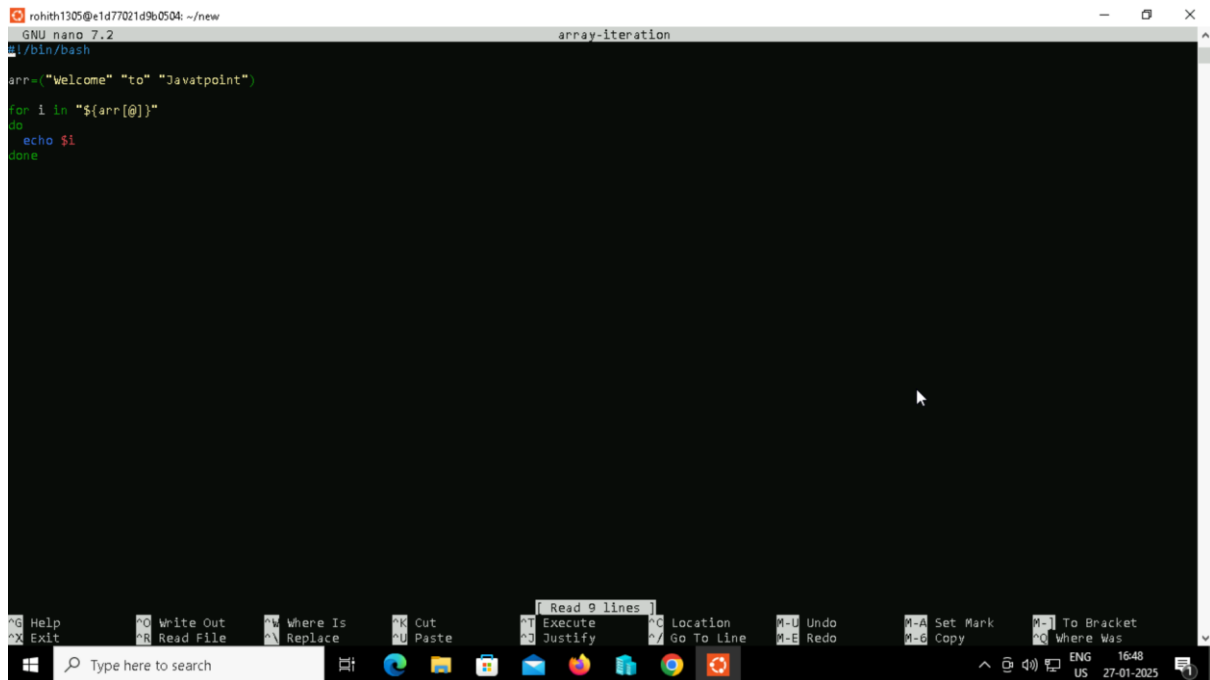


Array-iteration:

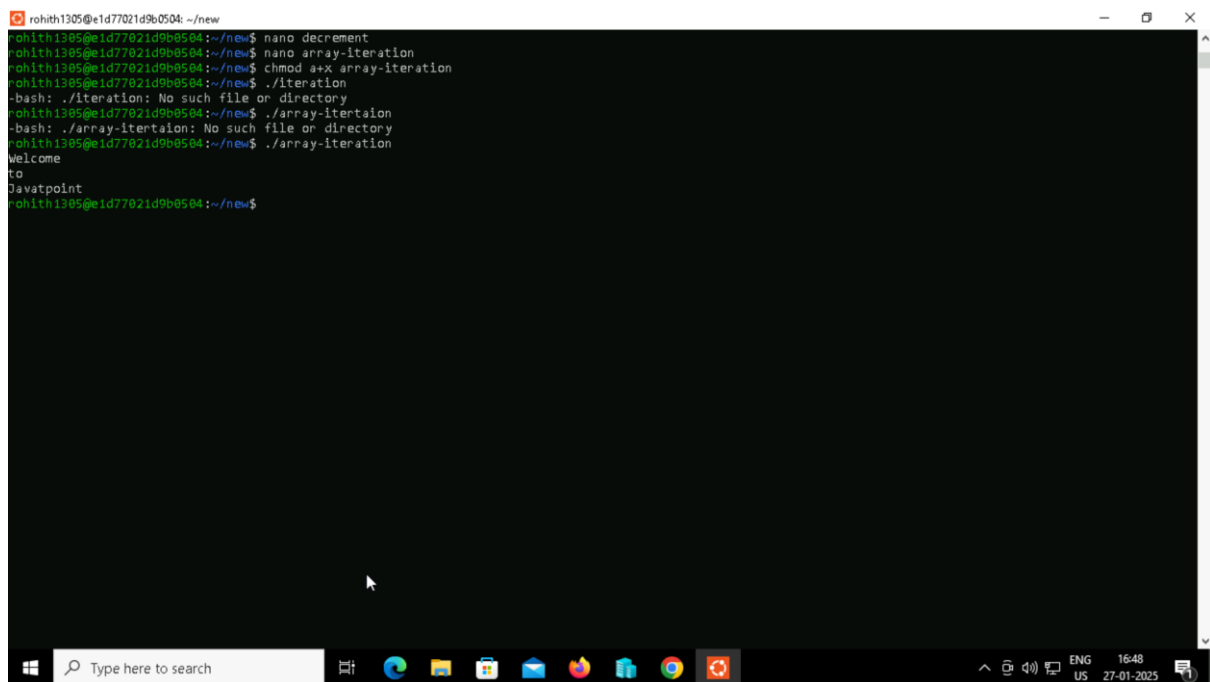


```
rohith1305@e1d77021d9b0504: ~/new
GNU nano 7.2 array-iteration
#!/bin/bash

arr=("Welcome" "to" "Javatpoint")

for i in "${arr[@]}"
do
    echo $i
done
```

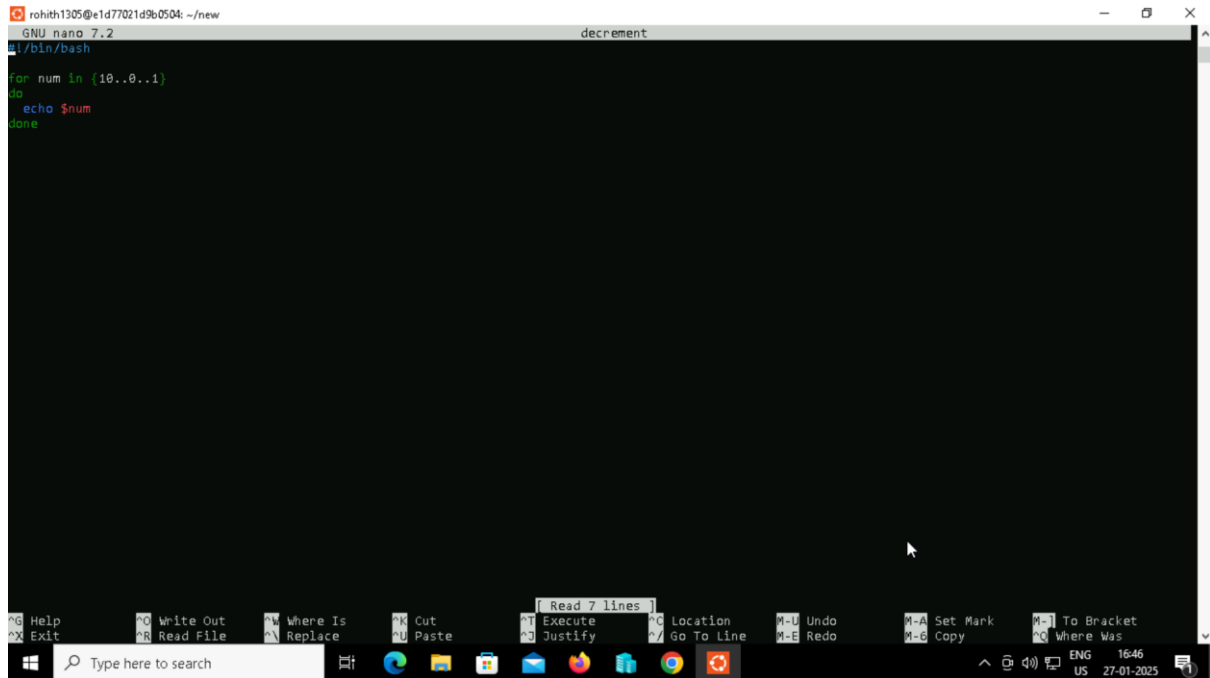
The screenshot shows a terminal window with the GNU nano 7.2 editor. The file is named 'array-iteration'. The code defines an array 'arr' with three elements: 'Welcome', 'to', and 'Javatpoint'. A 'for' loop iterates over each element of the array, printing it to the console. The terminal window has a Windows taskbar at the bottom with various application icons and a search bar.



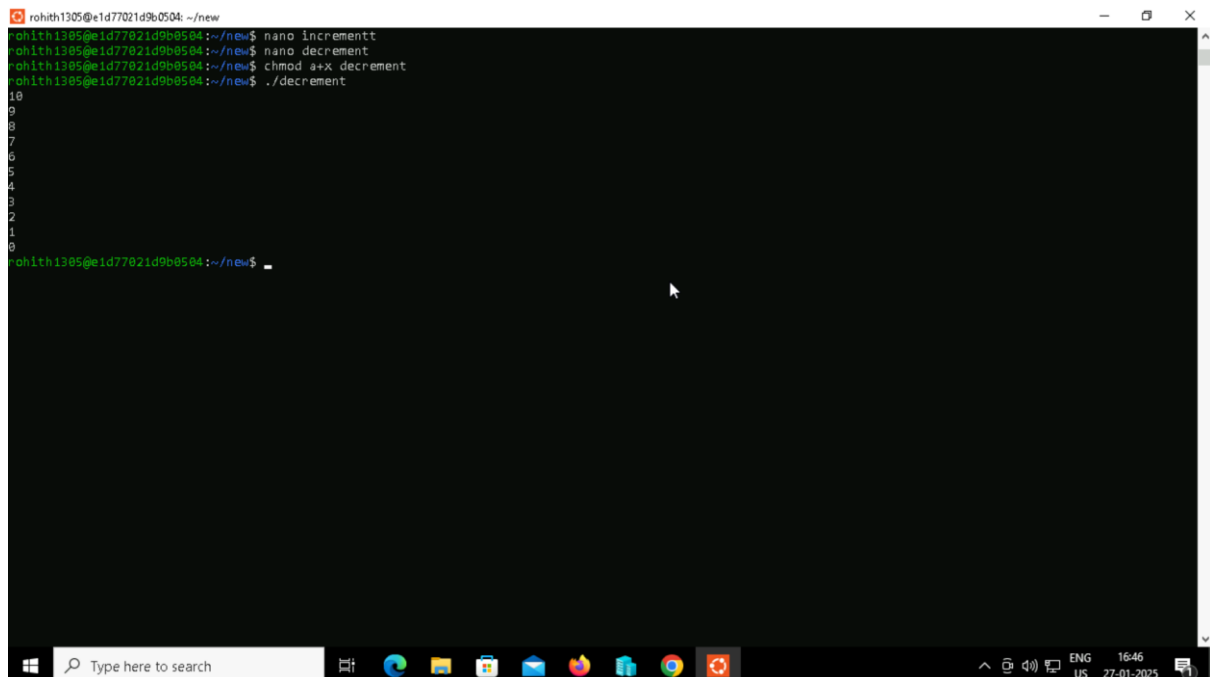
```
rohith1305@e1d77021d9b0504: ~/new
rohith1305@e1d77021d9b0504:~/new$ nano decrement
rohith1305@e1d77021d9b0504:~/new$ nano array-iteration
rohith1305@e1d77021d9b0504:~/new$ chmod a+x array-iteration
rohith1305@e1d77021d9b0504:~/new$ ./iteration
-bash: ./iteration: No such file or directory
rohith1305@e1d77021d9b0504:~/new$ ./array-iteration
-bash: ./array-iteration: No such file or directory
rohith1305@e1d77021d9b0504:~/new$ ./array-iteration
Welcome
to
Javatpoint
rohith1305@e1d77021d9b0504:~/new$
```

The screenshot shows a terminal window where the user has created a file named 'array-iteration' and made it executable with 'chmod a+x array-iteration'. The user then attempts to run the script with './iteration', which fails with the error 'No such file or directory'. The user then runs './array-iteration', which also fails with the same error. Finally, the user runs './array-iteration' again, which successfully executes the script and prints the output: 'Welcome', 'to', and 'Javatpoint' on separate lines. The terminal window has a Windows taskbar at the bottom with various application icons and a search bar.

Decrement using for:

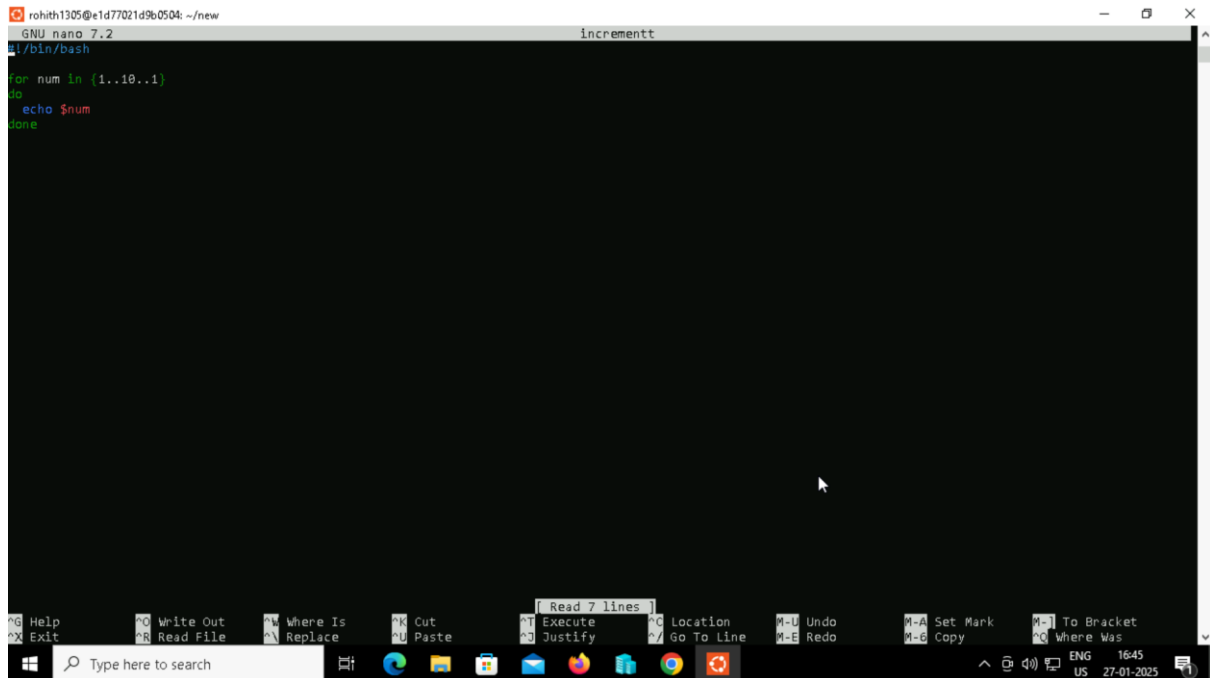


```
rohith1305@e1d77021d9b0504: ~/new
GNU nano 7.2 decrement
//bin/bash
for num in {10..0..1}
do
  echo $num
done
```



```
rohith1305@e1d77021d9b0504: ~/new
rohith1305@e1d77021d9b0504:~/new$ nano incrementt
rohith1305@e1d77021d9b0504:~/new$ nano decrement
rohith1305@e1d77021d9b0504:~/new$ chmod +x decrement
rohith1305@e1d77021d9b0504:~/new$ ./decrement
10
9
8
7
6
5
4
3
2
1
0
rohith1305@e1d77021d9b0504:~/new$
```

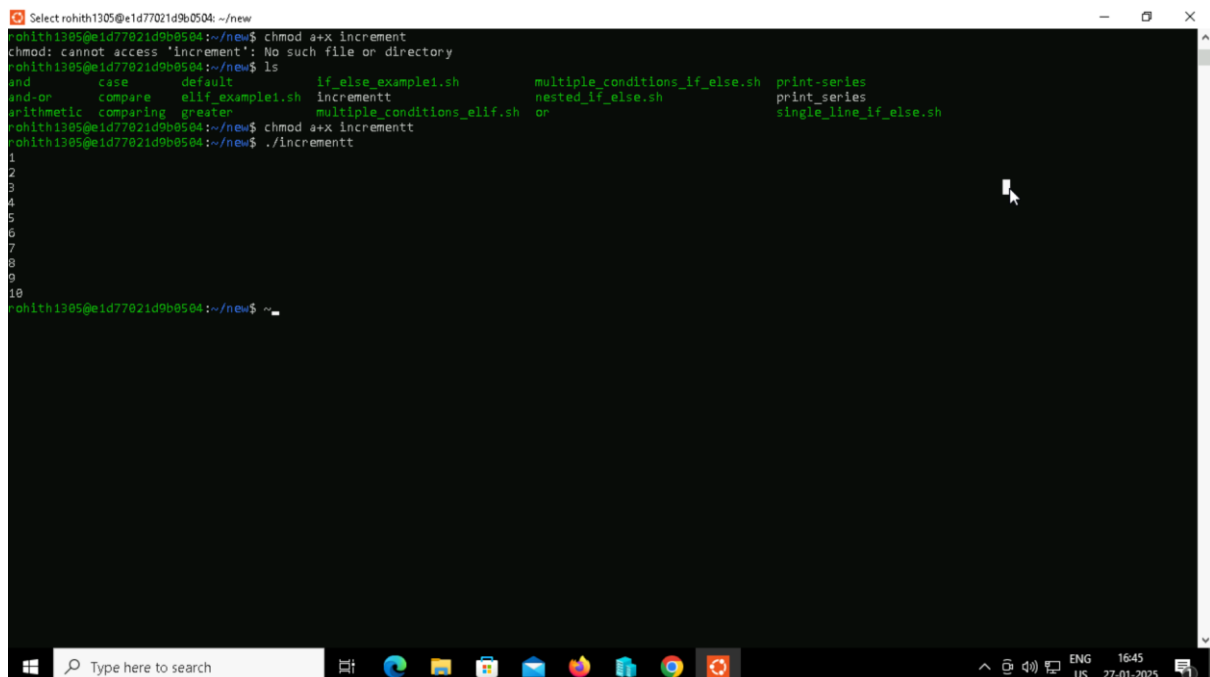
Increment using for:



A screenshot of a terminal window titled "incrementt" running GNU nano 7.2. The script content is as follows:

```
//bin/bash
for num in {1..10..1}
do
  echo $num
done
```

The terminal window includes a menu bar with options like Help, Write Out, Where Is, Cut, Paste, Execute, Justify, Location, Go To Line, Undo, Redo, Set Mark, Copy, To Bracket, and Where Was. The Windows taskbar is visible at the bottom.

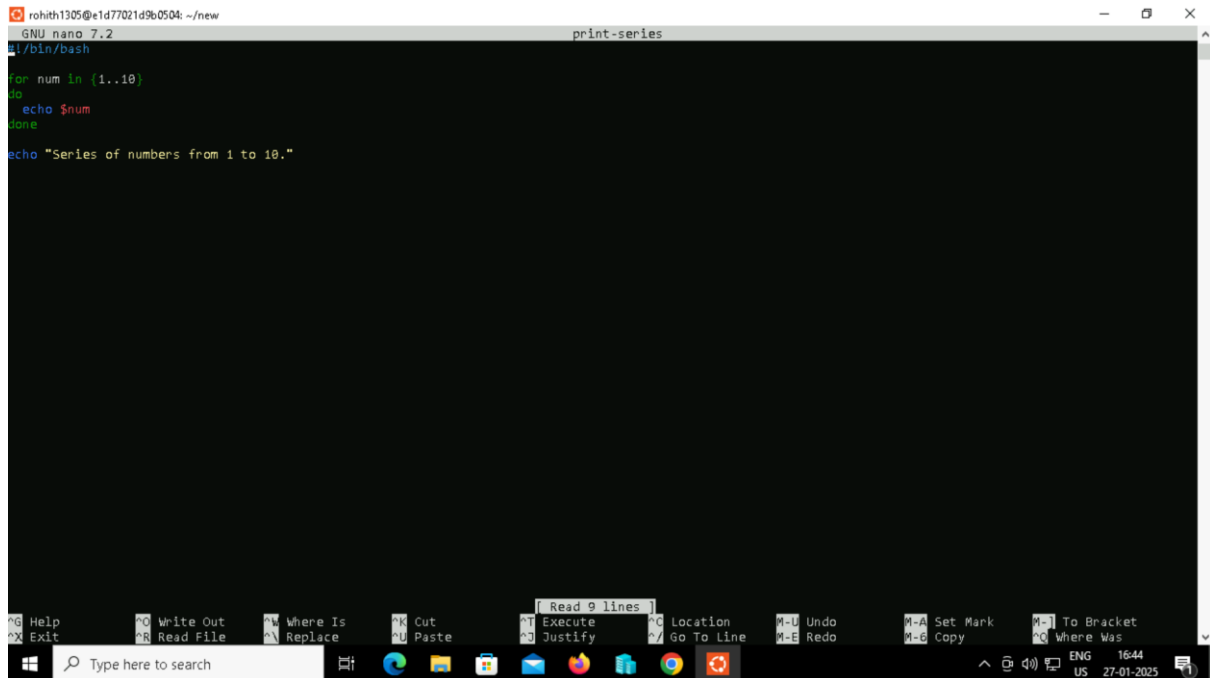


A screenshot of a terminal window showing the execution of the script. The output is as follows:

```
rohith1305@e1d77021d9b0504: ~/new
rohith1305@e1d77021d9b0504:~/new$ chmod a+x increment
chmod: cannot access 'increment': No such file or directory
rohith1305@e1d77021d9b0504:~/new$ ls
and          case          default      if_else_example1.sh  multiple_conditions_if_else.sh  print_series
and-or       compare      elif_example1.sh  incrementt           nested_if_else.sh             print_series
arithmetic   comparing   greater        multiple_conditions_elif.sh  or                             single_line_if_else.sh
rohith1305@e1d77021d9b0504:~/new$ chmod a+x incrementt
rohith1305@e1d77021d9b0504:~/new$ ./incrementt
1
2
3
4
5
6
7
8
9
10
rohith1305@e1d77021d9b0504:~/new$ ~
```

The terminal window includes a menu bar with options like Select, Help, Write Out, Where Is, Cut, Paste, Execute, Justify, Location, Go To Line, Undo, Redo, Set Mark, Copy, To Bracket, and Where Was. The Windows taskbar is visible at the bottom.

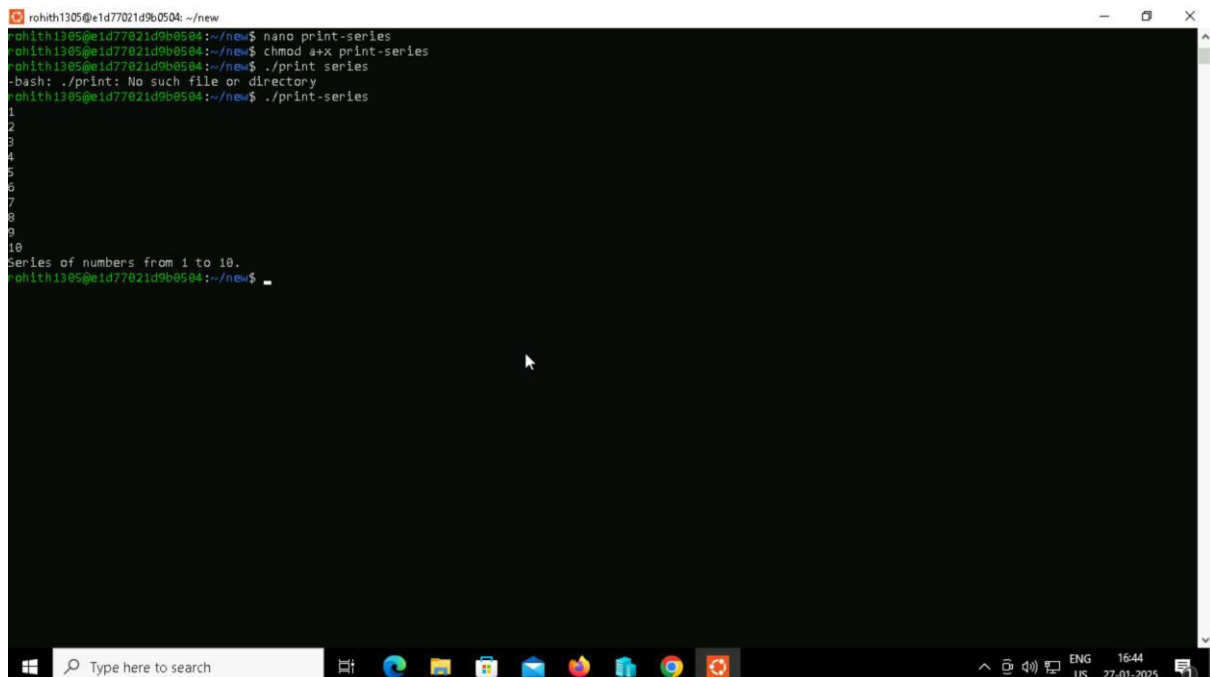
Print series using for:



```
rohith1305@e1d77021d9b0504: ~/new
GNU nano 7.2 print-series
#!/bin/bash

for num in {1..10}
do
    echo $num
done

echo "Series of numbers from 1 to 10."
```



```
rohith1305@e1d77021d9b0504: ~/new
rohith1305@e1d77021d9b0504:~/new$ nano print-series
rohith1305@e1d77021d9b0504:~/new$ chmod +x print-series
rohith1305@e1d77021d9b0504:~/new$ ./print-series
-bash: ./print: No such file or directory
rohith1305@e1d77021d9b0504:~/new$ ./print-series
1
2
3
4
5
6
7
8
9
10
Series of numbers from 1 to 10.
rohith1305@e1d77021d9b0504:~/new$
```

Using case statement:

```
rohith1305@e1d77021d9b0504: ~/new
GNU nano 7.2
/bin/bash

echo "Do you know Java Programming?"
read -p "Yes/No? :" Answer

case $Answer in
    Yes|yes|y|Y)
        echo "That's amazing."
        echo
        ;;
    No|no|N|n)
        echo "It's easy. Let's start learning from javatpoint."
        echo
        ;;
    *)
        ;;
esac
```

Help Write Out Where Is Cut Execute Location Undo Set Mark To Bracket
Exit Read File Replace Paste Justify Go To Line Redo Copy Where Was

Type here to search

```
rohith1305@e1d77021d9b0504: ~/new
/home/rohith1305/.hushlogin file.
rohith1305@e1d77021d9b0504:~$ cd new
rohith1305@e1d77021d9b0504:~/new$ ls
and          compare      greater      multiple_conditions_if_else.sh  single_line_if_else.sh
and-or       comparing   if_else     example1.sh  nested_if_else.sh
arithmetic   elif        example1.sh multiple_conditions_elif.sh    or
rohith1305@e1d77021d9b0504:~/new$ nano case
rohith1305@e1d77021d9b0504:~/new$ chmod +x case
rohith1305@e1d77021d9b0504:~/new$ ./case
Do you know Java Programming?
Yes/No? :Yes
That's amazing.

rohith1305@e1d77021d9b0504:~/new$
```

rohith1305@e1d77021d9b0504: ~/new

rohith1305@e1d77021d9b0504: ...

Default case statement:

```
rohith1305@e1d77021d9b0504: ~/new
GNU nano 7.2
default
#!/bin/bash

echo "Which Operating System are you using?"
echo "Windows, Android, Chrome, Linux, Others?"
read -p "Type your OS Name:" OS

case $OS in
    windows|windows)
        echo "That's common. You should try something new."
        echo
    ;;
    Android|android)
        echo "This is my favorite. It has lots of applications."
        echo
    ;;
    Chrome|chrome)
        echo "Cool!!! It's for pro users. Amazing Choice."
        echo
    ;;
    Linux|linux)
        echo "You might be serious about security!!"
        echo
    ;;
    *)
        echo "Sounds interesting. I will try that."
        echo
    ;;
esac
```

```
Select rohith1305@e1d77021d9b0504: ~/new

rohith1305@e1d77021d9b0504:~/new$ nano case
rohith1305@e1d77021d9b0504:~/new$ nano default
rohith1305@e1d77021d9b0504:~/new$ chmod +x default
rohith1305@e1d77021d9b0504:~/new$ ./default
Which Operating System are you using?
Windows, Android, Chrome, Linux, Others?
Type your OS Name:Chrome
Cool!!! It's for pro users. Amazing Choice.

rohith1305@e1d77021d9b0504:~/new$ _
```

While loop:



The image consists of two screenshots of a Windows terminal window. The top screenshot shows a nano editor editing a file named 'while'. The script inside the file is as follows:

```
#!/bin/bash
#Script to get specified numbers
read -p "Enter starting number: " snum
read -p "Enter ending number: " enum
while [[ $snum -le $enum ]];do
echo $snum
((snum++))
done
echo "This is the sequence that you wanted."
```

The bottom screenshot shows the terminal after the script has been executed. It displays the command sequence: `cd new`, `nano while`, `chmod a+x while`, and `./while`. The output shows the numbers 2 through 10 being printed, followed by the message "This is the sequence that you wanted."

While and OR:

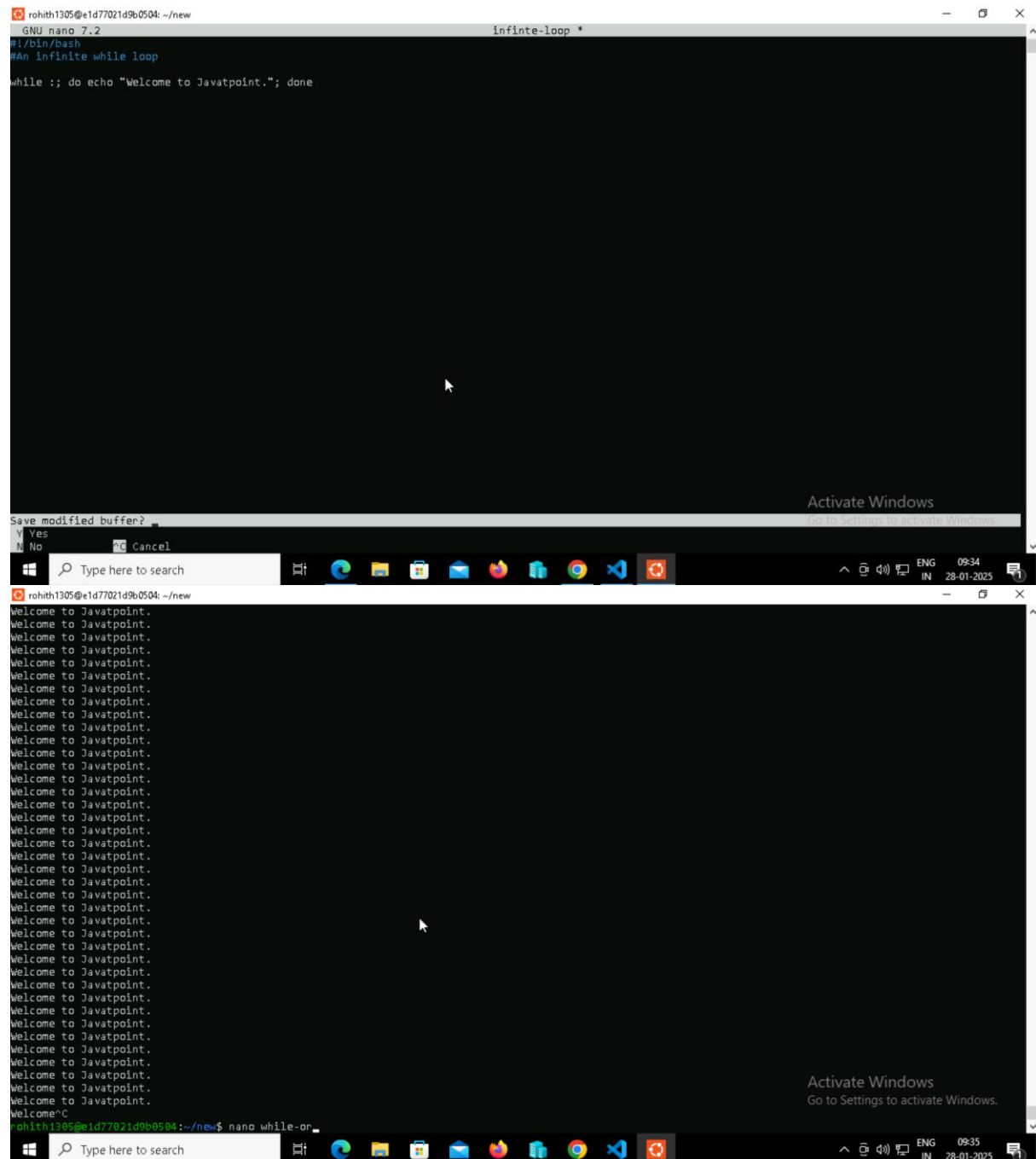


The image consists of two screenshots of a Windows terminal window. The top screenshot shows the creation of a script named 'while-or' using the nano text editor. The script prompts the user for a starting number (snum) and an ending number (enum), then prints a sequence of numbers from snum to enum using a while loop and the OR operator. The bottom screenshot shows the execution of the script. The user enters '2' for the starting number and '11' for the ending number, and the terminal displays the sequence of numbers from 2 to 11.

```
rohith1305@e1d77021d9b0504: ~/new
GNU nano 7.2
while-or
^C
script to get specified numbers
^C
read -p "Enter starting number: " snum
read -p "Enter ending number: " enum
while [[ $snum -lt $enum || $snum == $enum ]];
do
echo $snum
((snum++))
done
echo "This is the sequence that you wanted."
```

```
rohith1305@e1d77021d9b0504: ~/new
This is the sequence that you wanted.
rohith1305@e1d77021d9b0504:~/new$ nano while
rohith1305@e1d77021d9b0504:~/new$ chmod +x while-or
rohith1305@e1d77021d9b0504:~/new$ ./while-or
Enter starting number: 2
Enter ending number: 11
2
3
4
5
6
7
8
9
10
11
This is the sequence that you wanted.
rohith1305@e1d77021d9b0504:~/new$ nano while-or
rohith1305@e1d77021d9b0504:~/new$ ./while-or
```


Infinite loop using while:



The image consists of two screenshots of a Windows terminal window, demonstrating how to create and run an infinite loop using a `while` statement.

Top Screenshot: The terminal shows the user in a `new` directory, editing a file named `infinite-loop` using `GNU nano 7.2`. The code entered is:

```
#!/bin/bash
#An infinite while loop

while ;; do echo "Welcome to Javatpoint."; done
```

The prompt `Save modified buffer?` is visible at the bottom, with options `Y Yes` and `N No`.

Bottom Screenshot: The terminal shows the execution of the script. The user has run `./infinite-loop` (partially visible as `./infinite-loop` in the prompt area). The output is a continuous stream of the message "Welcome to Javatpoint." repeated many times, illustrating the infinite loop.

Break using while:

The screenshot displays a Windows 10 desktop with two terminal windows open.

The top terminal window is running the GNU nano 2.9.2 editor, editing a file named `break`. The script content is as follows:

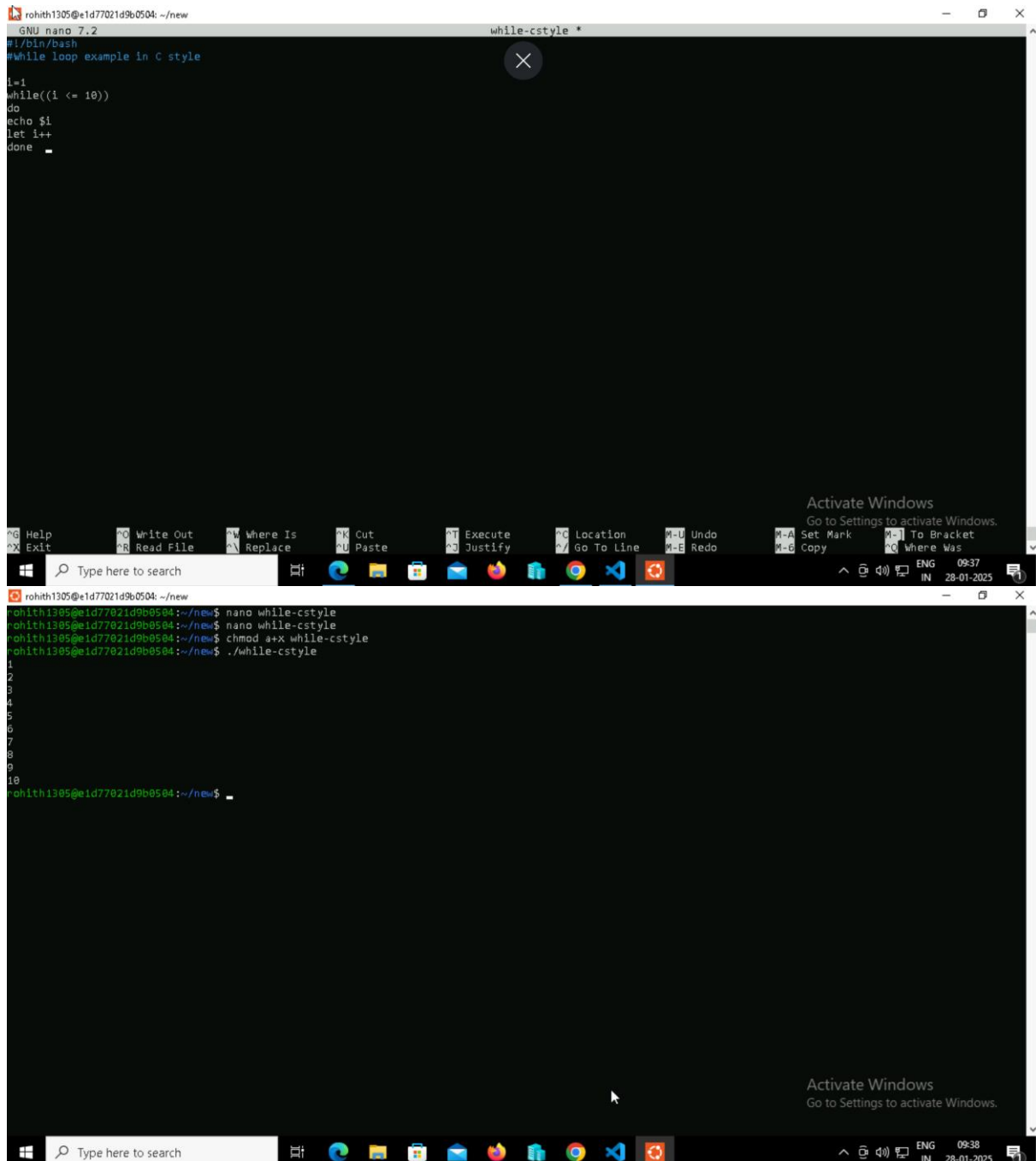
```
#!/bin/bash
#While Loop Example With a Break Statement

echo "Countdown for Website Launching..."
i=10
while [ $i -ge 1 ]
do
    echo [ $i == 2 ]
    if [ $i == 2 ]
    then
        echo "Mission Aborted, Some Technical Error Found."
        break
    fi
    echo "$i"
    (( i-- ))
done
```

The bottom terminal window shows the execution of the script. It prints "Welcome to Javatpoint." 10 times, followed by a countdown from 10 to 2. At the point where the script would print the number 2, it instead prints "Mission Aborted, Some Technical Error Found." and then returns to the prompt. The user has entered `break` at the prompt, which has been executed.

The Windows taskbar at the bottom shows the search bar, task view button, and several application icons including File Explorer, Edge, and the terminal application. The system tray on the right indicates the date as 28-01-2025 and the time as 09:36.

While using arithmetic symbol(()):



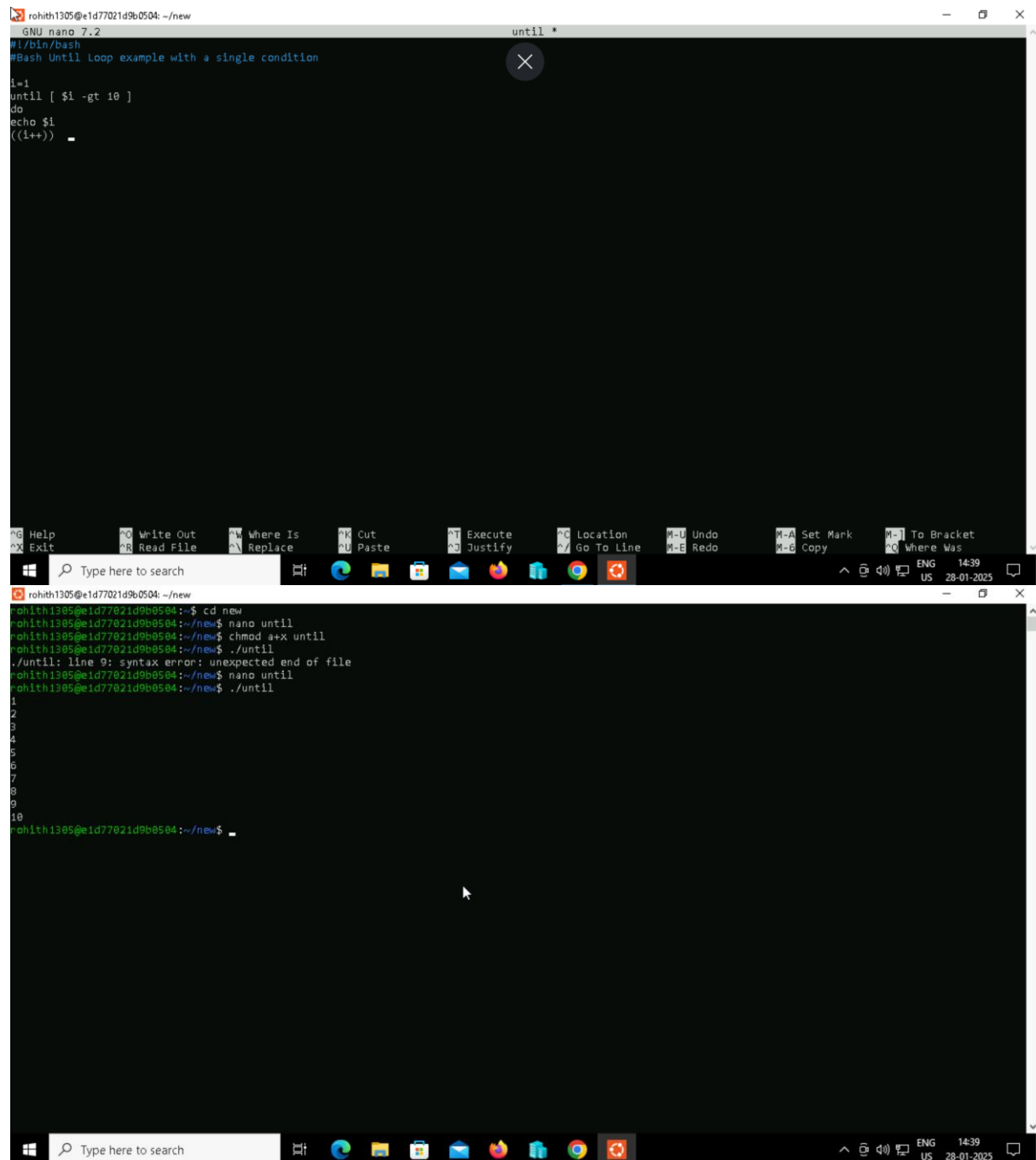
The image consists of two screenshots of a Windows terminal window. The top screenshot shows a nano editor editing a file named 'while-cstyle'. The code inside is a C-style while loop that prints numbers 1 through 10. The bottom screenshot shows the terminal after saving and running the script. The output is a list of numbers from 1 to 10, each on a new line.

```
rohith1305@e1d77021d9b0504: ~/new
GNU nano 7.2
#!/bin/bash
#While loop example in C style

i=1
while((i <= 10))
do
echo $i
let i++
done
```

```
rohith1305@e1d77021d9b0504: ~/new
rohith1305@e1d77021d9b0504:~/new$ nano while-cstyle
rohith1305@e1d77021d9b0504:~/new$ chmod a+x while-cstyle
rohith1305@e1d77021d9b0504:~/new$ ./while-cstyle
1
2
3
4
5
6
7
8
9
10
rohith1305@e1d77021d9b0504:~/new$
```

Print series of number using until:



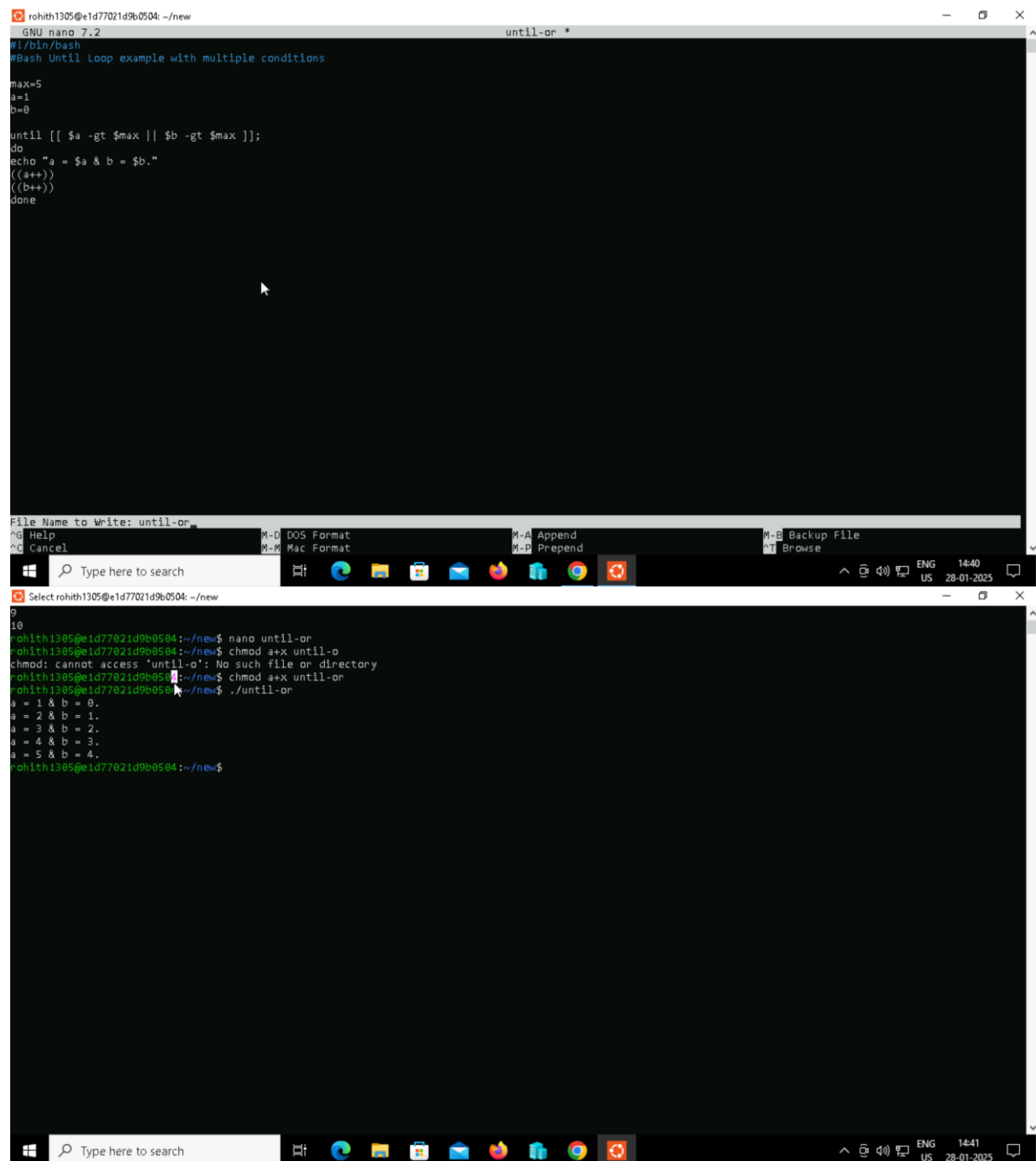
The image consists of two terminal window screenshots. The top screenshot shows a nano editor editing a file named 'until'. The script content is as follows:

```
GNU nano 7.2 until *
#!/bin/bash
#Bash Until Loop example with a single condition

i=1
until [ $i -gt 10 ]
do
echo $i
((i++))
```

The bottom screenshot shows the terminal execution of the script. The user navigates to the 'new' directory, creates the 'until' file with nano, sets permissions with 'chmod a+x until', and runs './until'. The output displays numbers 1 through 10, after which a syntax error is reported: 'syntax error: unexpected end of file'. The user then re-edits the file with nano and re-executes './until', which again produces the numbers 1 through 10.

Using multiple conditions:



The image consists of two screenshots of a Linux terminal window. The top screenshot shows a nano editor editing a file named 'until-or'. The script inside is a Bash until loop that increments variables 'a' and 'b' until 'a' is greater than or equal to 'max' or 'b' is greater than or equal to 'max'. The bottom screenshot shows the terminal after running the script, displaying the output of the loop.

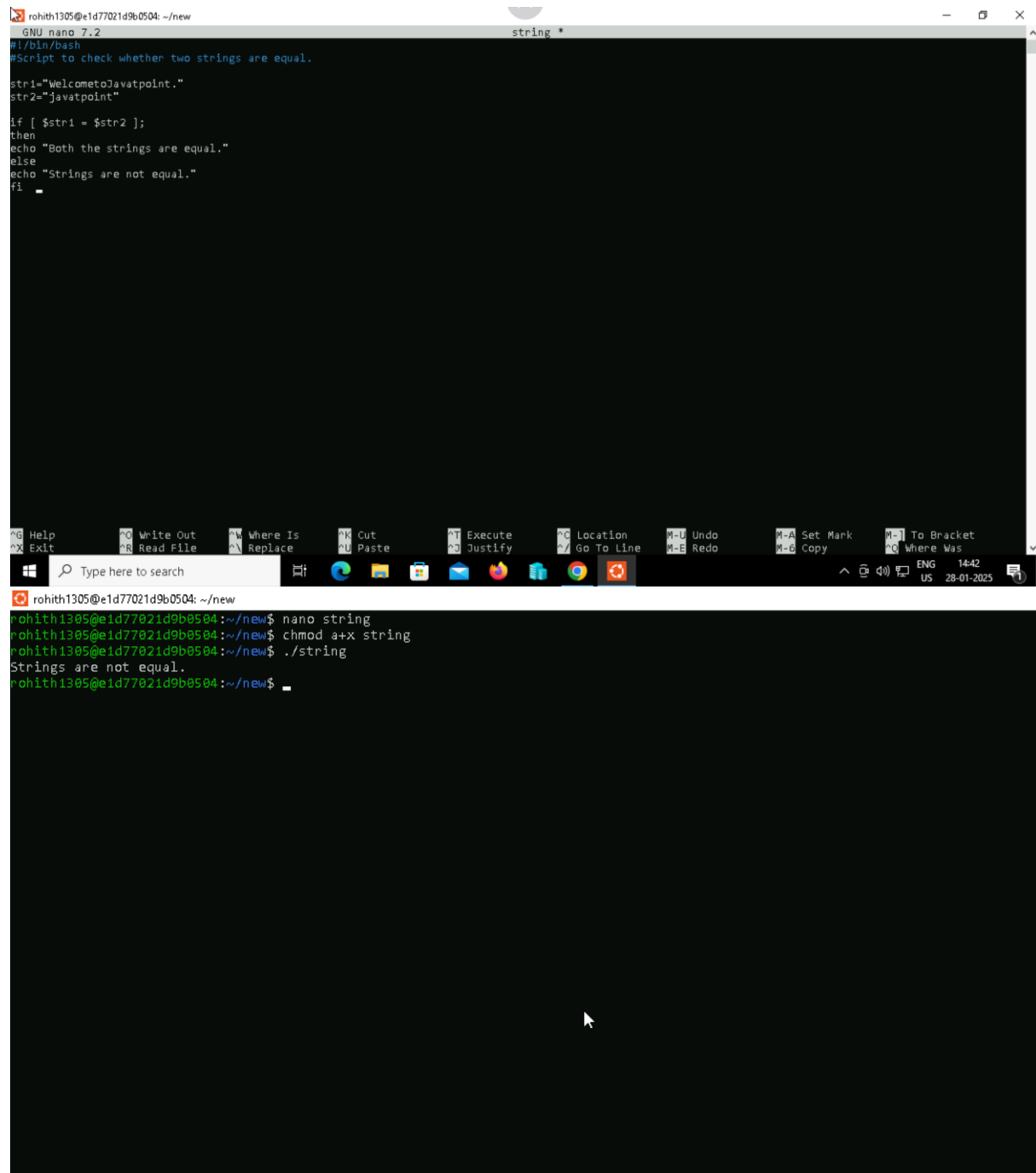
```
rohith1305@e1d77021d9b0504: ~/new
GNU nano 7.2      until-or *
#!/bin/bash
#Bash Until Loop example with multiple conditions

max=5
a=1
b=0

until [[ $a -gt $max || $b -gt $max ]];
do
echo "a = $a & b = $b."
((a++))
((b++))
done
```

```
rohith1305@e1d77021d9b0504: ~/new$ nano until-or
rohith1305@e1d77021d9b0504: ~/new$ chmod +x until-o
chmod: cannot access 'until-o': No such file or directory
rohith1305@e1d77021d9b0504: ~/new$ chmod +x until-or
rohith1305@e1d77021d9b0504: ~/new$ ./until-or
a = 1 & b = 0.
a = 2 & b = 1.
a = 3 & b = 2.
a = 4 & b = 3.
a = 5 & b = 4.
rohith1305@e1d77021d9b0504: ~/new$
```

Comparing two strings:



The image shows a Windows desktop environment with a terminal window open. The terminal window is titled "string *" and displays the contents of a file named "string" created with nano. The script inside the file compares two strings, "WelcometoJavatpoint." and "javatpoint", and prints a message based on whether they are equal. Below the terminal window, a command prompt window shows the execution of the script, which outputs "Strings are not equal.".

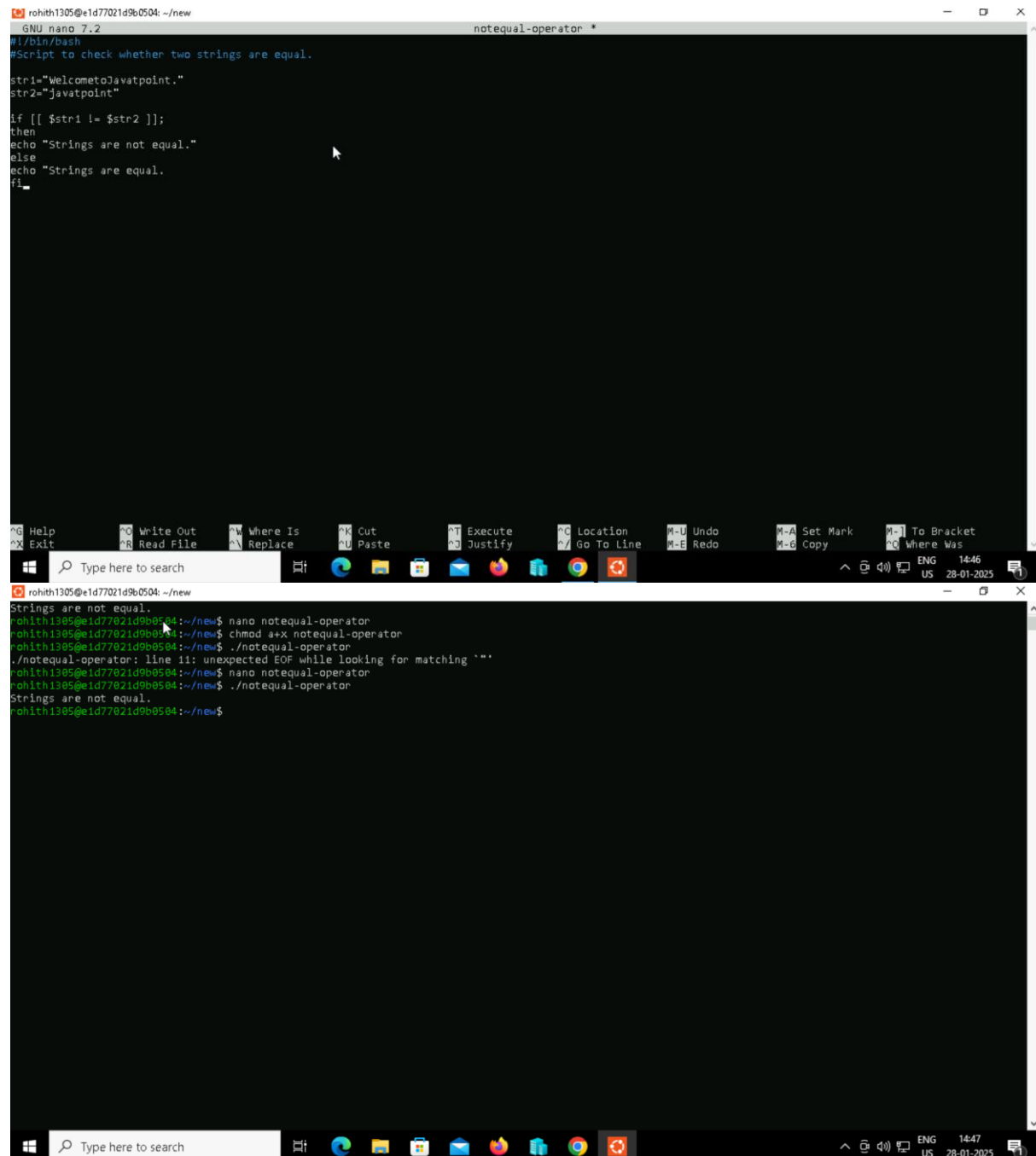
```
rohith1305@e1d77021d9b0504: ~/new
GNU nano 7.2
string *
#!/bin/bash
#Script to check whether two strings are equal.

str1="WelcometoJavatpoint."
str2="javatpoint"

if [ $str1 = $str2 ];
then
echo "Both the strings are equal."
else
echo "Strings are not equal."
fi
_

rohith1305@e1d77021d9b0504: ~/new
rohith1305@e1d77021d9b0504:~/new$ nano string
rohith1305@e1d77021d9b0504:~/new$ chmod a+x string
rohith1305@e1d77021d9b0504:~/new$ ./string
Strings are not equal.
rohith1305@e1d77021d9b0504:~/new$ _
```

Comparing strings using not equal operator:



The image consists of two screenshots of a terminal window, likely a Windows Subsystem for Linux (WSL) environment, showing the process of creating and running a shell script.

Top Screenshot: The terminal shows the user editing a file named `notequal-operator` using `GNU nano 7.2`. The script content is as follows:

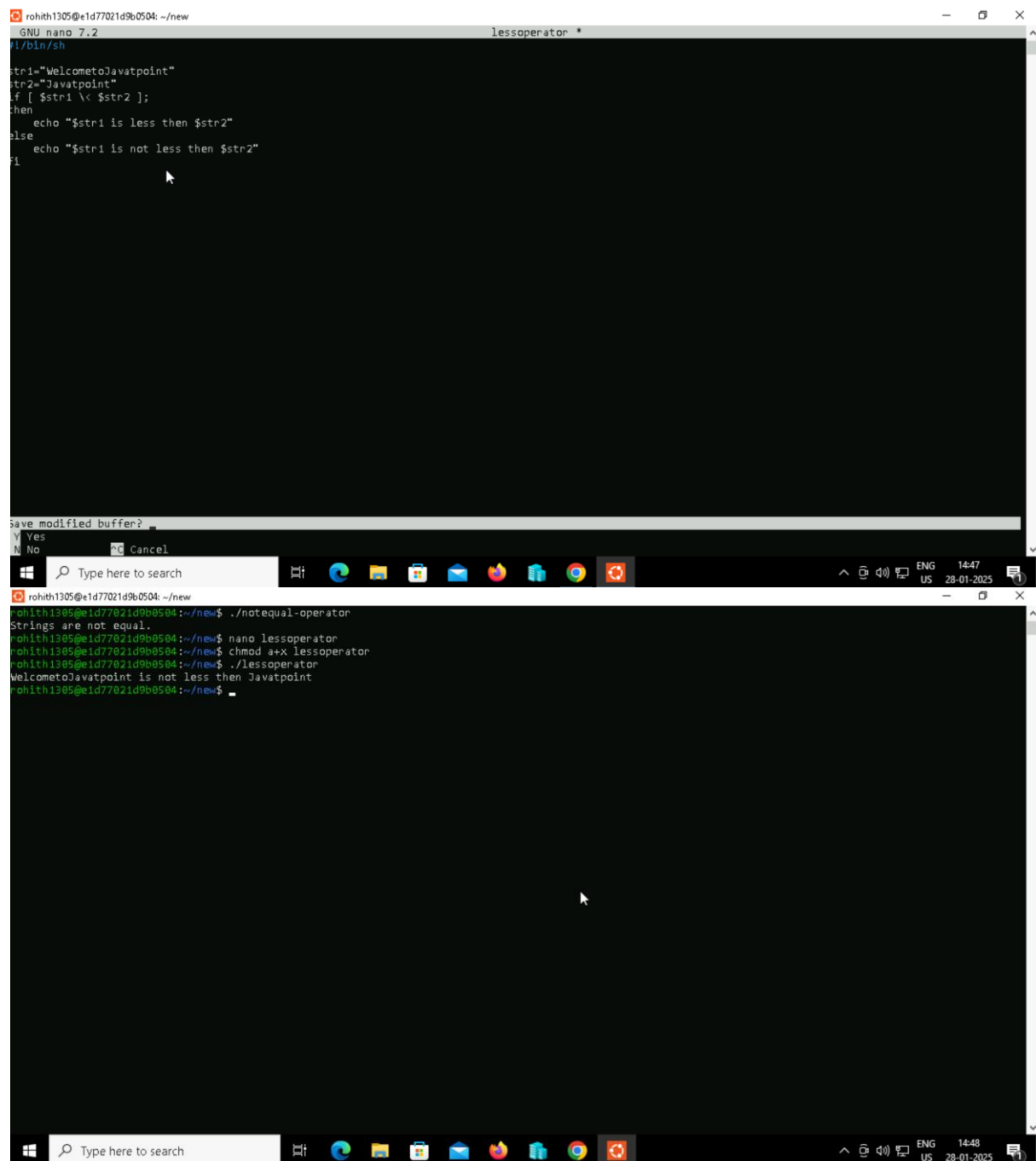
```
#!/bin/bash
#Script to check whether two strings are equal.

str1="WelcometoJavatpoint."
str2="javatpoint"

if [[ $str1 != $str2 ]];
then
echo "Strings are not equal."
else
echo "Strings are equal."
fi
```

Bottom Screenshot: The terminal shows the execution of the script. The user runs `./notequal-operator`, which outputs `Strings are not equal.`. The user then runs `chmod +x notequal-operator` to make the script executable. Subsequent runs of `./notequal-operator` continue to output `Strings are not equal.`. The terminal also shows the user editing the script again with `nano notequal-operator`.

Finding largest string:



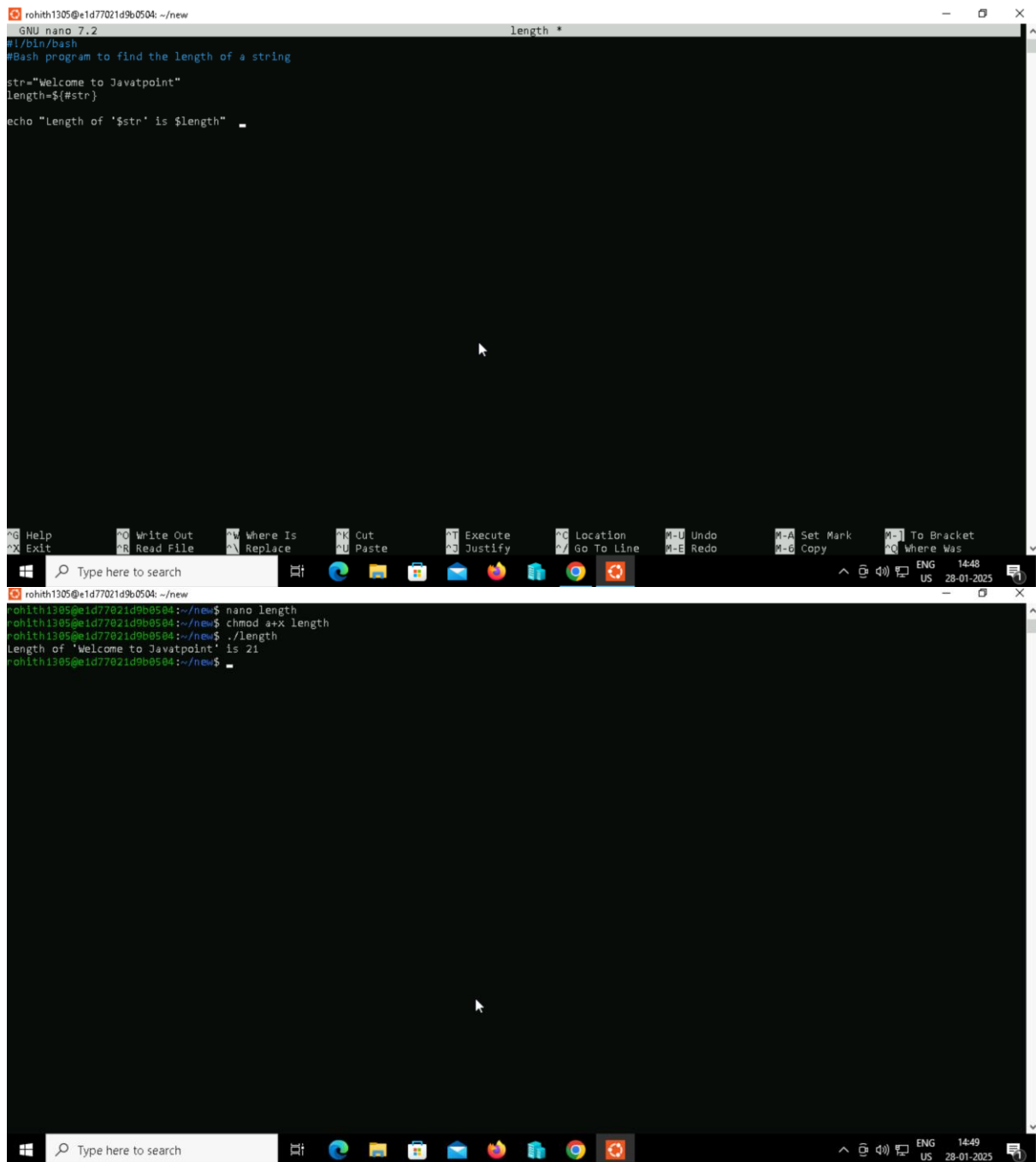
The image consists of two screenshots of a Linux terminal window. The top screenshot shows the user editing a file named 'lessoperator' using the 'nano' text editor. The script contains the following code:

```
str1="WelcometoJavatpoint"
str2="Javatpoint"
if [ $str1 \< $str2 ];
then
    echo "$str1 is less then $str2"
else
    echo "$str1 is not less then $str2"
fi
```

The bottom screenshot shows the user running the script. The terminal output is as follows:

```
rohith1305@e1d77021d9b0504: ~/new$ ./notequal-operator
Strings are not equal.
rohith1305@e1d77021d9b0504: ~/new$ nano lessoperator
rohith1305@e1d77021d9b0504: ~/new$ chmod a+x lessoperator
rohith1305@e1d77021d9b0504: ~/new$ ./lessoperator
WelcometoJavatpoint is not less then Javatpoint
rohith1305@e1d77021d9b0504: ~/new$
```


Finding length of string:



The image displays two sequential screenshots of a Linux terminal window, demonstrating the process of creating and running a shell script to find the length of a string.

Top Screenshot: The terminal shows the GNU nano 7.2 editor open with a file named `length`. The script content is as follows:

```
#!/bin/bash
#Bash program to find the length of a string

str="Welcome to Javatpoint"
length=${#str}

echo "Length of '$str' is $length"
```

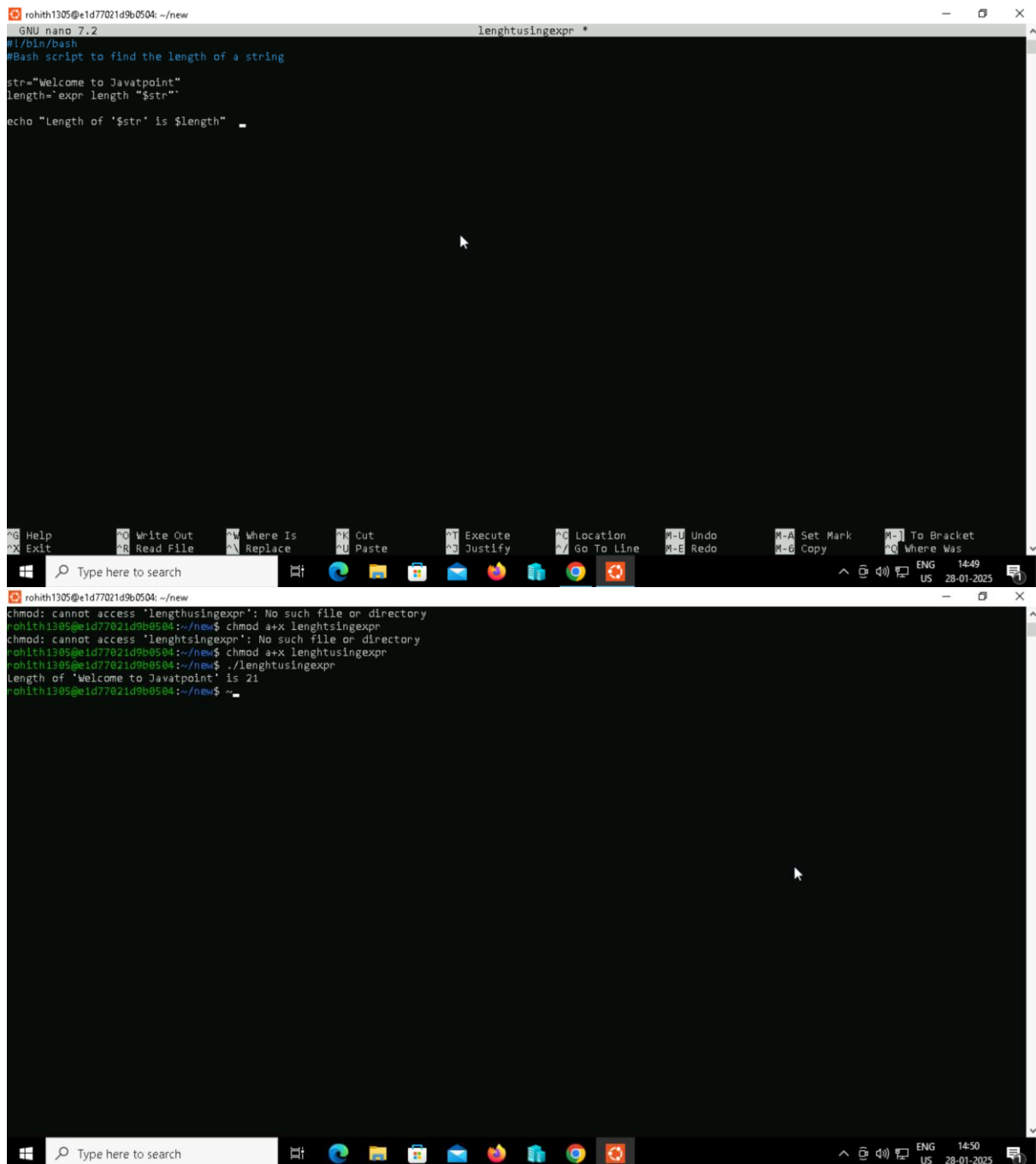
The bottom of the terminal shows the nano editor's menu bar with options like Help, Write Out, Where Is, Cut, Execute, Location, Undo, Set Mark, and To Bracket.

Bottom Screenshot: The terminal shows the execution of the script. The user runs `nano length`, then `chmod a+x length`, and finally `./length`. The output of the script is displayed:

```
rohith1305@e1d77021d9b0504: ~/new$ nano length
rohith1305@e1d77021d9b0504: ~/new$ chmod a+x length
rohith1305@e1d77021d9b0504: ~/new$ ./length
Length of 'Welcome to Javatpoint' is 21
rohith1305@e1d77021d9b0504: ~/new$
```

The terminal window's taskbar at the bottom shows various application icons and system status information, including the date 28-01-2025 and time 14:48.

Finding length of string using expr operator:



The image consists of two screenshots of a Linux terminal window. The top screenshot shows a nano editor editing a file named 'lengthusingexpr'. The script inside the file is as follows:

```
GNU nano 7.2 lengthusingexpr *
#!/bin/bash
#Bash script to find the length of a string

str="Welcome to Javatpoint"
length=`expr length "$str"`

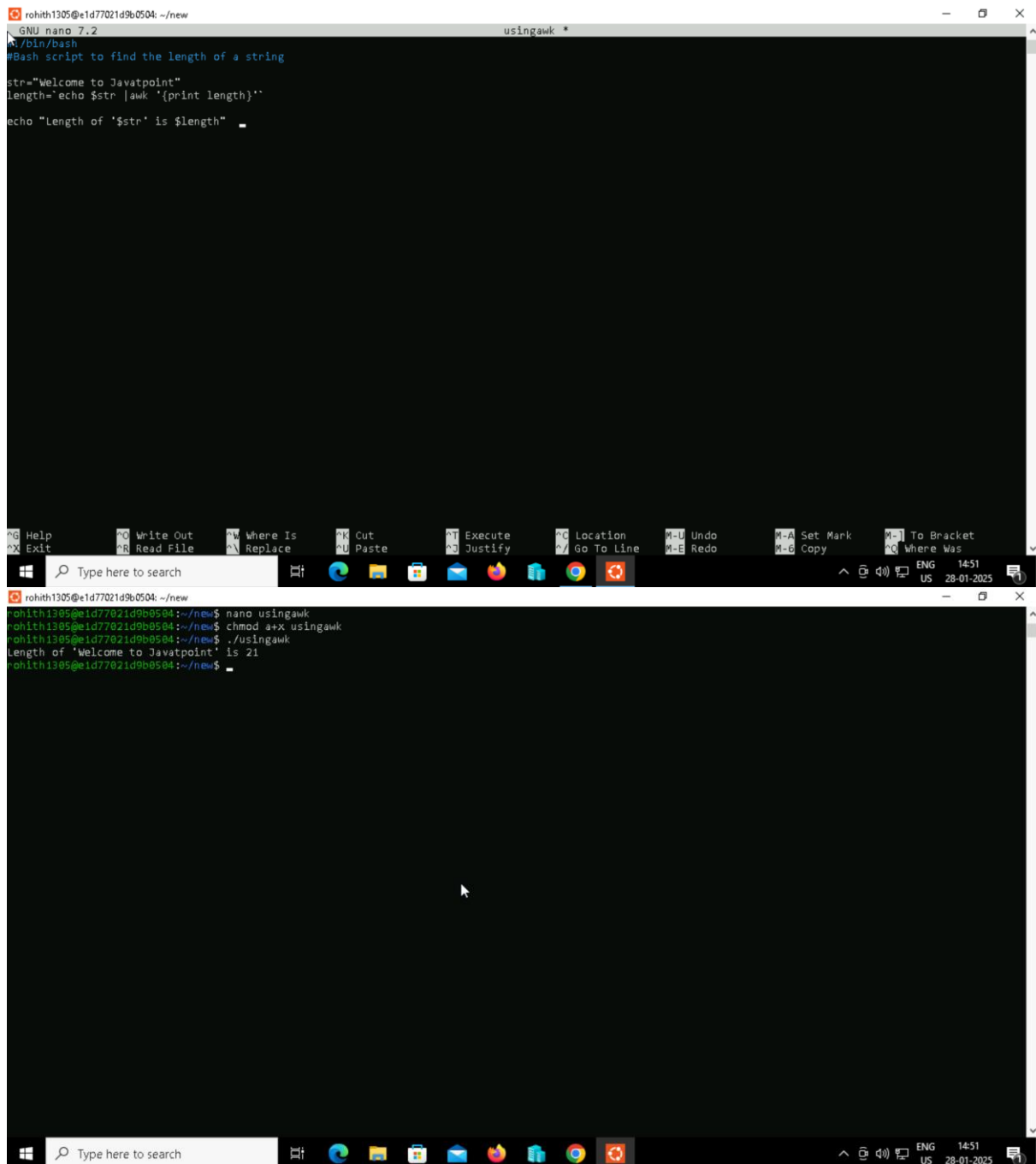
echo "Length of '$str' is $length"
```

The bottom screenshot shows the terminal after running the script. It displays the following commands and output:

```
rohith1305@e1d77021d9b0504: ~/new
chmod: cannot access 'lengthusingexpr': No such file or directory
rohith1305@e1d77021d9b0504:~/new$ chmod +x lengthusingexpr
chmod: cannot access 'lengthusingexpr': No such file or directory
rohith1305@e1d77021d9b0504:~/new$ chmod +x lengthusingexpr
rohith1305@e1d77021d9b0504:~/new$ ./lengthusingexpr
Length of 'Welcome to Javatpoint' is 21
rohith1305@e1d77021d9b0504:~/new$ ~
```

The terminal window includes a menu bar with options like Help, Write Out, Where Is, Cut, Execute, Location, Undo, Set Mark, To Bracket, Exit, Read File, Replace, Paste, Justify, Go To Line, Redo, Copy, and Where Was. The system tray at the bottom shows the date and time as 14:49 on 28-01-2025.

Finding length of string using awk:

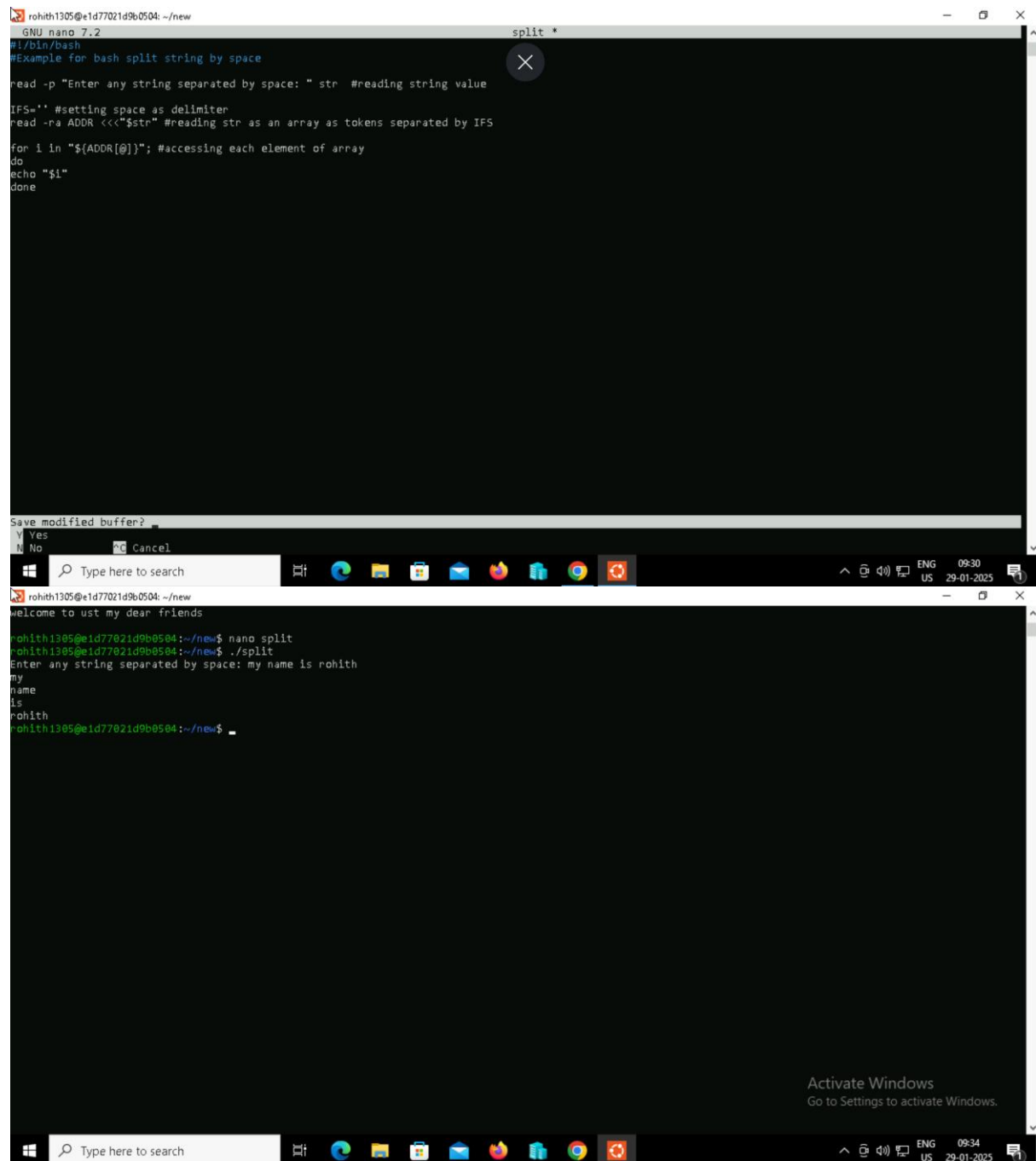


The image consists of two screenshots of a terminal window. The top screenshot shows the creation of a script named 'usingawk' using the 'nano' editor. The script contains a bash comment, a variable assignment for a string, and an 'echo' command that uses 'awk' to print the length of the string. The bottom screenshot shows the execution of the script after making it executable with 'chmod +x'. The output of the script is displayed in the terminal.

```
rohith1305@e1d77021d9b0504: ~/new
GNU nano 7.2
usingawk
#Bash script to find the length of a string
str="Welcome to Javatpoint"
length="echo $str |awk '{print length}'"
echo "Length of '$str' is $length"
```

```
rohith1305@e1d77021d9b0504: ~/new$ nano usingawk
rohith1305@e1d77021d9b0504: ~/new$ chmod +x usingawk
rohith1305@e1d77021d9b0504: ~/new$ ./usingawk
Length of 'Welcome to Javatpoint' is 21
rohith1305@e1d77021d9b0504: ~/new$
```

String split:



The image consists of two screenshots of a Linux terminal window. The top screenshot shows the creation of a script named 'split' using the nano text editor. The script's content is as follows:

```
GNU nano 7.2 split *
#!/bin/bash
#Example for bash split string by space

read -p "Enter any string separated by space: " str #reading string value

IFS="" #setting space as delimiter
read -ra ADDR <<<"$str" #reading str as an array as tokens separated by IFS

for i in "${ADDR[@]}"; #accessing each element of array
do
echo "$i"
done
```

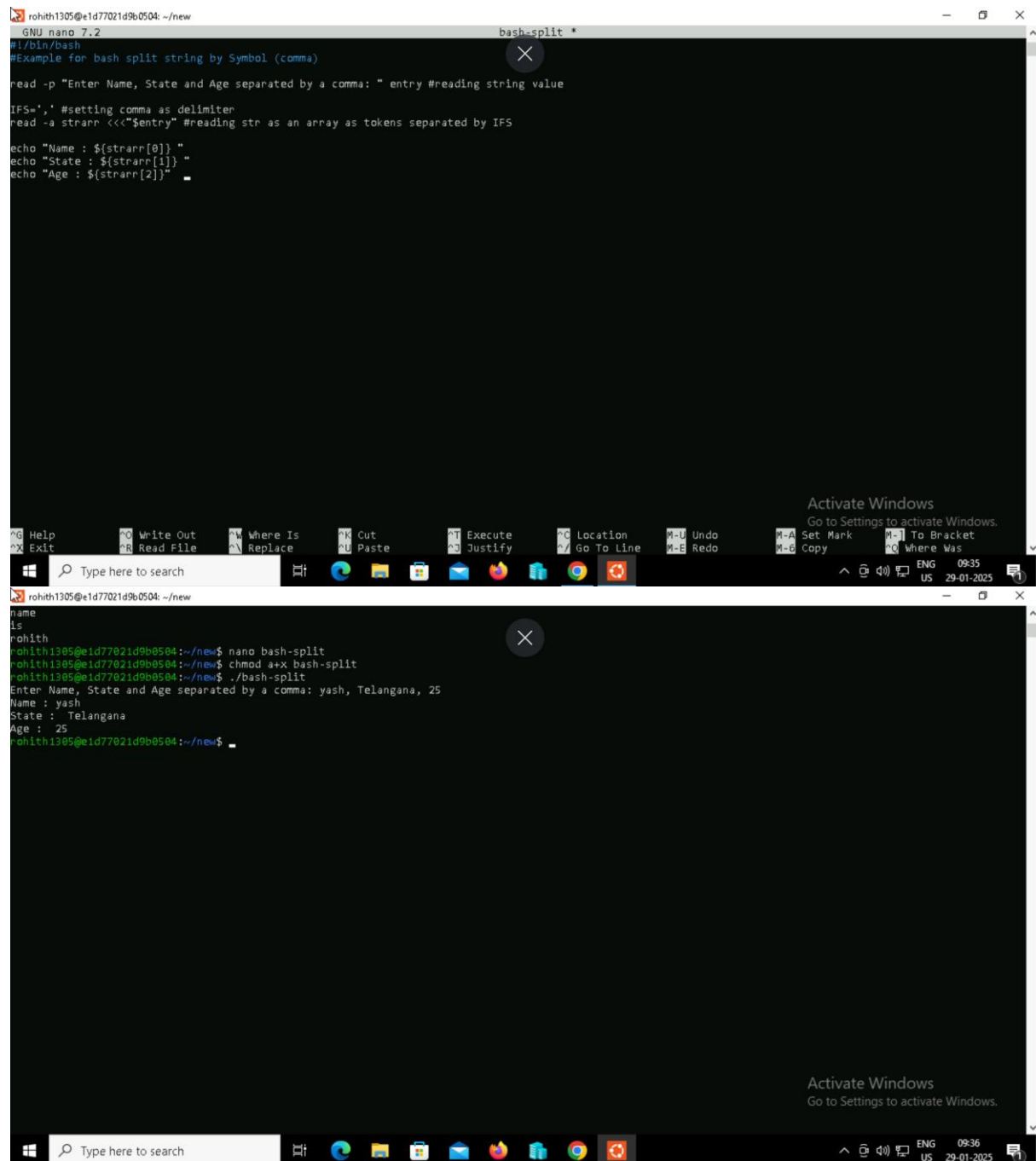
A 'Save modified buffer?' dialog box is visible at the bottom of the terminal.

The bottom screenshot shows the execution of the script. The user enters 'my name is rohith' in response to the prompt. The output of the script is:

```
rohith1305@e1d77021d9b0504: ~/new$ nano split
rohith1305@e1d77021d9b0504: ~/new$ ./split
Enter any string separated by space: my name is rohith
my
name
is
rohith
rohith1305@e1d77021d9b0504: ~/new$
```

The terminal window includes a taskbar at the bottom with various application icons and system status information (date: 29-01-2025, time: 09:34).

Split using arr:



The image consists of two screenshots of a Windows terminal window, showing the process of creating and running a bash script.

Top Screenshot: The terminal shows the creation of a file named `bash-split` using `nano`. The script content is as follows:

```
GNU nano 7.2 bash-split *
#!/bin/bash
#Example for bash split string by Symbol (comma)

read -p "Enter Name, State and Age separated by a comma: " entry #reading string value

IFS=',' #setting comma as delimiter
read -a strarr <<<"$entry" #reading str as an array as tokens separated by IFS

echo "Name : ${strarr[0]} "
echo "State : ${strarr[1]} "
echo "Age : ${strarr[2]} "
_
```

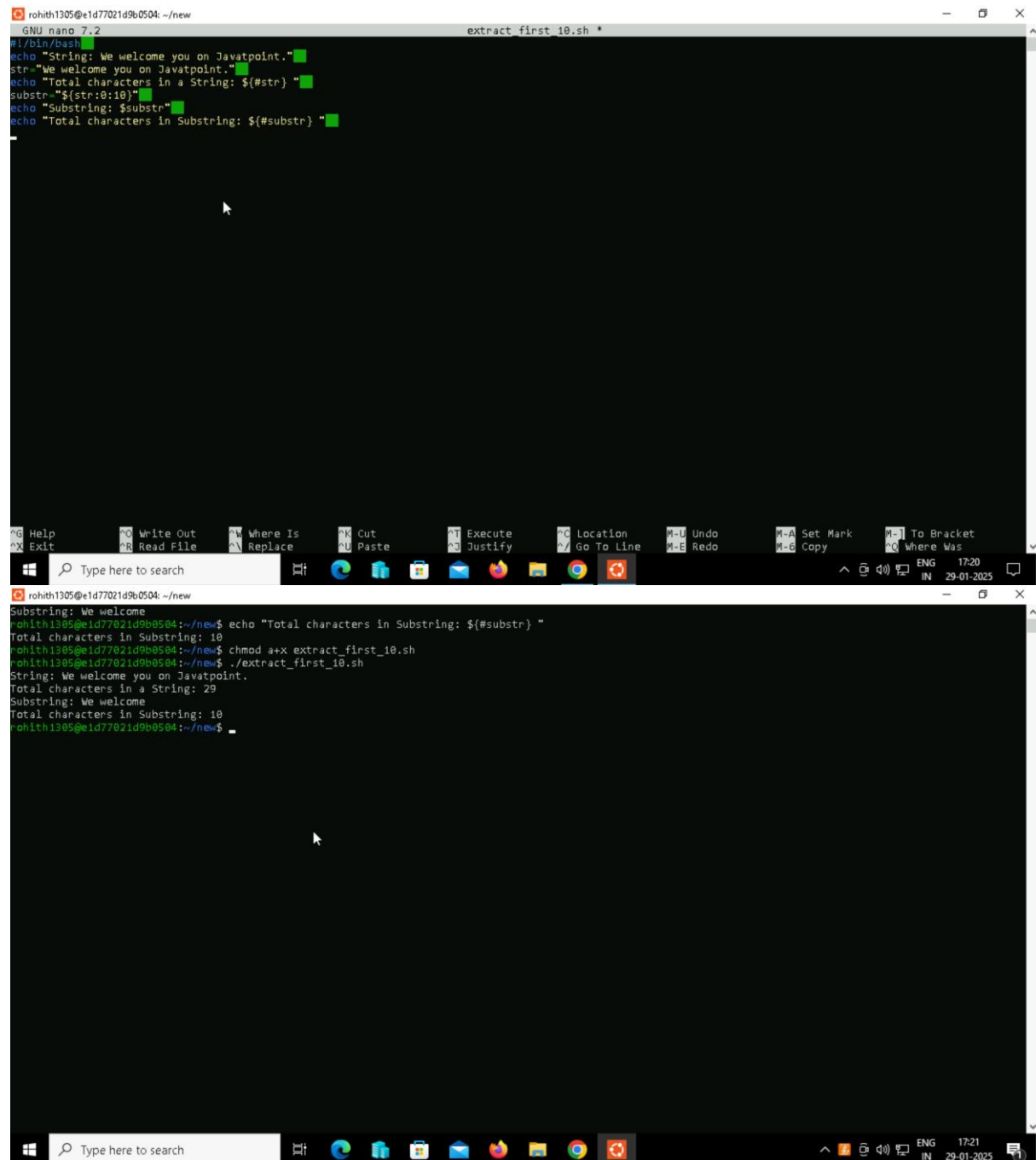
Bottom Screenshot: The terminal shows the execution of the script. The user enters `yash, Telangana, 25` in response to the prompt. The output is:

```
rohith1305@e1d77021d9b0504: ~/new
name
is
rohith
rohith1305@e1d77021d9b0504:~/new$ nano bash-split
rohith1305@e1d77021d9b0504:~/new$ chmod a+x bash-split
rohith1305@e1d77021d9b0504:~/new$ ./bash-split
Enter Name, State and Age separated by a comma: yash, Telangana, 25
Name : yash
State : Telangana
Age : 25
rohith1305@e1d77021d9b0504:~/new$ _
```

Split using loop:

```
rohith1305@e1d77021d9b0504: ~  
GNU nano 7.2 splitstring *  
#!/bin/bash  
#Example for bash split string by another string  
  
str="WeLearnWelcomeLearnYouLearnOnLearnJavatpoint"  
delimiter=Learn  
s=$str$delimiter  
array=();  
while [[ $s ]];  
do  
array+=( "${s%%$delimiter}" );  
s=${s#*$delimiter};  
done;  
declare -p array _  
  
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark  
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line  M-E Redo      M-G Copy  
  
rohith1305@e1d77021d9b0504: ~  
This message is shown once a day. To disable it please create the  
/home/rohith1305/.hushlogin file.  
rohith1305@e1d77021d9b0504:~$ nano splitstring  
rohith1305@e1d77021d9b0504:~$ chmod +x splitstring  
rohith1305@e1d77021d9b0504:~$ ./splitstring  
declare -a array=([0]="We" [1]="Welcome" [2]="You" [3]="On" [4]="Javatpoint")  
rohith1305@e1d77021d9b0504:~$
```

Extract string:



The image displays two screenshots of a terminal window, likely from a Windows environment, showing the execution of a shell script named `extract_first_10.sh`.

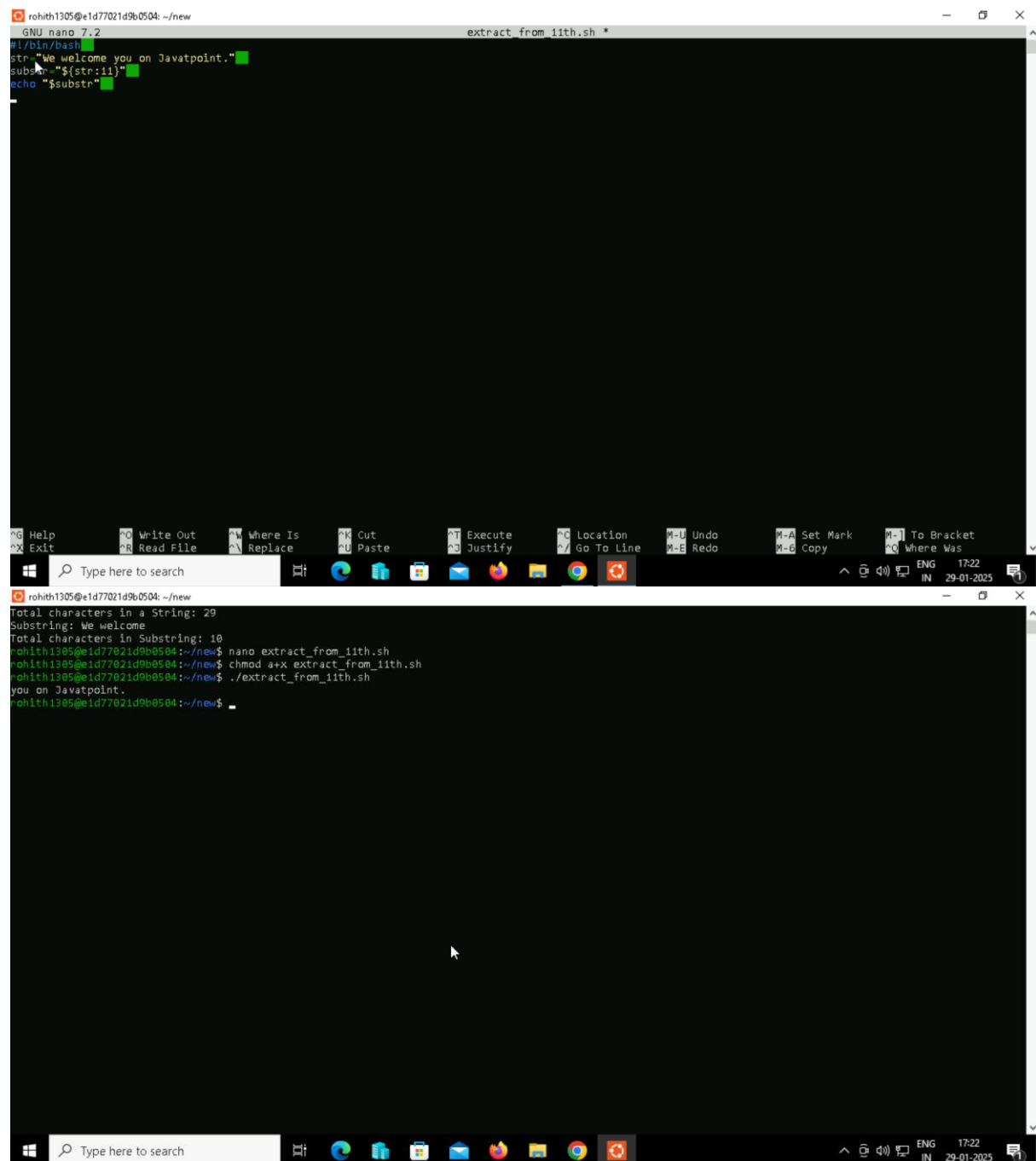
Top Screenshot: The terminal shows the script's content being edited in nano. The script defines a string `str="We welcome you on Javatpoint."`, calculates its total length using `echo "Total characters in a String: ${#str}"`, extracts the first 10 characters using `substr=${str:0:10}`, and then echoes the substring and its length.

```
GNU nano 7.2 extract_first_10.sh
#!/bin/bash
echo "String: We welcome you on Javatpoint."
str="We welcome you on Javatpoint."
echo "Total characters in a String: ${#str}"
substr=${str:0:10}
echo "Substring: $substr"
echo "Total characters in Substring: ${#substr}"
```

Bottom Screenshot: The terminal shows the script being executed. The output matches the script's logic, displaying the string, its total length (29), the extracted substring "We welcome", and the substring's length (10).

```
Substring: We welcome
rohith1305@e1d77021d9b0504: ~/new$ echo "Total characters in Substring: ${#substr}"
Total characters in Substring: 10
rohith1305@e1d77021d9b0504: ~/new$ chmod +x extract_first_10.sh
rohith1305@e1d77021d9b0504: ~/new$ ./extract_first_10.sh
String: We welcome you on Javatpoint.
Total characters in a String: 29
Substring: We welcome
Total characters in Substring: 10
rohith1305@e1d77021d9b0504: ~/new$
```

Extract certain position:

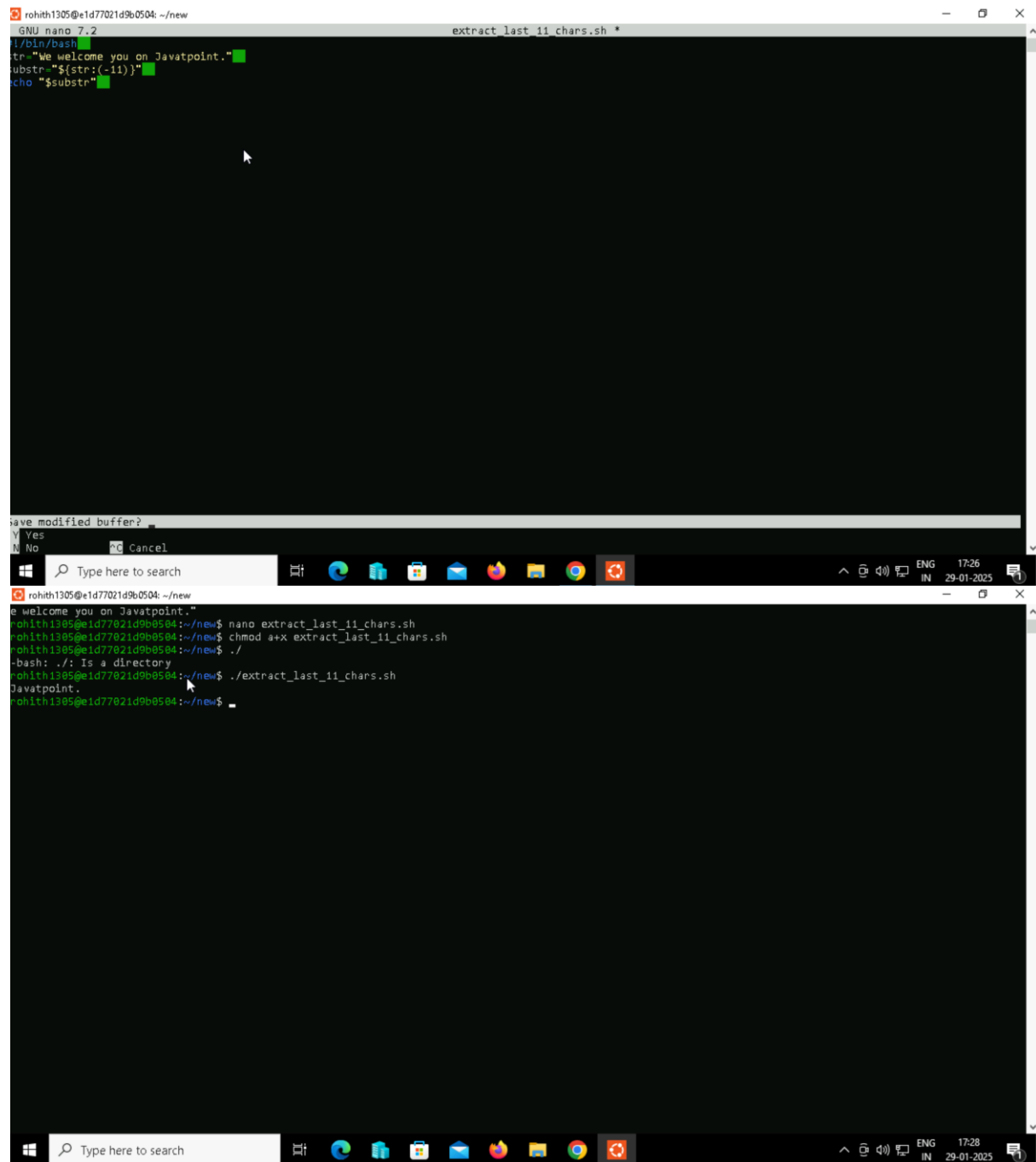


```
rohith1305@e1d77021d9b0504: ~/new
GNU nano 7.2 extract_from_11th.sh *
#!/bin/bash
str="We welcome you on Javatpoint."
substr="${str:11}"
echo "$substr"
```



```
rohith1305@e1d77021d9b0504: ~/new
Total characters in a String: 29
Substring: We welcome
Total characters in Substring: 10
rohith1305@e1d77021d9b0504:~/new$ nano extract_from_11th.sh
rohith1305@e1d77021d9b0504:~/new$ chmod a+x extract_from_11th.sh
rohith1305@e1d77021d9b0504:~/new$ ./extract_from_11th.sh
you on Javatpoint.
rohith1305@e1d77021d9b0504:~/new$
```


Extract last position:

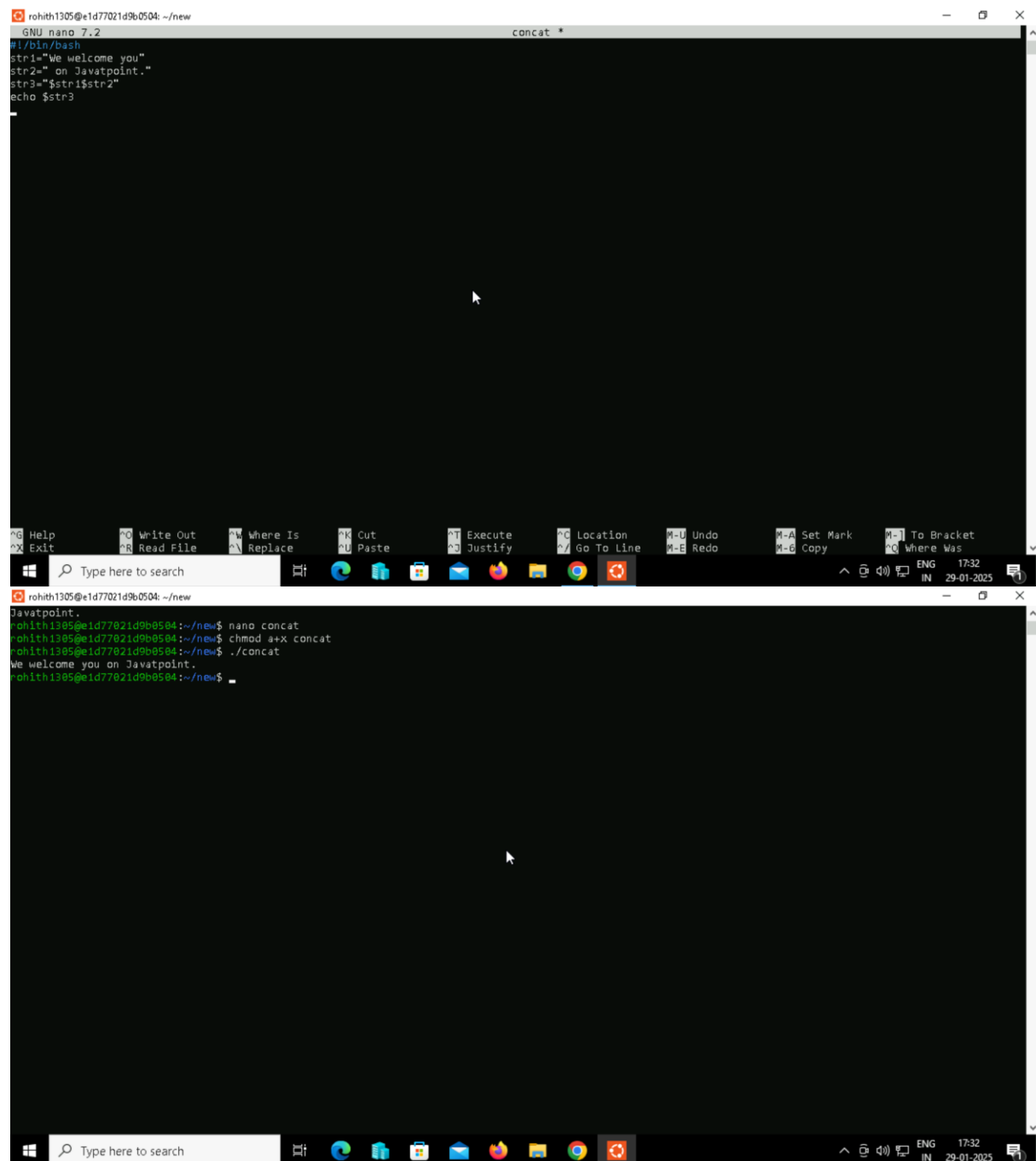


The image displays two sequential screenshots of a Windows terminal window. The top screenshot shows a nano editor session where a shell script named `extract_last_11_chars.sh` is being created. The script contains the following code:

```
#!/bin/bash
str="We welcome you on Javatpoint."
substr="${str:~-11}"
echo "$substr"
```

The bottom screenshot shows the terminal after the script has been saved and executed. The user runs `chmod +x extract_last_11_chars.sh` and then `./extract_last_11_chars.sh`, which outputs `Javatpoint.` The terminal window includes a taskbar at the bottom with various application icons and a system tray showing the date and time.

Concat string:



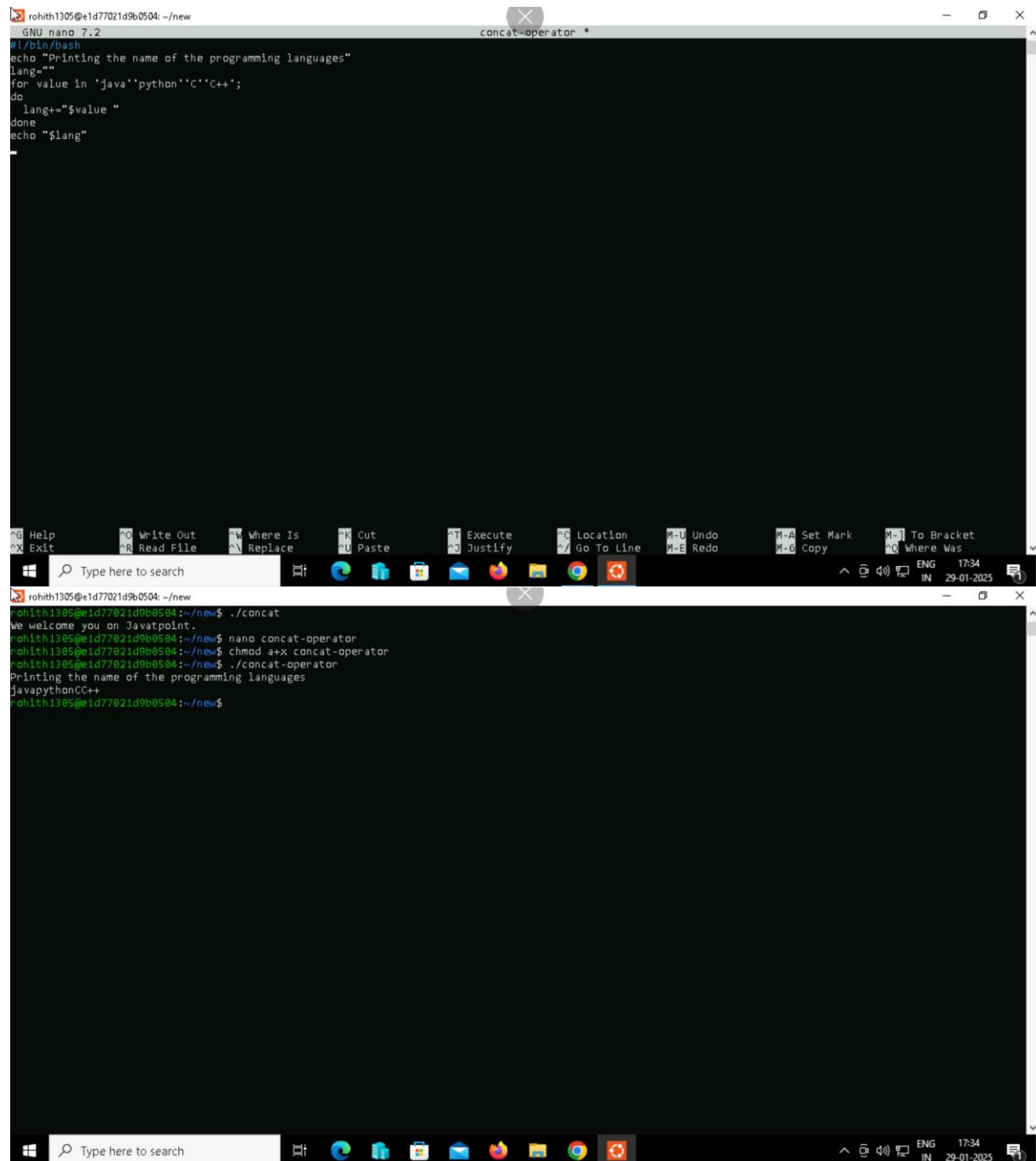
The image consists of two screenshots of a Windows environment. The top screenshot shows a terminal window with the GNU nano 7.2 editor open, editing a file named 'concat'. The script inside the editor is as follows:

```
#!/bin/bash
str1="We welcome you"
str2=" on Javatpoint."
str3="$str1$str2"
echo $str3
```

The bottom screenshot shows a terminal window where the script 'concat' has been executed. The output of the script is 'We welcome you on Javatpoint.' The terminal window shows the following commands and output:

```
Javatpoint.
rohith1305@e1d77021d9b0504: ~/new
rohith1305@e1d77021d9b0504:~/new$ nano concat
rohith1305@e1d77021d9b0504:~/new$ chmod a+x concat
rohith1305@e1d77021d9b0504:~/new$ ./concat
We welcome you on Javatpoint.
rohith1305@e1d77021d9b0504:~/new$
```

Concat string using operator:



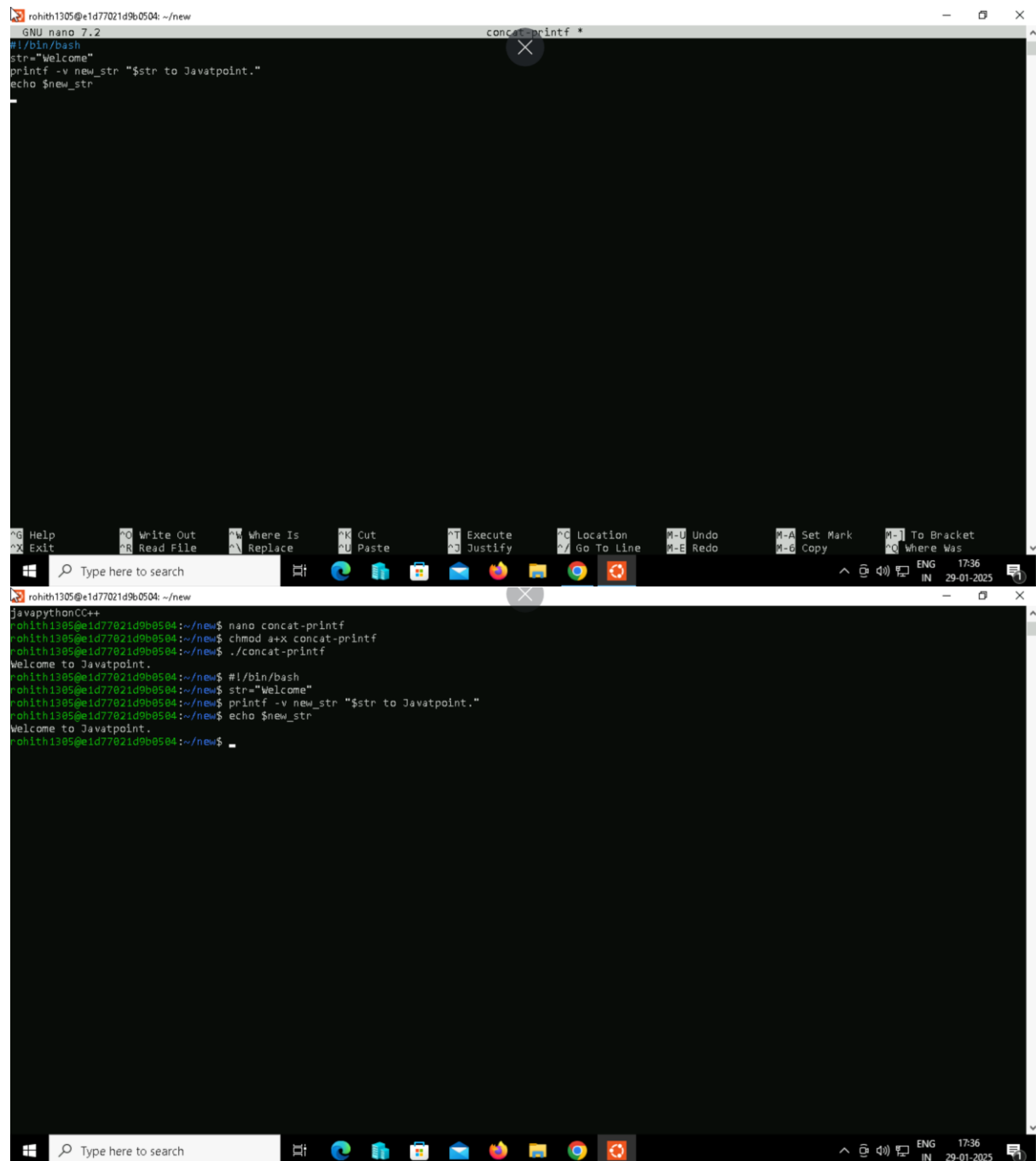
The image consists of two screenshots of a Linux terminal window. The top screenshot shows a nano editor editing a file named 'concat-operator'. The script inside the editor is as follows:

```
#!/bin/bash
echo "Printing the name of the programming languages"
lang=""
for value in 'java' 'python' 'C' 'C++';
do
    lang+="$value "
done
echo "$lang"
```

The bottom screenshot shows the terminal after the script has been executed. The user runs './concat', which outputs 'We welcome you on Javatpoint.'. Then, the user runs 'nano concat-operator' to edit the script, followed by 'chmod a+x concat-operator' to make it executable. Finally, the user runs './concat-operator', which outputs 'Printing the name of the programming languages' followed by 'javapythonCC++' on the next line.

```
rohith1305@e1d77021d9b0504: ~/new
rohith1305@e1d77021d9b0504:~/new$ ./concat
We welcome you on Javatpoint.
rohith1305@e1d77021d9b0504:~/new$ nano concat-operator
rohith1305@e1d77021d9b0504:~/new$ chmod a+x concat-operator
rohith1305@e1d77021d9b0504:~/new$ ./concat-operator
Printing the name of the programming languages
javapythonCC++
rohith1305@e1d77021d9b0504:~/new$
```

Concat string using printf:



The image displays two screenshots of a terminal window, likely from a Windows environment, showing the process of creating and running a C program that concatenates strings using printf.

Top Screenshot: The terminal shows the GNU nano 7.2 editor open with a file named `concat_printf.c`. The code inside the editor is:

```
#!/bin/bash
str="Welcome"
printf -v new_str "$str to Javatpoint."
echo $new_str
```

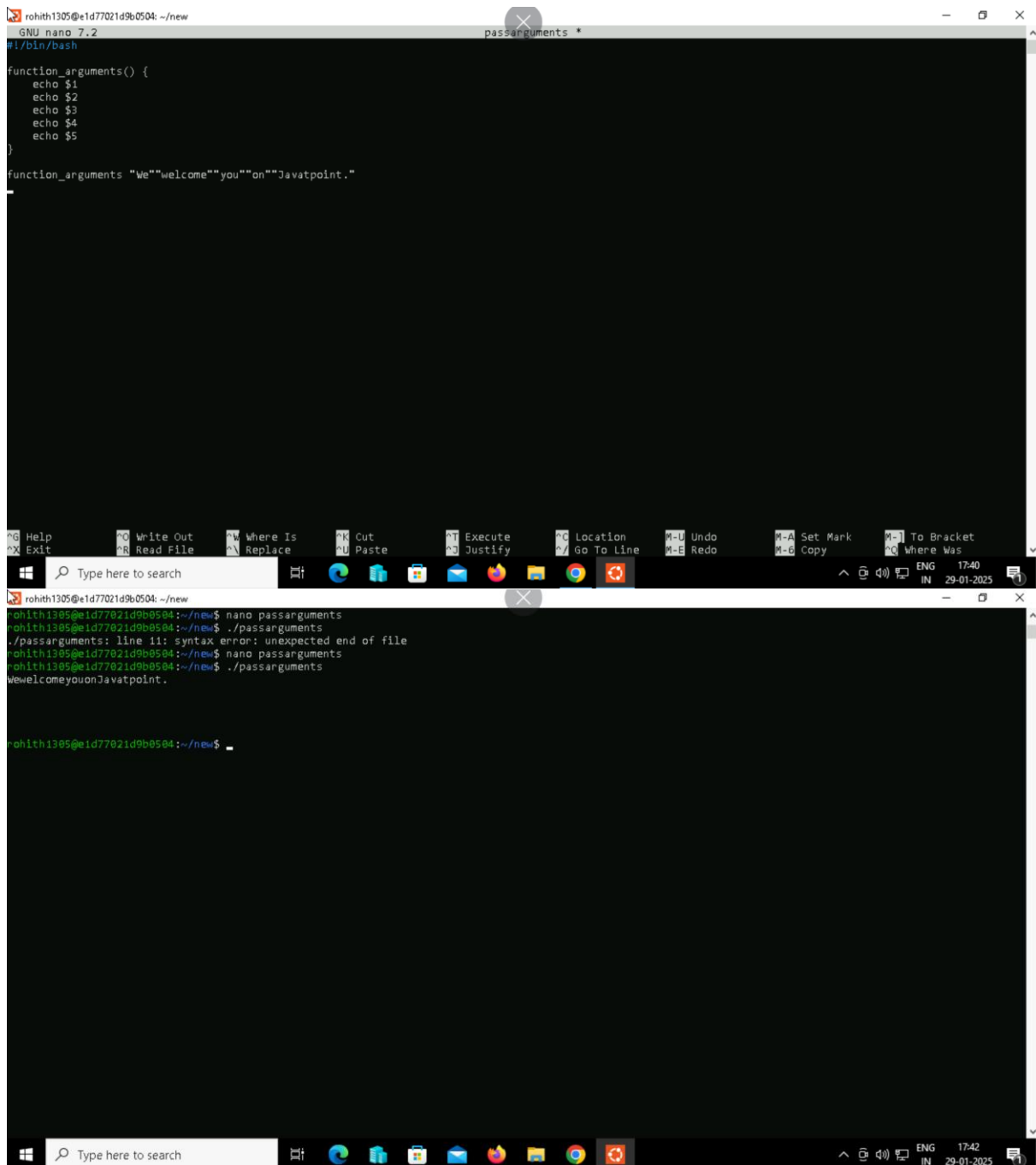
The terminal window includes a menu bar with options like Help, Write Out, Where Is, Cut, Execute, Location, Undo, Set Mark, and To Bracket.

Bottom Screenshot: The terminal shows the execution of the program. The user runs `nano concat-printf`, then `chmod a+x concat-printf`, and finally `./concat-printf`. The output of the program is:

```
Welcome to Javatpoint.
```

The terminal window also shows the user's prompt `rohith1305@e1d77021d9b0504: ~/new` and the system date and time (17:36, 29-01-2025).

Function pass arguments:



The image consists of two screenshots of a terminal window, likely from a Windows Subsystem for Linux (WSL) environment, showing the process of creating and running a shell script.

Top Screenshot: The terminal shows the user editing a file named `passarguments` using the `nano` text editor. The content of the file is as follows:

```
function_arguments() {  
    echo $1  
    echo $2  
    echo $3  
    echo $4  
    echo $5  
}  
  
function_arguments "We""welcome""you""on""Javatpoint."
```

The terminal window title is `passarguments *`. The bottom status bar shows the time as 17:40 on 29-01-2025.

Bottom Screenshot: The terminal shows the user running the script `./passarguments` from the `~/new` directory. The output of the script is displayed:

```
~rohith1305@e1d77021d9b0504: ~/new$ nano passarguments  
~rohith1305@e1d77021d9b0504: ~/new$ ./passarguments  
./passarguments: line 11: syntax error: unexpected end of file  
~rohith1305@e1d77021d9b0504: ~/new$ nano passarguments  
~rohith1305@e1d77021d9b0504: ~/new$ ./passarguments  
WewelcomeyouonJavatpoint.  
  
~rohith1305@e1d77021d9b0504: ~/new$
```

The bottom status bar shows the time as 17:42 on 29-01-2025.

Variable-scope:

```
GNU nano 7.2 variable-scope *
#!/bin/bash

v1='A'
v2='B'

my_var () {
    local v1='C'
    v2='D'
    echo "Inside Function"
    echo "v1 is $v1."
    echo "v2 is $v2."
}

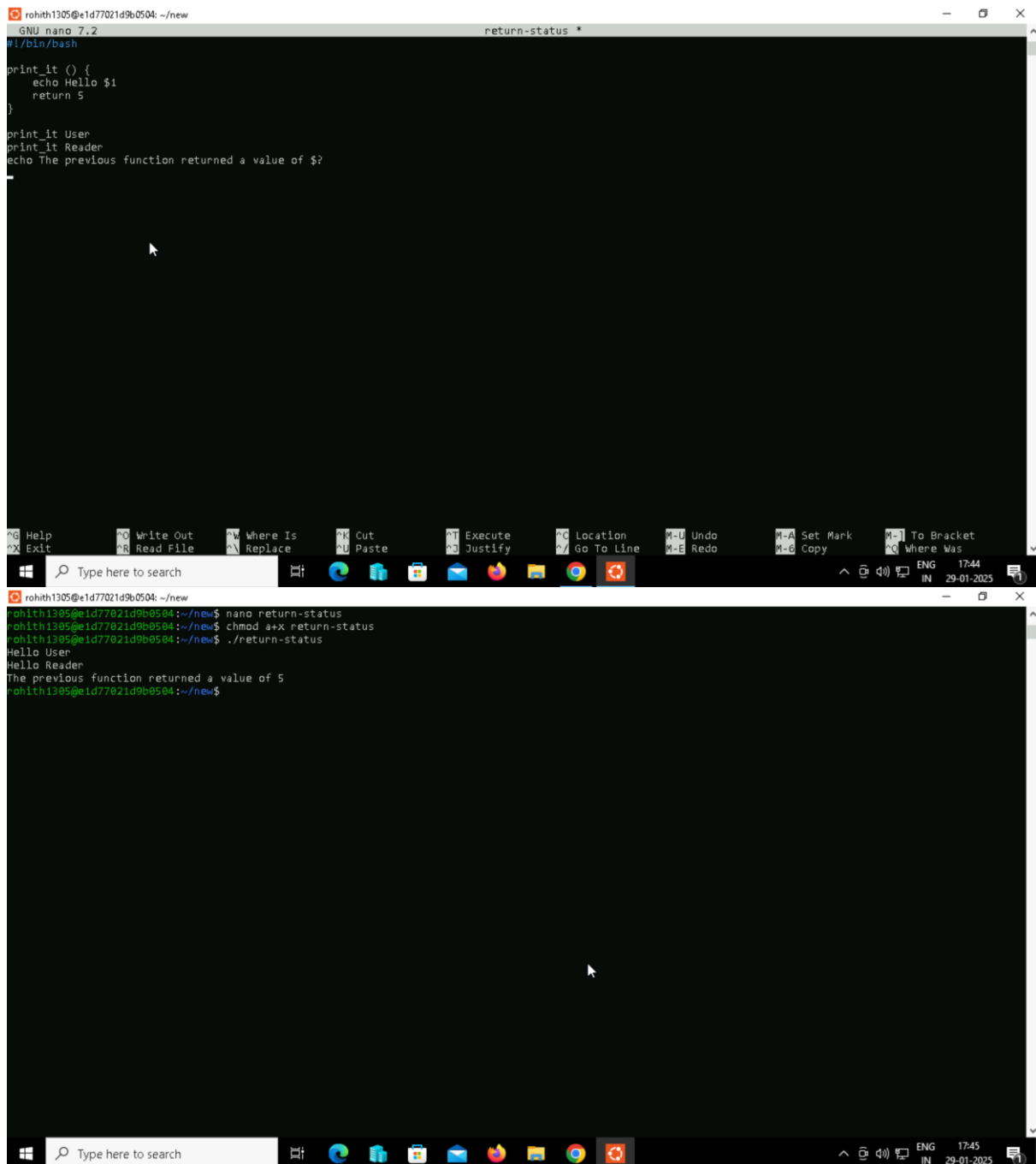
echo "Before Executing the Function"
echo "v1 is $v1."
echo "v2 is $v2."

my_var

echo "After Executing the Function"
echo "v1 is $v1."
echo "v2 is $v2."
```

```
rohith1305@e1d77021d9b0504: ~/new
rohith1305@e1d77021d9b0504:~/new$ nano variable-scope
rohith1305@e1d77021d9b0504:~/new$ chmod +x variable-scope
rohith1305@e1d77021d9b0504:~/new$ ./variable-scope
-bash: ./variable-scope: No such file or directory
rohith1305@e1d77021d9b0504:~/new$ ./variable-scope
Before Executing the Function
v1 is A.
v2 is B.
Inside Function
v1 is C.
v2 is D.
After Executing the Function
v1 is A.
v2 is D.
rohith1305@e1d77021d9b0504:~/new$
```

Function return status:



The image consists of two screenshots of a terminal window. The top screenshot shows the nano text editor editing a file named 'return-status'. The code defines a function 'print_it' that takes an argument, prints 'Hello \$1', and returns the value 5. Below the function definition, there are three lines of code: 'print_it User', 'print_it Reader', and 'echo The previous function returned a value of \$?'. The bottom screenshot shows the terminal after running the commands to create the file, set permissions, and execute it. The output shows 'Hello User', 'Hello Reader', and 'The previous function returned a value of 5'.

```
rohith1305@e1d77021d9b0504: ~/new
GNU nano 7.2 return-status *
#!/bin/bash

print_it () {
    echo Hello $1
    return 5
}

print_it User
print_it Reader
echo The previous function returned a value of $?

NG Help      WQ Write Out  W Where Is   C Cut         AT Execute   G Location  M-U Undo     M-A Set Mark M-T To Bracket
OX Exit      R Read File  R Replace    U Paste       T Justify    G Go To Line M-E Redo     M-G Copy     Q Where Was
```

```
rohith1305@e1d77021d9b0504: ~/new
rohith1305@e1d77021d9b0504:~/new$ nano return-status
rohith1305@e1d77021d9b0504:~/new$ chmod +x return-status
rohith1305@e1d77021d9b0504:~/new$ ./return-status
Hello User
Hello Reader
The previous function returned a value of 5
rohith1305@e1d77021d9b0504:~/new$
```