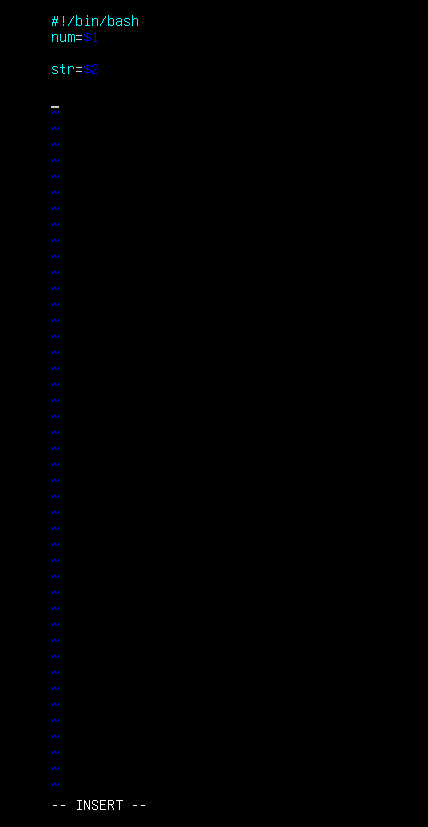
str=$2

* Save and quit (:wq)
* Make executable and run with examples:

chmod +x setup.sh

./setup.sh 10 Student

* Screenshots:
  + vim content: task11\_b0\_vim.png
  + run output: task11\_b0\_run.png





1. add the -eq test (equal)

* Append to setup.sh:

if [ "$num" -eq 10 ]; then

echo "$num is equal to 10 (-eq)."

else

echo "$num is NOT equal to 10 (-eq)."

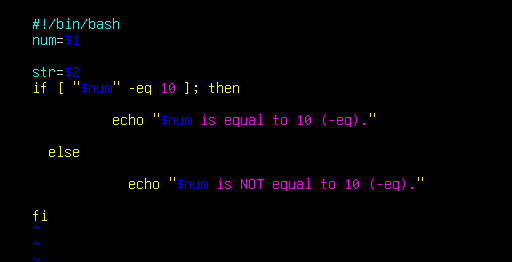
fi

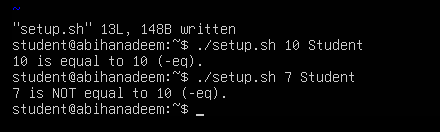
* Save and quit; then run these commands (capture both in one terminal screenshot):

./setup.sh 10 Student

./setup.sh 7 Student

* Screenshots:
  + vim content after edit: task11\_b1\_vim.png
  + run output demonstrating both cases: task11\_b1\_run.png





1. add the -ne test (not equal)

* Append to setup.sh:

if [ "$num" -ne 10 ]; then

echo "$num is not equal to 10 (-ne)."

else

echo "$num is equal to 10 (-ne false)."

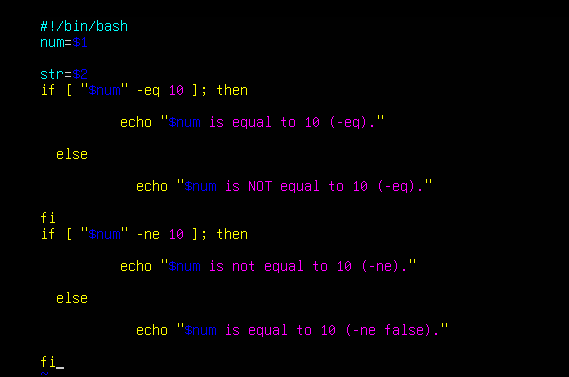
fi

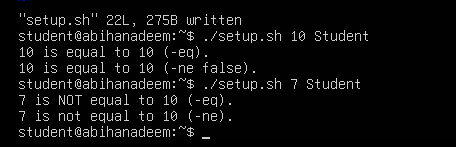
* Save and quit; run:

./setup.sh 7 Student

./setup.sh 10 Student

* Screenshots:
  + vim content: task11\_b2\_vim.png
  + run output: task11\_b2\_run.png





1. add the -gt test (greater than)

* Append:

if [ "$num" -gt 10 ]; then

echo "$num is greater than 10 (-gt)."

else

echo "$num is NOT greater than 10 (-gt)."

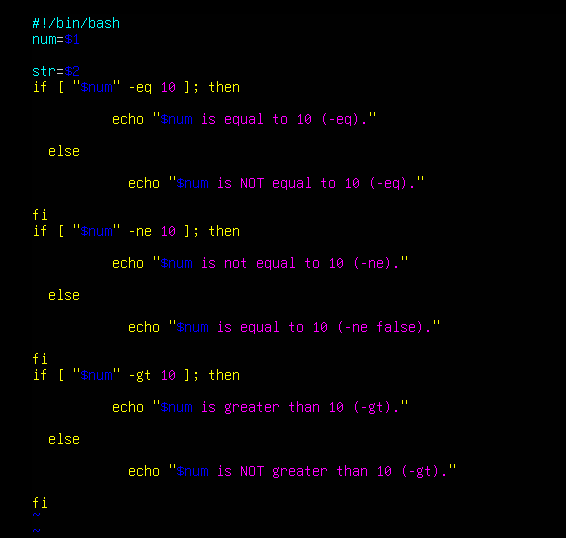
fi

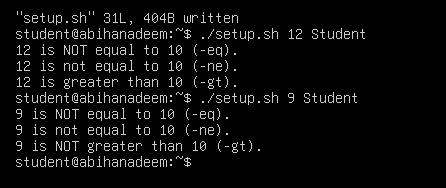
* Run:

./setup.sh 12 Student

./setup.sh 9 Student

* Screenshots:
  + vim content: task11\_b3\_vim.png
  + run output: task11\_b3\_run.png





1. add the -lt test (less than)

* Append:

if [ "$num" -lt 10 ]; then

echo "$num is less than 10 (-lt)."

else

echo "$num is NOT less than 10 (-lt)."

fi

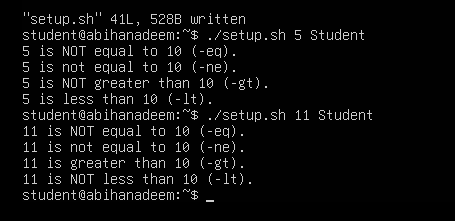
* Run:

./setup.sh 5 Student

./setup.sh 11 Student

* Screenshots:
  + vim content: task11\_b4\_vim.png
  + run output: task11\_b4\_run.png





1. add the -ge test (greater than or equal)

* Append:

if [ "$num" -ge 10 ]; then

echo "$num is greater than or equal to 10 (-ge)."

else

echo "$num is NOT greater than or equal to 10 (-ge)."

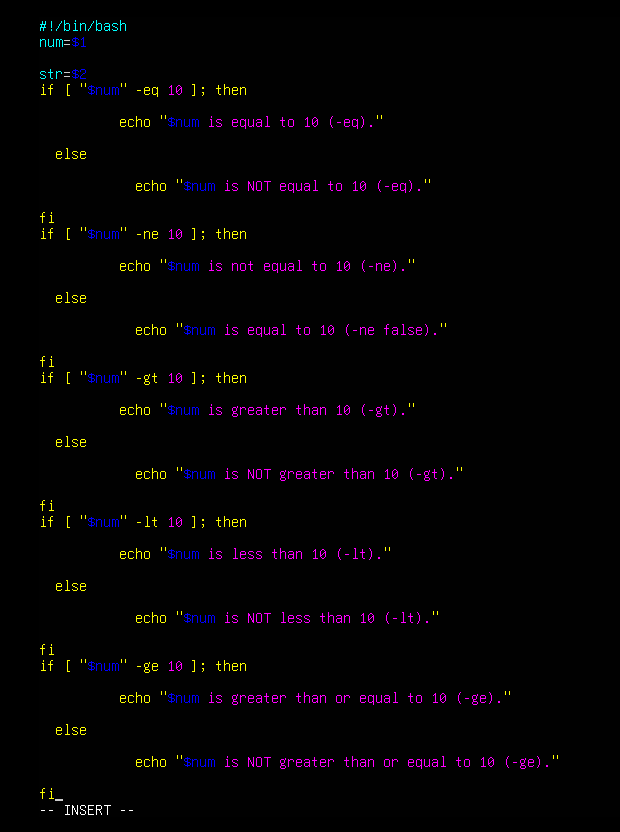
fi

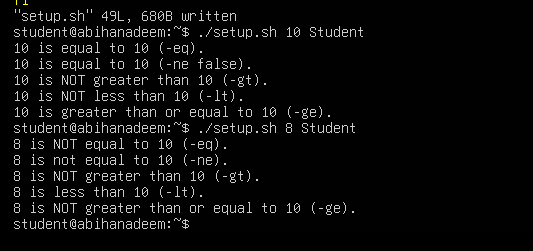
* Run:

./setup.sh 10 Student

./setup.sh 8 Student

* Screenshots:
  + vim content: task11\_b5\_vim.png
  + run output: task11\_b5\_run.png





1. add the -le test (less than or equal)

* Append:

if [ "$num" -le 10 ]; then

echo "$num is less than or equal to 10 (-le)."

else

echo "$num is NOT less than or equal to 10 (-le)."

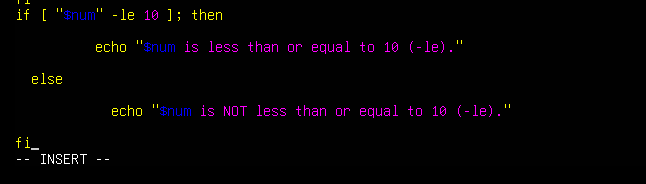
fi

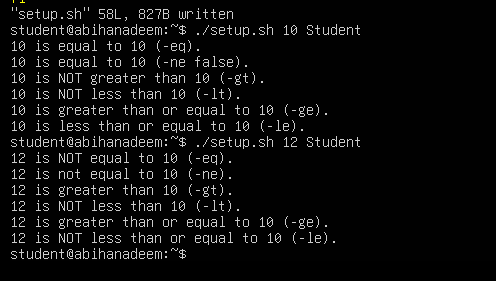
* Run:

./setup.sh 10 Student

./setup.sh 12 Student

* Screenshots:
  + vim content: task11\_b6\_vim.png
  + run output: task11\_b6\_run.png





1. string equality test ( = )

* Ensure str=$2 exists at top (1.). Append:

if [ "$str" = "Student" ]; then

echo "Second argument equals 'Student' ( = )."

else

echo "Second argument does NOT equal 'Student' ( = )."

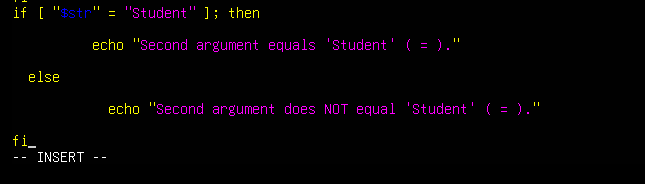
fi

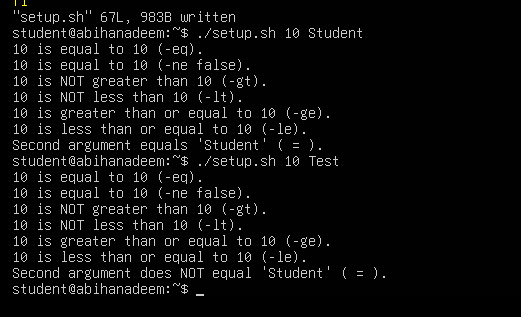
* Run:

./setup.sh 10 Student

./setup.sh 10 Test

* Screenshots:
  + vim content: task11\_b7\_vim.png
  + run output: task11\_b7\_run.png





1. string inequality test ( != )

* Append:

if [ "$str" != "Student" ]; then

echo "Second argument is not equal to 'Student' ( != )."

else

echo "Second argument equals 'Student' ( != false)."

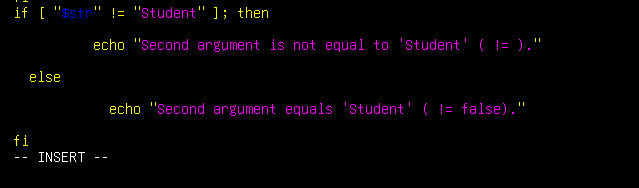
fi

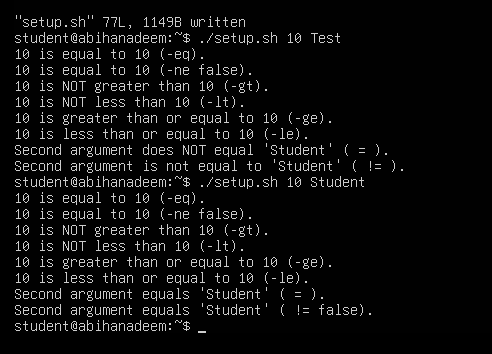
* Run:

./setup.sh 10 Test

./setup.sh 10 Student

* Screenshots:
  + vim content: task11\_b8\_vim.png
  + run output: task11\_b8\_run.png





1. check if second argument is empty (zero-length)

* Append:

if [ -z "$str" ]; then

echo "Second argument is empty (zero-length)."

else

echo "Second argument is not empty."

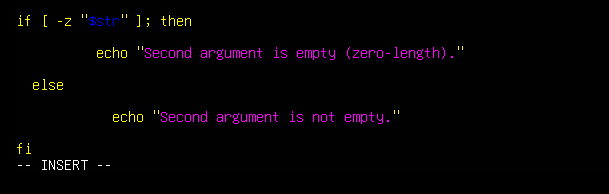
fi

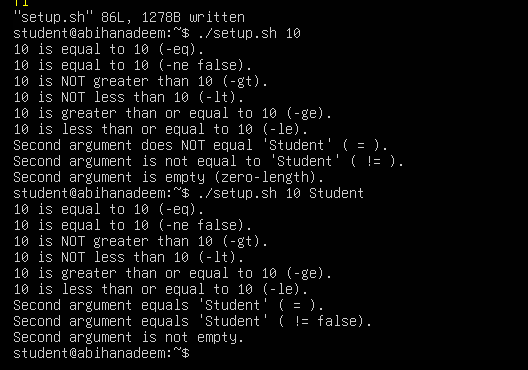
* Run:

./setup.sh 10

./setup.sh 10 Student

* Screenshots:
  + vim content: task11\_b9\_vim.png
  + run output: task11\_b9\_run.png





**Task 12 – Script setup.sh – print all arguments with a for loop**

1. Create the script with shebang and basic structure

* Open vim and overwrite setup.sh:

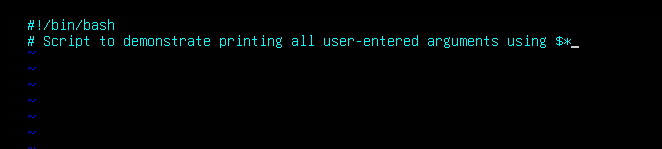
vim setup.sh

* Insert these lines (first step — shebang and a short comment):

#!/bin/bash

# Script to demonstrate printing all user-entered arguments using $\*

* Save and quit (:wq)
* Screenshots:
  + vim editor showing the shebang and comment: task12\_b1\_vim.png
  + run (no output expected but show ./setup.sh run): task12\_b1\_run.png





1. Append the for loop using $\* and print each argument

* Re-open setup.sh in vim and append the following lines:

# Print all arguments using $\*

echo "Printing all arguments using \$\*:"

for arg in $\*; do

echo "Argument: $arg"

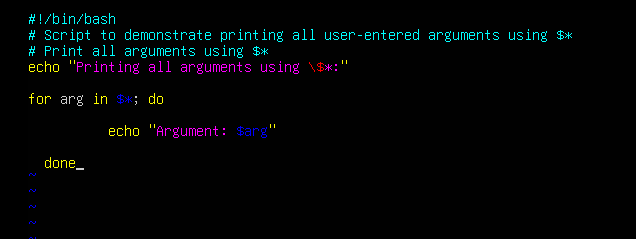
done

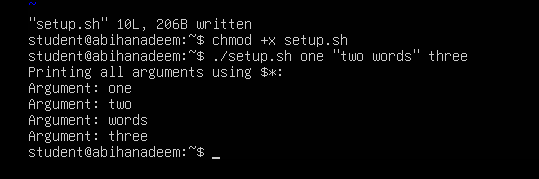
* Save and quit (:wq)
* Make the script executable and run it with example arguments:

chmod +x setup.sh

./setup.sh one "two words" three

* Screenshots:
  + vim editor showing the for-loop appended: task12\_b2\_vim.png
  + script run output showing the printed arguments: task12\_b2\_run.png





**Task 13 – Script setup.sh – while loop summation and functions**

Clear the previous code of setup.sh and write a new script, step-by-step, that:

* Starts with a shebang line
* Implements an interactive while loop that prompts the user to enter numbers and keeps a running total until the user types q to quit; after each input the script echoes "Total Score: <current\_total>"
* Implements a function sum\_two() that runs its own interactive while loop doing the same accumulation and echoes the running totals
* Adds a second function that takes two numeric arguments, sums them, and returns the result via echo (demonstrated in the script)
* Important: if you move the while-loop logic into the sum\_two() function, delete the standalone while-loop code to avoid running the same loop twice

1. Add the shebang line

* Open vim and overwrite setup.sh with the shebang line:

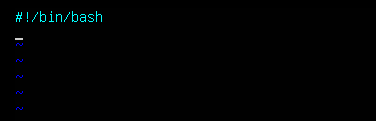
#!/bin/bash

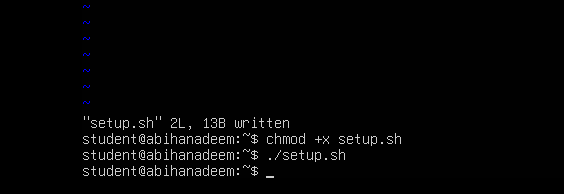
* Save and quit (:wq)
* Make executable and run (no output expected):

chmod +x setup.sh

./setup.sh

* Screenshots:
  + vim editor showing shebang: task13\_b1\_vim.png
  + run output: task13\_b1\_run.png





1. Add the while-loop summation (interactive)

* Re-open setup.sh in vim and append the while-loop:

# While-loop summation (interactive)

sum=0

while true; do

read -p "Enter a number (or 'q' to quit): " input

if [ "$input" = "q" ]; then

break

fi

sum=$((sum + input))

echo "Total Score: $sum"

done

echo "Final total: $sum"

* Save and quit (:wq)
* Run the script and demonstrate a short session (example): enter 5, then 7, then q

./setup.sh

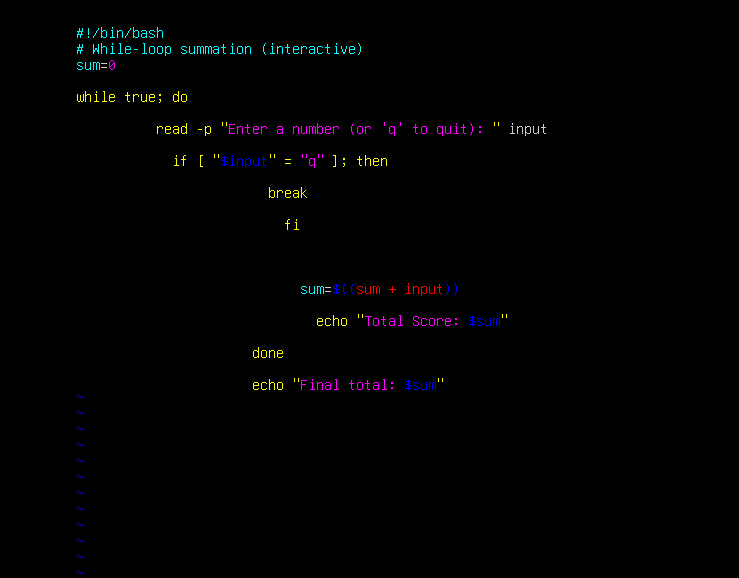
# interactively enter:

# 5

# 7

# q

* Screenshots:
  + vim editor showing while-loop appended: task13\_b2\_vim.png
  + run output showing the interactive session and totals: task13\_b2\_run.png





1. Add the interactive summation function and demonstrate it

* Re-open setup.sh in vim and append the function sum\_two() which contains its own interactive while-loop:

# Function to accumulate scores interactively

sum\_two() {

sum=0

while true; do

read -p "Enter a number (or 'q' to quit): " input

if [ "$input" = "q" ]; then

break

fi

sum=$((sum + input))

echo "Total Score: $sum"

done

echo "Function final total: $sum"

}

# Demonstrate the function

echo "Now calling sum\_two function:"

sum\_two

* Save and quit (:wq)
* Important: If you have the standalone while-loop from step 2 and you place this function into the script, delete the standalone loop to avoid executing the same interactive logic twice when running the script.
* Run the script and demonstrate a short session (example): enter 3, 4, q when prompted by the function:

./setup.sh

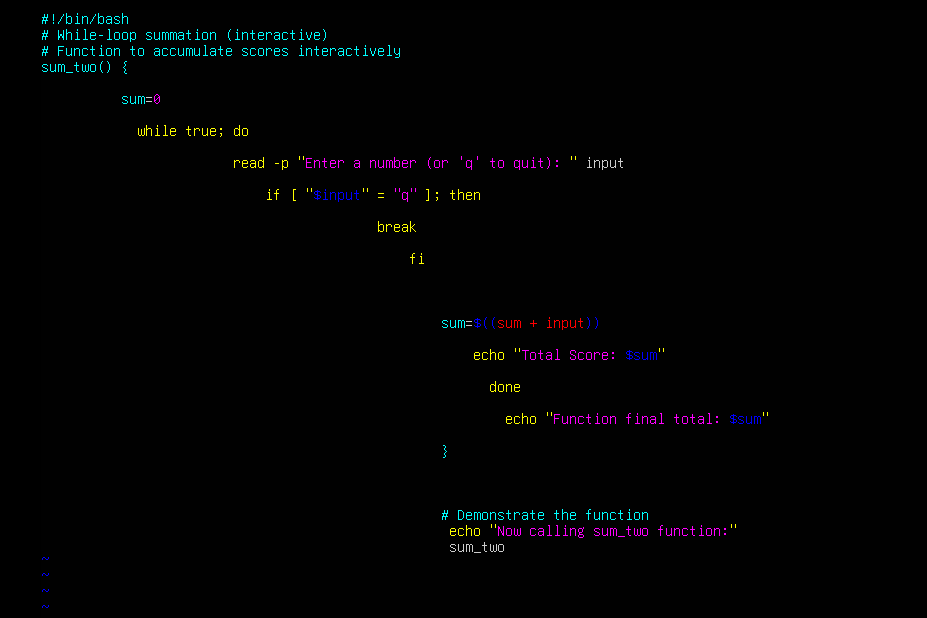
# when prompted by the function enter:

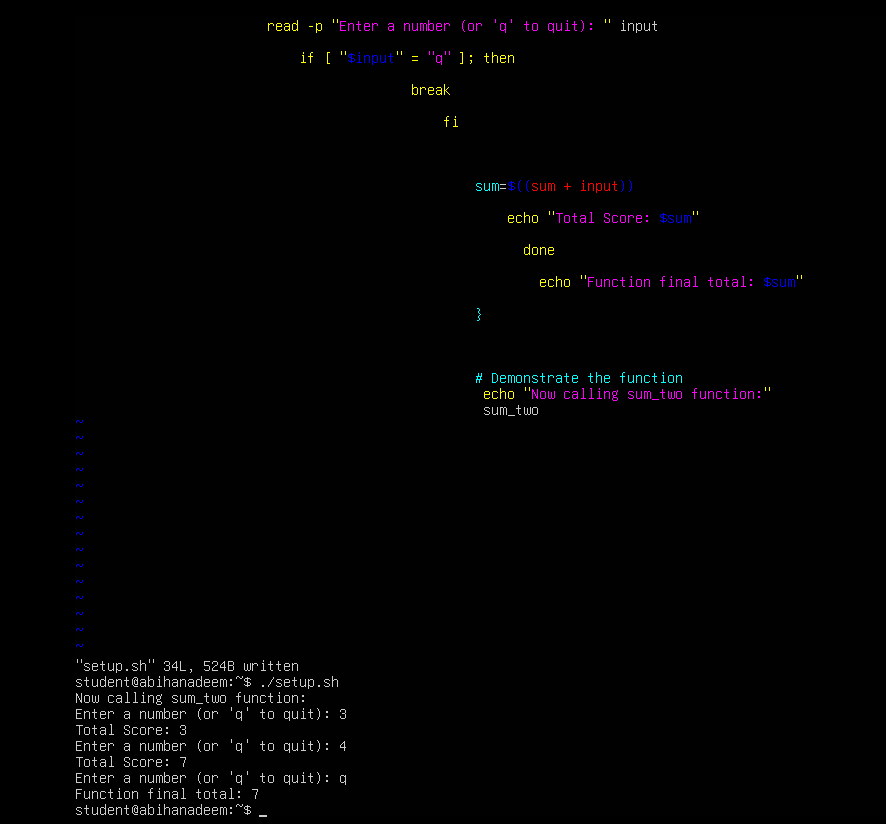
# 3

# 4

# q

* Screenshots:
  + vim editor showing function appended: task13\_b3\_vim.png
  + run output showing the function prompts and final total: task13\_b3\_run.png





1. Add a function that takes two numeric arguments, sums them, and returns the result (echo)

* Re-open setup.sh in vim and append the following function and demonstration. This function accepts two numeric arguments, adds them, and return the sum. The script then captures that output and displays it.

# Function that sums two arguments and returns the result

sum\_args() {

a=$1

b=$2

return $((a + b))

}

# Demonstrate sum\_args function

echo "Now demonstrating sum\_args function:"

sum\_args 3 4

result=$?

echo "sum\_args(3,4) returned: $result"

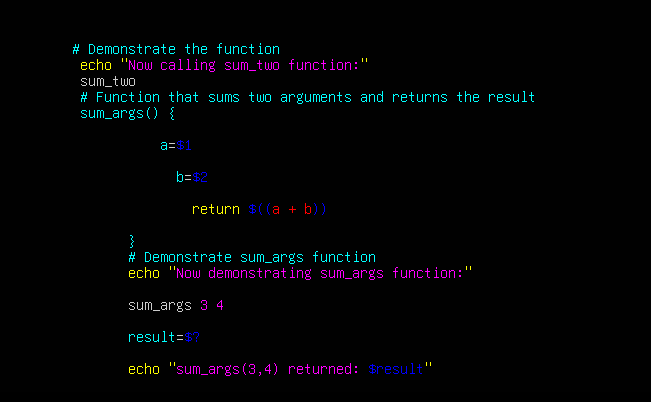
* Save and quit (:wq)
* Run the script and capture the demonstration output:

chmod +x setup.sh

./setup.sh

# Observe the output that shows "sum\_args(3,4) returned: 7"

* Screenshots:
  + vim editor showing function appended: task13\_b4\_vim.png
  + run output showing function demonstration and returned sum: task13\_b4\_run.png





**Task 14 – Codespaces GUI — fork repo, run start-desktop.sh, open VNC, stop GUI**

Goal: Fork the specified repository to your GitHub account, open it in GitHub Codespaces, run the provided script to start a desktop GUI, connect to the GUI via the Codespaces forwarded port (6080) -> vnc.html, and then stop the GUI using the provided stop script.

Important notes before starting:

* GitHub Codespaces must be enabled for your account/org. Codespaces availability and billing may apply.
* The instructions below assume you have permission and capacity to create a Codespace for your fork.
* If Codespaces is not available, you may perform this step on another cloud environment that exposes the same port and scripts, but the screenshot filenames below assume Codespaces.

Steps:

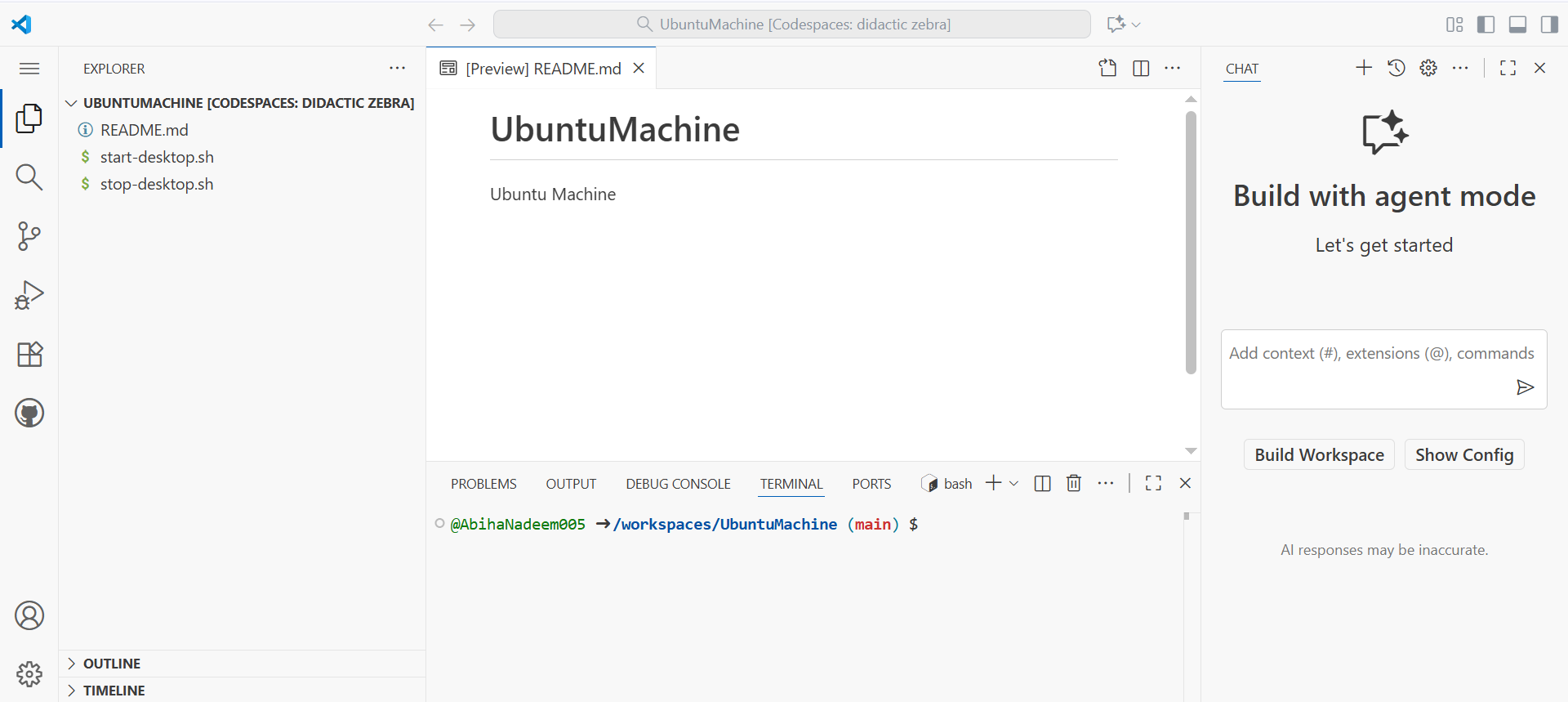
1. Fork the repository to your GitHub account

* Open the repo URL in your browser:
  + [Ubuntu Machine](https://github.com/WaqasSaleem97/UbuntuMachine)
* Click "Fork" (top-right) and fork it to your account.
* Save screenshot as: task14\_fork.png



1. Open a Codespace on your fork

* In your forked repository on GitHub, click the green "Code" button → "Open with Codespaces" → "Create codespace on main" (or appropriate branch).
* Wait for the Codespace to initialize.
* Save screenshot as: task14\_codespace\_launch.png



1. Verify the start script is present and executable (capture evidence)

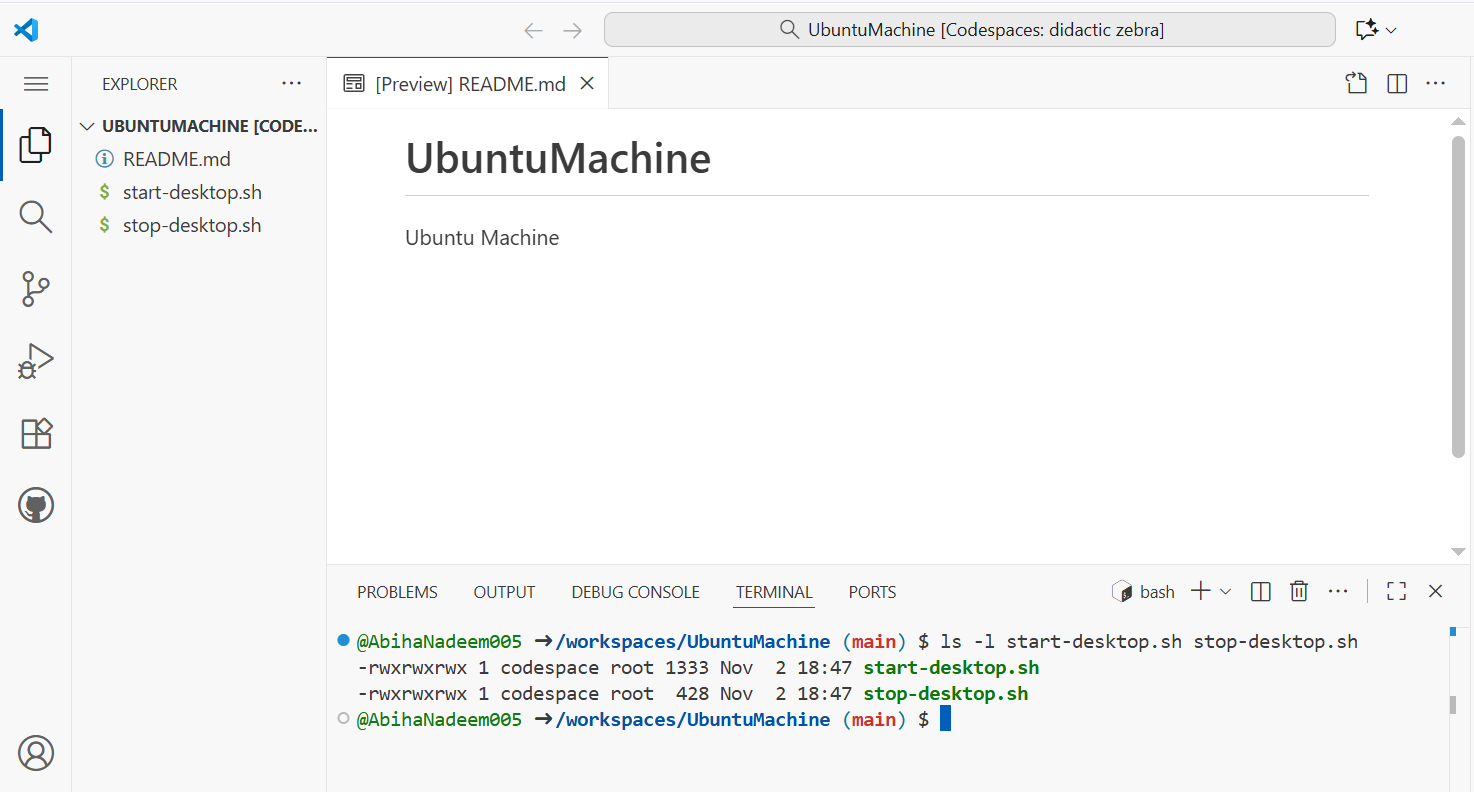
* In the Codespace terminal list files in the repo root and show the start script and stop script exist:

ls -l start-desktop.sh stop-desktop.sh

* If not executable, make it executable:

chmod +x start-desktop.sh stop-desktop.sh

* Save a screenshot showing the ls -l output (file listing) and the chmod command if applied:
  + task14\_start\_script\_ls.png



1. Run the start script inside the Codespace terminal

* In the Codespace terminal run:

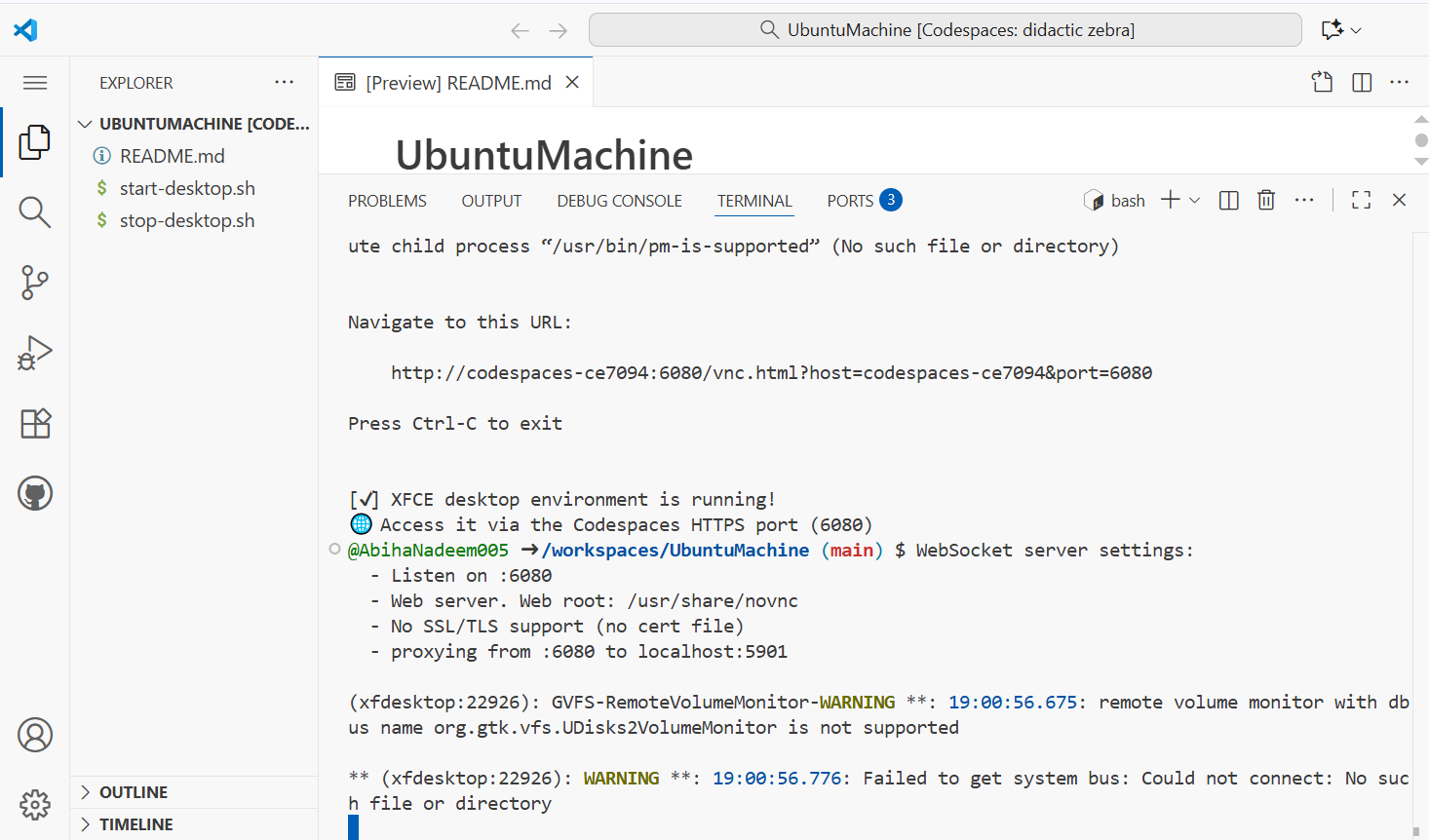
# Ensure the start script is executable

chmod +x start-desktop.sh

# Start the desktop GUI

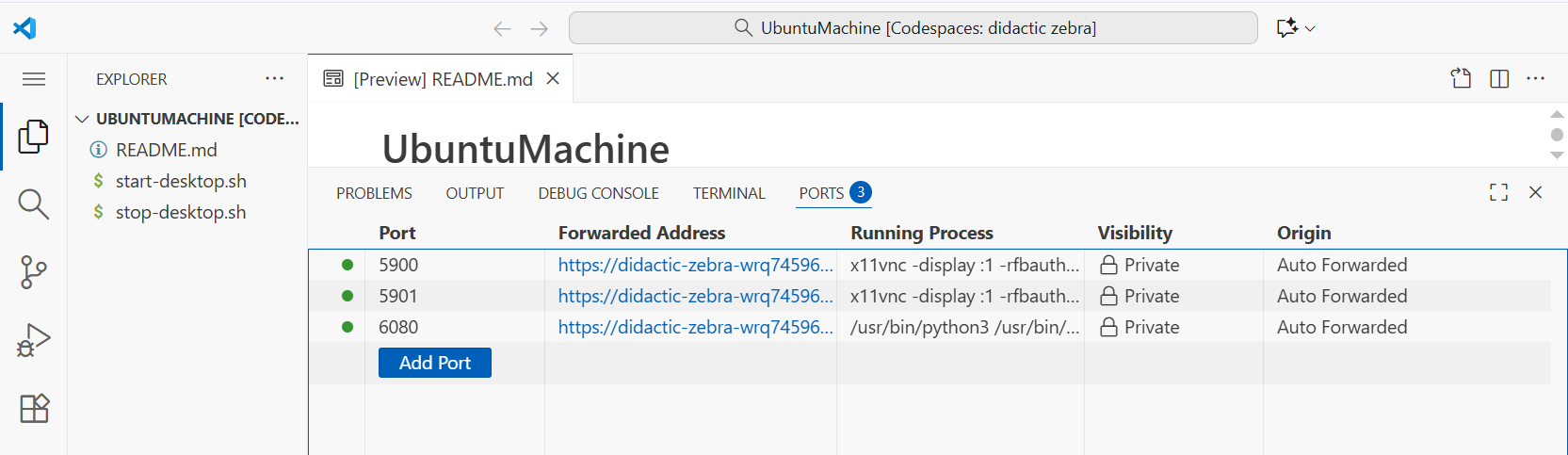
./start-desktop.sh

* Capture the terminal output showing successful start messages.
* Save screenshot as: task14\_start\_run.png



1. Verify forwarded ports in Codespaces (Ports view)

* Open the Codespaces "Ports" panel / view and confirm port 6080 is forwarded and visible.
* Save a screenshot of the Ports view showing port 6080 and its status:
  + task14\_ports\_view.png

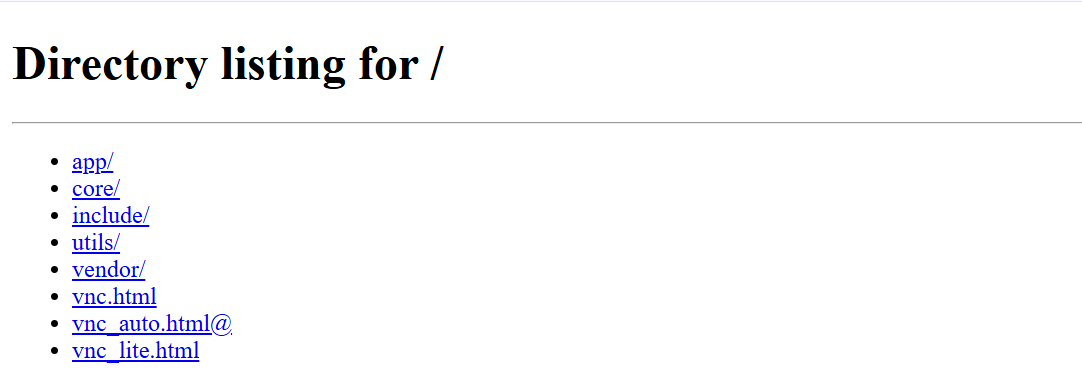


1. Open forwarded port 6080 and connect to VNC HTML page

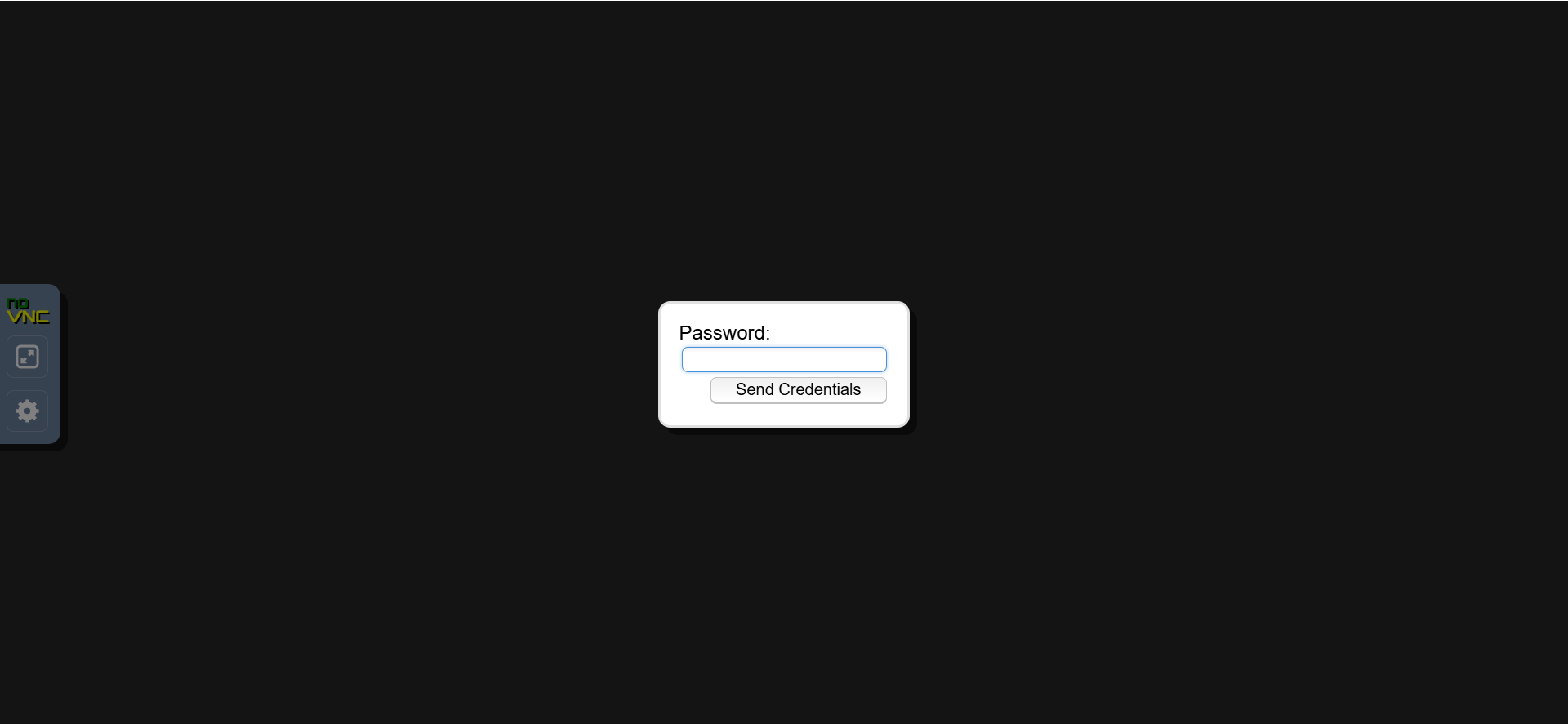
* In the Codespaces UI, open the forwarded port's preview URL or copy the forwarded URL and open it in your browser.
* Visit the port 6080 address and click the vnc.html link.
* When prompted for a password enter:

codespace

* Capture screenshots of:
  + The browser showing the forwarded port URL in the address bar / Codespaces preview: task14\_vnc\_url.png



* + The VNC password prompt (showing password field; do NOT include typed password in a screenshot): task14\_vnc\_password\_prompt.png



* + The VNC session after successful connection showing the GUI/desktop: task14\_vnc\_desktop.png



* + (Optional) A focused screenshot of vnc.html UI showing the "Connect" button before/after connecting: task14\_vnc\_connect.png

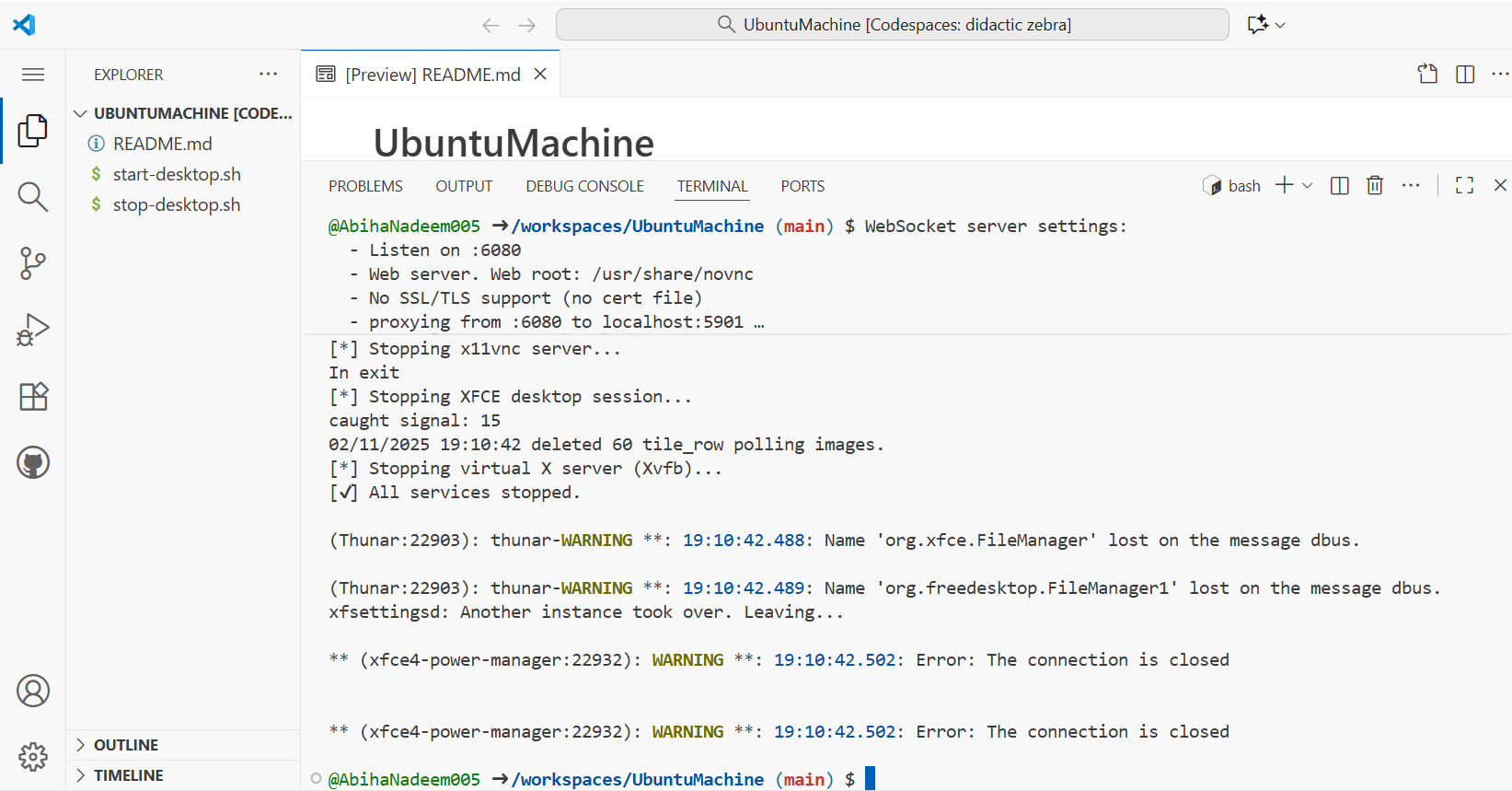


1. Stop the GUI

* When finished, return to the Codespace terminal and run:

./stop-desktop.sh

* Capture the terminal output that shows the GUI stopping and any cleanup messages.
* Save screenshot as: task14\_stop\_run.png



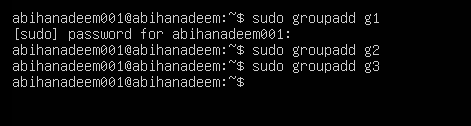
**Exam Evaluation Questions**

**1. Group Management and Membership**

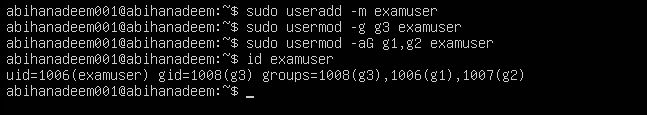
**Scenario:**  
Create groups and manage a user’s primary and supplementary group memberships.

**Steps:**

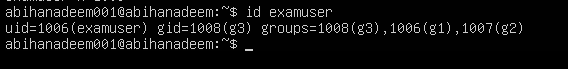
1. Create groups g1, g2, and g3.
   * Screenshot: Q1\_groups\_created.png



1. Change examuser’s primary group to g3 and add g1 and g2 as supplementary groups.
   * Screenshot: Q1\_group\_changes.png



1. Show the final id and /etc/group lines that prove the changes.
   * Screenshot: Q1\_group\_verification.png



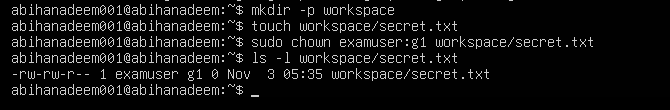


**2. Ownership and Permission Tasks**

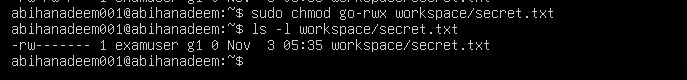
**Scenario:**  
Demonstrate ownership changes and apply both symbolic and numeric permission changes.

**Steps:**

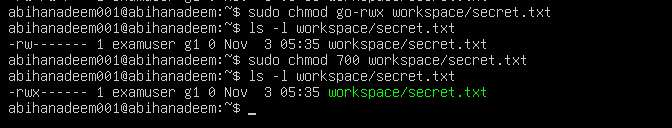
1. Create workspace/secret.txt, change its owner to examuser and group to g1.
   * Screenshot: Q2\_chown\_chgrp.png



1. Remove all permissions for group and others using a symbolic command, then using a numeric command to achieve the same result.
   * Screenshot: Q2\_symbolic\_numeric.png



1. Show ls -l for the file after each change to document the permission bits.
   * Screenshot: Q2\_permissions\_ls.png

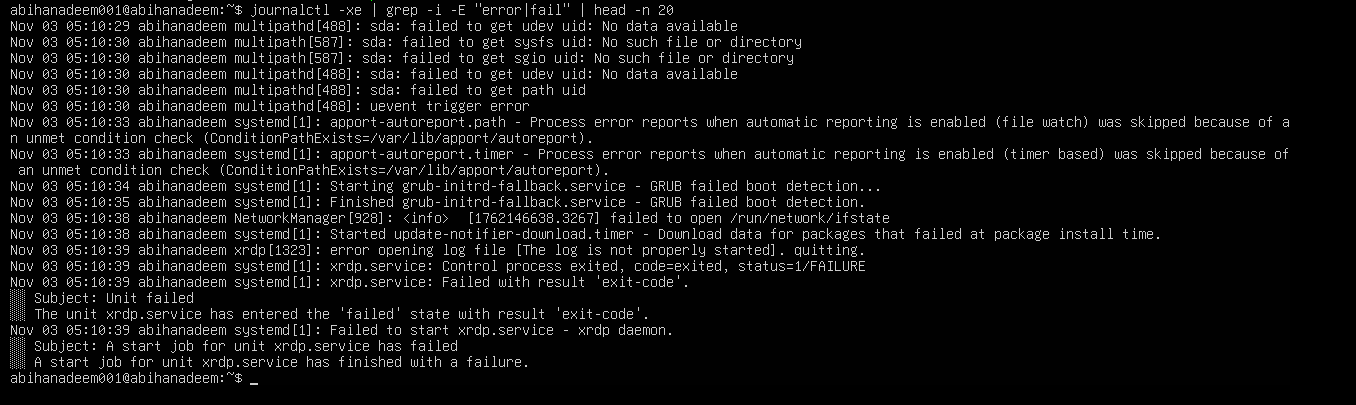


**3. Pipes, Grep, and Redirection Practice**

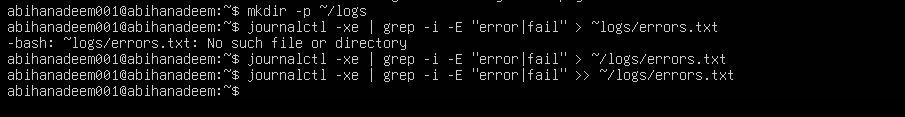
**Scenario:**  
Filter system logs and save results using redirection and piping.

**Steps:**

1. Use grep (or journalctl where applicable) with a pipe to find lines containing "error" or "fail" and show the first 20 results.
   * Screenshot: Q3\_grep\_pipe.png



1. Save the filtered results to a file ~/logs/errors.txt using overwrite, then append additional matching lines using append redirection.
   * Screenshot: Q3\_redirect\_overwrite\_append.png



1. Use a pager to view the saved file.
   * Screenshot: Q3\_pager\_view.png

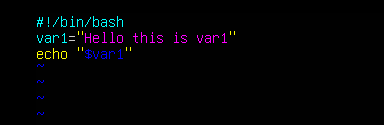


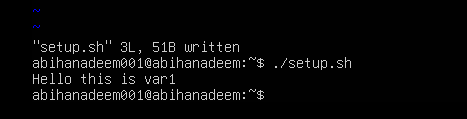
**4. Script: Variables, Command Substitution, File & Dir Checks**

**Scenario:**  
Build and run a script incrementally that demonstrates variables, command substitution, and filesystem checks.

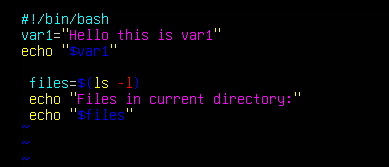
**Steps:**

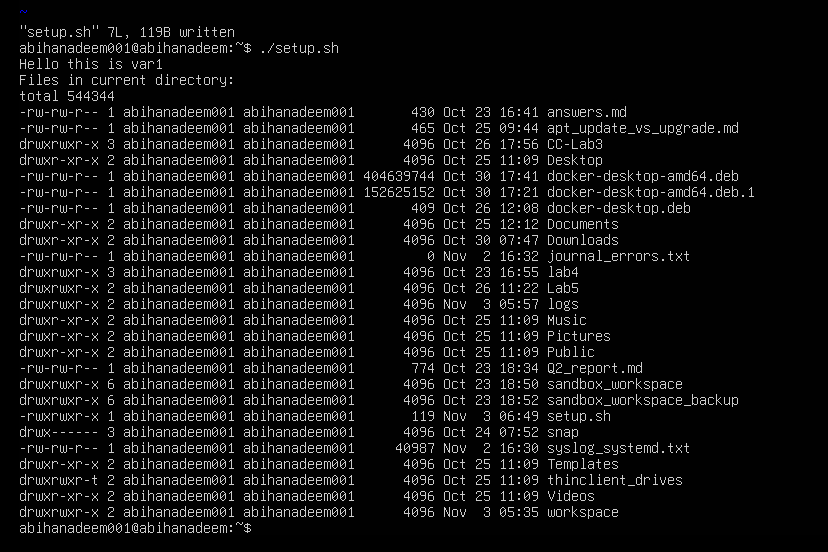
1. Create setup.sh with a shebang and a variable var1 that you echo.
   * Screenshot: Q4\_step1\_var1.png



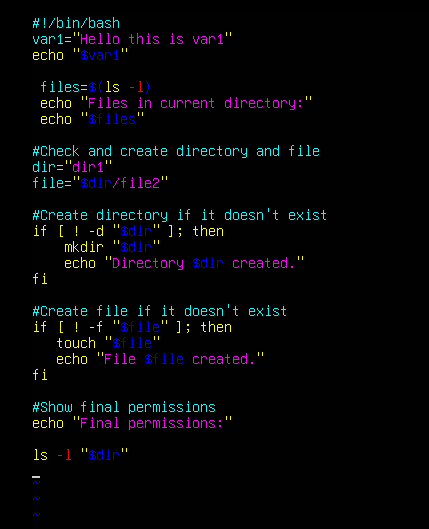


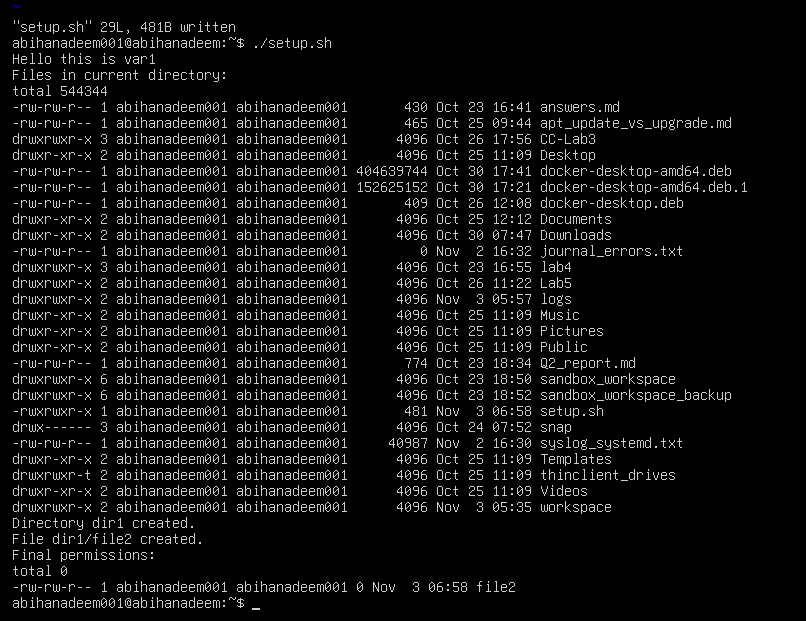
1. Append command substitution that stores ls -l output into a variable and echo it.
   * Screenshot: Q4\_step2\_allfiles.png





1. Append directory and file checks that create dir1 and dir1/file2 if missing, and display their final permissions.
   * Screenshot: Q4\_step3\_dirfile\_checks.png



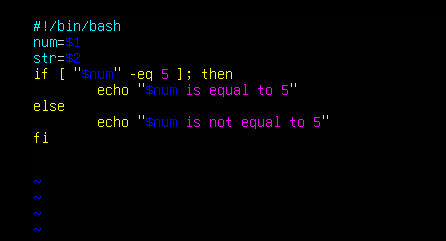


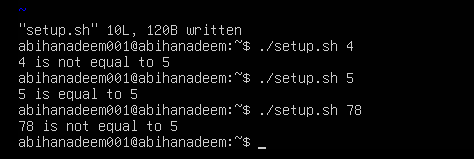
**5. Script: Comparisons and String Tests**

**Scenario:**  
Incrementally add numeric and string comparison tests to a script and show both true/false cases.

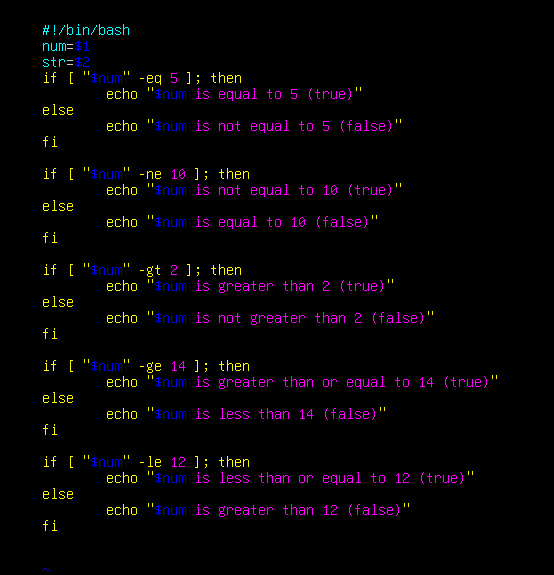
**Steps:**

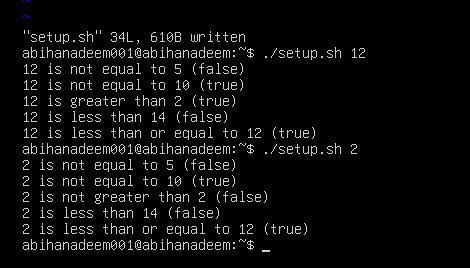
1. Overwrite setup.sh to set num=$1 and str=$2, and add an -eq test showing true and false examples.
   * Screenshot: Q5\_eq\_examples.png



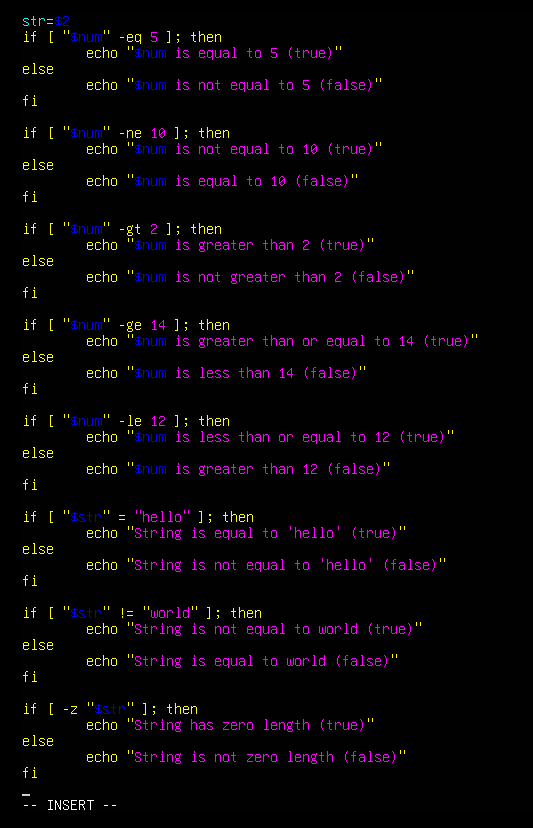


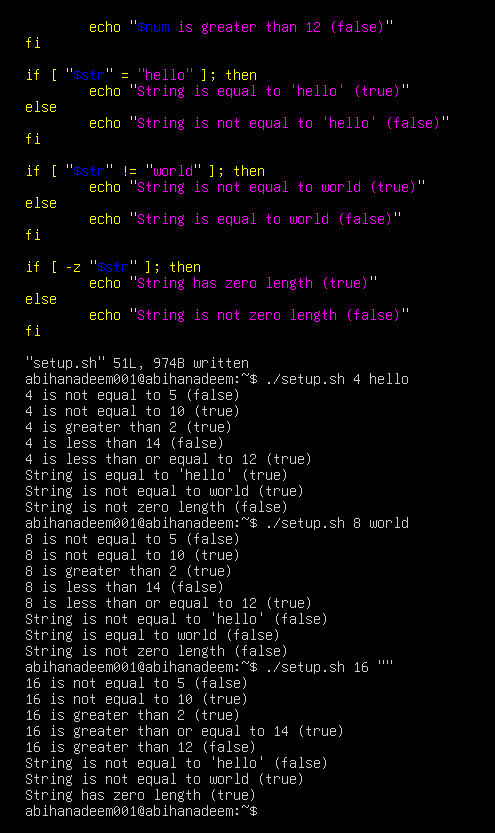
1. Append -ne, -gt, -lt, -ge, and -le tests and demonstrate at least one true and one false invocation for each.
   * Screenshot: Q5\_numeric\_tests.png





1. Append string equality (=) and inequality (!=) checks and a -z (zero-length) test for the second argument, demonstrating true/false cases.
   * Screenshot: Q5\_string\_tests.png



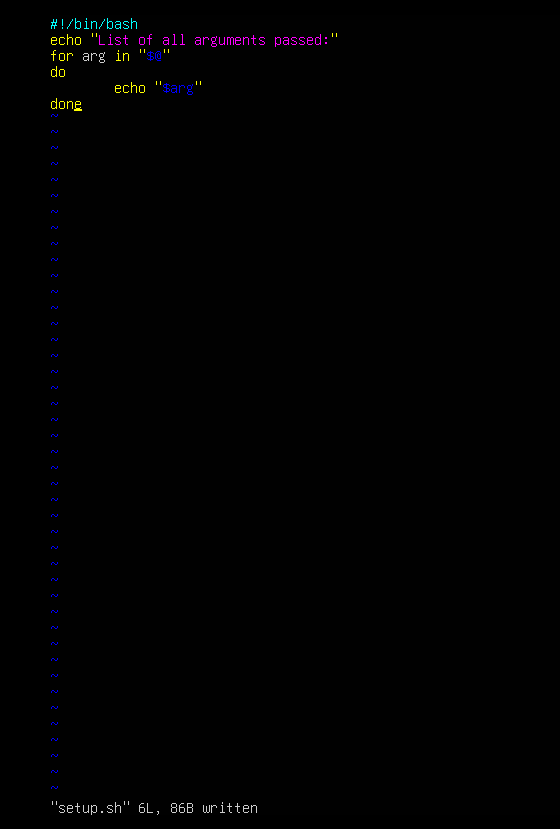


**6. Script: For Loop and Argument Handling**

**Scenario:**  
Write a script that prints all provided arguments and demonstrate correct handling of quoted multi-word arguments.

**Steps:**

1. Create/overwrite setup.sh to print every argument using "$@" in a for loop and save the file.
   * Screenshot: Q6\_script\_forloop\_vim.png



1. Run the script with mixed single and quoted multi-word arguments and capture the output showing each argument on its own line.
   * Screenshot: Q6\_forloop\_run.png

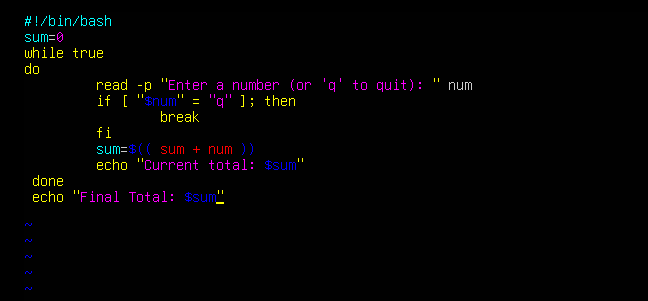
****

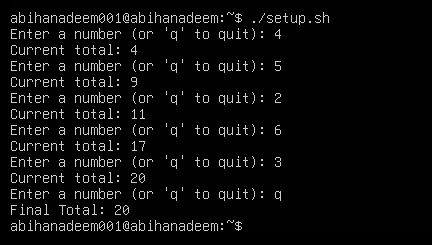
**7. Script: While Loop Summation and Functions**

**Scenario:**  
Implement an interactive or non-interactive summation function and a demonstrated function that returns a numeric result.

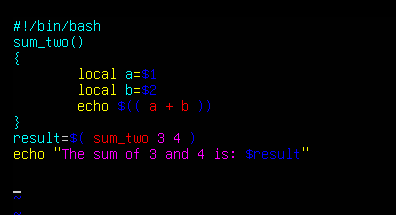
**Steps:**

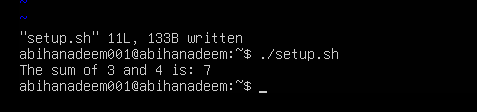
1. Write an interactive while-loop that accumulates numbers until q is entered and shows running totals.
   * Screenshot: Q7\_while\_session.png





1. Add a function that accepts two numeric arguments, returns their sum, and demonstrate capturing its result in a variable.
   * Screenshot: Q7\_function\_sum.png





\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*