

## 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.

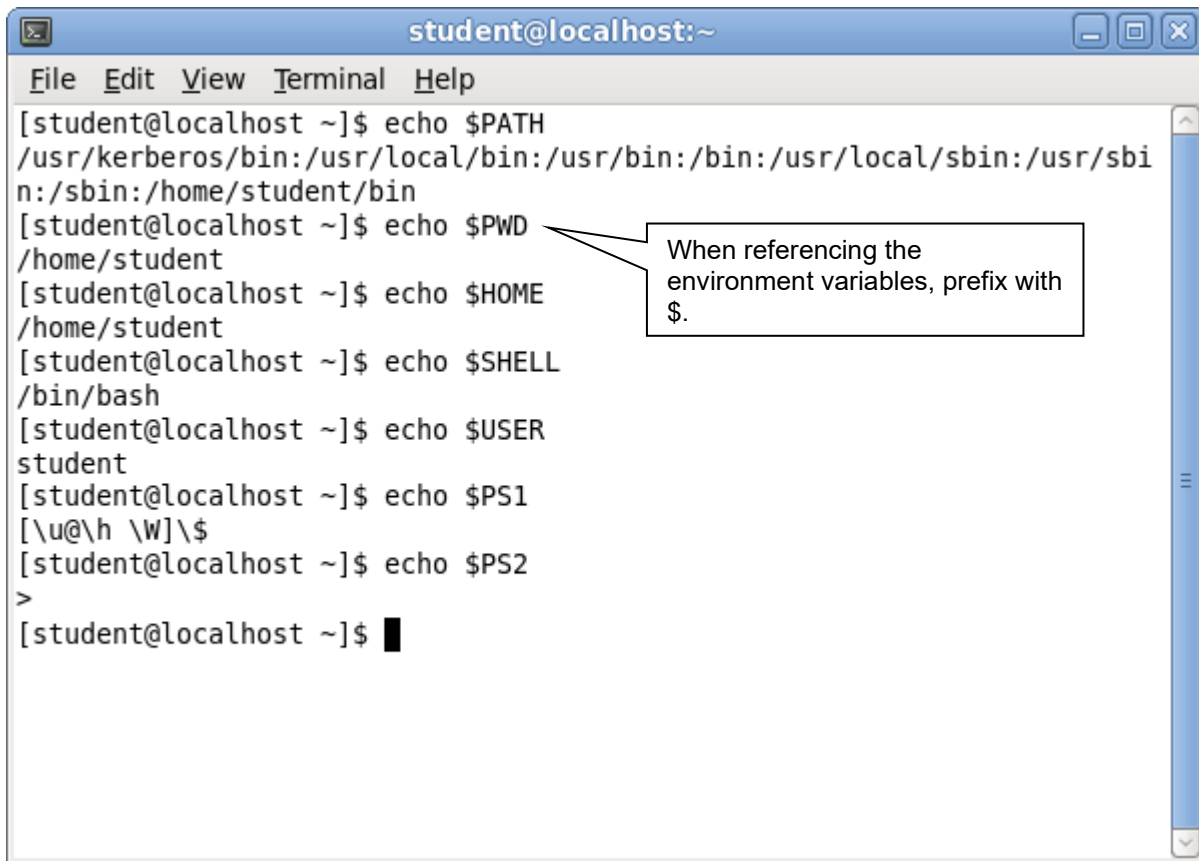
**Note**

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 <b>while</b> and <b>for</b> 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.



A terminal window titled "student@localhost:~" with a menu bar (File, Edit, View, Terminal, Help). The terminal displays the following commands and outputs:

```
[student@localhost ~]$ echo $PATH
/usr/kerberos/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/student/bin
[student@localhost ~]$ 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 $PS2
>
[student@localhost ~]$
```

A callout box points to the "\$PWD" command with the text: "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.

```

student@fedora:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
student@fedora:~$ ls | grep D
Desktop
Documents
Downloads
student@fedora:~$ ls | grep D | wc -l
3
student@fedora:~$
  
```

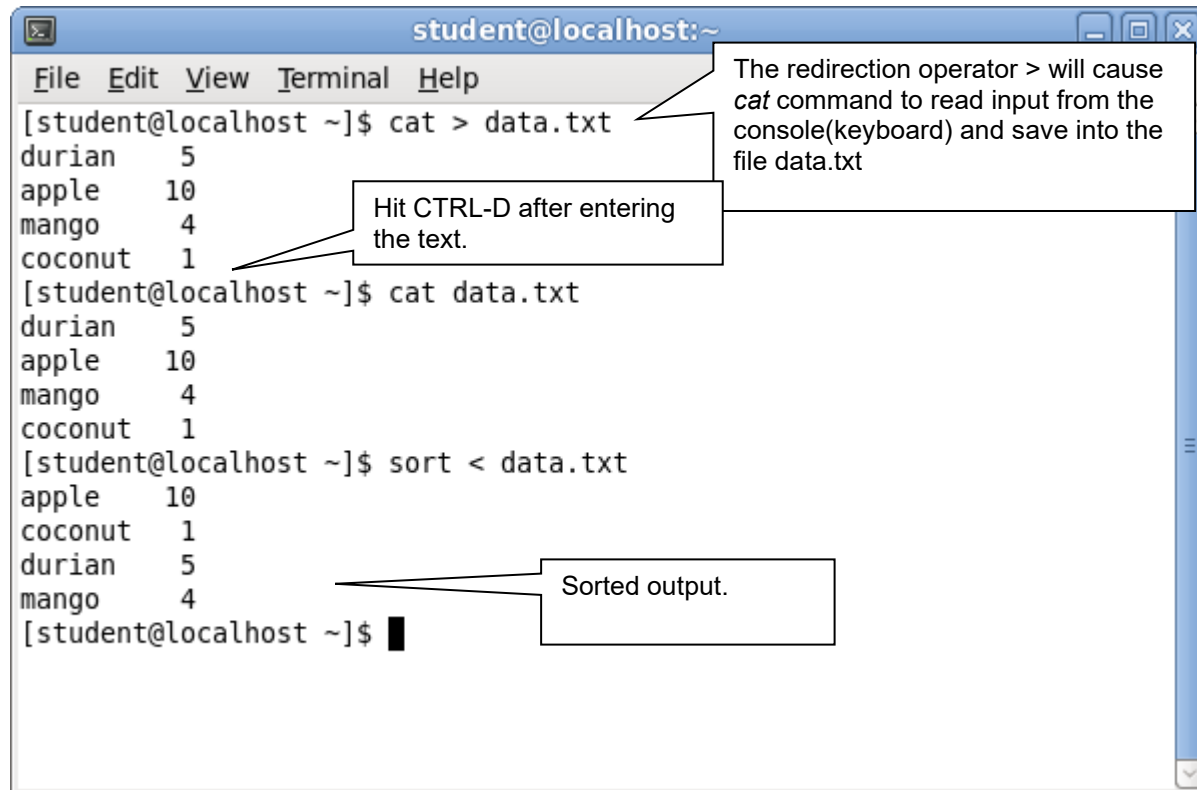
The output of `ls` is piped to the `grep` command. The `grep` command looks for lines which contains the string D.

Output of `grep` is piped to the `wc` command. `wc` counts the number of lines, words or bytes depending on the options. `-l` causes it to count the number of lines.

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



The image shows a terminal window titled 'student@localhost:~' with a menu bar (File, Edit, View, Terminal, Help). The terminal displays the following commands and output:

```
[student@localhost ~]$ cat > data.txt
durian    5
apple    10
mango     4
coconut   1
[student@localhost ~]$ cat data.txt
durian    5
apple    10
mango     4
coconut   1
[student@localhost ~]$ sort < data.txt
apple    10
coconut   1
durian    5
mango     4
[student@localhost ~]$
```

Annotations in the terminal window:

- A callout box points to the command `cat > data.txt` with the text: "The redirection operator > will cause *cat* command to read input from the console(keyboard) and save into the file data.txt".
- A callout box points to the end of the input after the first `cat` command with the text: "Hit CTRL-D after entering the text.".
- A callout box points to the output of the `sort` command with the text: "Sorted output.".

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	<b>ls   less</b>
Takes input from a file	<b>sort &lt; data.txt</b>
Sends output to a file	<b>echo hello &gt; hello.txt</b> <b>ls &gt; list.txt</b>
Append output to a file	<b>echo world &gt;&gt; hello.txt</b>
Takes input from a file and send output to a file	<b>sort &lt; data.txt &gt; sorted.txt</b>
Redirect STDERR to a file	<b>sort no_such_file 2&gt; error.txt</b>
Redirect STDOUT and STDERR to different files	<b>cat hello.txt 1&gt;data.bak 2&gt;error.txt</b>

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

```

student@fedora:~$ echo hello > hello.txt
student@fedora:~$ ls -l hello.txt
-rw-r--r--. 1 student student 6 Jan 29 14:12 hello.txt
student@fedora:~$ cat hello.txt
hello
student@fedora:~$ wc < hello.txt
1 1 6
student@fedora:~$ wc < hello.txt > wc.txt
student@fedora:~$ cat wc.txt
1 1 6
student@fedora:~$
  
```

Counts number of lines, words and bytes and display on terminal window

Counts number of lines, words and bytes and save to file wc.txt

#### 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.

```
student@fedora:~$ NAME=student
student@fedora:~$ echo $NAME
student
student@fedora:~$ NAME=
student@fedora:~$ echo $NAME

student@fedora:~$
```

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

The array index should be an integer.

```
student@fedora:~$ FRUIT[1]=apple
student@fedora:~$ FRUIT[2]=mango
student@fedora:~$ FRUIT[3]=durian
student@fedora:~$ echo ${FRUIT[1]}
apple
student@fedora:~$ echo ${FRUIT[*]}
apple mango durian
student@fedora:~$ echo ${#FRUIT[*]}
3
student@fedora:~$
```

## Exercise 5 - Compound Commands

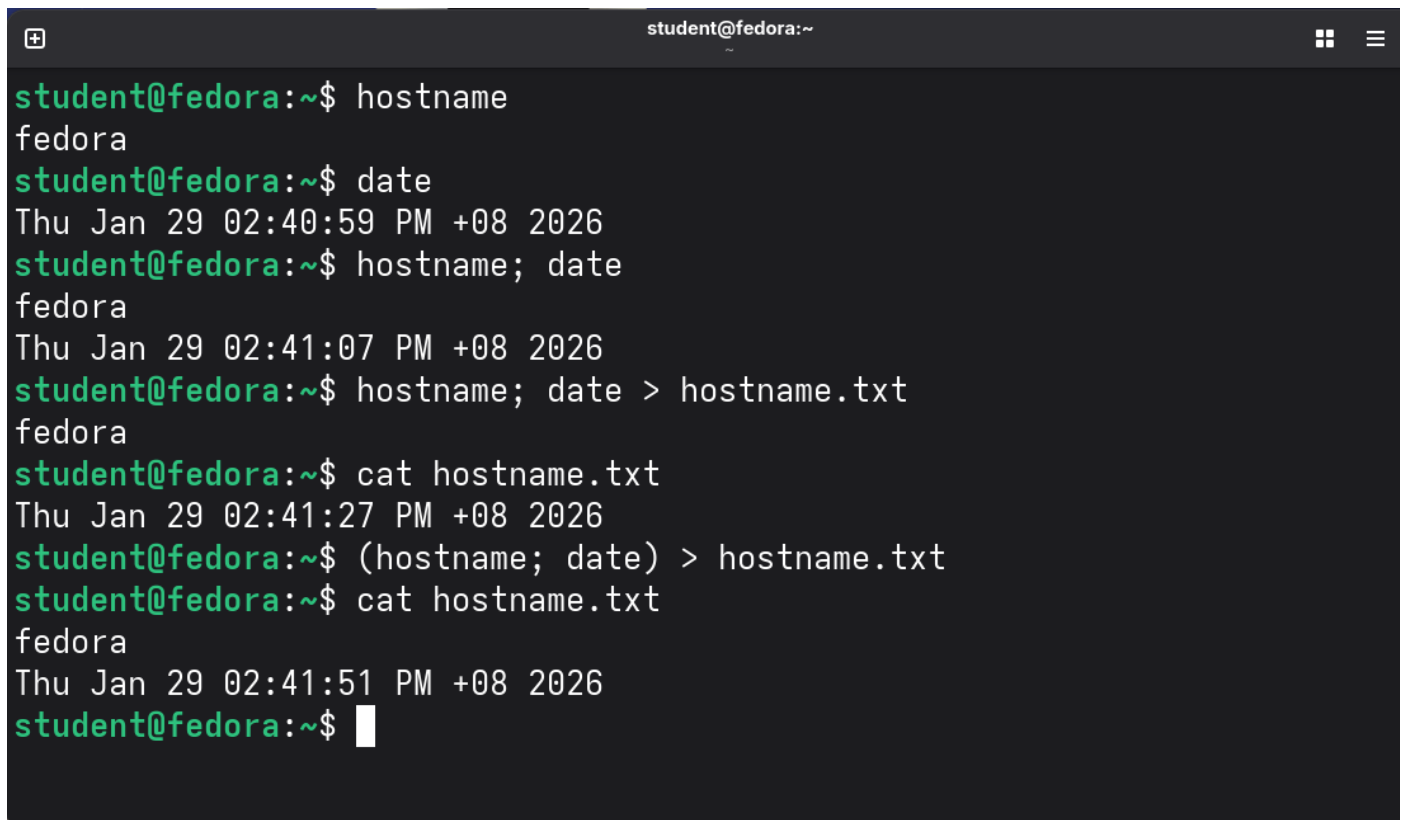
1. Multiple commands can be written in one single line of statement by delimiting the commands with command separators (;)

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:**

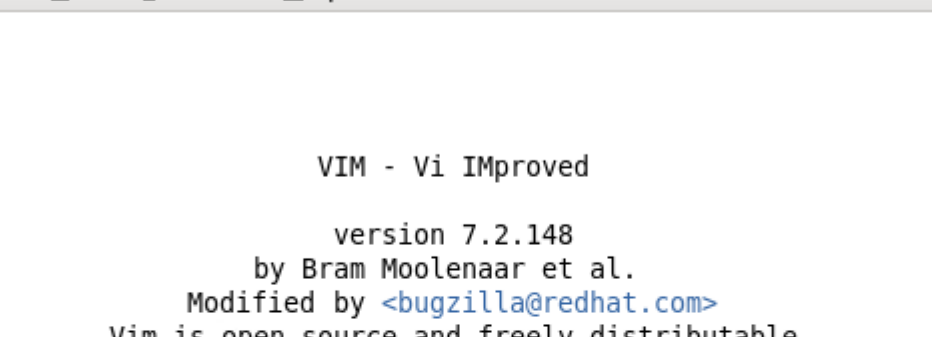
In a command group, the outputs from all individual commands are redirected.  
Without command group, only the output of the last command is redirected.

A terminal window titled 'student@fedora:~' with a dark background and light green text. It shows a series of commands and their outputs. The commands are: 'hostname', 'date', 'hostname; date', 'hostname; date > hostname.txt', 'cat hostname.txt', '(hostname; date) > hostname.txt', and 'cat hostname.txt'. The outputs are: 'fedora', 'Thu Jan 29 02:40:59 PM +08 2026', 'fedora', 'Thu Jan 29 02:41:07 PM +08 2026', 'fedora', 'Thu Jan 29 02:41:27 PM +08 2026', 'fedora', and 'Thu Jan 29 02:41:51 PM +08 2026'. The terminal window has a standard Linux desktop environment header with window controls and a system menu icon.

```
student@fedora:~$ hostname
fedora
student@fedora:~$ date
Thu Jan 29 02:40:59 PM +08 2026
student@fedora:~$ hostname; date
fedora
Thu Jan 29 02:41:07 PM +08 2026
student@fedora:~$ hostname; date > hostname.txt
fedora
student@fedora:~$ cat hostname.txt
Thu Jan 29 02:41:27 PM +08 2026
student@fedora:~$ (hostname; date) > hostname.txt
student@fedora:~$ cat hostname.txt
fedora
Thu Jan 29 02:41:51 PM +08 2026
student@fedora:~$
```

## Exercise 6 - Creating and Running Scripts

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



```
student@localhost:~
File Edit View Terminal Help

VIM - Vi IMproved

        version 7.2.148
        by Bram Moolenaar et al.
    Modified by <bugzilla@redhat.com>
Vim is open source and freely distributable


        Become a registered Vim user!
type  :help register<Enter>    for information

type  :q<Enter>                to exit
type  :help<Enter> or <F1>     for on-line help
type  :help version7<Enter>   for version info


[student@localhost ~]$
```

### 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.



```
student@localhost:~  
File Edit View Terminal Help  
  
~  
~  
~  
~  
~  
~  
VIM - Vi IMproved  
~  
~  
version 7.2.148  
by Bram Moolenaar et al.  
Modified by <bugzilla@redhat.com>  
Vim is open source and freely distributable  
~  
~  
~  
~  
~  
Become a registered Vim user!  
type :help register<Enter> for information  
~  
~  
type :q<Enter> to exit  
type :help<Enter> or <F1> for on-line help  
type :help version7<Enter> for version info  
~  
~  
-- INSERT --  
[student@localhost ~]$
```

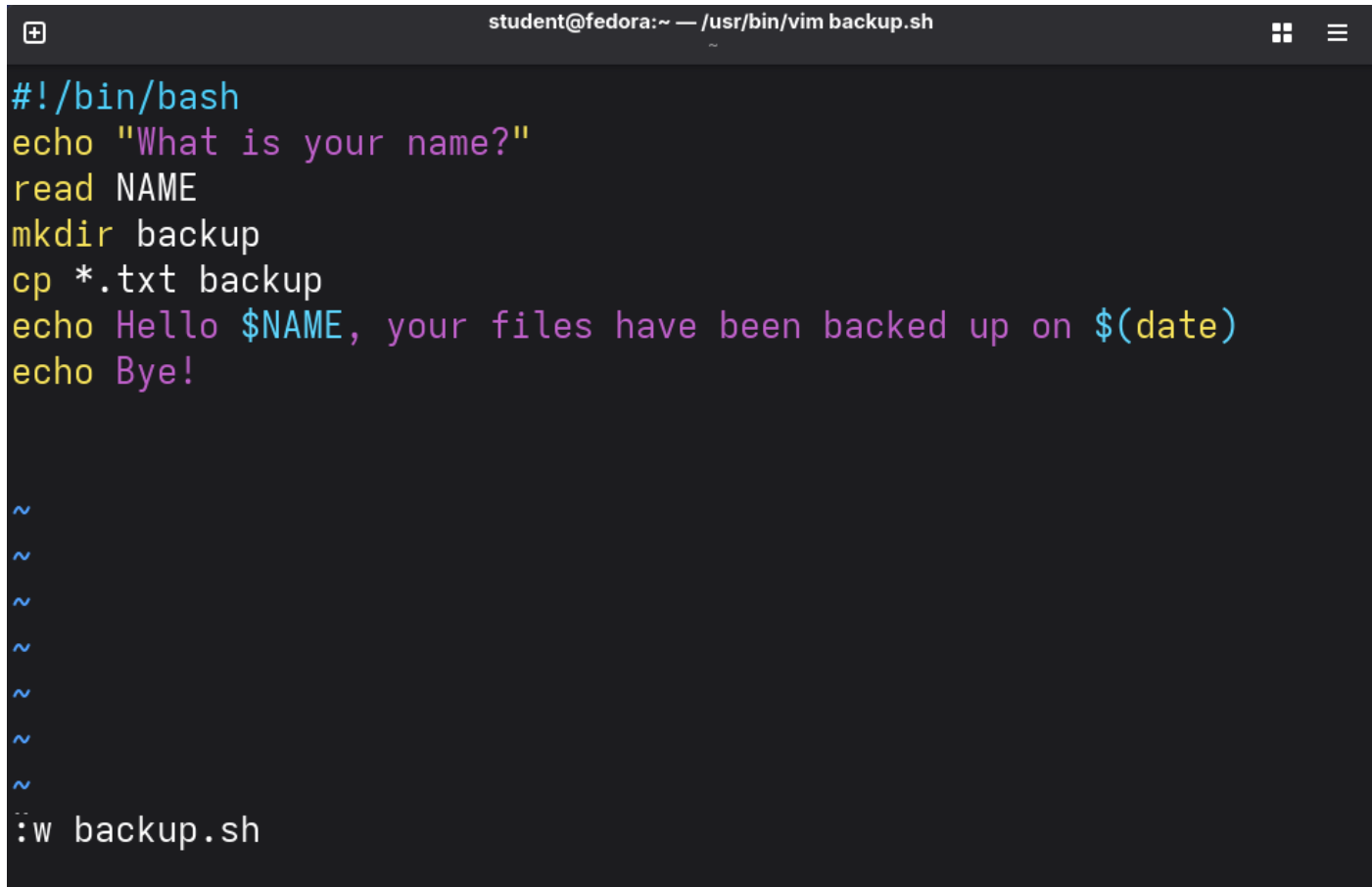
Type the script as shown.

### 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".

A screenshot of a terminal window with a dark background. The title bar at the top shows 'student@fedora:~ — /usr/bin/vim backup.sh'. The terminal content shows a shell script being edited in vim. The script includes a shebang, an echo statement asking for a name, a read command, mkdir, cp, and two more echo statements. There are several tilde characters (~) representing new lines. The command ':w backup.sh' is entered at the bottom.

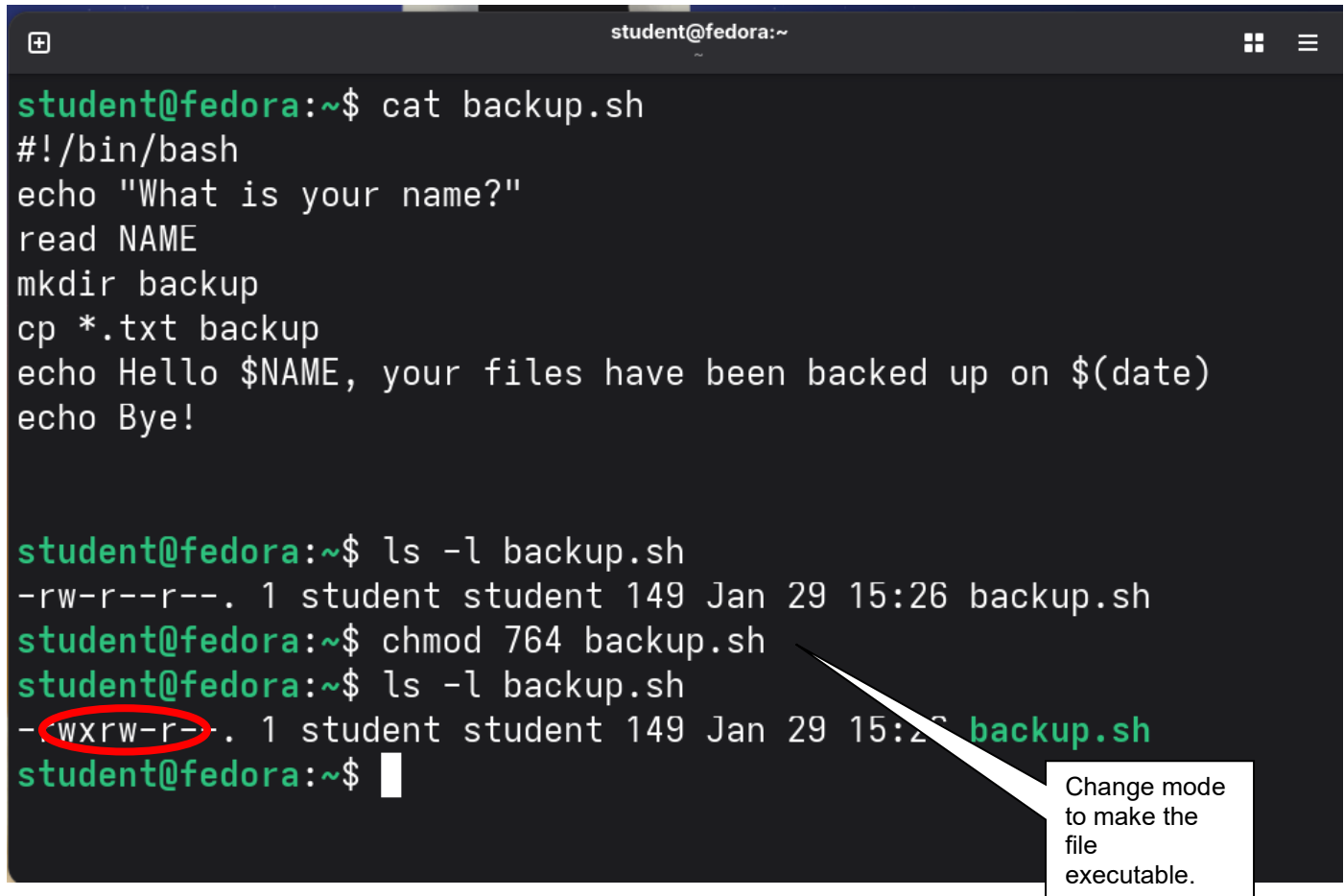
```
#!/bin/bash
echo "What is your name?"
read NAME
mkdir backup
cp *.txt backup
echo Hello $NAME, your files have been backed up on $(date)
echo Bye!

~
~
~
~
~
~
~
~
:w backup.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.



A terminal window titled 'student@fedora:~' showing the following commands and output:

```
student@fedora:~$ cat 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!

student@fedora:~$ ls -l backup.sh
-rw-r--r--. 1 student student 149 Jan 29 15:26 backup.sh
student@fedora:~$ chmod 764 backup.sh
student@fedora:~$ ls -l backup.sh
-rwxr--r--. 1 student student 149 Jan 29 15:26 backup.sh
student@fedora:~$
```

The permissions `-rwxr--r--` in the final `ls` output are circled in red. A callout box with an arrow pointing to this line contains the text: "Change mode to make the file executable."

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`

```
student@fedora:~$ ./backup.sh
What is your name?
Jason
Hello Jason, your files have been backed up on Thu Jan 29 03:41:02
PM +08 2026
Bye!
student@fedora:~$ ls -l backup
total 16
-rw-r--r--. 1 student student 6 Jan 29 15:41 hello.txt
-rw-r--r--. 1 student student 39 Jan 29 15:41 hostname.txt
-rw-r--r--. 1 student student 6 Jan 29 15:41 wc.txt
-rw-r--r--. 1 student student 8 Jan 29 15:41 who.txt
student@fedora:~$ PATH=$PATH
student@fedora:~$ backup.sh
What is your name?
Jason
mkdir: cannot create directory 'backup': File exists
Hello Jason, your files have been backed up on Thu Jan 29 03:42:18
PM +08 2026
Bye!
student@fedora:~$
```

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.