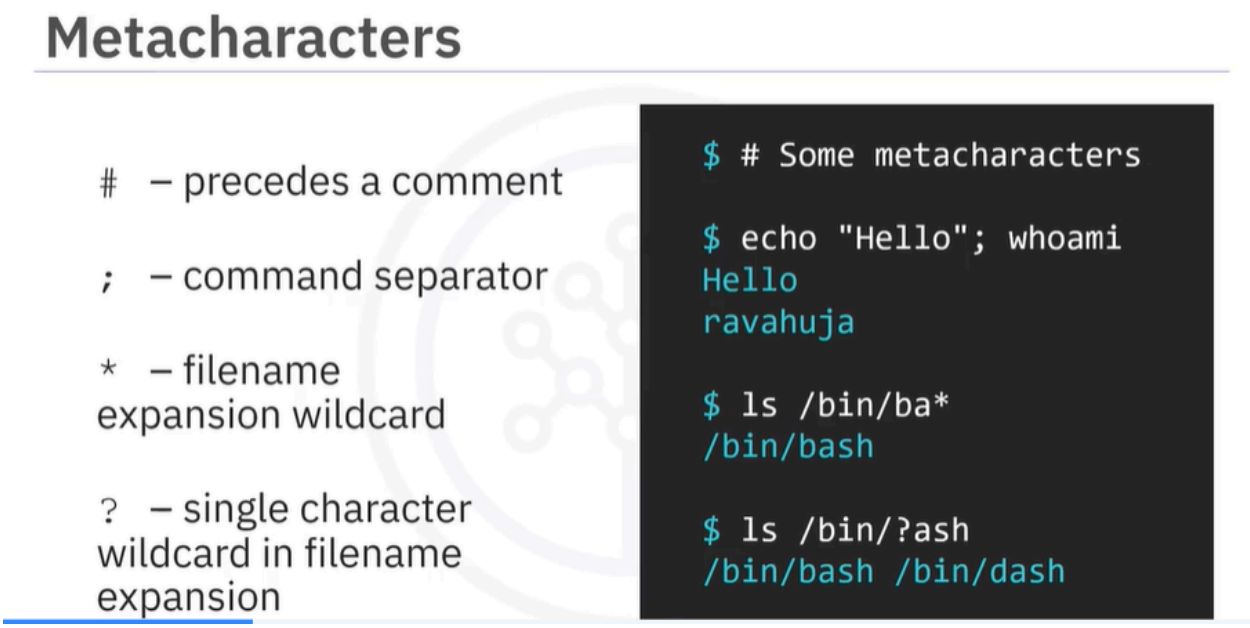
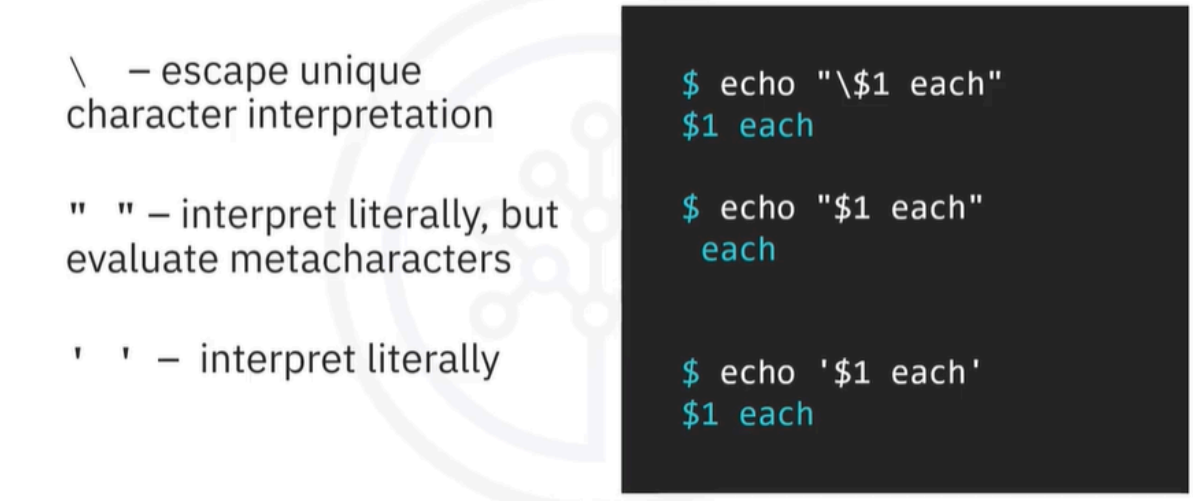
Useful Features of the Bash Shell



- Metacharacters are special characters that have meaning to the shell. The pound symbol is used to include comments that the shell ignores. For example, this comment does not return anything. The semicolon is a metacharacter that separates commands typed on the same line. For example, these two commands return their outputs on two lines like this. The asterisk represents any number of consecutive characters within a filename pattern. For example, this expression returns all objects in the slash bin directory with a name starting with ba and followed by any characters, which in this case is Bash. And the question mark acts as a single-character version of the asterisk metacharacter. For example, this expression lists all objects in the slash bin directory having a single character in place of the question mark, namely the bash and dash paths.



- Quoting specifies whether the shell should interpret special characters as metacharacters or ‘escape’ them. You can use the backslash to escape the interpretation of a single character as a metacharacter. Here, the backslash tells Bash to interpret the dollar sign as text rather than the default variable name, so the output is literally ‘dollar one each.’ Double quotes will interpret the text literally, except for any metacharacters, which will be interpreted according to their special meanings. For example, the ‘dollar one’ expression, without the preceding backslash, is interpreted as a variable, which in this case is empty, as we see here. Notice that it printed the space before “each.” Single quotes, on the other hand, are used to interpret all contents as literal characters. For example, this expression returns the same result as the first example above.

I/O redirection

Input/Output, or **I/O redirection**, refers to a set of features used for redirecting

```
>     – Redirect output to the file
>>    – Append output to the file
2>    – Redirect standard error to file
2>>   – Append standard error to file
<     – Redirect file contents to standard input
```

I/O redirection examples

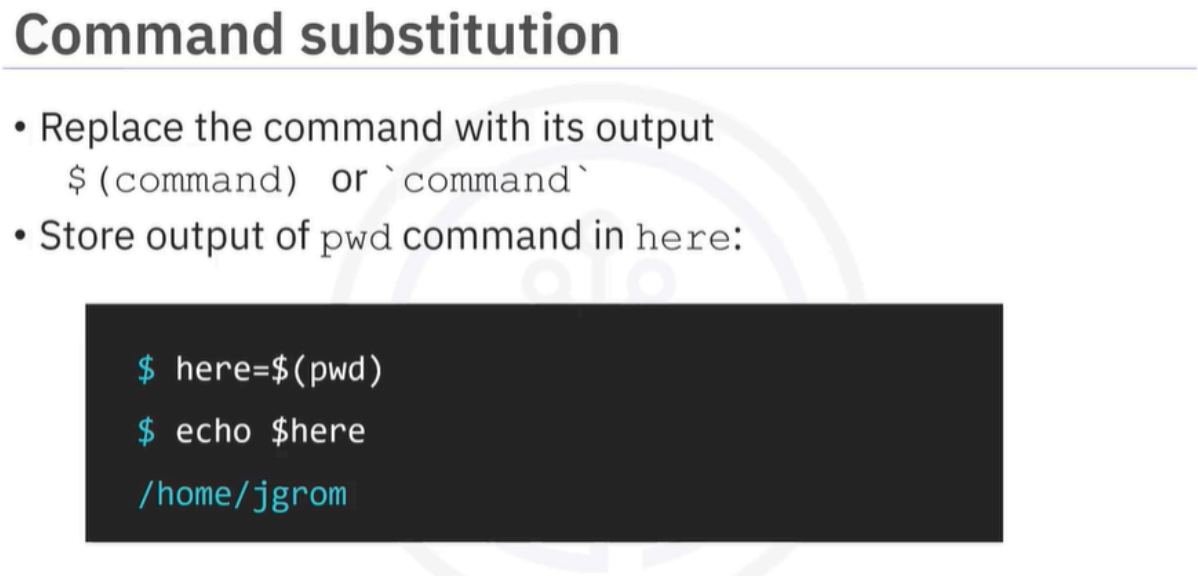
```
$ echo "line1" > eg.txt
$ cat eg.txt
line1
$ echo "line2" >> eg.txt
$ cat eg.txt
line1
line2
```

```
$ garbage
garbage: command not found
$ garbage 2> err.txt
$ cat err.txt
garbage: command not found
```

- Input/Output, or ‘I’ ‘O’ redirection, refers to a set of features used for redirecting either the standard input, the keyboard, or the standard output, the terminal. The ‘>’ symbol is used to redirect the standard output of a command to a file. It also creates the file if it doesn’t exist and overwrites its contents if it already exists. You can avoid overwriting by using the double greater than symbol, which appends output to existing content. The combination ‘2>’ redirects an error message to a file. To append the error message, for example, to an error log file, add another greater than sign. And lastly, the less

than sign is a redirection that is used to pass file contents as input to the standard input.

- Let's look at a few examples. You can start by simultaneously creating a file and populating it with some text, like this. You can then cat the file to see what's there. As you can see, 'line1' was written to 'eg.text'. Now, you can try adding another line to the file. And viewing the contents again, you can see that the file contains exactly the two lines you added. Entering something like the word garbage returns an error. This expression catches the error message and redirects it to the file 'err.txt.' And indeed, the contents of 'err.txt' is what can be expected.



- You can use command substitution to replace a command with its output. There are two equivalent notations: in the first, the command is encapsulated by parentheses with a dollar sign out front, while in the second, the command is encapsulated within backticks, also known as backquotes. Let's say you want to store the current directory path in a variable called 'here.' You can use command substitution on the pwd (or present working directory) command to capture its output and assign it to the variable 'here.' And indeed, echoing its value returns the current directory.

Command line arguments

- Program arguments specified on the command line
- A way to pass arguments to a shell script
- Usage:

```
$ ./MyBashScript.sh arg1 arg2
```

- Command line arguments are arguments used by a program specified on the command line. In particular, they provide a way to pass arguments to a shell script, a program. Command line arguments for a Bash script are specified: The arguments ‘arg1’ and ‘arg2’ are passed to ‘MyBashScript.sh’. Bash has two main modes of operation: Batch mode, which is the usual mode, runs commands sequentially.

Batch versus concurrent modes

Batch mode:

- Commands run sequentially

```
command1; command2
```

Concurrent mode:

- Commands run in parallel

```
command1 & command2
```

- For example, these two commands, separated by the semicolon metacharacter, will be executed in a particular order. Command 2 only runs after command 1 is completed. In concurrent mode, commands run in parallel. The ampersand operator, after command 1, directs command 1 to operate in the background and passes control to command 2 in the foreground.

Recap

In this video, you learned that:

- Metacharacters are special characters that have meaning to the shell
- Quoting is used to interpret or escape metacharacter meaning
- I/O redirection refers to a set of features used for redirecting
- Command substitution to replace a command with its output
- Command line arguments are used to pass arguments to a shell script
- Concurrent mode allows commands to run in parallel