Higher Diploma in Computer Science

Computer Systems & Networks

SPRINGBOARD
www.springboardcourses.ie

Ireland's European Structural and Investment Funds Programmes 2014-2020
Co-funded by the Irish Government and the European Union

OIDEACHAS AGUS EOLAÍOCHTA | EDUCATION AND SCIENCE

HEA | HIGHER EDUCATION AUTHORITY
AN tÚDARÁS um ARD-OIDEACHAS

European Union
Investing in your future
European Social Fund

SETU
Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Quoting

Double Quotes

- Double quotes stop the shell from *interpreting* some metacharacters, including glob characters.

> Glob characters, also called wild cards, are symbols that have special meaning to the shell (i.e, `*`, `?`).

- Double quotes still allow for *command substitution*, *variable substitution,* and permit some other shell metacharacters (i.e., the `PATH` variable)

```
compsys@compsys-virtualbox:~/tmp/tutorial$ echo The glob chars are * ? and []
The glob chars are Dec Oct Sept ? and []
compsys@compsys-virtualbox:~/tmp/tutorial$ echo "The glob chars are * ? and []"
The glob chars are * ? and []
```

Single Quotes

- Single quotes prevent the shell from doing **any interpreting** of special characters, including globs, variables, command substitution and other metacharacters.

```
compsys@compsys-virtualbox:~$ echo The car costs $100
The car costs 00
compsys@compsys-virtualbox:~$ echo 'The car costs $100'
The car costs $100
```

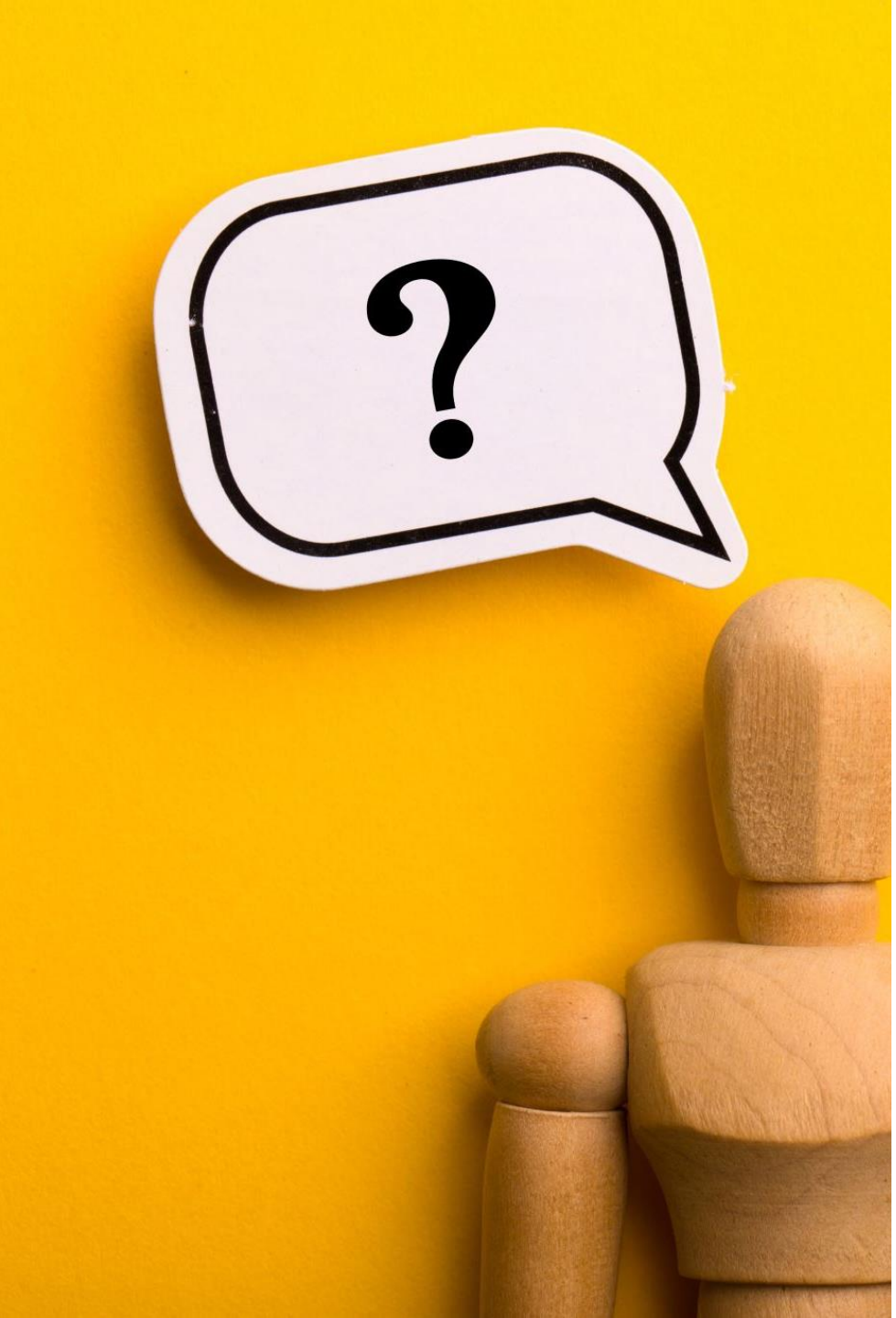
Backslash Character

- A technique to essentially **single quote a single character** is to use the backslash character `\`.
- If the phrase below is placed in quotes, `$1`and `$PATH` are not variables:

```
compsys@compsys-virtualbox:~$ echo "The service costs $1 and the path is $PATH"

The service costs  and the path is
/usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
:/usr/games
```

- What if you want to have `$PATH` treated as a variable and `$1` not?

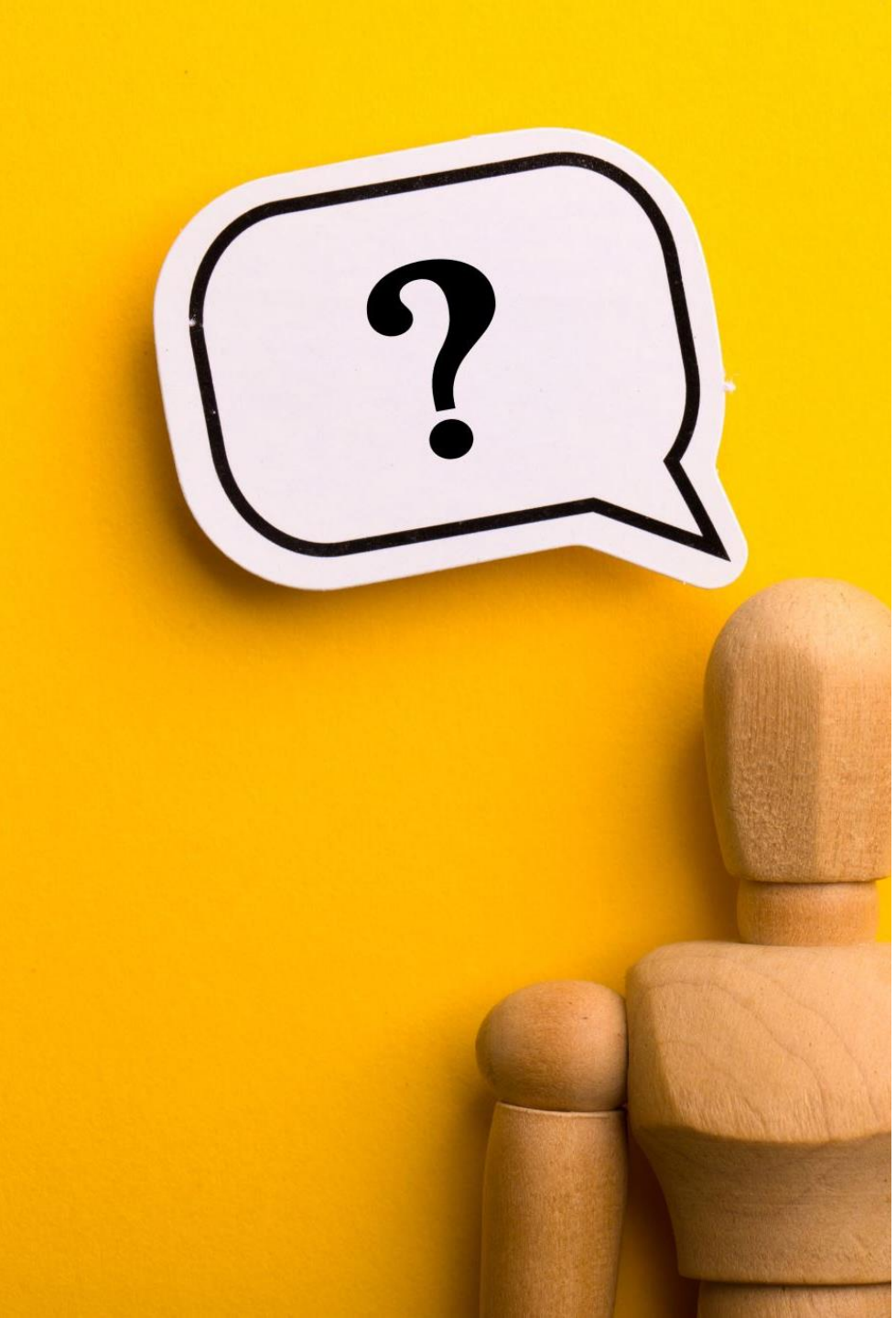
```
compsys@compsys-virtualbox:~$ echo The service costs \$1 and the path is $PATH

The service costs $1 and the path is
/usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bi
n:/usr/games
```



Single or Double Quotes?

- SINGLE ‘ → will literally echo what you have between them

- When ' ' is used around anything, there is no "transformation or translation" done.
- It is printed as it is.

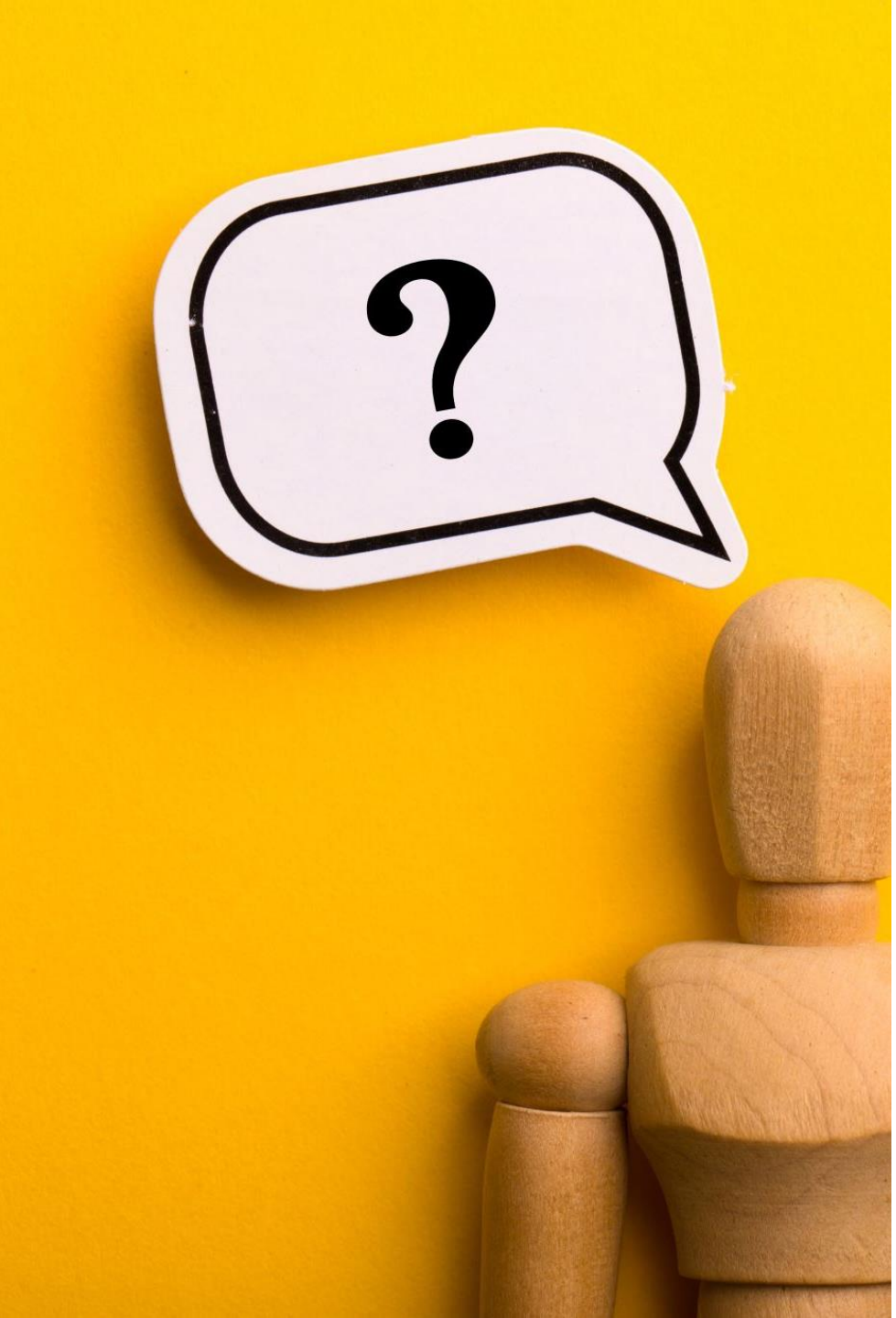


Single or Double Quotes?

- DOUBLE “ →will evaluate variables between them and output the value of the variable

- Double quotes will expand the variable

  - With " ", whatever it surrounds, is "translated or transformed" into its value.



But it works without “…. ”

In almost all situations you want to add quotes.

- quoted version is not subject to field splitting by the shell

*Enclosing characters in double quotes preserves the literal value of all characters within the quotes, with the exception of $, `, * **[man bash]**

Backquotes

- Backquotes, or backticks, are used to specify **a command within a command**, a process called *command substitution*.

```
compsys@compsys-virtualbox:~$ echo Today is date
Today is date
```

- To execute the `date` command so the output of that command is sent to the `echo` command, put the `date` command inside of two backquotes:

```
compsys@compsys-virtualbox:~$ echo Today is `date`
Today is Wed 1 Sept 2021 12:40:04 IST
```

Control Statements

Control Statements

- Control statements allow you to use multiple commands at once or run additional commands.
- Control statements include:
  - Semicolon (`;`)
  - Double ampersand (`&&`)
  - Double pipe (`||`)

Control Statements

- The semicolon can be used to run **multiple** commands, one after the other:

```
compsys@compsys-virtualbox:~$ cal 1 2021; cal 2 2021; cal 3 2021
```

- The double ampersand `&&` acts as a logical "**and**" if the first command is **successful**, then the second command (to the right of the `&&`) will also run:

```
compsys@compsys-virtualbox:~$ ls /etc/perl && echo That listed it perfectly
Net
That listed it perfectly
```

- The double pipe `||` is a logical "or". It works similarly to `&&`; depending on the result of the first command, the second command will either run or be skipped:

```
compsys@compsys-virtualbox:~$ ls /etc/xml || echo failed
ls: cannot access /etc/xml: No such file or directory
failed
```

Help

Man Pages

- UNIX is the operating system that Linux was modeled after.
- The developers of UNIX created help documents called *man pages* (short for *manual page*).
- Man pages provide a basic description of the purpose of the command, as well as details regarding available options.

Searching Man Pages

- To search a man page for a term, press the / and type the term followed by the **Enter** key.

```
/all
```

- If a match is found, to move to the next match of the term, press **n**. To return to a previous match of the term, press **N**.

Finding Commands and Documentation

Finding Commands and Documentation

To search for

- the location of a command *or*
- the man pages for a command

use the `whereis` command.

- This command searches for commands, source files and man pages in specific locations where these files are typically stored:

```
compsys@compsys-virtualbox:~$ whereis ls
ls: /bin/ls /usr/share/man/man1p/ls.1.gz /usr/share/man/man1/ls.1.gz
```

- Man pages are easily distinguished from commands as they are typically compressed with a program called `gzip`, resulting in a filename that ends in `.gz`.

Find Any File or Directory

- To find any file or directory, use the `locate` command.

- The output can be quite large so it may be helpful to use the following options:
  - The `-c` option to the `locate` command will list how many files match:

```
compsys@compsys-virtualbox:~$ locate -c passwd
97
```

  - The `-b` option only includes listings that have the search term in the *basename* of the filename. To limit the output even further, place a `\` character in front of the search term:

```
compsys@compsys-virtualbox:~$ locate -b "\passwd"
```