# Input/Output and Shell Fundamentals: C file handling + Bash metacharacters/variables

## Metacharacters

Metacharacters are symbols with special meaning to the shell. These symbols can be used for: Comment, Filename expansion (wildcards), variable referencing, separating commands, executing commands as background jobs, character escaping (removing the special meaning of another metacharacter).

Metacharacters may include: **‘ \ # ~ $ \* . ` & “**

**The effect of quotations**

Bash uses quotations differently compared to other computer languages. Single and double quotes are not used to delineate strings or characters, but to control how the shell groups characters and interprets special characters within a string. Bash calls this process **word splitting**.

**No Quotes**

**Without quotes**, the shell interprets all metacharacters and attempts to expand their meaning.

A black screen with white text

AI-generated content may be incorrect.

**Filenames:** The second line, you can see that the asterisk has been expanded to become a filename. The directory the command was executed in has that file name: **mymath**. So, the asterisk was expanded to be all ‘all filenames’ that could be in the directory. If there were more than one file, all would have appeared.

**Variables expansion**

Notice the output of the second line? The $ has disappeared in the output because the shell recognises the dollar as a special character. It is a metacharacter for variable expansion.

A black background with white text

AI-generated content may be incorrect.

This is an example of correct variable expansion.

A black background with white text

AI-generated content may be incorrect.

**Double quotes:**

Double quotes are used to prevent **filename expansion** (the recognition of filename wildcard metacharacters)

A black screen with white text

AI-generated content may be incorrect.

However, double quotes do not prevent **variable expansion**, as shown below.

A black screen with white text

AI-generated content may be incorrect.

In this example, the quotation marks prevent the asterisk from being replaced with a list of files. The first line doesn’t include the quotes. The output is a list of files in that directory.

A screen shot of a computer code

AI-generated content may be incorrect.

**Single Quotes**

Single quotes are used to prevent filename and variable expansion (the recognition of filename wildcard metacharacters and the dollar sign). It means single quotes leave all characters unchanged.

A black screen with white text

AI-generated content may be incorrect.

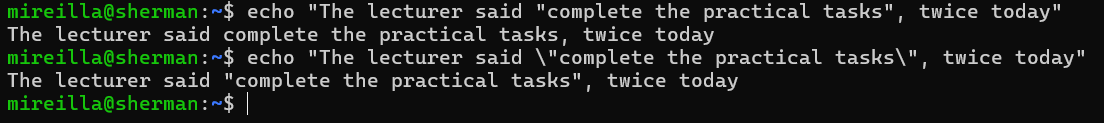
Single quotes also prevent variable expansion:

A black and white screen with white text

AI-generated content may be incorrect.

**Character Escaping**

To remove the special character meaning of one character, use character escaping. A **back-slash** is placed before the character to be escaped.



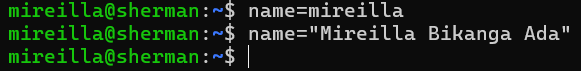
In this example, you can see that the special meaning of double quotes “” is removed by using the backslash. So, the output is a normal double quote: "complete the practical tasks"

## Variables

**Defining and assigning a variable**

Defining and naming a variable occur at the same time. In this example, the variable 'name' is assigned the value 'mireilla' using an equal sign (=), in the first line of code. It is assumed to be a string. Variable names should start with a letter but may include numbers in other positions. Variable names should not include spaces.

As seen in the second line, when the **value** has a space or other metacharacter, use quotes.



**Referencing a variable**

To reference a variable, use the variable name with a dollar ($) sign before it:

A black screen with green and white text

AI-generated content may be incorrect.

If the value of the variable is placed adjacent to the other characters, this prevents proper variable expansion. To prevent this, use the curly braces (**{}**).

A screen shot of a computer

AI-generated content may be incorrect.

**Unsetting Variables**

A variable that has been assigned can be unassigned using **unset** as seen in the example below

A screen shot of a computer

AI-generated content may be incorrect.

**Arithmetic**

The shell does not like maths. This example shows that the shells prefer literal interpretation.

A screen shot of numbers and symbols

AI-generated content may be incorrect.

However, you can force the shell to perform arithmetic as seen below.

A screen shot of a computer

AI-generated content may be incorrect.

Example of addition and subtraction

A screen shot of a computer

AI-generated content may be incorrect.

Example multiplication and division

A screen shot of a computer

AI-generated content may be incorrect.

**Getting help**

The **help** command only describes Bash commands. To get help on Linux commands, you need to use the **man** (manual) command.

A black background with white text

AI-generated content may be incorrect.

## Input/Output

Functions: printf, scanf, fopen, fgets

Exercise. Write a C program that:

1. Prompts the user for a filename (**scanf**).
2. Opens the file (**fopen**).
3. Reads it line by line (**fgets**).
4. Counts the number of lines.
5. Prints the result (**printf**).

**Create a simple text file**

My file simple.txt and its content.

A black background with white text

AI-generated content may be incorrect.

**Program (count\_lines.c)**

Now, the content of the **count\_lines.c** program I created using **vim**:

A screenshot of a computer program

AI-generated content may be incorrect.

**Compile and run**

A black screen with white text

AI-generated content may be incorrect.

**Printf (print formatted)**

This built-in ***printf*** command is very similar to the C standard I/O ***printf()*** function, but they are not identical. For instance, single- and double-quoted strings are treated differently in shell scripts than in C programs.

In this example, I have used the ***printf*** command to print the value of my variable. Notice that in the second line, I have used a formatting code. The last line shows the output when I do not use the formatting code. The special formatting code **“%s”** means print a string argument. **\n** = newline (line break).

A screen shot of a computer code

AI-generated content may be incorrect.

You can also use the ***printf*** command to display simple messages. So put the message in place of the formatting code. Of course, you could output the same message using the **echo** command.

A black background with white text

AI-generated content may be incorrect.

The results of a command can be assigned to a variable using backquotes.

A black screen with white text and red line

AI-generated content may be incorrect.

The date shown is the date when the variable DATE is assigned its value. The value of the variable remains the same until a new value is assigned.

Here, what could be the reason for this ***printf*** output?

A screen shot of a computer

AI-generated content may be incorrect.

We can do the same with **echo, which is simpler.** However, echo can behave differently across shells. Meanwhile, printf **"%s\n"** is more predictable, most importantly in scripts.

Try with echo.

A black background with red and blue text

AI-generated content may be incorrect.

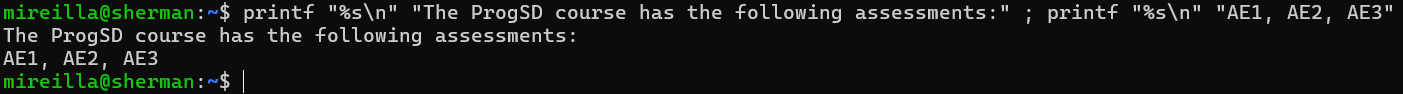
**Other examples of special formatting code. “**%d” represents an integer number and “%f” represents a floating-point number.

**A screen shot of a computer screen

AI-generated content may be incorrect.**

## Multiple Commands

You can combine multiple commands on a single line.

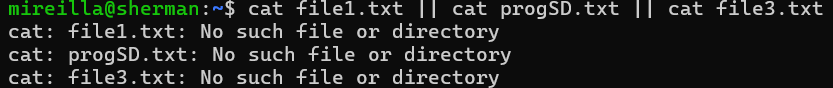


If a double ampersand separates each command (**&&**)(this is AND operator), the commands are executed until one of them fails or until the commands are executed. So the next command runs only if the previous one succeeds.



If each command is separated by a double vertical bar (**||**) **(OR operator)**, the commands are executed as long as each one fails until the commands are executed. In other words, it keeps trying until something works. So, execution stops at the first success.

In this example, none of the file exists.



A screen shot of a computer

AI-generated content may be incorrect.

Semicolons, double ampersands, and double vertical bars can be mixed in a single line. They are primarily intended as command-line shortcuts. When combined with redirection operators such as >, a long command chain can be difficult to read, and it is best avoided in scripts.

**; (semicolon):** run commands one after another, regardless of success/failure.

**&& (double ampersand):** Run the next command only if the previous succeeds.

**|| (double vertical bar):** Run the next command only if the previous one fails.

A screen shot of a computer

AI-generated content may be incorrect.