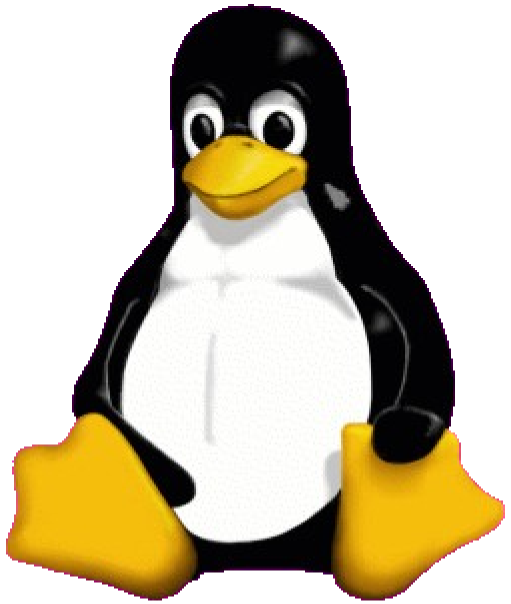
Lesson 7

By Dr. Amir

GNU/Linux

Control Operators

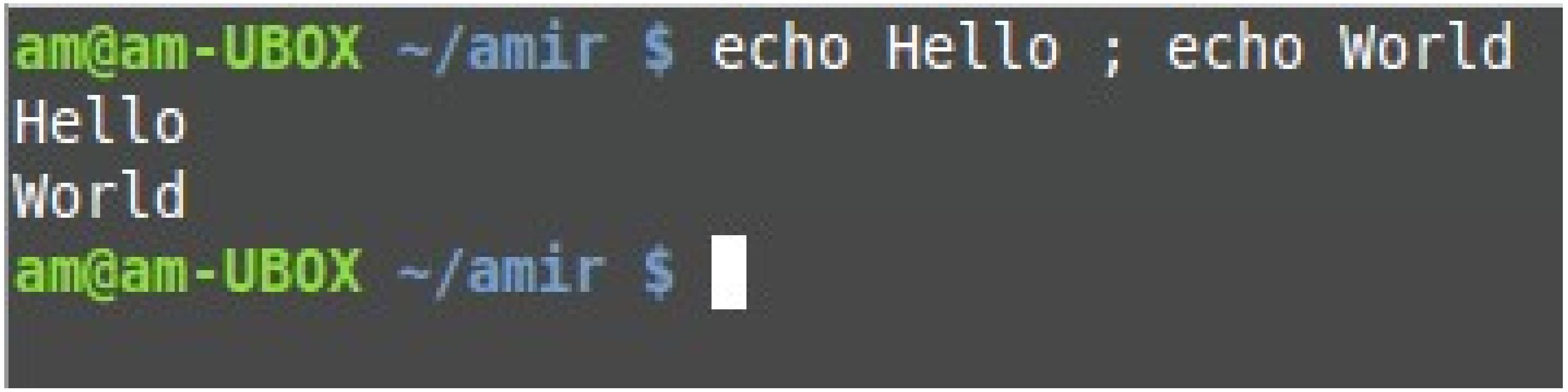


Control Operators

Linux command

; (semicolon)

To enter more than one command on a line

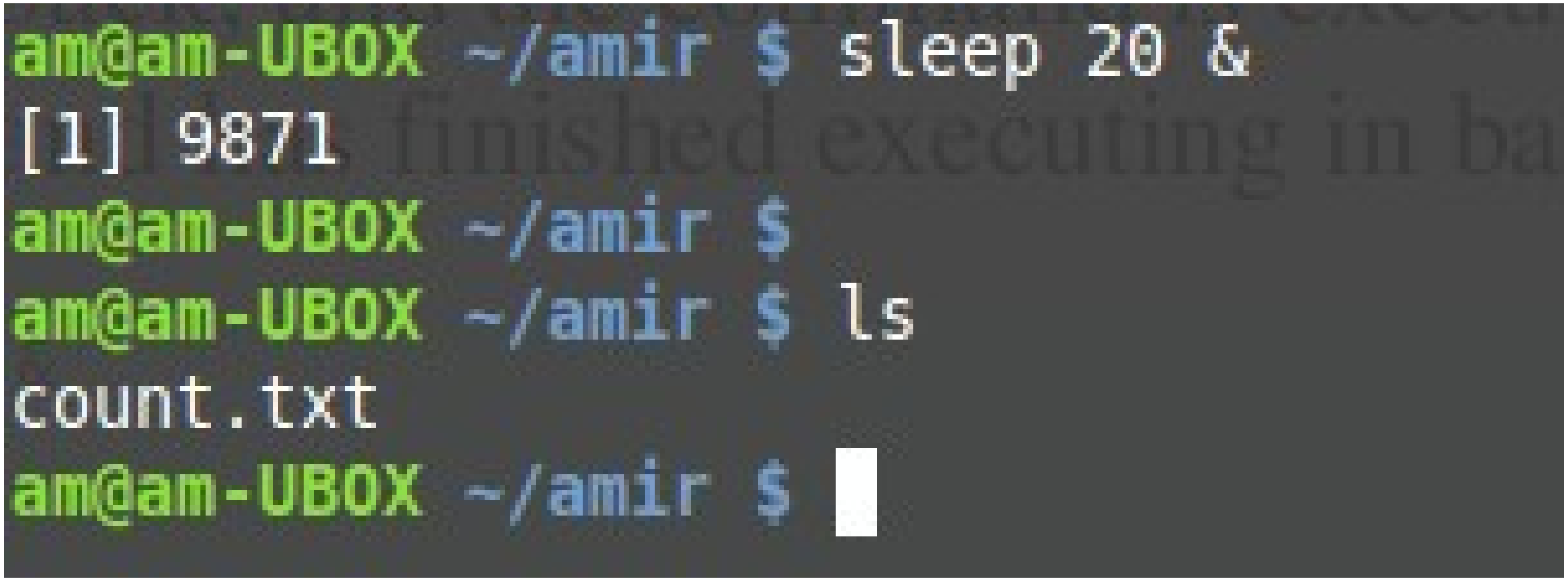


```
am@am-UBOX ~/amir $ echo Hello ; echo World
Hello
World
am@am-UBOX ~/amir $ 
```

& (ampersand)

& (ampersand)

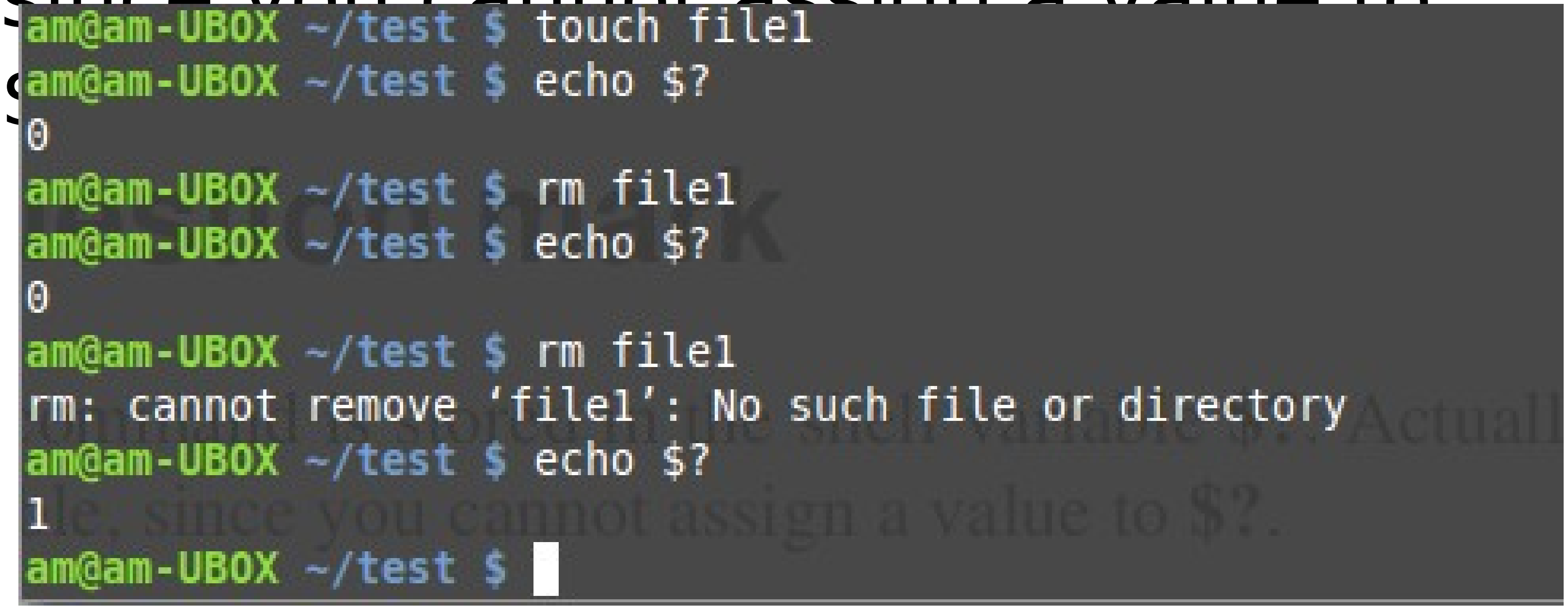
When a line ends with an ampersand &, the shell will not wait for the command to finish.



```
am@am-UBOX ~/amir $ sleep 20 &
[1] 9871
am@am-UBOX ~/amir $
am@am-UBOX ~/amir $ ls
count.txt
am@am-UBOX ~/amir $ 
```

$? (dollar question mark)

The exit code of the previous command is stored in the shell variable $?. Actually $? is a shell parameter and not a variable, since you cannot assign a value to $

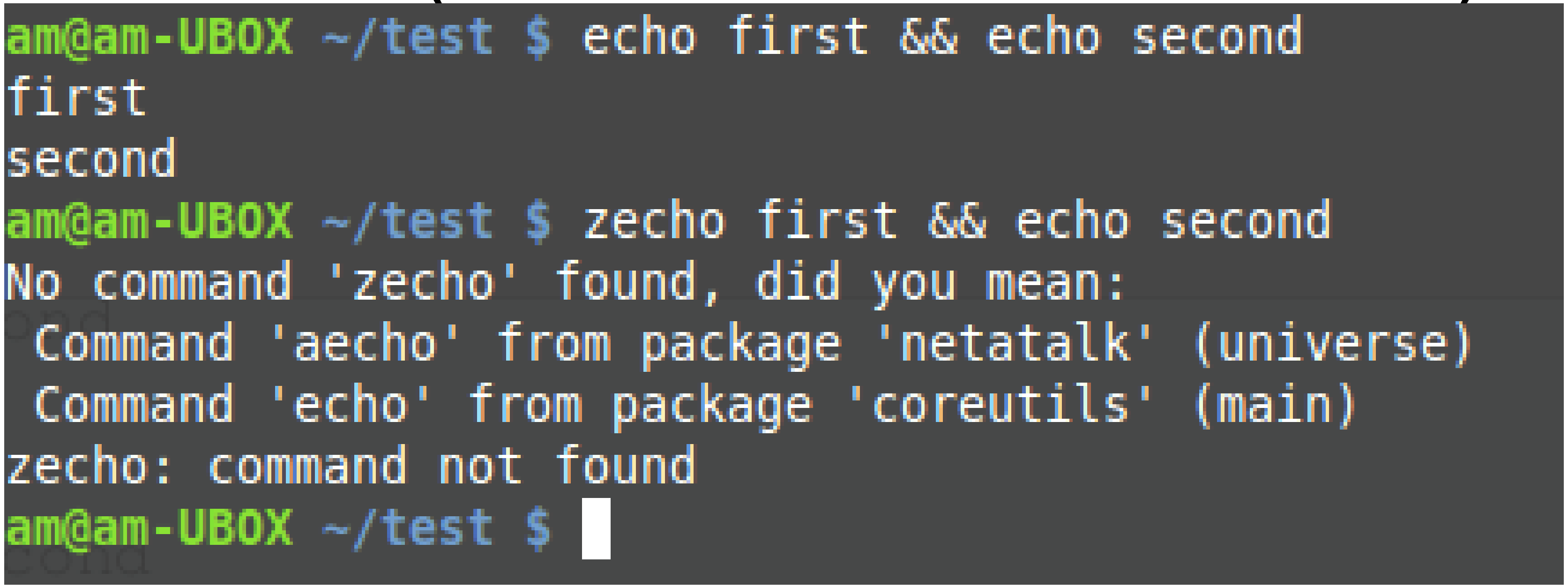


```
am@am-UBOX ~/test $ touch file1
am@am-UBOX ~/test $ echo $?
0
am@am-UBOX ~/test $ rm file1
am@am-UBOX ~/test $ echo $?
0
am@am-UBOX ~/test $ rm file1
rm: cannot remove ‘file1’: No such file or directory
am@am-UBOX ~/test $ echo $?
1
am@am-UBOX ~/test $ 
```

&& (double ampersand)

The shell will interpret && as a logical AND. When using && the second command is

executed only if the first one succeeds (returns a zero exit status).



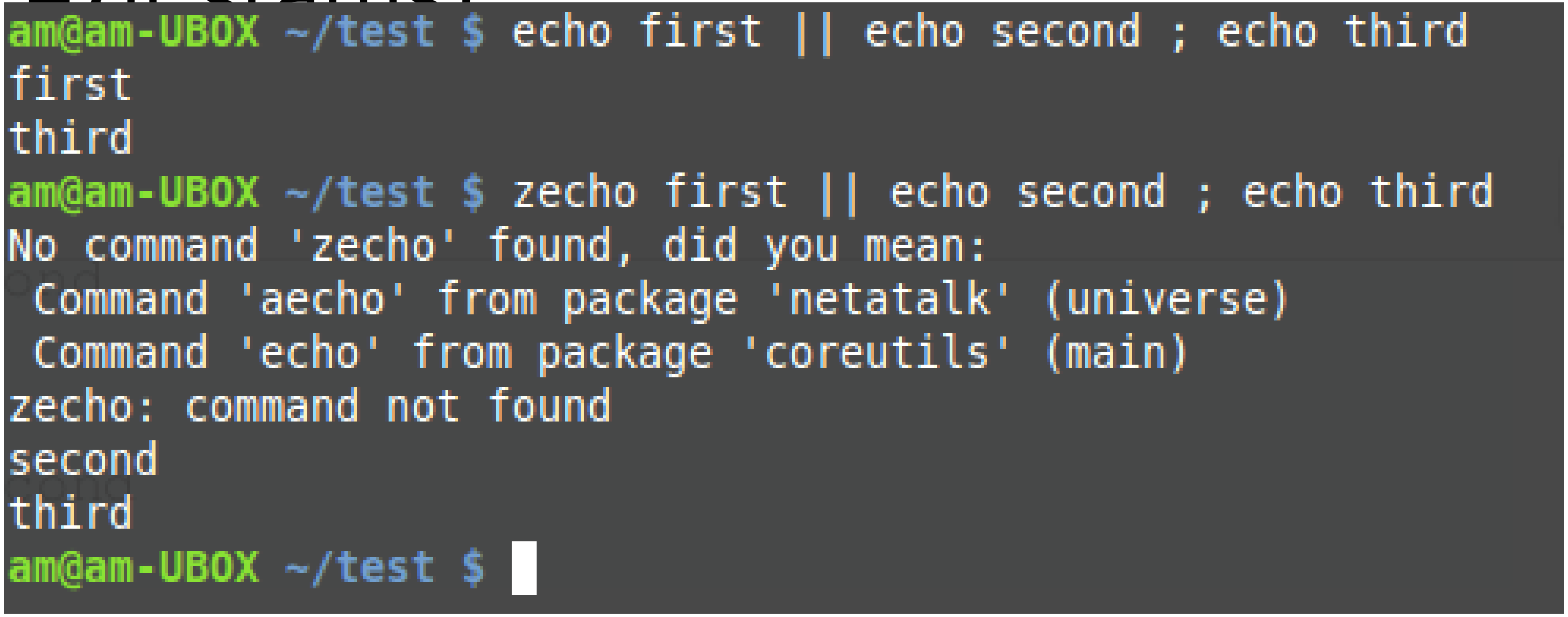
```
am@am-UBOX ~/test $ echo first && echo second
first
second
am@am-UBOX ~/test $ zecho first && echo second
No command 'zecho' found, did you mean:
 Command 'aecho' from package 'netatalk' (universe)
 Command 'echo' from package 'coreutils' (main)
zecho: command not found
am@am-UBOX ~/test $ 
```

Linux command

|| (double vertical bar)

The || represents a logical OR. The second command is executed only when the first

command fails (returns a non-zero exit status).

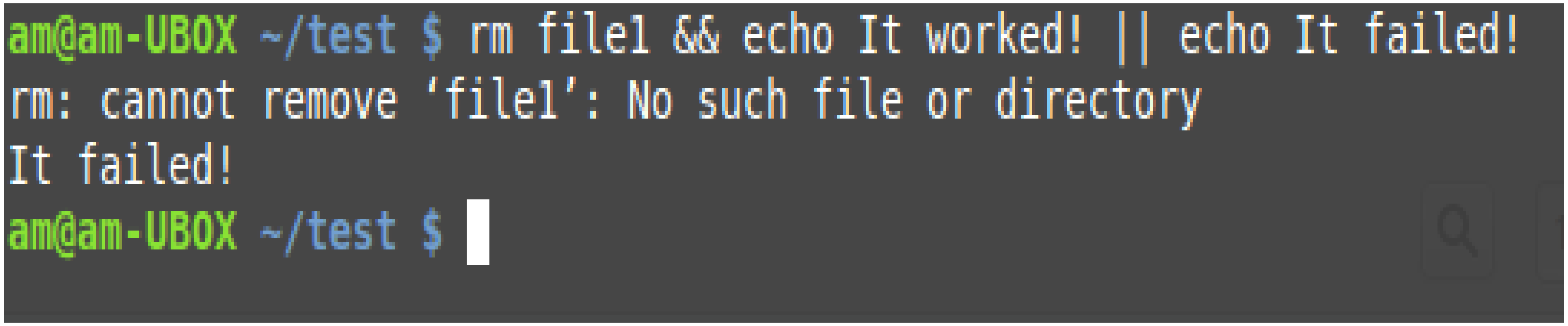


```
am@am-UBOX ~/test $ echo first || echo second ; echo third
first
third
am@am-UBOX ~/test $ zecho first || echo second ; echo third
No command 'zecho' found, did you mean:
 Command 'aecho' from package 'netatalk' (universe)
 Command 'echo' from package 'coreutils' (main)
zecho: command not found
second
third
am@am-UBOX ~/test $
```

combining && and ||

You can use this logical AND and logical OR to write an if-then-else structure on the

command line. This example uses echo to display whether the rm command was successful.

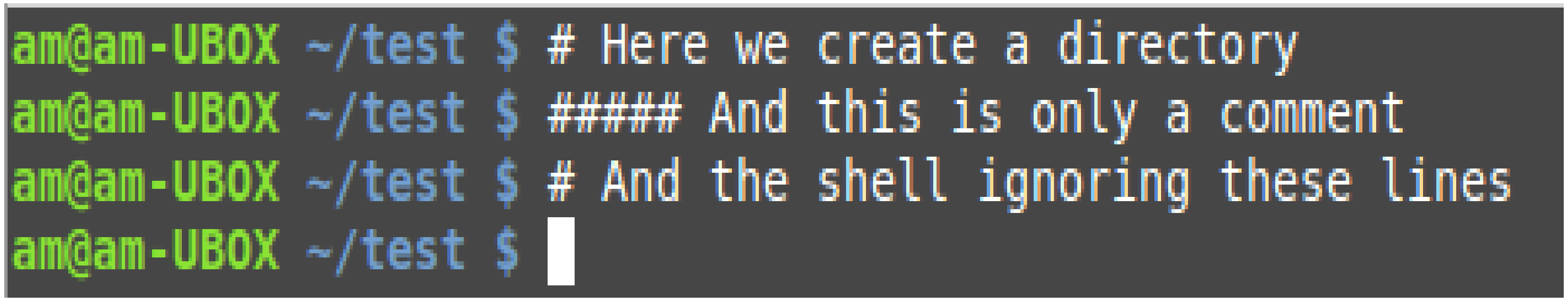


```
am@am-UBOX ~/test $ rm file1 && echo It worked! || echo It failed!
rm: cannot remove ‘file1’: No such file or directory
It failed!
am@am-UBOX ~/test $
```

# (pound sign)

Everything written after a pound sign (#) is ignored by the shell. This is useful to write a

shell comment, but has no influence on the command execution or shell expansion.

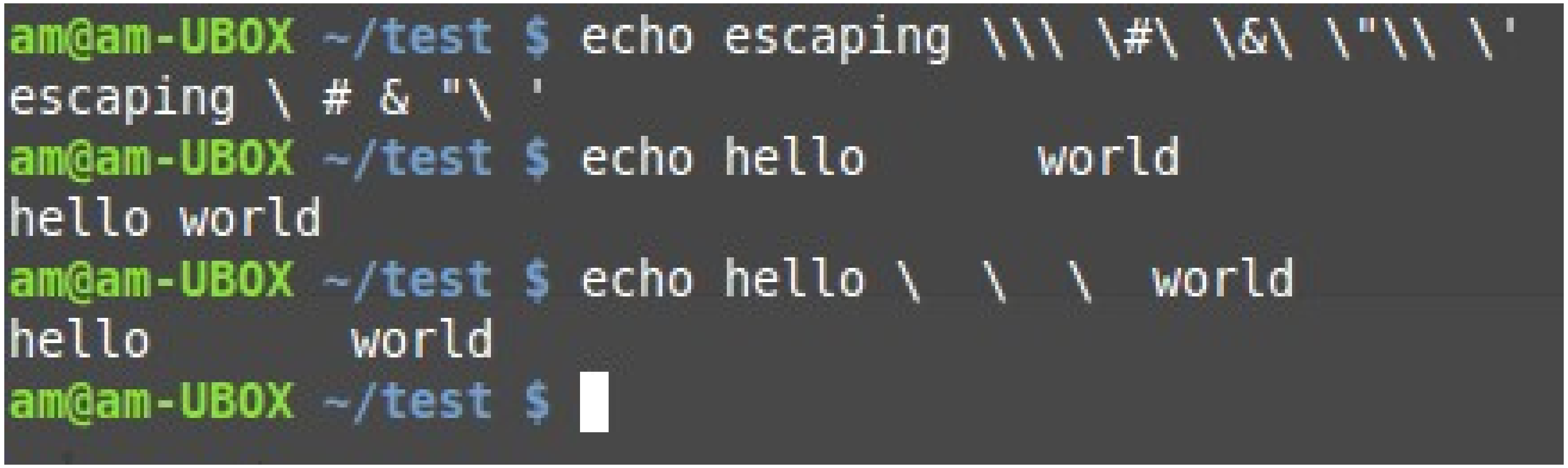


```
am@am-UBOX ~/test $ # Here we create a directory
am@am-UBOX ~/test $ ###### And this is only a comment
am@am-UBOX ~/test $ # And the shell ignoring these lines
am@am-UBOX ~/test $ 
```

\ (escaping special characters)

The backslash \ character enables the use of control characters, but without the shell

interpreting it, this is called escaping characters.

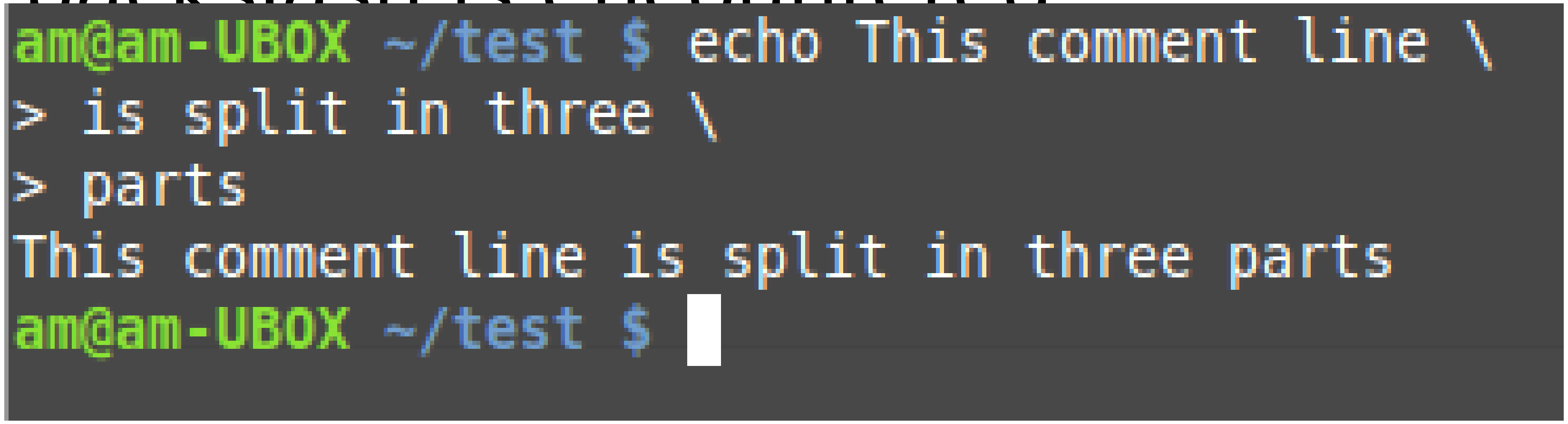


```
am@am-UBOX ~/test $ echo escaping \\\ \#\ \&\ \"\\ \'
escaping \ # & "\ '
am@am-UBOX ~/test $ echo hello      world
hello world
am@am-UBOX ~/test $ echo hello \  \  \  world
hello       world
am@am-UBOX ~/test $ 
```

end of line backslash

Lines ending in a backslash are continued on the next line. The shell does not interpret the

newline character and will wait on shell expansion and execution of the command line until a newline without backslash is encountered



```
am@am-UBOX ~/test $ echo This comment line \
> is split in three \
> parts
This comment line is split in three parts
am@am-UBOX ~/test $
```

Exercises

1. When you type passwd, which file is executed ?

```
which passwd
```

Exercises

2. What kind of file is that ?

```
file /usr/bin/passwd
```

Exercises

3. Execute the pwd command twice.

```
pwd ; pwd
```

Exercises

4. Execute ls after cd /etc, but only if cd /etc did not error.

```
cd /etc && ls
```

Exercises

5. Execute cd /etc after cd etc, but only if cd etc fails.

```
cd etc || cd /etc
```

Exercises

6. Echo it worked when touch test42 works, and echo it failed when the touch failed. All

on one command line as a normal user (not root).

```
cd ; touch test42 && echo it worked || echo it failed
it worked
```

Exercises

7. Execute sleep 6, what is this command doing ?

pausing for six seconds

Exercises

Execute sleep 200 in background (do not wait for it to finish).

```
sleep 200 &
```

Exercises

9. Write a command line that executes rm file55. Your command line should print 'success'

if file55 is removed, and print 'failed' if there was a problem.

```
rm file55 && echo success || echo failed
```