

## Practical 1: Linux Commands

### a) Linux Directory Commands

Command: `pwd`

Purpose: वर्तमान कार्यशील directory दिखाता है।

Syntax: `pwd`

Example Output: `/home/user`

Command: `mkdir`

Purpose: एक नई directory बनाता है।

Syntax: `mkdir new_directory`

Check: `ls` command का उपयोग करके आप वर्तमान फ़ोल्डर में directory देख सकते हैं।

Command: `rm -rf`

Purpose: directory और उसकी सामग्री को recursively (क्रमिक रूप से) हटाता है।

Syntax: `rm -rf new_directory`

Caution: इस command का उपयोग सावधानी से करें क्योंकि यह directory को स्थायी रूप से हटा देता है।

Command: `ls`

Purpose: वर्तमान directory में उपलब्ध फाइलों और directories की सूची दिखाता है।

Syntax: `ls`

Command: `cd`

Purpose: वर्तमान directory को बदलता है।

Syntax: `cd new_directory`

Command: cd -

Purpose: पिछले directory पर वापस स्विच करता है।

Syntax: cd -

## b) Linux File Commands

Command: touch

Purpose: एक खाली फाइल बनाता है।

Syntax: touch file.txt

Command: cat

Purpose: किसी फाइल की सामग्री दिखाता है। content

Syntax: cat file.txt

Command: rm

Purpose: एक फाइल को हटाता है।

Syntax: rm file.txt

Command: cp

Purpose: एक फाइल को एक स्थान से दूसरे स्थान पर कॉपी करता है।

Syntax: cp file.txt /path/to/destination/

Command: mv

Purpose: फाइल को स्थानांतरित (move) या नाम बदलने (rename) के लिए उपयोग किया जाता है।

Syntax (Move): mv file.txt /path/to/destination/

Syntax (Rename): mv old\_name.txt new\_name.txt

Command: rename

Purpose: पैटर्न के आधार पर फाइलों का नाम बदलता है।

Syntax: rename 's/old/new/' \*.txt

यह command सभी .txt फाइलों के नाम "old" से "new" में बदल देगा।

### c) Linux Permission Commands

Command: su

Purpose: सुपरयूज़र (root) खाते में स्विच करता है।

Syntax: su

Command: id

Purpose: उपयोगकर्ता ID और समूह ID दिखाता है।

Syntax: id

Command: useradd

Purpose: एक नया उपयोगकर्ता जोड़ता है।

Syntax: sudo useradd new\_user

Command: passwd

Purpose: उपयोगकर्ता का पासवर्ड बदलता है या सेट करता है।

Syntax: sudo passwd new\_user

Command: groupadd

Purpose: एक नया समूह जोड़ता है।

Syntax: sudo groupadd new\_group

Command: chmod

Purpose: फाइल की अनुमतियों को बदलता है।

Syntax: chmod 755 file.txt

यह command मालिक को पढ़ने, लिखने, और निष्पादन की अनुमति देता है, जबकि अन्य उपयोगकर्ताओं को केवल पढ़ने और निष्पादन की अनुमति देता है।

Command: chown

Purpose: फाइल के मालिक को बदलता है।

Syntax: `sudo chown user file.txt`

`sudo chown user file.txt` चलाकर, आप उपयोगकर्ता को `file.txt` का स्वामी बना रहे हैं। इसका मतलब यह है कि उपयोगकर्ता के पास फ़ाइल की अनुमतियों (जैसे पढ़ना, लिखना और इसे निष्पादित करना) पर पूर्ण नियंत्रण होगा, यह मानते हुए कि फ़ाइल अनुमतियाँ इसकी अनुमति देती हैं।

#### d) Linux File Content & Filter Commands

Command: `head`

Purpose: फाइल की पहले 10 लाइनों को दिखाता है।

Syntax: `head file.txt`

Command: `tail`

Purpose: फाइल की आखिरी 10 लाइनों को दिखाता है।

Syntax: `tail file.txt`

Command: `grep`

Purpose: फाइल में किसी विशेष पैटर्न की खोज करता है।

Syntax: `grep 'pattern' file.txt`

#### e) Linux Utility Commands

Command: find

Purpose: किसी directory में फाइलों की खोज करता है।

Syntax: find /path/to/search -name "filename"

Command: locate

Purpose: फाइलों को नाम के आधार पर खोजता है।

यह एक डेटाबेस पर निर्भर करता है जो आपके सिस्टम पर फ़ाइलों को अनुक्रमित करता है, जिसे नियमित रूप से बनाने और अद्यतन करने की आवश्यकता होती है।

Syntax: locate filename

Command: df

Purpose: डिस्क स्पेस उपयोग को दिखाता है।

डीएफ कमांड का मतलब डिस्क फ्री है।-एच: इसका मतलब "मानव-पठनीय" प्रारूप है। यह डिस्क स्थान को केवल बाइट्स के बजाय जीबी, एमबी या केबी जैसे अधिक उपयोगकर्ता-अनुकूल तरीके से प्रदर्शित करके आउटपुट को समझना आसान बनाता है।

Syntax: df -h

## f) Linux Networking Commands

Command: ip

Purpose: आईपी पता और नेटवर्क कॉन्फिगरेशन दिखाता है।

Syntax: ip addr show

### Command: ping

Purpose: नेटवर्क कनेक्टिविटी का परीक्षण करता है।

पिंग कमांड का उपयोग आपके सिस्टम और किसी नेटवर्क (जैसे इंटरनेट) पर रिमोट सर्वर या डिवाइस के बीच कनेक्टिविटी का परीक्षण करने के लिए किया जाता है। यह निर्दिष्ट पते पर डेटा के छोटे पैकेट भेजता है और प्रतिक्रिया की प्रतीक्षा करता है। यदि इसे कोई प्रतिक्रिया मिलती है, तो यह इंगित करता है कि आपके सिस्टम और लक्ष्य के बीच नेटवर्क कनेक्टिविटी है।

Syntax: ping google.com

### g) Editing Crontab for Scheduling

#### Command: crontab -e

Purpose: क्रोनटैब फ़ाइल को संपादित करता है ताकि कार्यों को शेड्यूल किया जा सके।

Example: सिस्टम-व्यापी संदेश को रोजाना 10 बजे शेड्यूल करना:

```
wall "Hello, this is a scheduled message."
```

Iska matlab hoga ki /path/to/your/script.sh script har din subah 6:00 baje run hoga.

### h) Using the vi Editor=vi editor usig guidelines=used to edit files using command line.

Open a file in vi:

```
vi filename.txt
```

Insert text:

Press i to go into insert mode.

Type your text.

Save and quit:

Press Esc to exit insert mode.

Type :wq to save and exit.

Search for a term:

Press / followed by the term you want to search.

Press n to jump to the next occurrence.

(Vi में एक फ़ाइल खोलें:

vi फ़ाइलनाम.txt

पाठ सम्मिलित करें:

इन्सर्ट मोड में जाने के लिए i दबाएँ।

अपना टेक्स्ट टाइप करें.

सहेजें और छोड़ें:

इन्सर्ट मोड से बाहर निकलने के लिए Esc दबाएँ।

सहेजने और बाहर निकलने के लिए :wq टाइप करें।

कोई शब्द खोजें:

जिस शब्द को आप खोजना चाहते हैं उसके बाद / दबाएँ।

अगली घटना पर जाने के लिए n दबाएँ।)

## Practical 2: Shell Scripting

### a) Shell Script to Print a Message

```
#!/bin/bash
```

लाइन `#!/bin/bash` को शेबैंग कहा जाता है, जो सिस्टम को बताता है कि इस स्क्रिप्ट को बैश शेल में चलाया जाना चाहिए। इको कमांड एक संदेश प्रिंट करता है जिसे आप टर्मिनल पर देख सकते हैं।



शब्द "शेबंग" "शार्प" (#) और "बैंग" (!) के संयोजन से आया है। यह एक स्क्रिप्ट की शुरुआत में एक विशेष वर्ण अनुक्रम (#!) है जो ऑपरेटिंग सिस्टम को बताता है कि फ़ाइल को निष्पादित करने के लिए किस दुभाषिया (जैसे बैश, पायथन, आदि) का उपयोग करना है

Create the script:

code

nano message.sh

Write the script:

code

`#!/bin/bash`

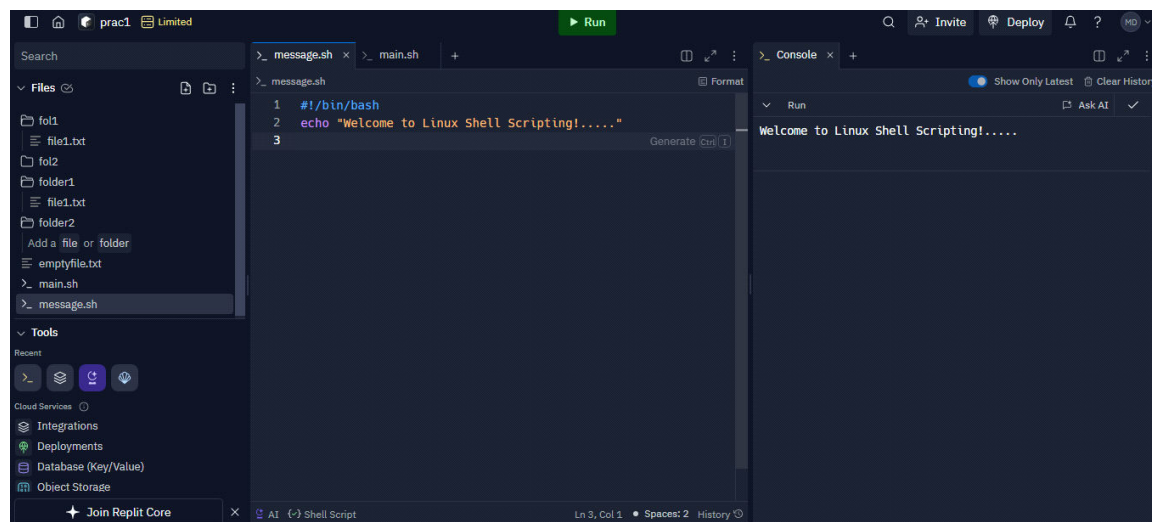
`echo "Welcome to Linux Shell Scripting!"`

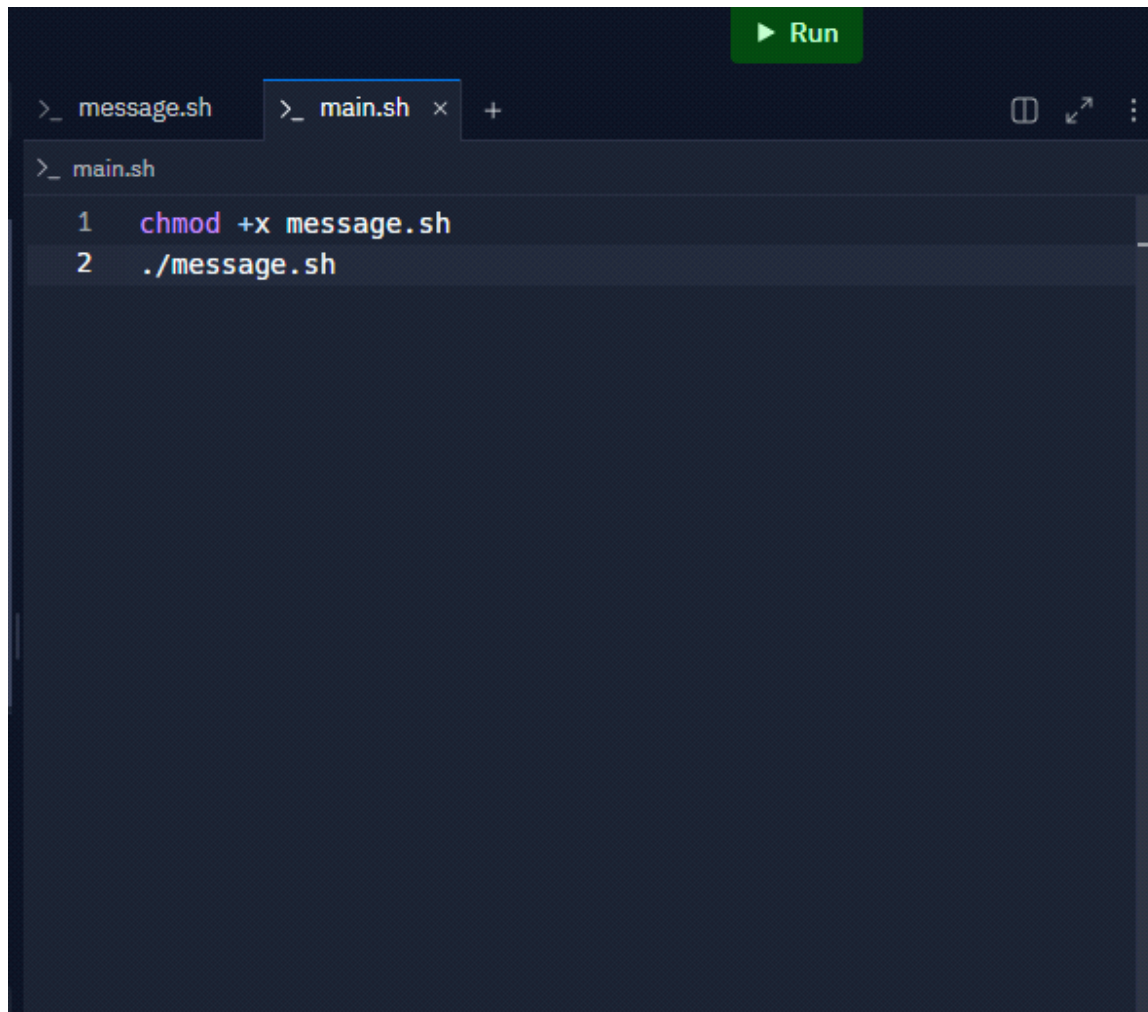
Save and run the script:

code

`chmod +x message.sh`

`./message.sh`





```
>_ message.sh >_ main.sh x +
>_ main.sh
1  chmod +x message.sh
2  ./message.sh
```

## b) Shell Script to Access Command-Line Arguments

Create the script:

code

nano args.sh

Write the script:

code

```
#!/bin/bash
```

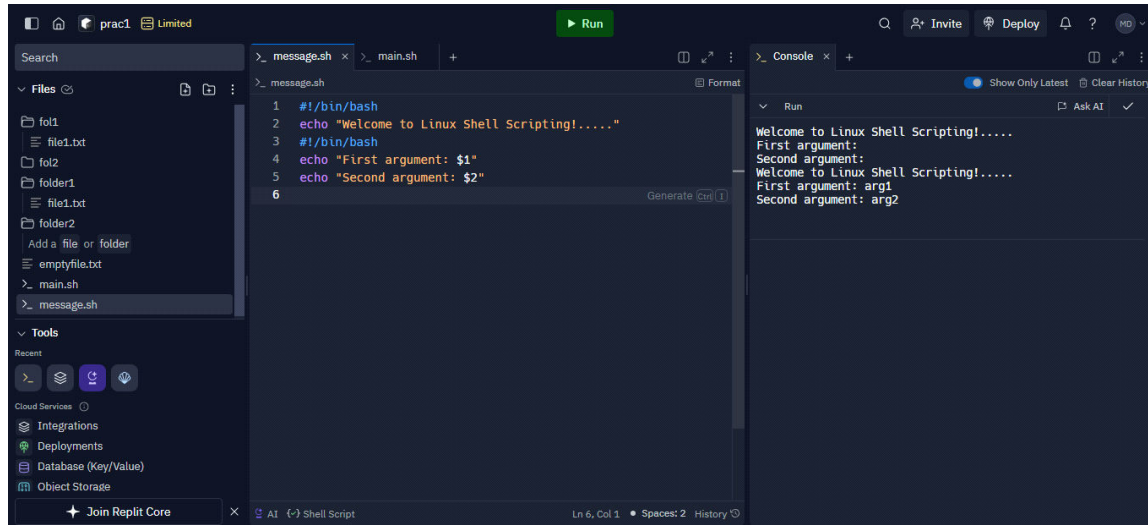
```
echo "First argument: $1"
```

```
echo "Second argument: $2"
```

Save and run the script with arguments:

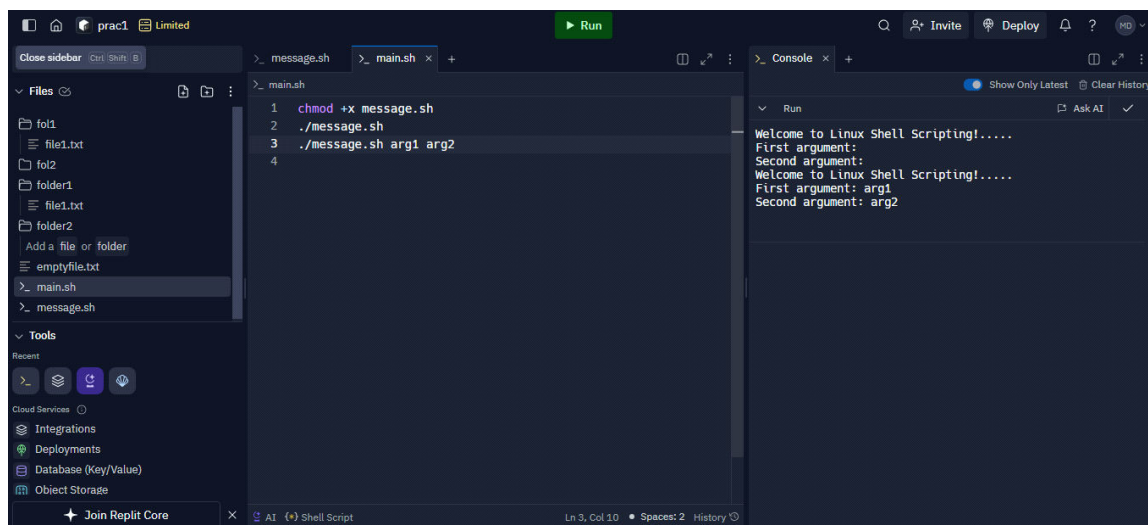
code

`./args.sh arg1 arg2`



```
1 #!/bin/bash
2 echo "Welcome to Linux Shell Scripting!...."
3 #!/bin/bash
4 echo "First argument: $1"
5 echo "Second argument: $2"
6
```

```
Welcome to Linux Shell Scripting!....
First argument:
Second argument:
Welcome to Linux Shell Scripting!....
First argument: arg1
Second argument: arg2
```



```
1 chmod +x message.sh
2 ./message.sh
3 ./message.sh arg1 arg2
4
```

```
Welcome to Linux Shell Scripting!....
First argument:
Second argument:
Welcome to Linux Shell Scripting!....
First argument: arg1
Second argument: arg2
```

### c) Shell Script to Create Files with Command-Line Names

Create the script:

code

`nano create_files.sh`

Write the script:

code

```
#!/bin/bash
```

```
for file in "$@"; do
```

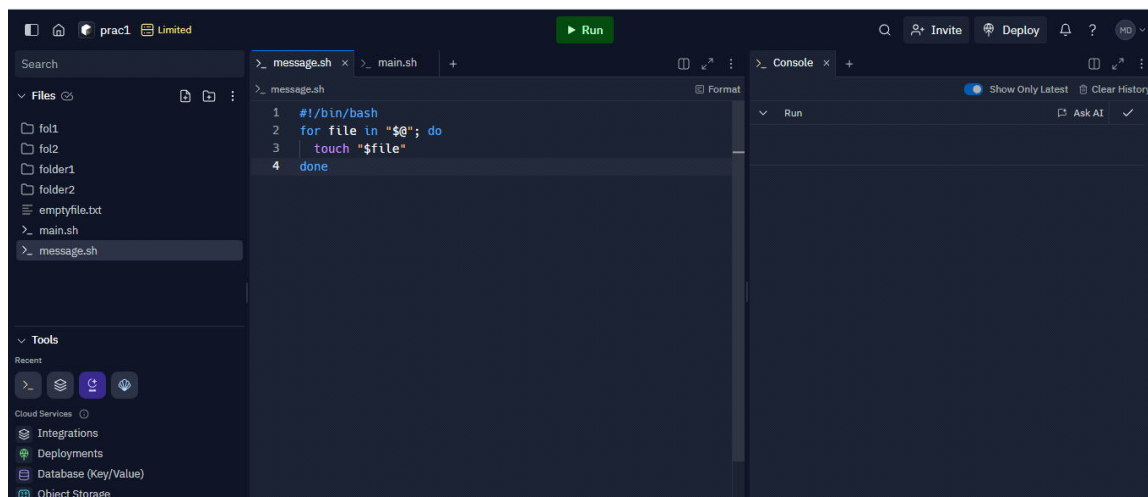
```
    touch "$file"
```

```
done
```

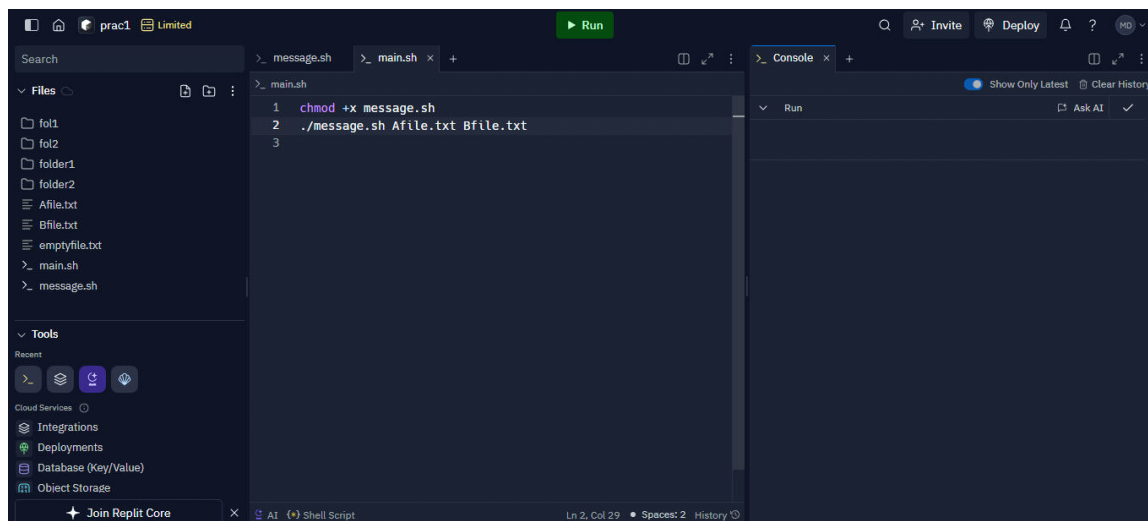
Save and run the script:

code

```
./create_files.sh file1.txt file2.txt
```



```
>_ message.sh x >_ main.sh x +
message.sh
1  #!/bin/bash
2  for file in "$@"; do
3    touch "$file"
4  done
```



```
>_ message.sh x >_ main.sh x +
main.sh
1  chmod +x message.sh
2  ./message.sh Afile.txt Bfile.txt
3
```

```
for file in "$@"; do ... done:
```

The for loop iterates over each argument passed to the script.

"\$@" is a special variable that represents all the arguments passed to the script as a list, preserving any spaces within each argument.

Each individual argument (file name in this case) is stored in the variable file during each iteration of the loop.

`touch "$file":`

The touch command is used to create an empty file with the specified name, or to update the timestamp of an existing file.

Here, "\$file" is each file name from the list of arguments. So, touch creates each file with the name provided in the command line.

#### d) Shell Script to Find Factorial of a Number

Create the script:

code

`nano factorial.sh`

Write the script:

code

`#!/bin/bash`

`echo "Enter a number:"`

`read num`

`fact=1`

`for (( i=1; i<=num; i++ )); do`

`fact=$((fact * i))`

`done`

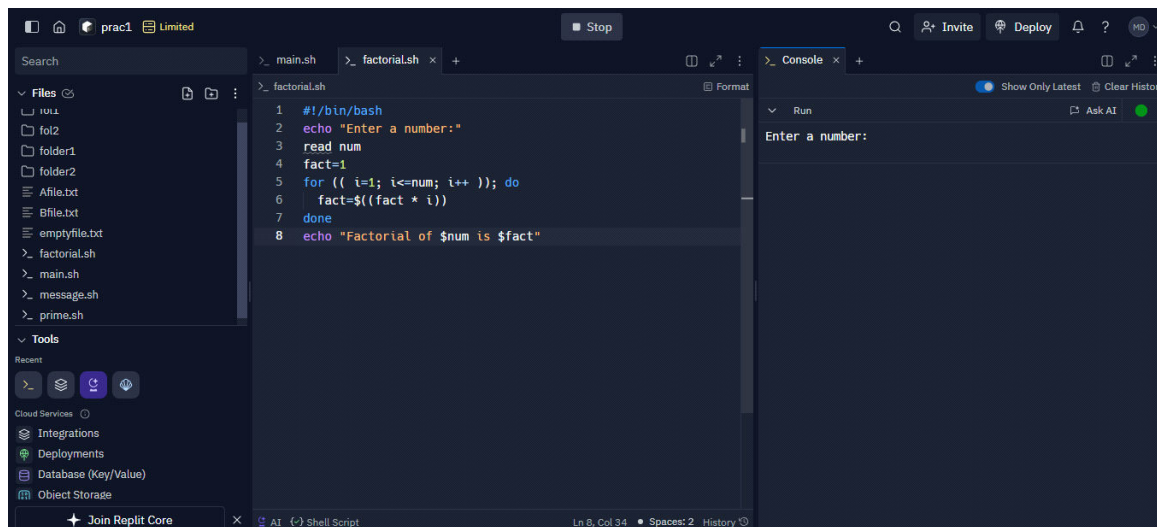
`echo "Factorial of $num is $fact"`

Save and run the script:

code

`chmod +x factorial.sh`

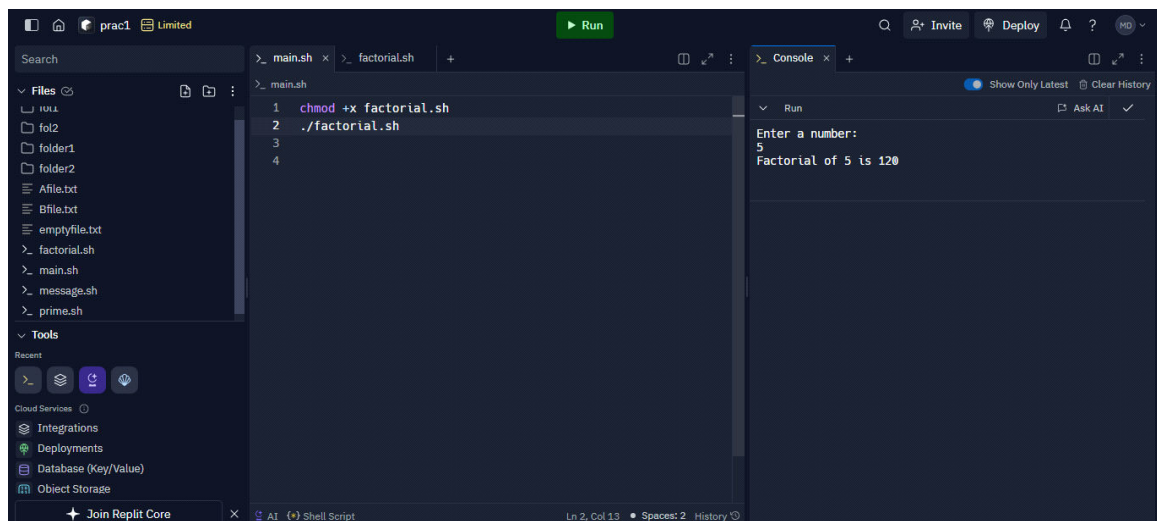
`./factorial.sh`



The screenshot shows a Replit IDE with a file explorer on the left containing files like `fol2`, `folder1`, `folder2`, `Afile.txt`, `Bfile.txt`, `emptyfile.txt`, `factorial.sh`, `main.sh`, `message.sh`, and `prime.sh`. The main editor displays the contents of `factorial.sh`:

```
1 #!/bin/bash
2 echo "Enter a number:"
3 read num
4 fact=1
5 for (( i=1; i<=num; i++ )); do
6     fact=$((fact * i))
7 done
8 echo "Factorial of $num is $fact"
```

The console on the right shows the prompt `Enter a number:`.



The screenshot shows the same Replit IDE, but now the `main.sh` file is open in the editor:

```
1 chmod +x factorial.sh
2 ./factorial.sh
3
4
```

The console on the right shows the output of the script execution:

```
Enter a number:
5
Factorial of 5 is 120
```

## e) Shell Script to Check if a Number is Prime

Create the script:

code

`nano prime.sh`

Write the script:

code

`#!/bin/bash`

```
echo "Enter a number:"

read num

is_prime=1

for ((i=2; i<=num/2; i++)); do

    if ((num % i == 0)); then

        is_prime=0

        break

    fi  # Close the 'if' statement
done

if ((is_prime == 1)); then

    echo "$num is a prime number."

else

    echo "$num is not a prime number."

fi
```

Save and run the script:

```
code

chmod +x prime.sh

./prime.sh
```

This screenshot shows a Replit editor interface with a file explorer on the left, a code editor in the center, and a console on the right. The file explorer shows a project named 'prac1' with a 'Limited' status. The code editor is open to a file named 'prime.sh' and contains a shell script for checking if a number is prime. The script prompts the user to 'Enter a number:', reads the input, and then uses a for loop to check divisibility from 2 to num/2. If a divisor is found, it sets 'is\_prime' to 0 and breaks the loop. Otherwise, it remains 1. Finally, it prints a message based on the value of 'is\_prime'.

```
1 #!/bin/bash
2 echo "Enter a number:"
3 read num
4 is_prime=1
5
6 for ((i=2; i<=num/2; i++)); do
7     if ((num % i == 0)); then
8         is_prime=0
9         break
10    fi # Close the 'if' statement
11 done
12
13 if ((is_prime == 1)); then
14     echo "$num is a prime number."
15 else
16     echo "$num is not a prime number."
17 fi
18
```

The console on the right shows the prompt 'Enter a number:'.

This screenshot shows the same Replit editor interface, but now the script has been executed. The code editor shows the first four lines of the script: setting permissions, running the script, and a 'Generate' button. The console on the right shows the output of the script after the user entered '5'.

```
1 chmod +x prime.sh
2 ./prime.sh
3
4
```

The console shows the prompt 'Enter a number:' followed by the user input '5' and the output '5 is a prime number.'.