Appendix G: Perl Environment and Scripting

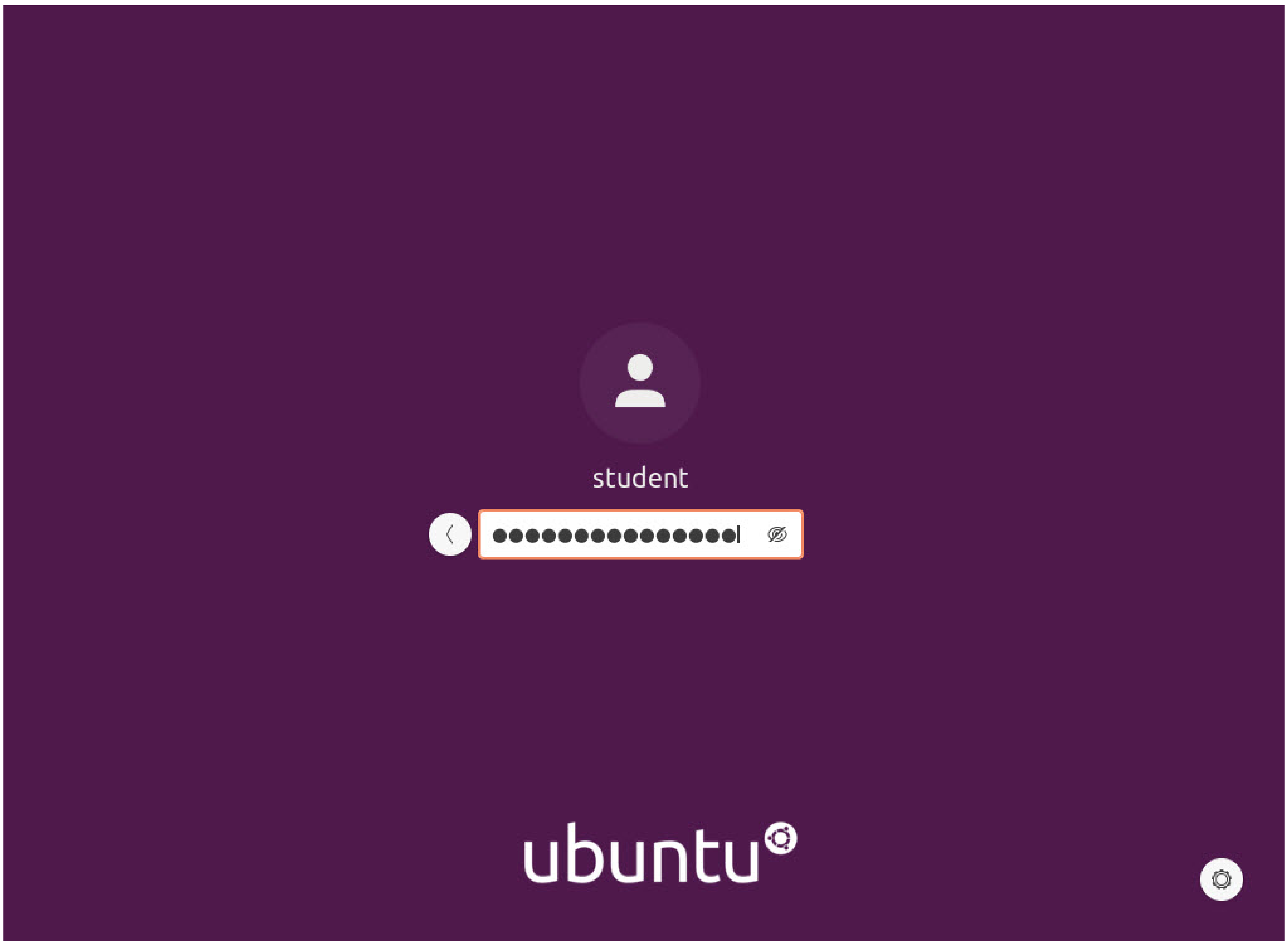
Exercise 1: Basic PERL Queries

Objectives

- Create basic Perl queries and extract data and explore the features and components of Perl

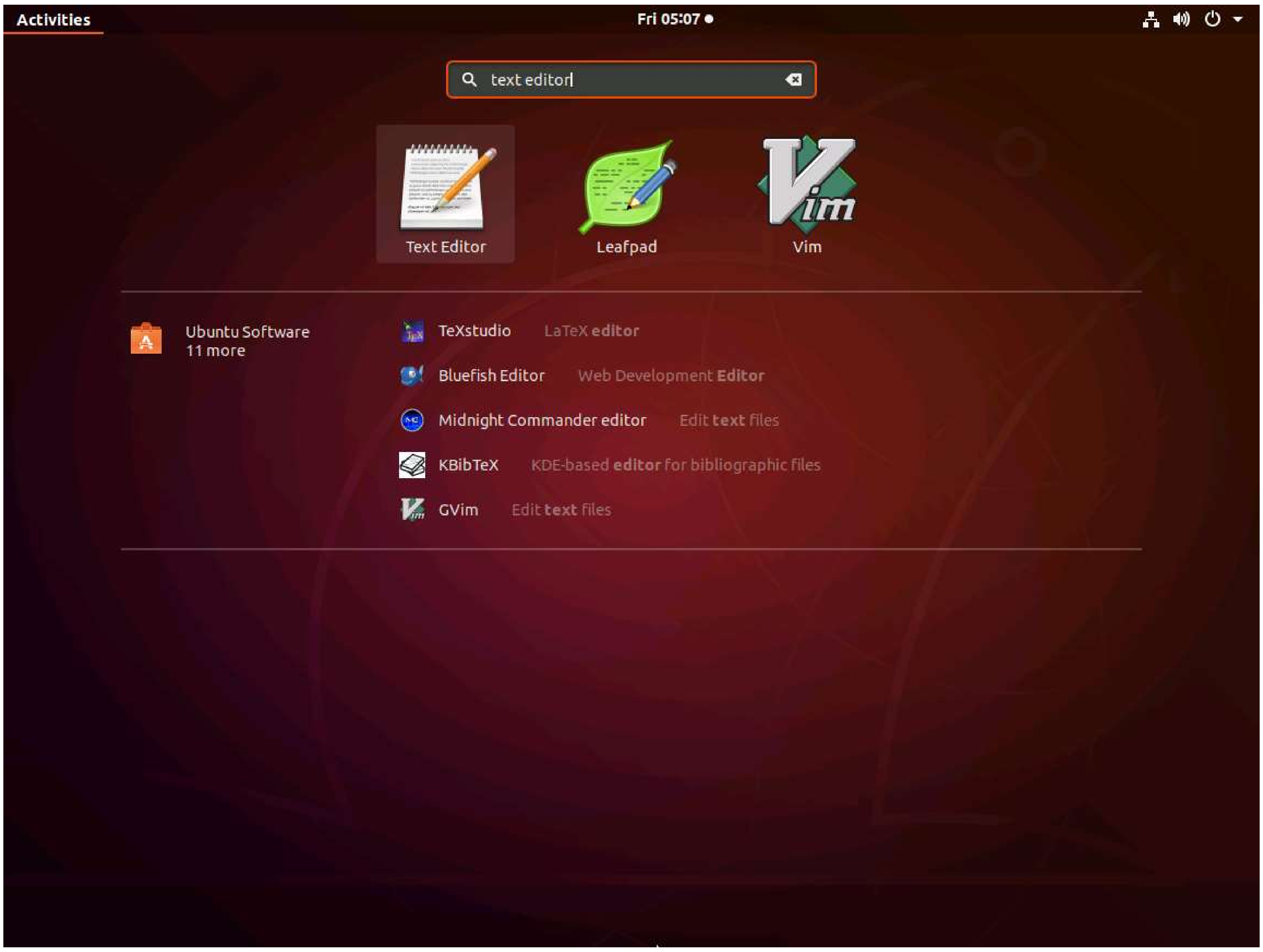
Lab Duration: **20** Minutes

1. By default **CPENT Ubuntu-Perl** selected, click **student**.
2. Type **studentpassword** in the Password field and press **Enter** or click **Sign In** to login

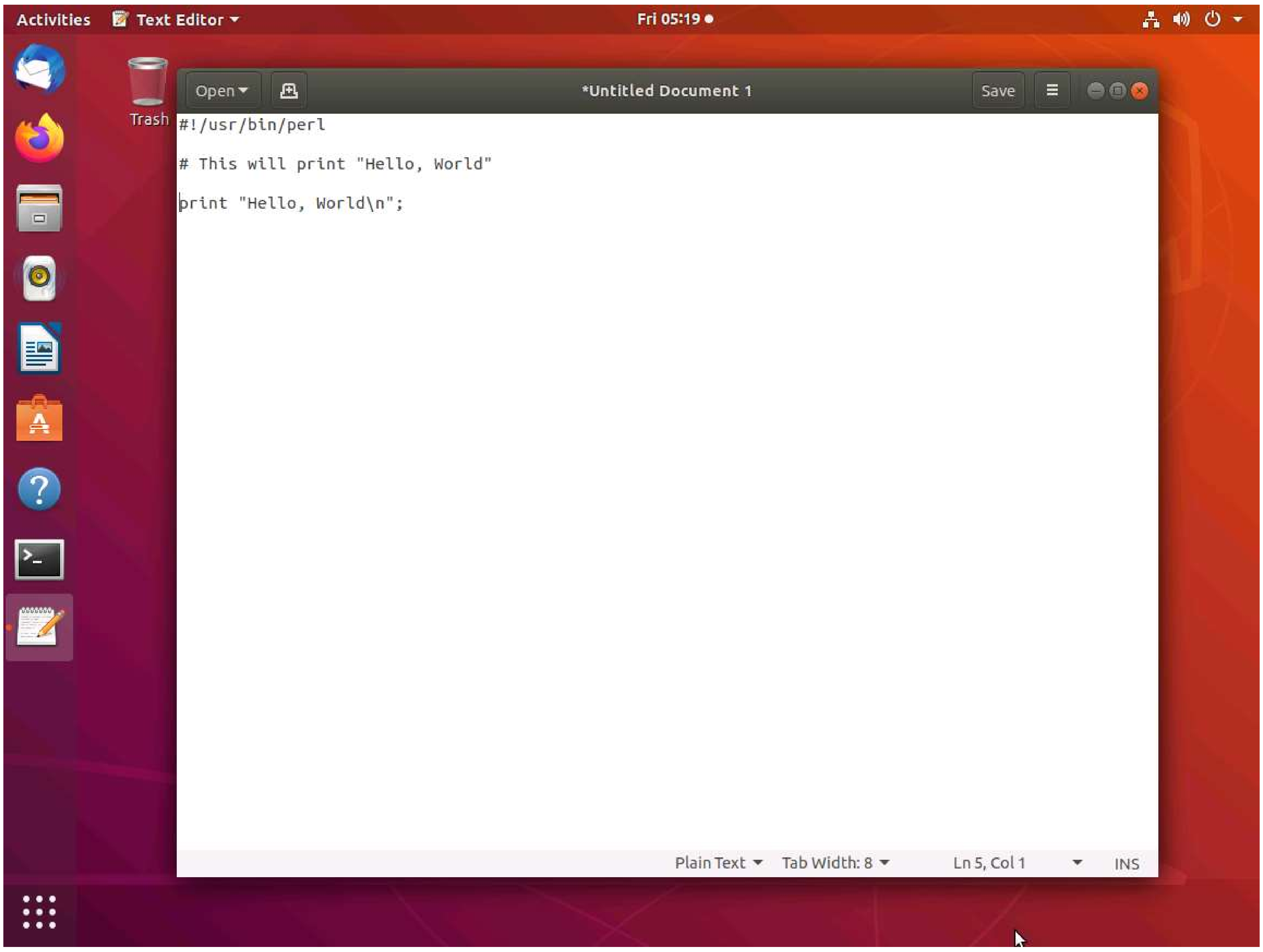


3. Click **Show Applications** at lower left corner of the Desktop. In the Search field type **text editor**. From the search results click **Text Editor**.

Note: You can launch any text editor of your choice.

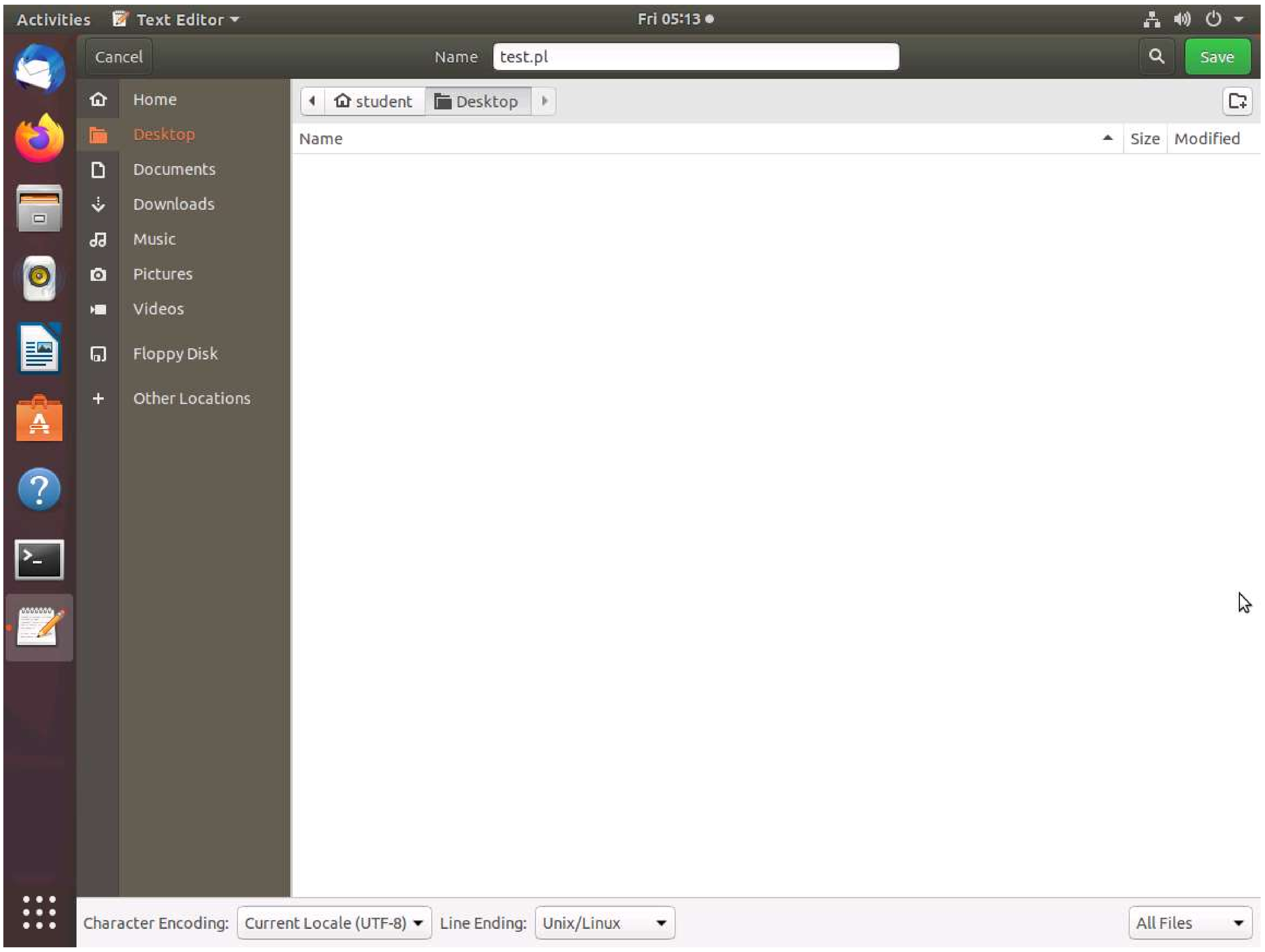

Activities
Fri 05:07 ●
text editor
Text Editor
Leafpad
Vim
Ubuntu Software
11 more
TeXstudio LaTeX editor
Bluefish Editor Web Development Editor
Midnight Commander editor Edit text files
KBibTeX KDE-based editor for bibliographic files
GVim Edit text files

4. Create the first **simple program**. Type the following code in the **Text Editor** as shown in the screenshot. Once you have entered the code click **Save** at top right corner of the Text Editor.

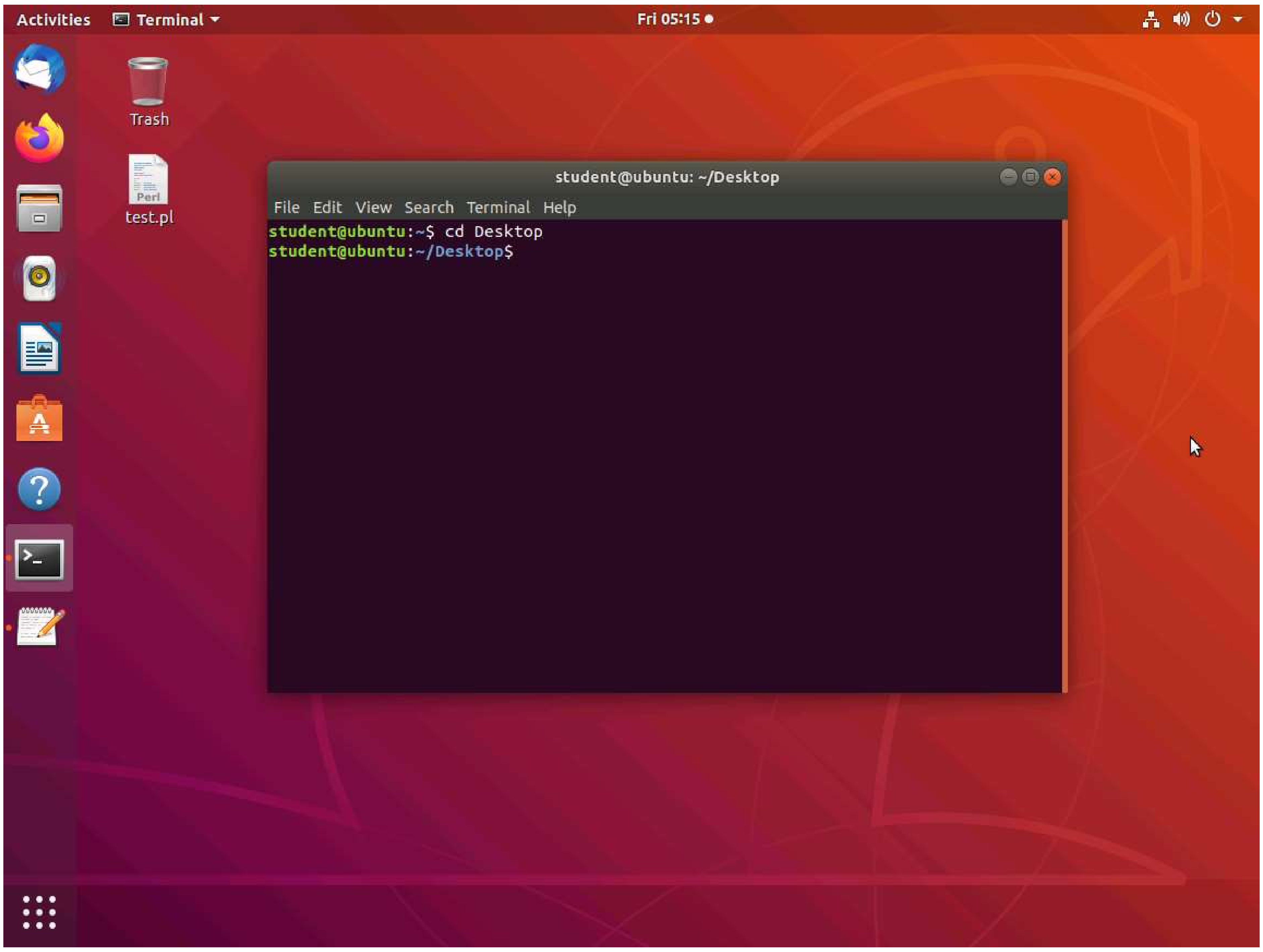


```
#!/usr/bin/perl

# This will print "Hello, World"

print "Hello, World\n";
```

5. Save As window appears, name file as **test.pl** choose the location to save the file (here, **Desktop**) and click **Save**. Minimize the text editor.

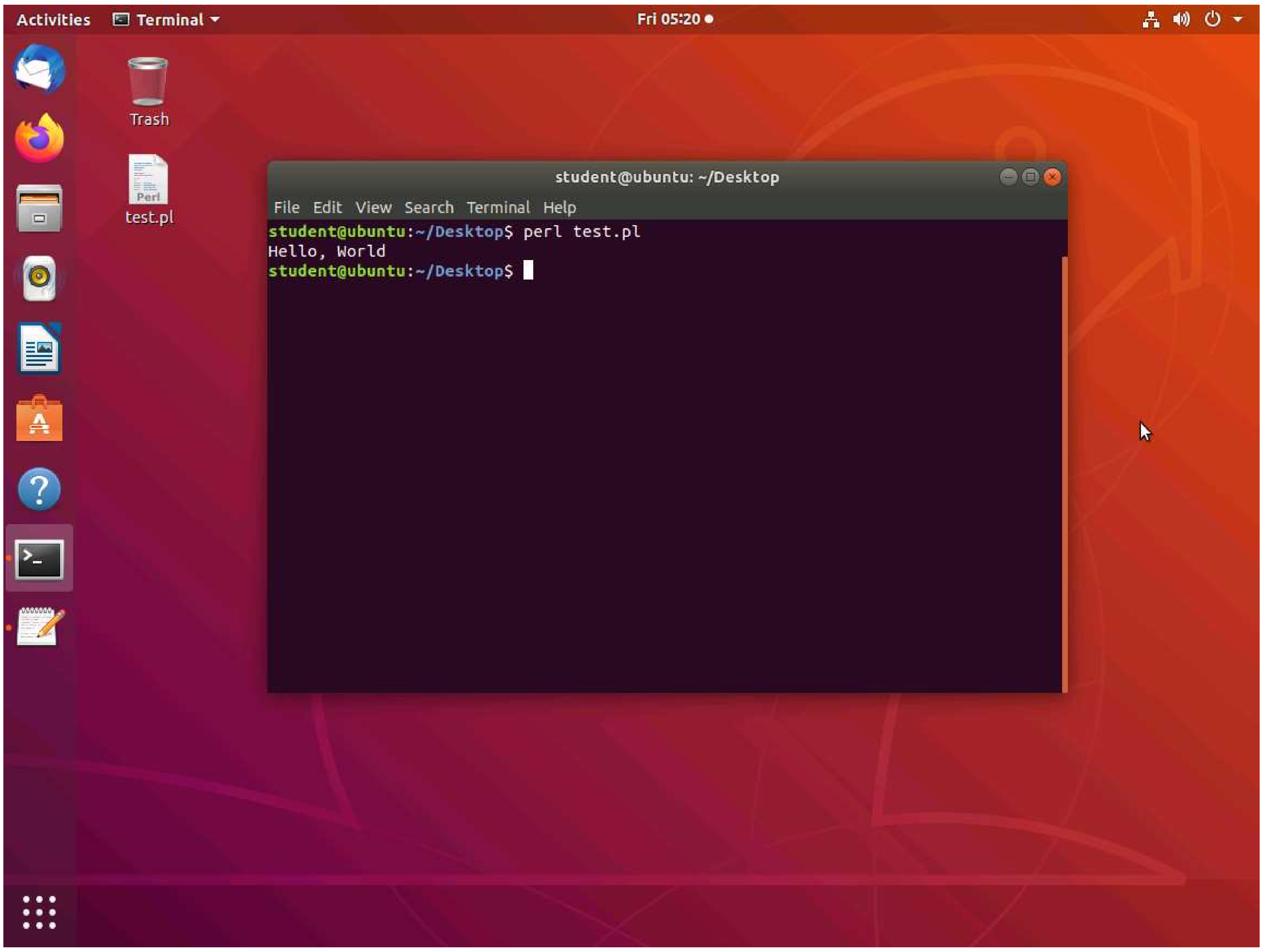


6. Launch a terminal. In the terminal type **cd Desktop** and press **Enter** to change the directory location to Desktop.


Activities
Terminal
Fri 05:15
Trash
test.pl
student@ubuntu: ~/Desktop
File Edit View Search Terminal Help

```
student@ubuntu:~$ cd Desktop
student@ubuntu:~/Desktop$
```

7. Type **perl test.pl** and press **Enter** to execute.

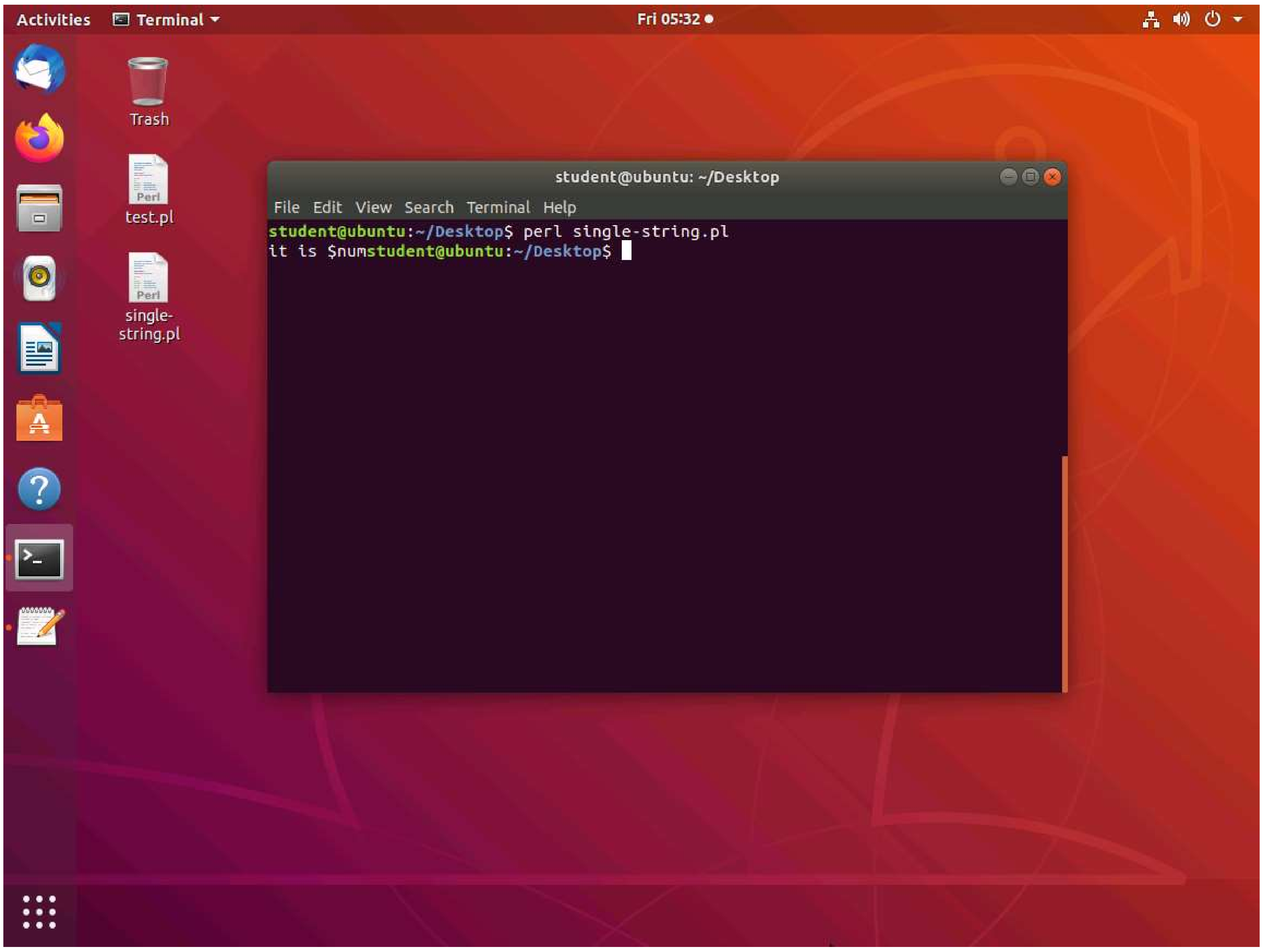

Activities
Terminal
Fri 05:20
Trash
test.pl
student@ubuntu: ~/Desktop
File Edit View Search Terminal Help

```
student@ubuntu:~/Desktop$ perl test.pl
Hello, World
student@ubuntu:~/Desktop$ 
```

8. Explore some of the basics of the **syntax** for Perl. Now, open a new Text Editor and type following code in the text editor. **Save** the file on **Desktop** with the name **single-string.pl**. Minimize the Text Editor.

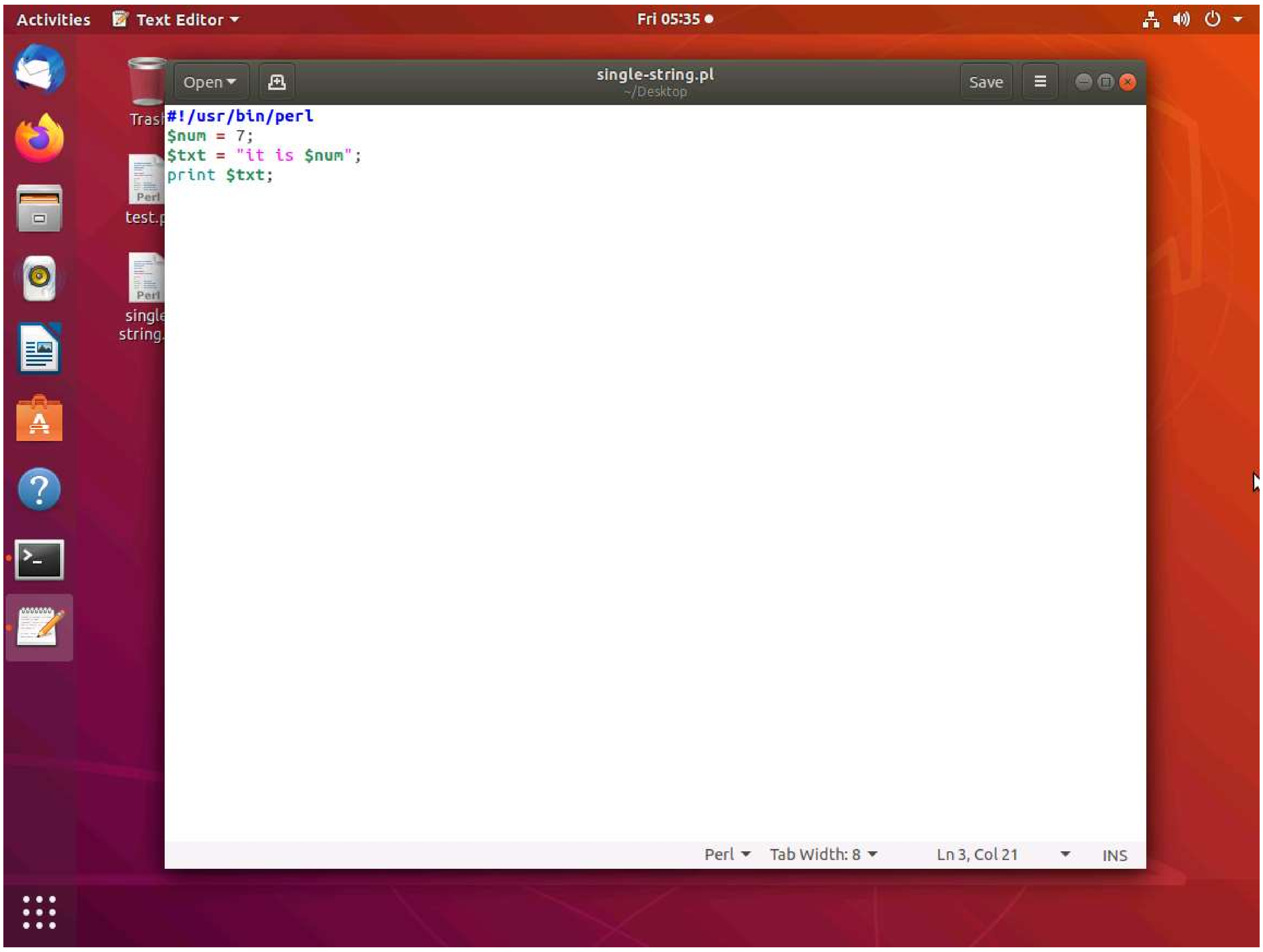
- # !/usr/bin/perl
- $num = 7;
- $txt = 'it is $num';
- print $txt;

9. In the terminal, type **perl single-string.pl** and press **Enter**. Because of single quotes, the value of **$num** is not taken, and the literal characters “$,” “n,” “u,” and “m” are added to the value of $txt.


Activities
Terminal
Fri 05:32
Trash
test.pl
single-string.pl
student@ubuntu: ~/Desktop
File Edit View Search Terminal Help

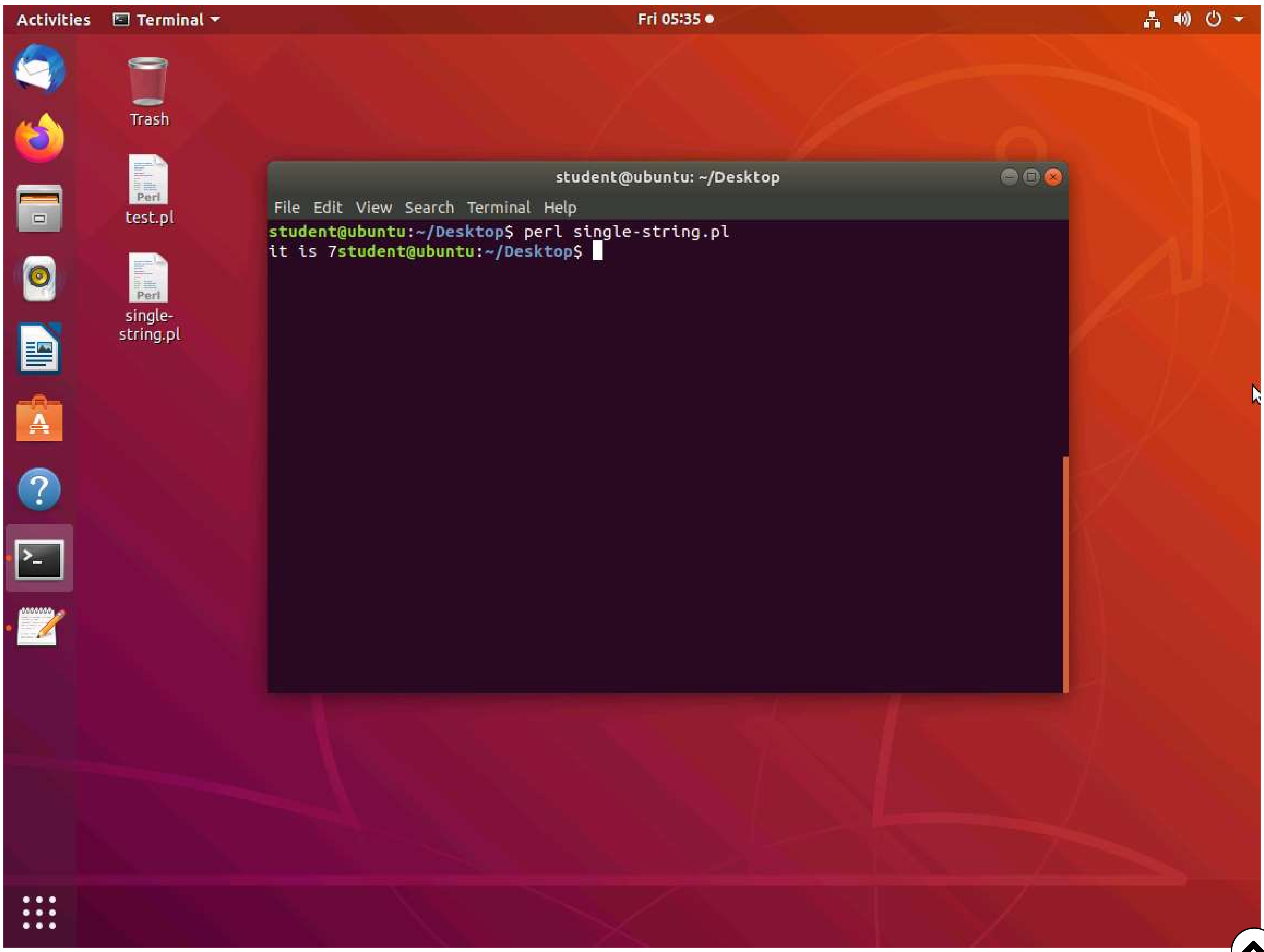
```
student@ubuntu:~/Desktop$ perl single-string.pl
it is $numstudent@ubuntu:~/Desktop$
```

10. Next, consider the double quote result. Maximize the Text Editor and enter the same code, but use the **double quote** as shown in the screenshot. **Save** the file.



```
#!/usr/bin/perl
$num = 7;
$txt = "it is $num";
print $txt;
```

11. Switch to terminal and type **perl single-string.pl** and press **Enter**. The output **7** from the double quotes value of $num is taken and added to the value of **$txt**. Double-quotes **interpolate scalar** and **array variables**, but not hashes. On the other hand, you can use double-quotes to interpolate slices of both arrays and hashes.

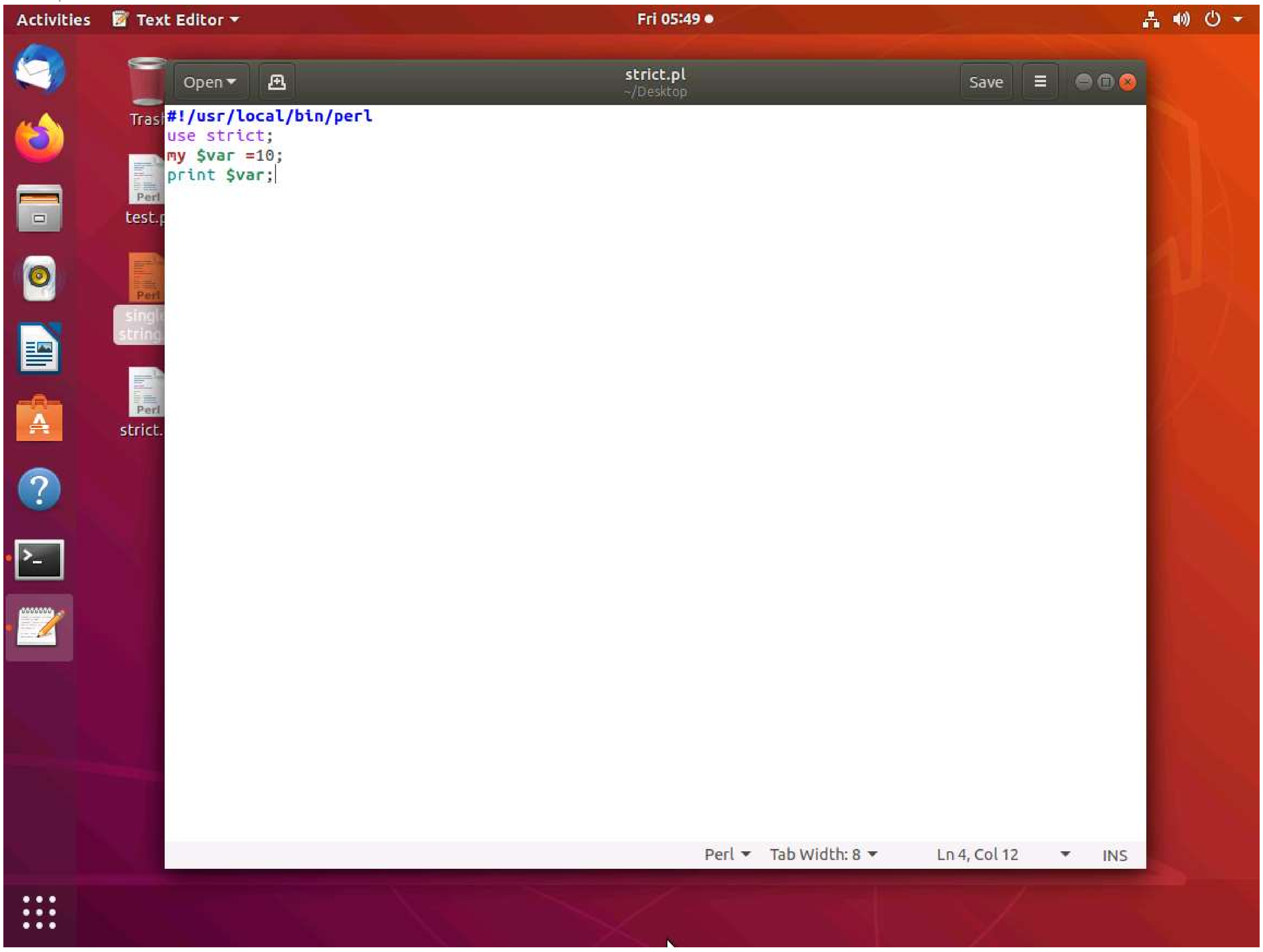


```
student@ubuntu:~/Desktop$ perl single-string.pl
it is 7student@ubuntu:~/Desktop$ 
```

12. The concatenation operator "." unites two or more strings. Remember, if the string contains quotes, carriage returns, and backslashes, all these special characters need to be escaped using a backslash.

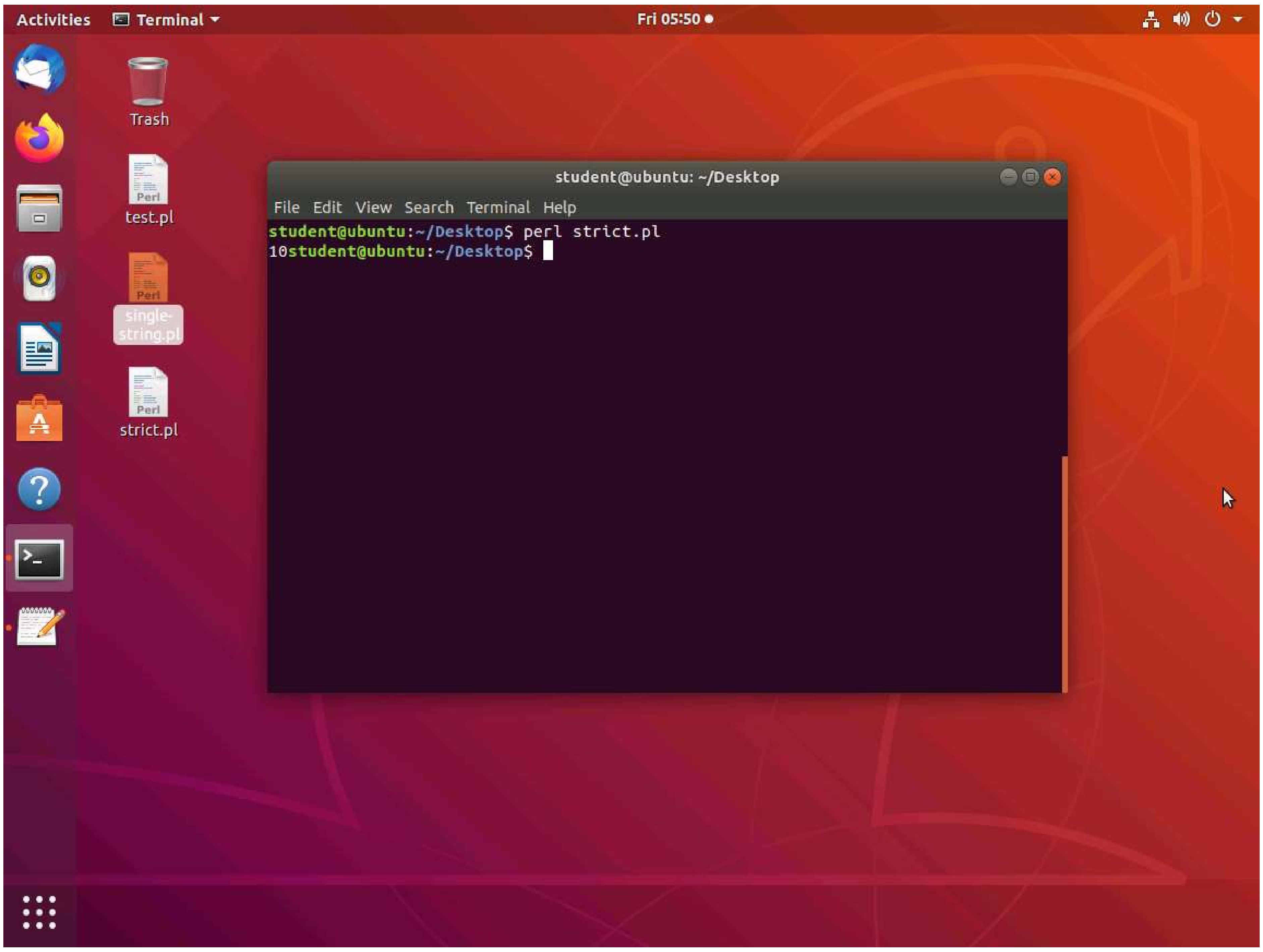
13. Use **"strict;"** to write **better** and **cleaner** code; it turns on strict pragma, which will make you declare the variables with the "my" keyword. This is good programming practice, type the following code in the Text Editor and Save the file on Desktop.

    - # !/usr/local/bin/perl
    - use strict;
    - my $var = 10;
    - print "$var";



```
#!/usr/local/bin/perl
use strict;
my $var =10;
print $var;
```

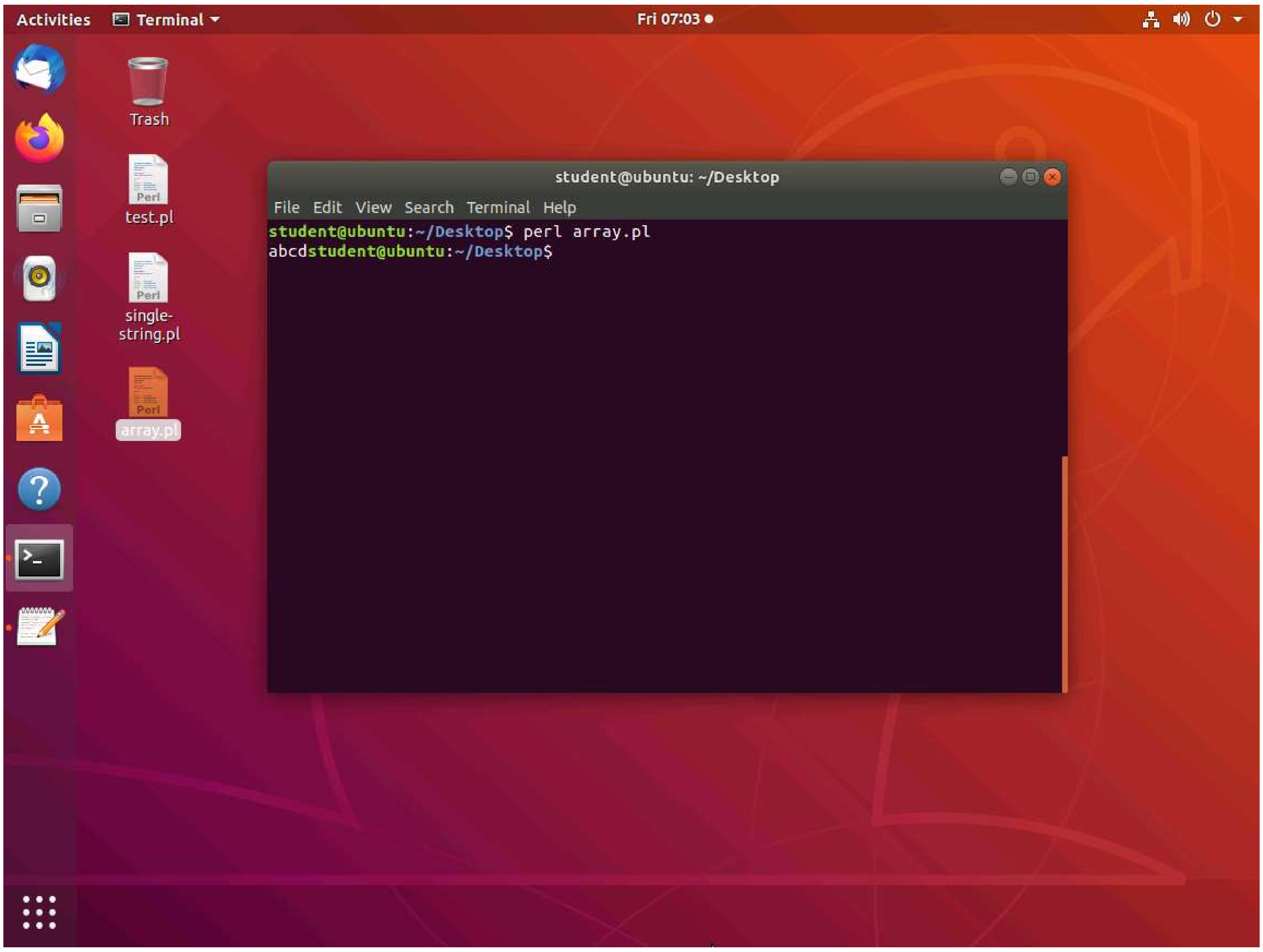
14. Since you have used "**strict;**", the variable has to be declared with the "**my**" keyword. The screenshot shows a scenario when this does not occur. This will save you debugging time in future when you use the strict option.



```
student@ubuntu:~/Desktop$ perl strict.pl
10student@ubuntu:~/Desktop$
```

15. This is how you define an **array** in Perl. Further, there are different ways of storing data in an array. Open a Text editor and type the following code. Save the file with the name array.pl. In the terminal type perl **array.pl** and press **Enter**.

   - # !/usr/local/bin/perl
   - my @array=(a,b,c,d);
   - print @array;



```
student@ubuntu:~/Desktop$ perl array.pl
abcdstudent@ubuntu:~/Desktop$
```

16. One array is already available. If you do not know the size of the array, then use the following options:
    - @array= qw/a b c d e/;
    - print $size=scalar (@array);
17. The push, pop, shift, and unshift for Perl arrays functions can be used in Perl to add/delete to array elements
    - Perl Push: Adds array element at the end of an existing array
    - Perl Pop: Removes the last element from an array
    - Perl Shift: Removes the first element from an array
    - Perl Unshift: Adds an element at the beginning of an array
18. To see how this works, you can enter the following code into a file and save it.
    - @days = ("Mon","Tue","Wed");
    - print "1st : @days\n";
    - push(@days, "Thu"); # adds one element at the end of an array
    - print "2nd when push : @days\n";
    - unshift(@days, "Fri"); # adds one element at the beginning of an array
    - print "3rd when unshift : @days\n";
    - pop(@days);
    - print "4th when pop : @days\n"; # remove one element from the last of an array.
    - shift(@days); # remove one element from the beginning of an array.
    - print "5th when shift : @days\n";
19. Majority of the commands and operators are the same for other languages, but you must review the syntax. An example of file handling is shown below:

    open(FH,"<file.txt");

    my @content=\; # specifying the input of the array is FH.

    foreach(@content)

    {

    ```
      print "$_";
    ```

    }

close FH;

20. Step 19 will help you print the file content on the output screen. Write a program to perform File I/O.
21. The first feature to use is the Tell feature; enter the following code, and then run it to see the results:

    open(FH, "test.pl");

    while(\)

    {

    ```
        $a=tell FH;
    print "$a";
    ```

    }
22. Next, always perform a file seek. You might be familiar with the fseek program that has a similar function. You can execute similarly in Perl. Enter the following:

    open FH, '+<','input.txt';

    seek FH, 5, 0; # This will start reading data after 5 bytes.

    $/ = undef;

    $out = \;

    print $out;

    close FH;
23. This will seek the file 5 bytes before it starts. If the input file has the string “Hello” in front of it, only the characters after the first five ones will be printed.
24. To delete a file in Perl, use the Unlink command as follows:

    unlink("filename or complete file path");
25. We can also work with Directories, and this is shown in the next sequence of code:

    opendir(DIR,"/etc/var");

    @content=readdir(DIR);

    foreach(@content)

    {

    ```
        print "$_\n";
    ```

    }

    closedir(DIR);
26. This code will print all files in the directory specified.
27. You have thus covered the basics of the command. This concludes the lab exercise.

Exercise 2: Perl Socket Handling

Objectives

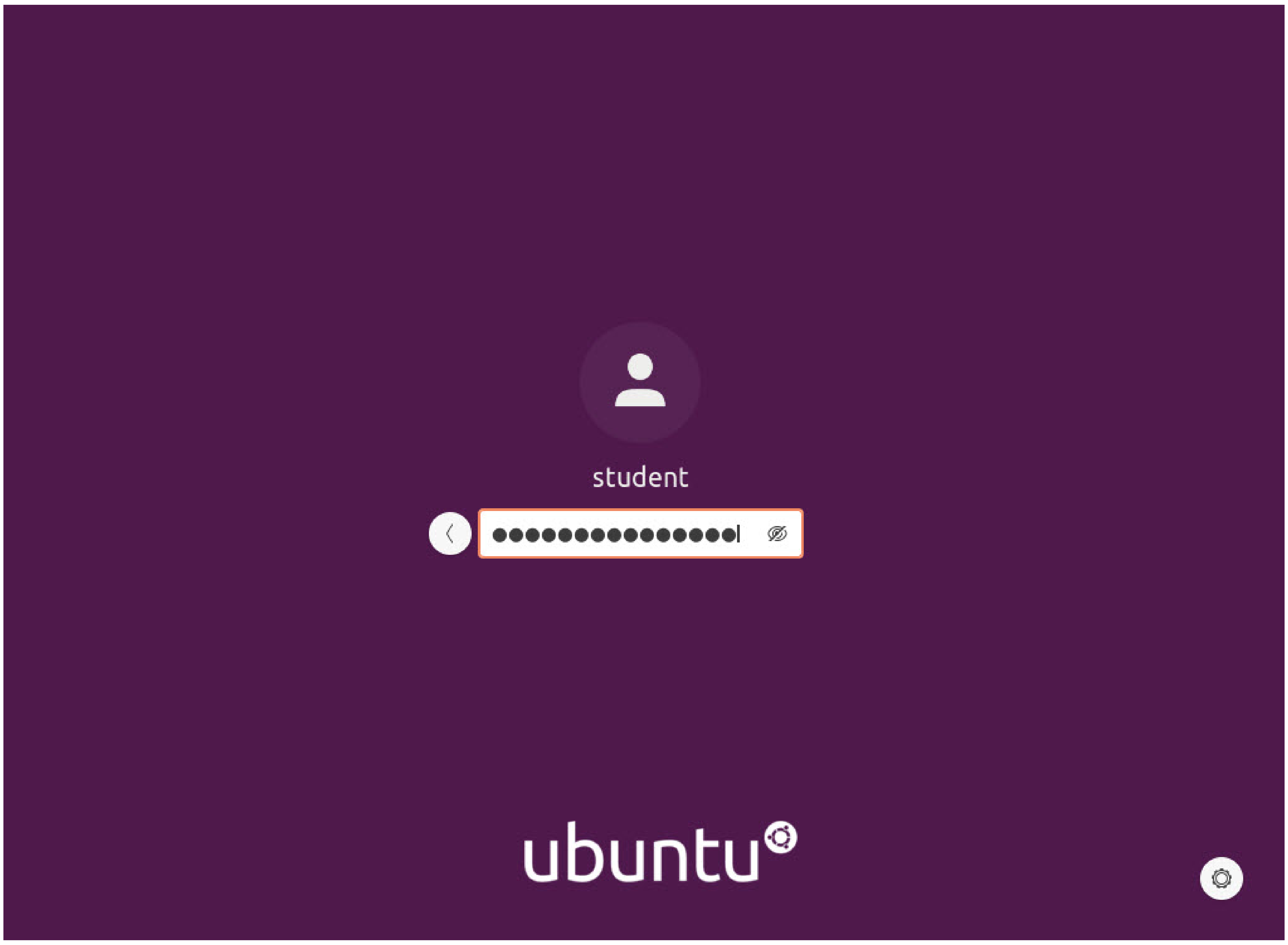
- Create and manipulate sockets using Perl
- Create custom Perl scripts

Lab Duration: **20** Minutes

1. As we have shown throughout, the manipulation of sockets is essential to master as a penetration tester.
2. By default **CPENT Ubuntu-Perl** selected, click **student**.

Note: If you are already logged in skip to step **4**.

3. Type **studentpassword** in the Password field and press **Enter** or click **Sign In** to login



4. Perl has a native **application programming interface** (API) through which sockets can be implemented. For ease, there are many **Comprehensive Perl Archive Network** (CPAN) modules that can assist you when you write socket programs.

5. For the server, you need to perform these steps:

    - Create Socket
    - Bind socket with address and port
    - Listen to the socket on that port address
        - Accept the client connections that attempt to connect using the port and IP of the server
    - Perform operations

6. For the client

    - Create Socket
    - Connect to Server using its port address
    - Perform operations

7. To create the server, enter the following code in any text editor of your choice and save the file with the name **server-perl.pl**.

8. In this we are saving the file on **Desktop**.

```
use IO::Socket;

use strict;

use warnings;

my $socket = new IO::Socket::INET (

LocalHost => 'localhost',

LocalPort => '12345',

Proto => 'tcp',

Listen => 1,
```

Reuse => 1,

);

die "Could not create socket: $!n" unless $socket;

print "Waiting for the client to send datan";

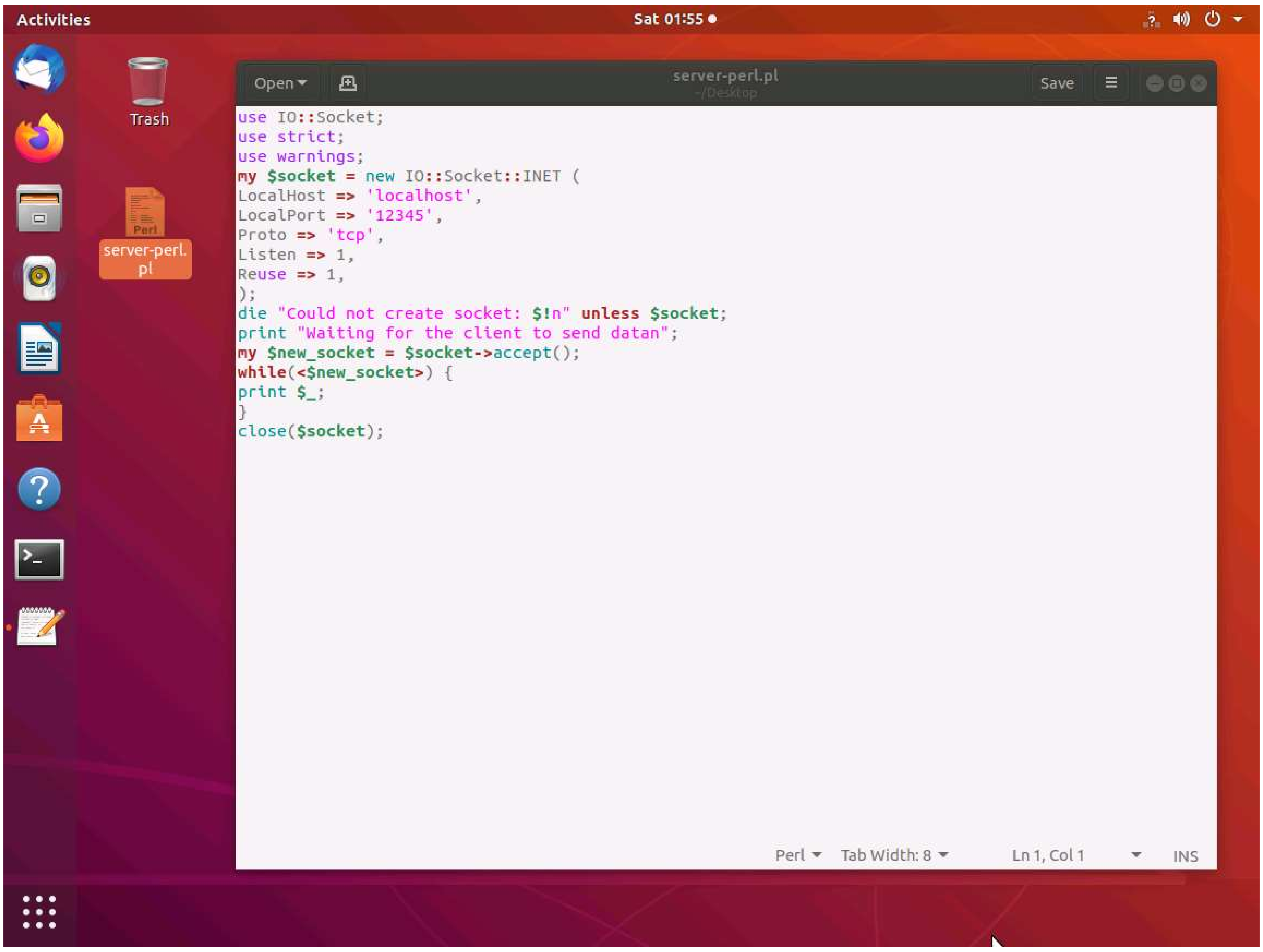
my $new_socket = $socket->accept();

while(<$new_socket>) {

print $_;

}

close($socket);



```
use IO::Socket;
use strict;
use warnings;
my $socket = new IO::Socket::INET (
LocalHost => 'localhost',
LocalPort => '12345',
Proto => 'tcp',
Listen => 1,
Reuse => 1,
);
die "Could not create socket: $!n" unless $socket;
print "Waiting for the client to send datan";
my $new_socket = $socket->accept();
while(<$new_socket>) {
print $_;
}
close($socket);
```

9. Next, enter the following code to create the client

use strict;

use warnings;

use IO::Socket;

my $socket = new IO::Socket::INET (

PeerAddr => 'localhost',

PeerPort => '12345',

Proto => 'tcp',

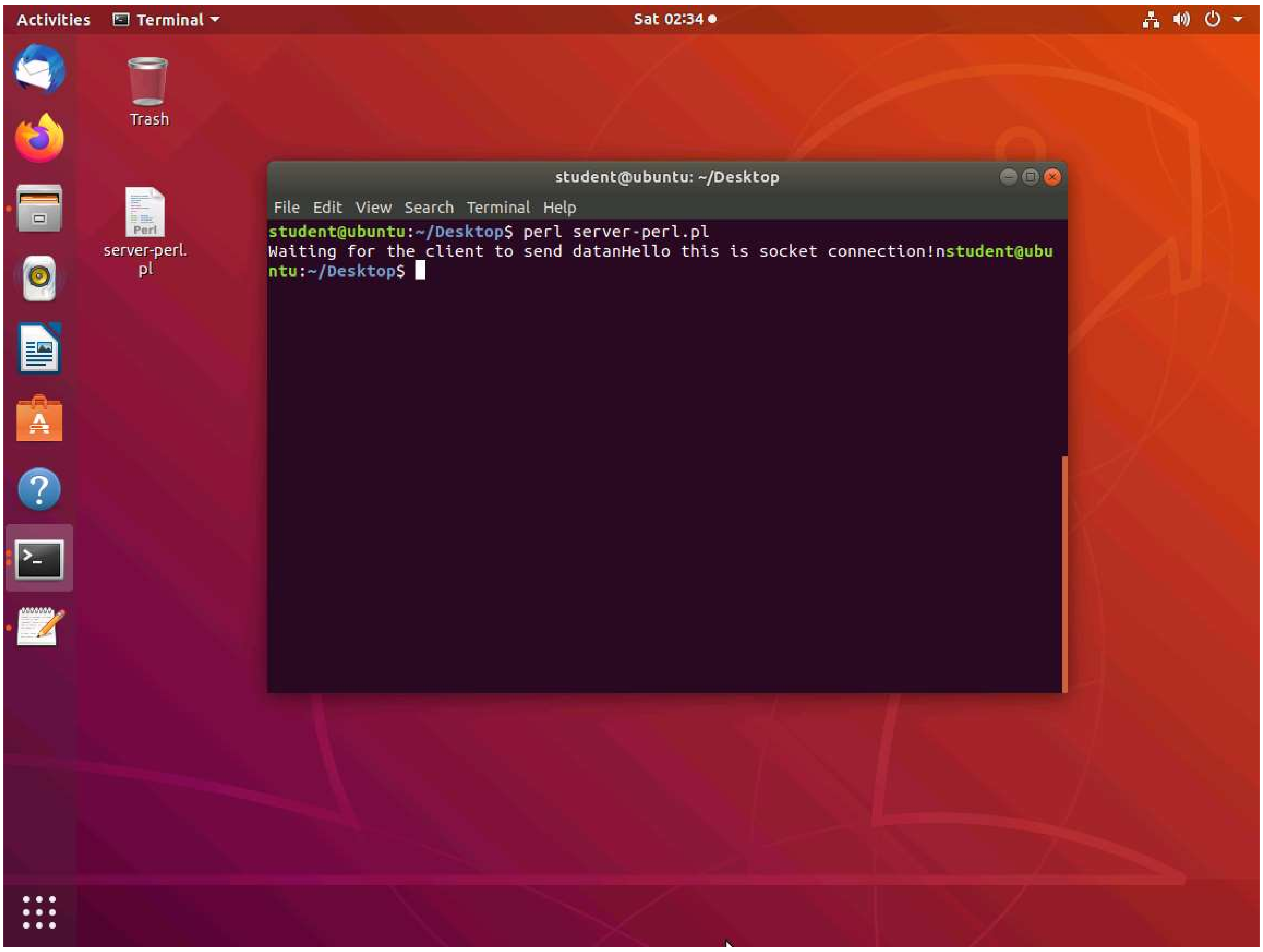
);

die "Could not create socket: $!n" unless $socket;

print $socket "Hello this is socket connection!n";

close($socket);

10. Run both in their own windows to connect, as shown in the screenshot. You have so far created the sockets and connections, and then sent the message, creating a chat program.


Activities
Terminal
Sat 02:34
Trash
server-perl.pl
student@ubuntu: ~/Desktop
File Edit View Search Terminal Help

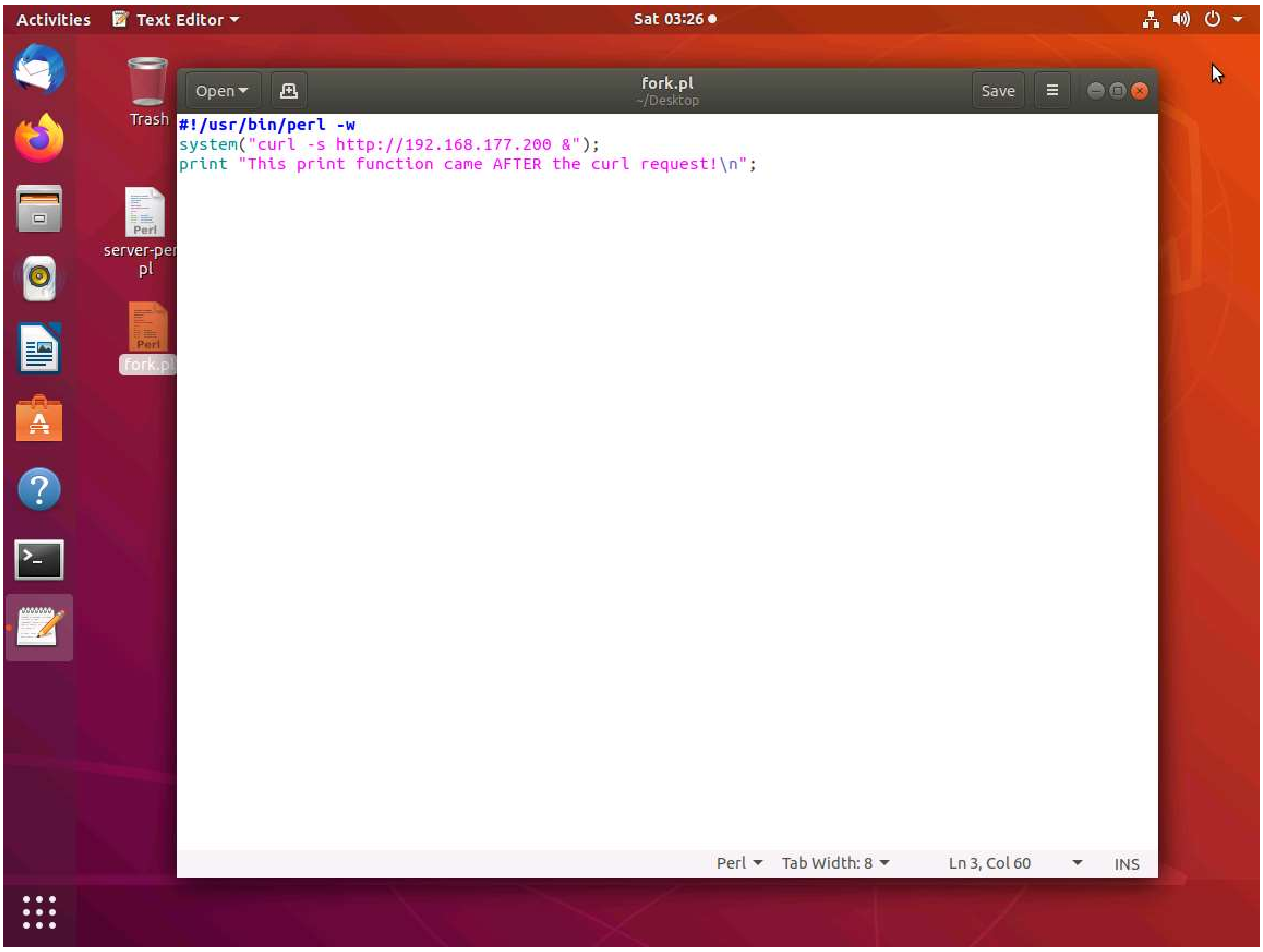
```
student@ubuntu:~/Desktop$ perl server-perl.pl
Waiting for the client to send datanHello this is socket connection!nstudent@ubu
ntu:~/Desktop$
```

11. Programming in Perl does not cause portability issues, which is common when using different shells in shell scripting. Error handling is easy in Perl.

12. You can write long and complex programs within Perl easily because of its vastness. This is in contrast with Shell, which does not support namespaces, modules, object, or inheritance.

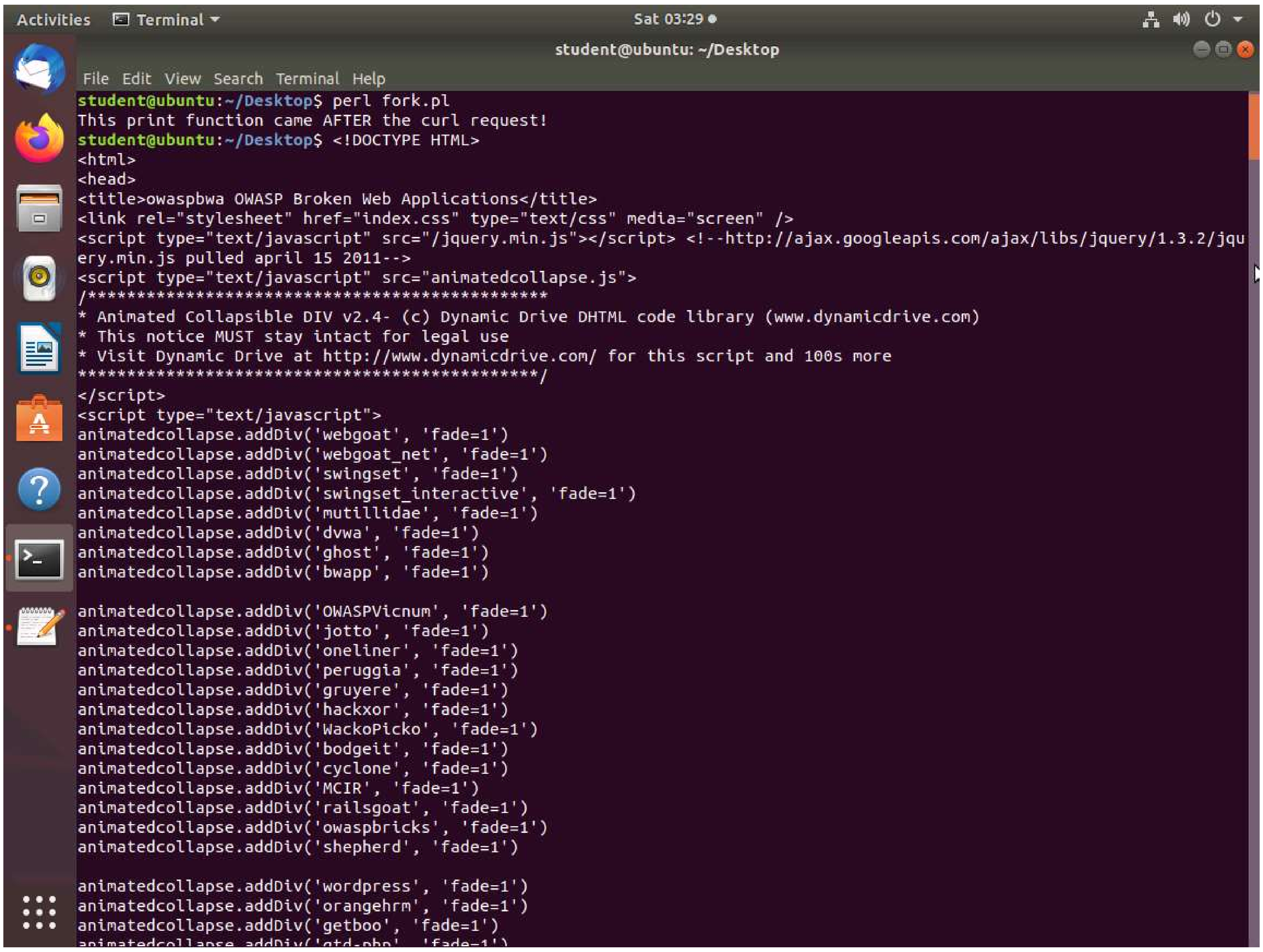
13. Shell has fewer reusable libraries available unlike Perl's CPAN. Shell is less secure. It calls external functions (commands such as mv and cp depend on the shell being used). Perl does most of its useful work while using internal functions.

14. Now, consider the **forking** of a process; enter the following code, as shown in the screenshot in the Text Editor and save the file with the name **fork.pl** on Desktop.



```
#!/usr/bin/perl -w
system("curl -s http://192.168.177.200 &");
print "This print function came AFTER the curl request!\n";
```

15. Then, run the code as shown in the screenshot.


Activities
Terminal
Sat 03:29
student@ubuntu: ~/Desktop
File Edit View Search Terminal Help

```
student@ubuntu:~/Desktop$ perl fork.pl
This print function came AFTER the curl request!
student@ubuntu:~/Desktop$ <!DOCTYPE HTML>
<html>
<head>
<title>owaspbwa OWASP Broken Web Applications</title>
<link rel="stylesheet" href="index.css" type="text/css" media="screen" />
<script type="text/javascript" src="/jquery.min.js"></script> <!--http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jqu
ery.min.js pulled april 15 2011-->
<script type="text/javascript" src="animatedcollapse.js">
/***********************************************
* Animated Collapsible DIV v2.4- (c) Dynamic Drive DHTML code library (www.dynamicdrive.com)
* This notice MUST stay intact for legal use
* Visit Dynamic Drive at http://www.dynamicdrive.com/ for this script and 100s more
***********************************************/
</script>
<script type="text/javascript">
animatedcollapse.addDiv('webgoat', 'fade=1')
animatedcollapse.addDiv('webgoat_net', 'fade=1')
animatedcollapse.addDiv('swingset', 'fade=1')
animatedcollapse.addDiv('swingset_interactive', 'fade=1')
animatedcollapse.addDiv('mutillidae', 'fade=1')
animatedcollapse.addDiv('dvwa', 'fade=1')
animatedcollapse.addDiv('ghost', 'fade=1')
animatedcollapse.addDiv('bwapp', 'fade=1')

animatedcollapse.addDiv('OWASPVicnum', 'fade=1')
animatedcollapse.addDiv('jotto', 'fade=1')
animatedcollapse.addDiv('oneliner', 'fade=1')
animatedcollapse.addDiv('peruggia', 'fade=1')
animatedcollapse.addDiv('gruyere', 'fade=1')
animatedcollapse.addDiv('hackxor', 'fade=1')
animatedcollapse.addDiv('WackoPicko', 'fade=1')
animatedcollapse.addDiv('bodgeit', 'fade=1')
animatedcollapse.addDiv('cyclone', 'fade=1')
animatedcollapse.addDiv('MCIR', 'fade=1')
animatedcollapse.addDiv('railsgoat', 'fade=1')
animatedcollapse.addDiv('owaspbricks', 'fade=1')
animatedcollapse.addDiv('shepherd', 'fade=1')

animatedcollapse.addDiv('wordpress', 'fade=1')
animatedcollapse.addDiv('orangehrm', 'fade=1')
animatedcollapse.addDiv('getboo', 'fade=1')
```