Practical 11 – Linux Administration – Environment Variables and Scripts

Objectives

- Exercise 1 - Environment Variables
- Exercise 2 - Pipelines and Filters
- Exercise 3 - Input and Output Redirection
- Exercise 4 - Shell Variable
- Exercise 5 - Compound Commands
- Exercise 6 - Creating and Running Scripts

Exercise 1 - Environment Variables

1. Environment variables are variables containing values of system properties.

   For example, the *bash* shell keeps a list of directories to look for commands in the environment variable called "PATH".

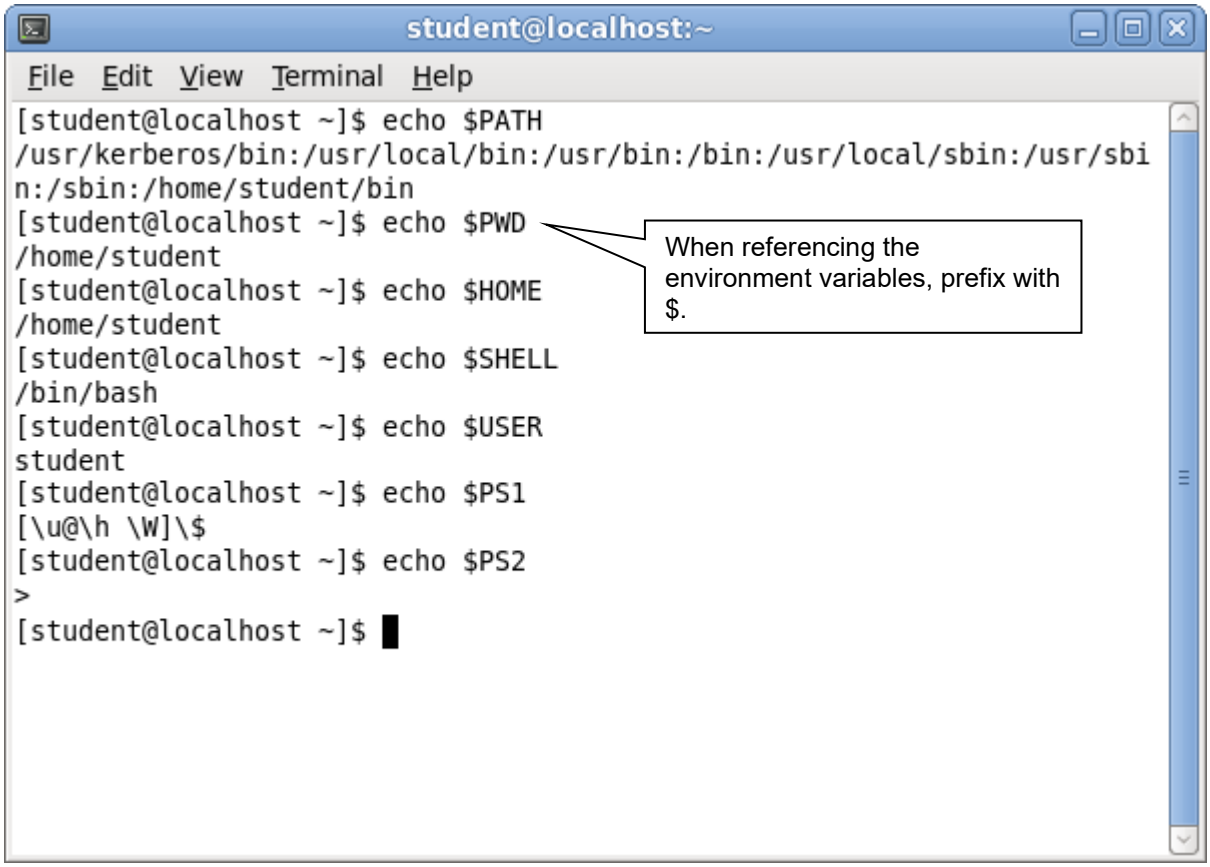
   Unlike shell script variables, environment variables are available to all instances of the shell.

   Note
   Run the **printenv** or the **env** commands to display a list of environment variables.

   These are the important environment variables which you need to know.

Environment Variable	Description
HOME	The absolute path of the home directory
PATH	The value of the command search path.
PS1	The value of the primary prompt.
PS2	The value of the secondary prompt (used in **while** and **for** commands).
PWD	The absolute path of the current work directory.
SHELL	The absolute path of the login shell.
USER	The user name of the current logged in user.

   Run the **echo** command to display the value of the environment variables as shown.


student@localhost:~
File Edit View Terminal Help
[student@localhost ~]$ echo $PATH
/usr/kerberos/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbi
n:/sbin:/home/student/bin
[student@localhost ~]$ echo $PWD
/home/student
[student@localhost ~]$ echo $HOME
/home/student
[student@localhost ~]$ echo $SHELL
/bin/bash
[student@localhost ~]$ echo $USER
student
[student@localhost ~]$ echo $PS1
[\u@\h \W]\$
[student@localhost ~]$ echo $PS2
>
[student@localhost ~]$
When referencing the environment variables, prefix with $.

Exercise 2 - Pipelines and Filters

1. The standard output of one command may be connected to become the standard input of another command by using a pipe (|).

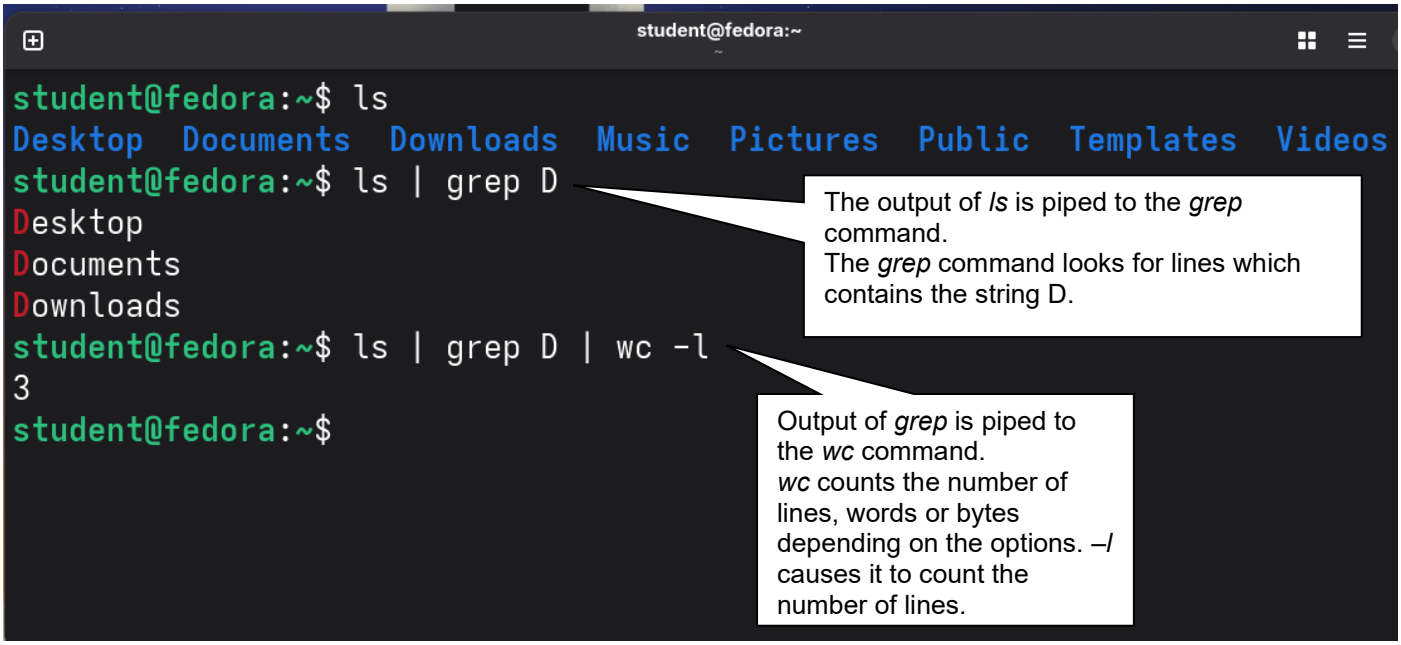
   The shell uses this mechanism to create a more sophisticated program by connecting a number of small single purpose programs together.

   For example, "**ls** | **wc**". These two commands connected in this way constitute a pipeline.

> **Note**
> Two processes connected by a pipe run in parallel. Pipes are unidirectional and synchronization is achieved by halting producer when the pipe is full and halting the consumer when there is nothing to read. In the example, the producer is **ls** and the consumer is **wc**.

> **Note**
> A pipeline may consist of more than two commands.
> For example, "**ls** | **grep** txt | **wc** -l".
> This command counts the number of files in the current directory with file names containing the string "txt".

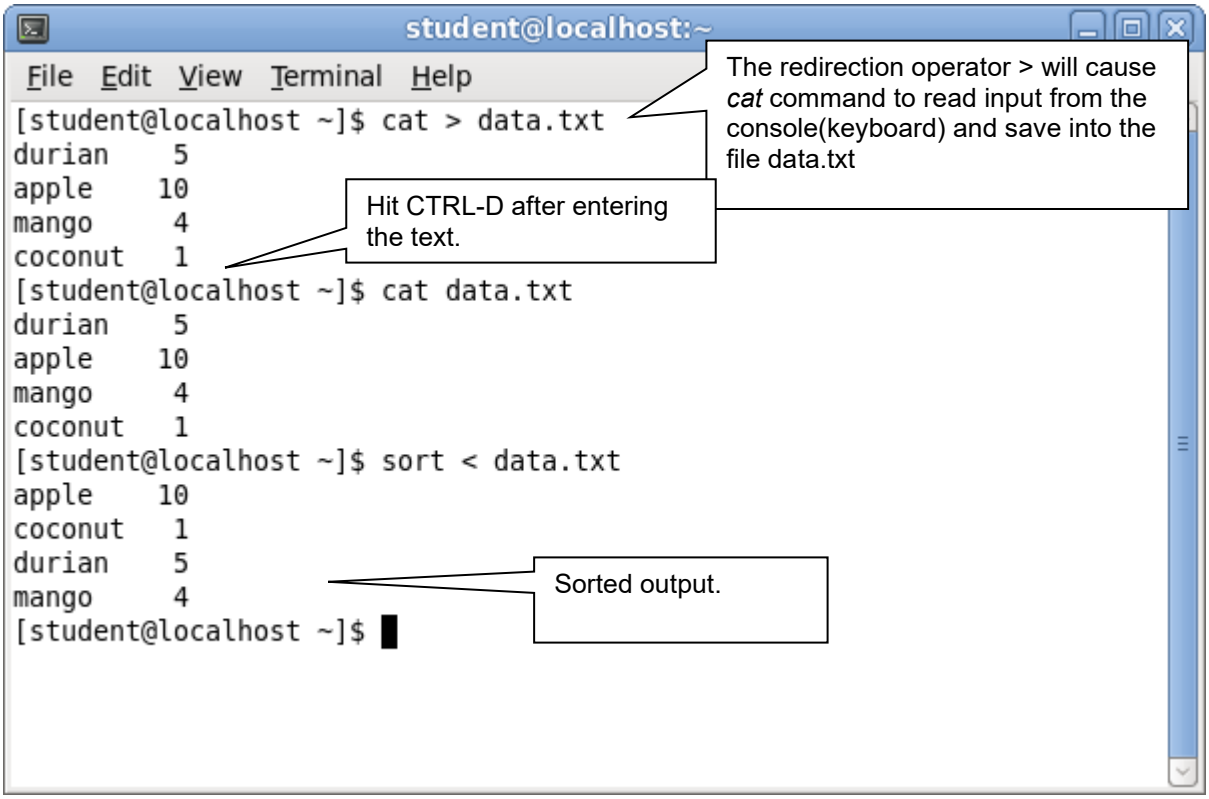
2. A filter is a command that transforms the inputs in some way.



   The **grep** command is a filter which selects and prints lines matching a pattern.

Exercise 3 – Input and Output Re-direction

The **sort** command is another filter which sort lines of text.
Let's first create a data.txt with 4 lines and then perform sort



.

Linux provides three input output channels when a program is executed.

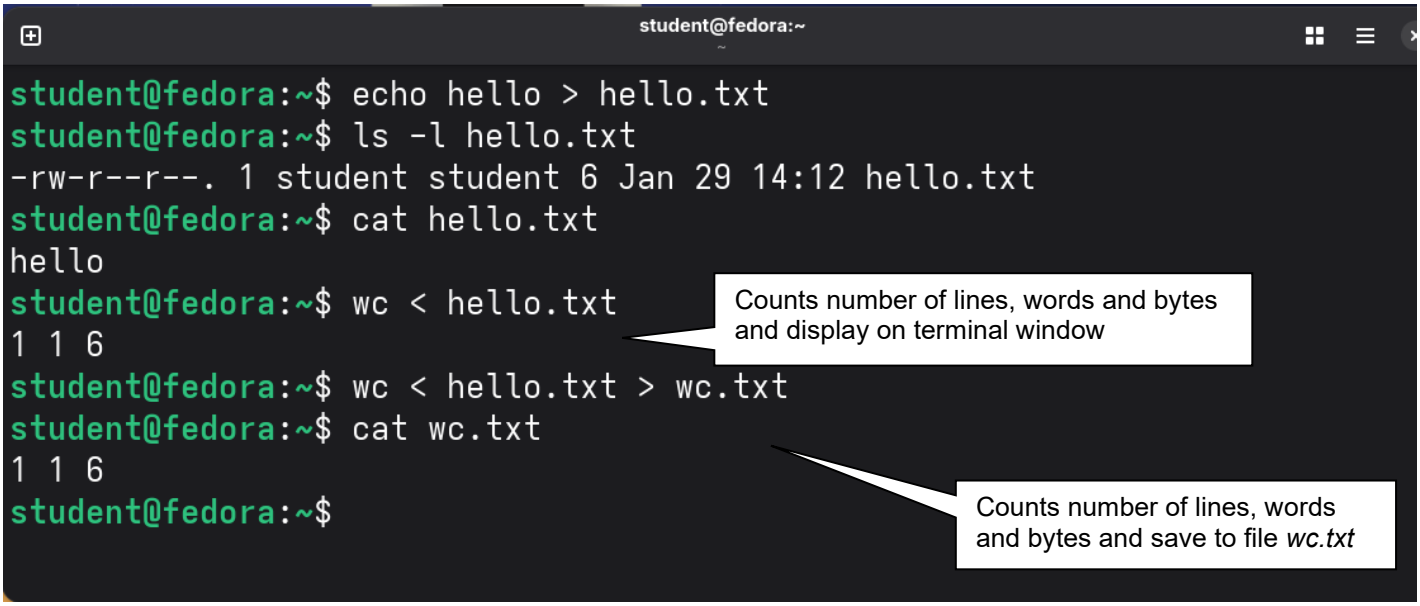
File Descriptor (FD)	Name	Description
0	STDIN (standard input)	Defaults to the keyboard.
1	STDOUT (standard output)	Defaults to the terminal window.
2	STDERR (standard error)	Defaults to the terminal window.

Redirection is an important feature which allows program to receive input from any sources and output to any destinations.

Explanation :

Redirection	Example
Sends output to another command	**ls** | **less**
Takes input from a file	**sort** < data.txt
Sends output to a file	**echo** hello > hello.txt **ls** > list.txt
Append output to a file	**echo** world >> hello.txt
Takes input from a file and send output to a file	**sort** < data.txt > sorted.txt
Redirect STDERR to a file	**sort** no_such_file 2> error.txt
Redirect STDOUT and STDERR to different files	**cat** hello.txt 1>data.bak 2>error.txt

Follow the example as shown to redirect from STDIN and to STDOUT in the same command.



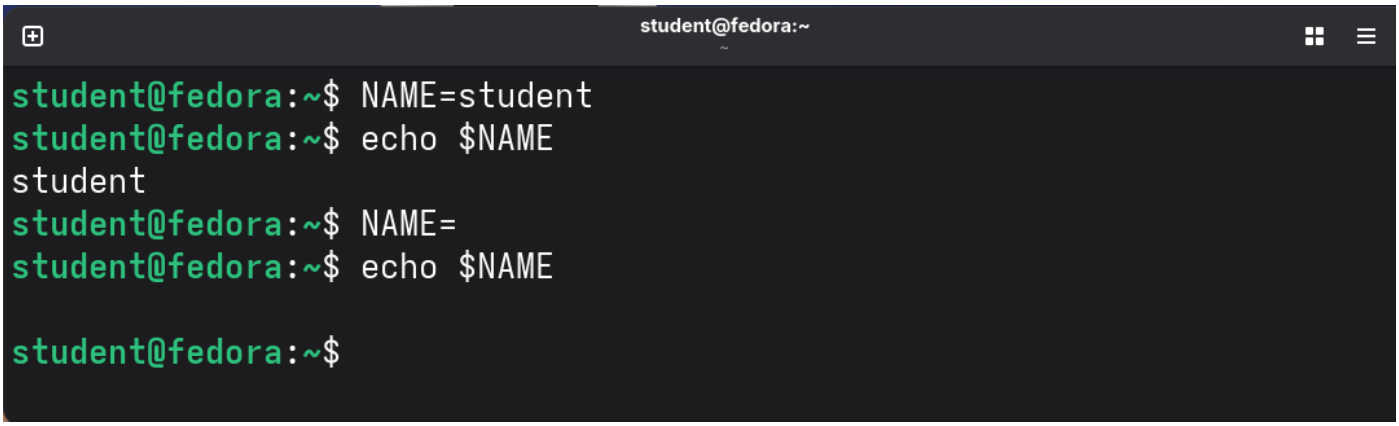
Note
The **wc** word count command counts the number of lines, words and characters in a file.

Exercise 4 - Shell Variables

1. Shell variable names begin with a letter and consist of letters, digits and underscores.

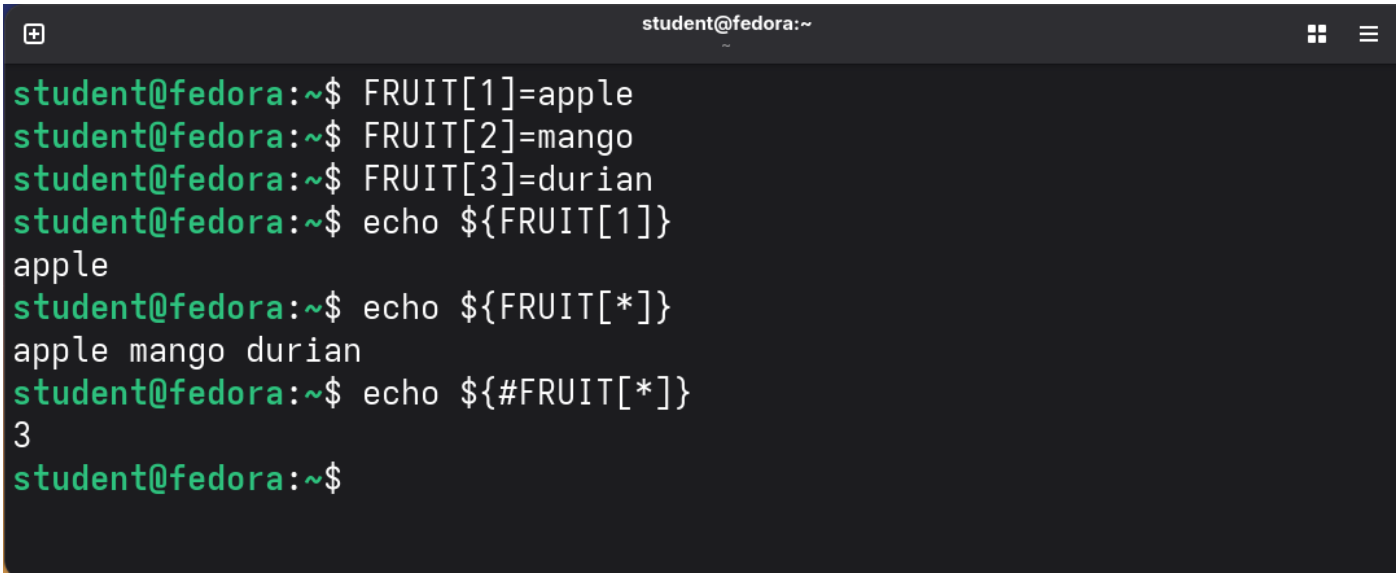
   The value of a variable is substituted in a command statement by preceding its name with dollar sign ($).

   Follow the example as shown to create some variables and display their results.



2. Follow the example as shown to create an array by assigning every array elements individually.

   The array index should be an integer.



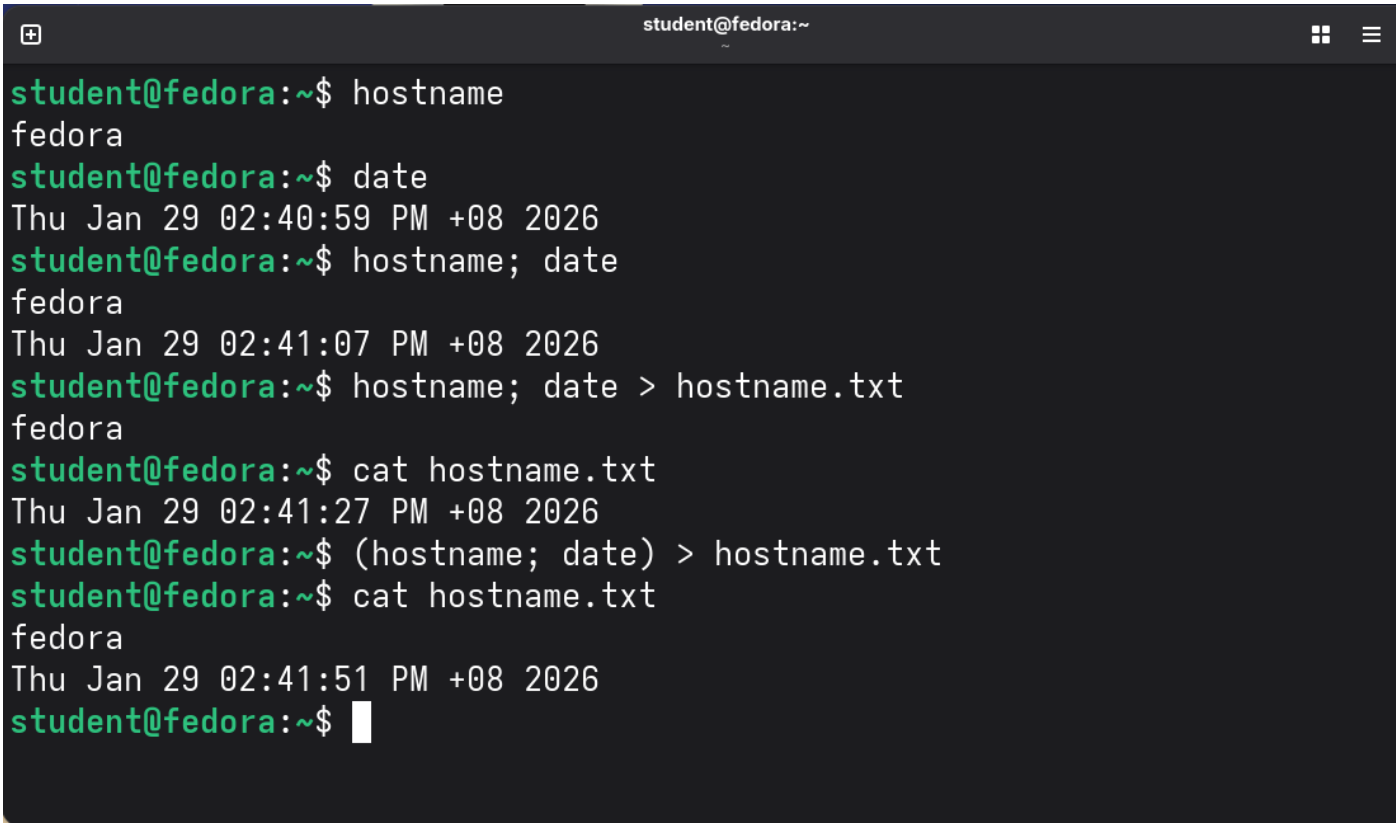
Exercise 5 - Compound Commands

1. Multiple commands can be written in one single line of statement by delimiting the commands with command separators (;)

   Multiple commands can be grouped together by enclosing them with brackets, forming a command group.

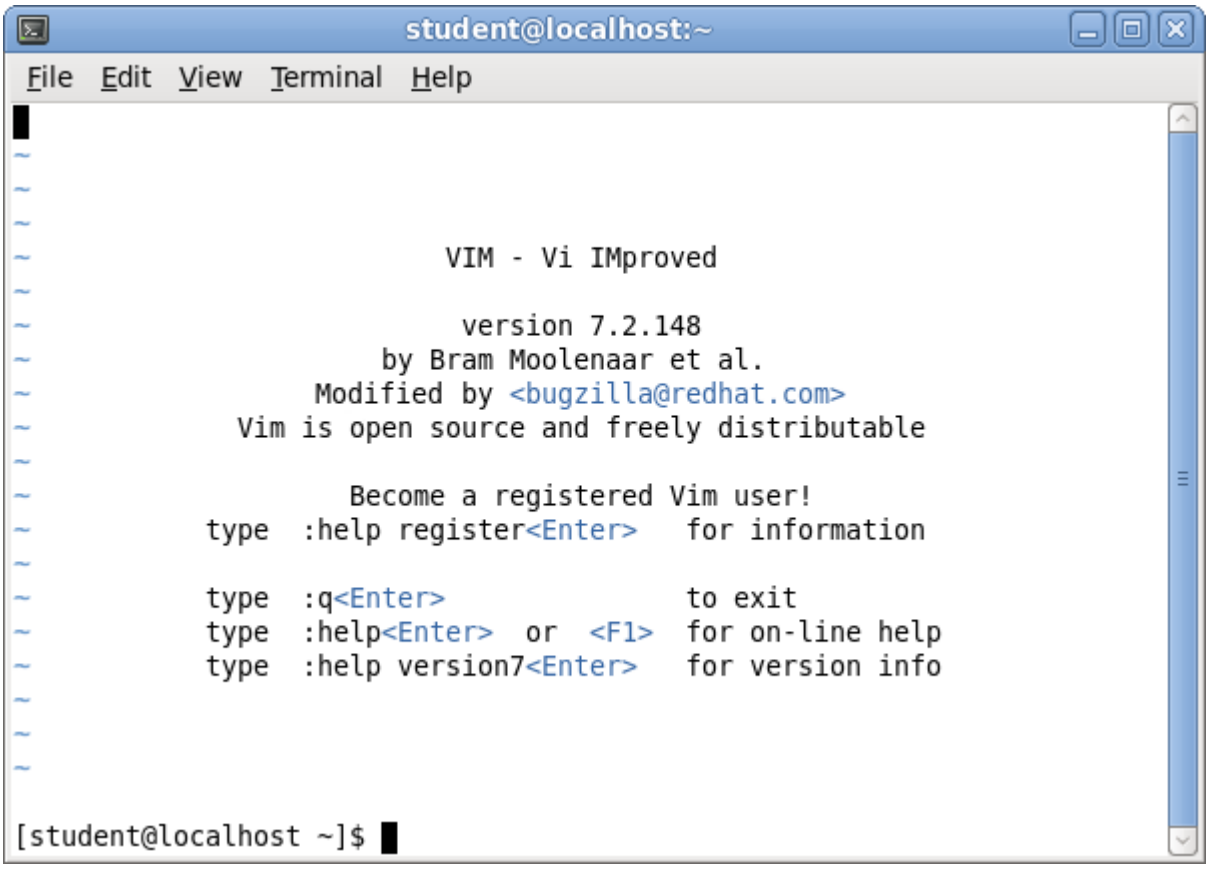
   Follow the example as shown to execute the multiple commands in a single statement.

   Note:
   In a command group, the outputs from all individual commands are redirected.
   Without command group, only the output of the last command is redirected.



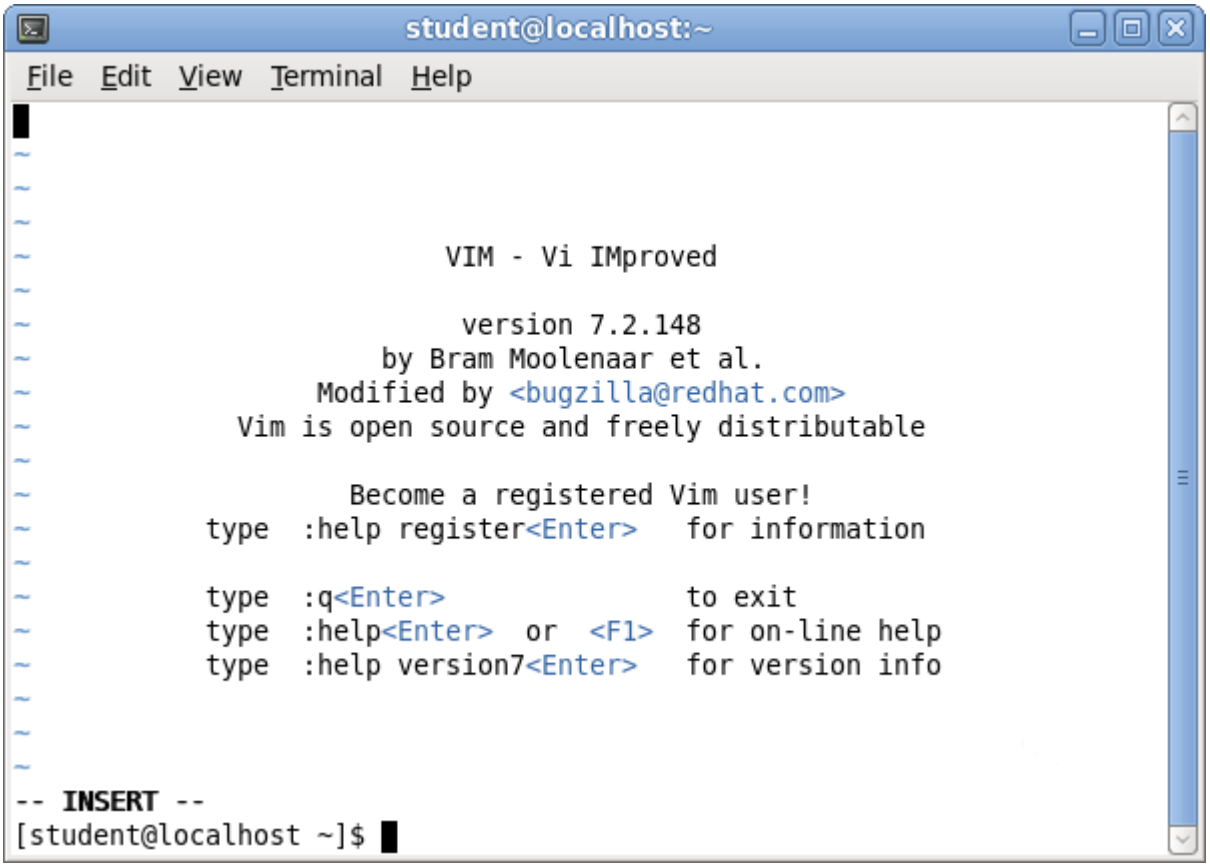
Exercise 6 - Creating and Running Scripts

1. Run **vi** at the command prompt.



Note
The VI editor is a powerful screen-based text editor available in most Unix/Linux systems.

Type the lowercase "i" to get into the "INSERT" mode.



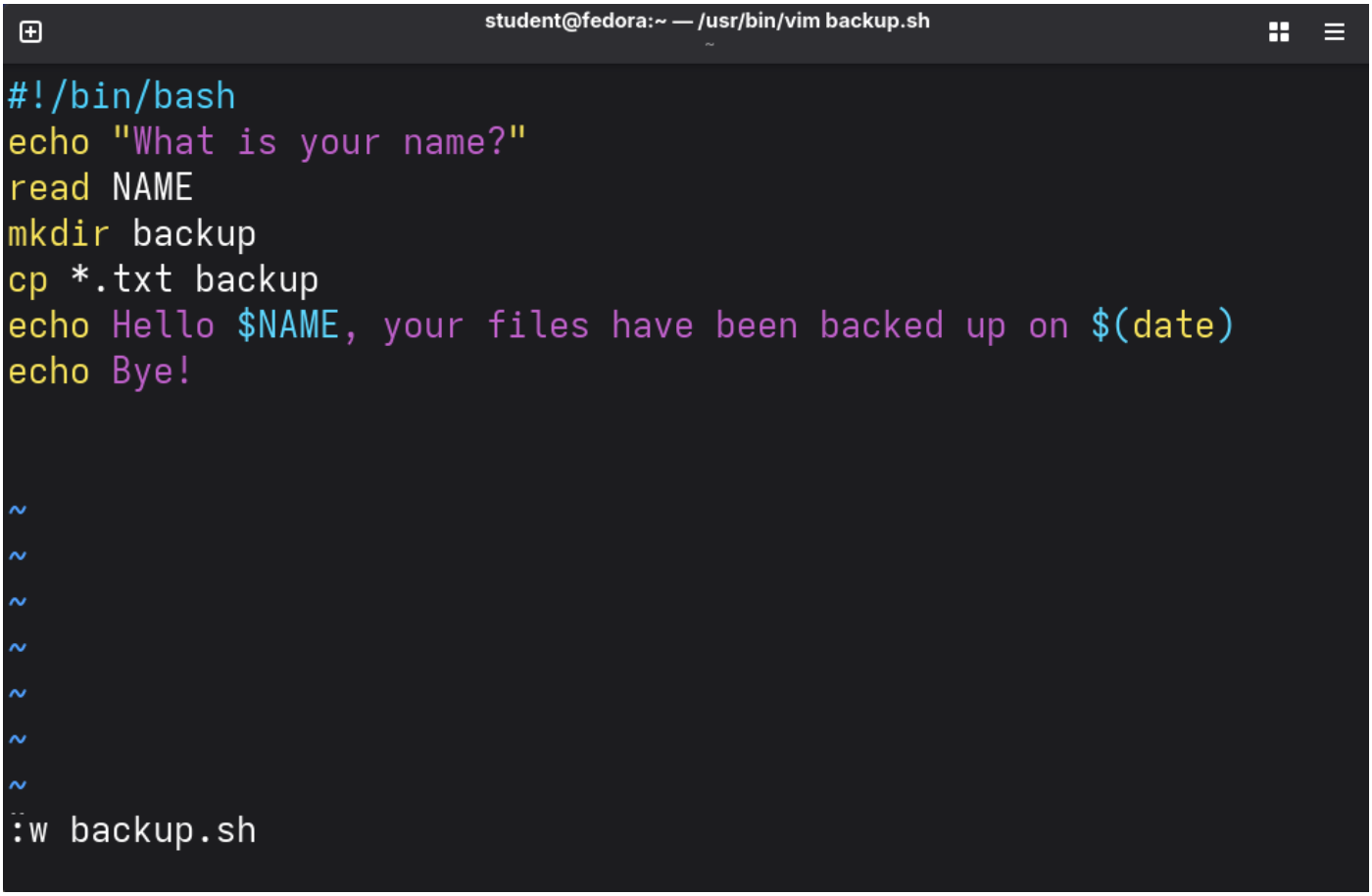
Type the script as shown.

LISTING : backup.sh

```
#!/bin/bash
echo "What is your name?"
read NAME
mkdir backup
cp *.txt backup
echo Hello $NAME, your files have been backed up on $(date)
echo Bye!
```

Note

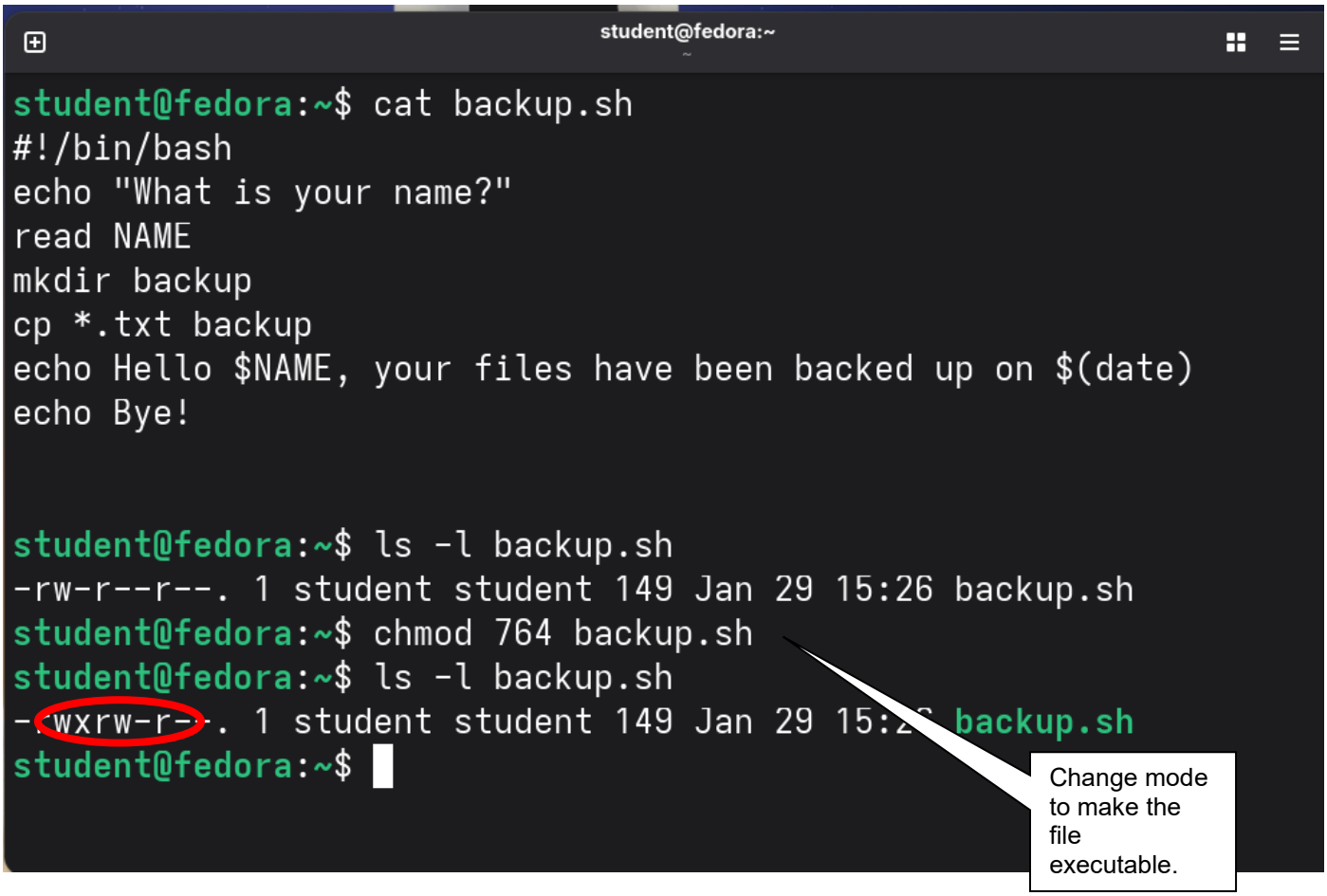
The first line of the script must specify the shell to use for executing the script.
You may run the "**echo** $SHELL" command to display the location of the default shell.
Usually, shell script uses the file extension ".sh".



Press the <ESC> key. Type ":w backup.sh" (write command) and press the <ENTER> key. The file is now saved.

Next, press the <ESC> key. Type ":q" (quit command) and press the <ENTER> key to exit vi editor.

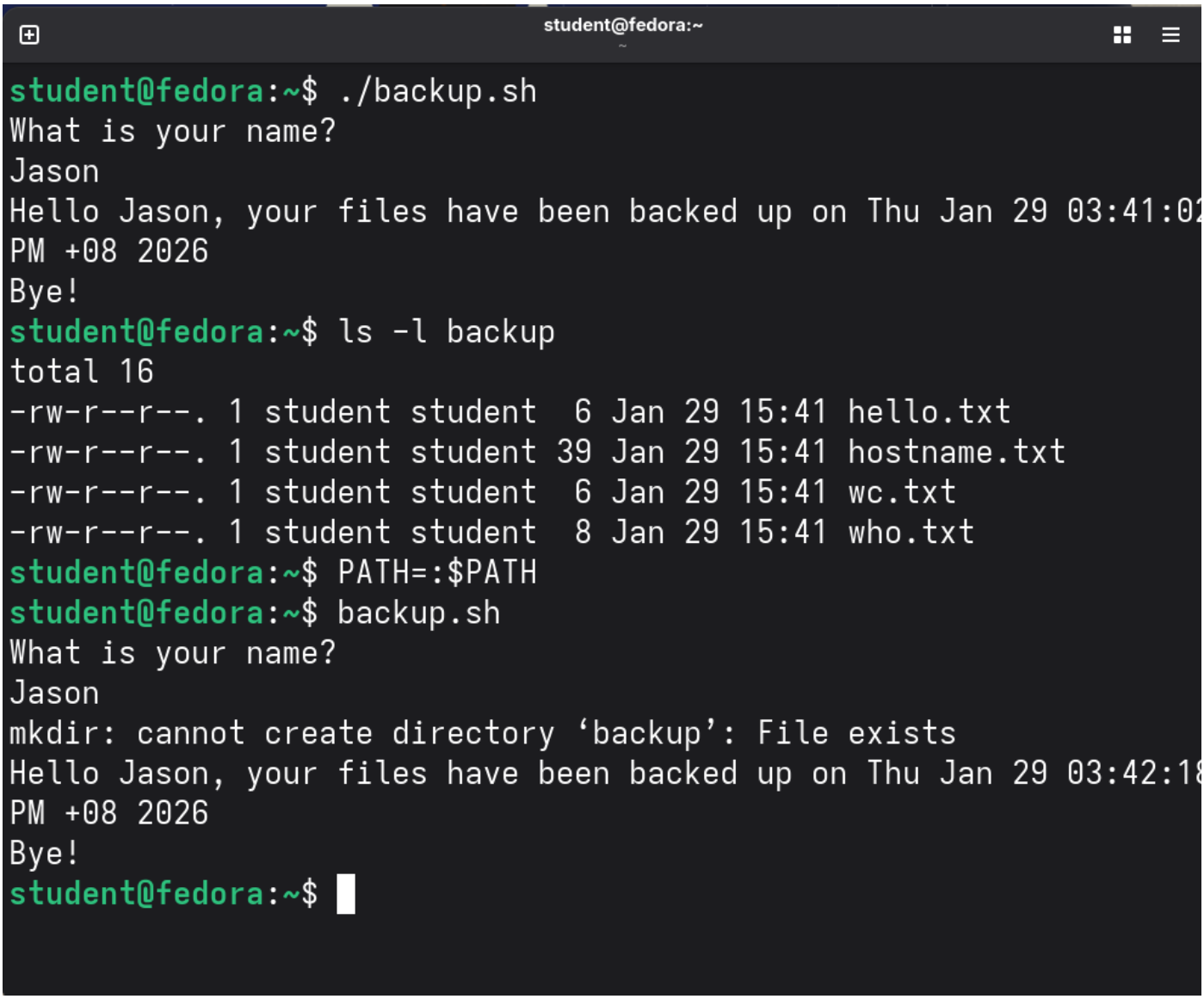
2. Run the "**cat** backup.sh" command to display the content of the file.



Run the "**ls** -l backup.sh" to list the file. Notice that there is no execution permission.
Run the "**chmod** 764 backup.sh" to set the execute permission.
Run the "**ls** -l backup.sh" to check the file permissions.

> **Note**
> A script cannot be executed unless it is given the execution right.

Run your backup script by typing ./backup.sh



Can you execute the script without specifying the path ./ ?

Enter "PATH=:$PATH" at the command prompt and press <Enter> to include the current directory in the command search path. An empty entry (":" or "::") in PATH means the current directory. Run the "backup.sh" script again.