# Lab 1: WSL Setup

## Learning Outcomes

- Create a fresh installation of Linux
- Understand the concepts of the Windows Subsystem for Linux (WSL)
- Demonstrate an ability to use a Linux shell.
- Use the 'man' command to obtain documentation about Linux commands
- Explain how to list the contents of a directory in multiple forms.
- Navigate the Linux file system by changing directories.
- Manage the creation and deletion of new files and directories from within the command shell.

## Editing a Text File

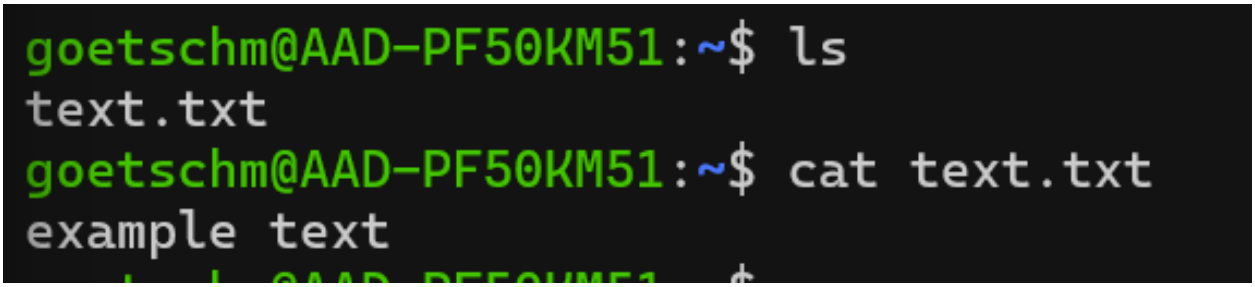


**Figure 1:** A screenshot of the commands used to print out the example file.

[1]

## Building and Running a Program

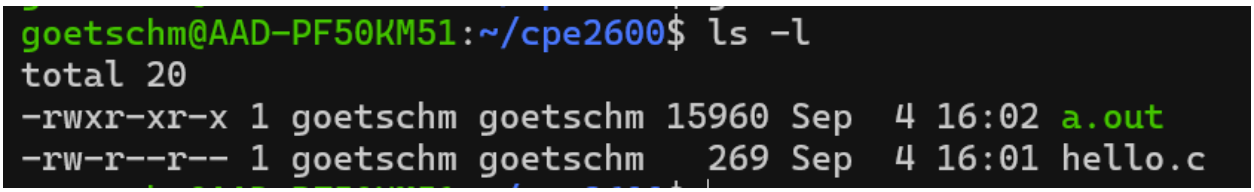


**Figure 2:** A long listing of the files in the directory that contains hello.c

The files in the directory are the c file `hello.c` and the compiled program `a.out`.[2]

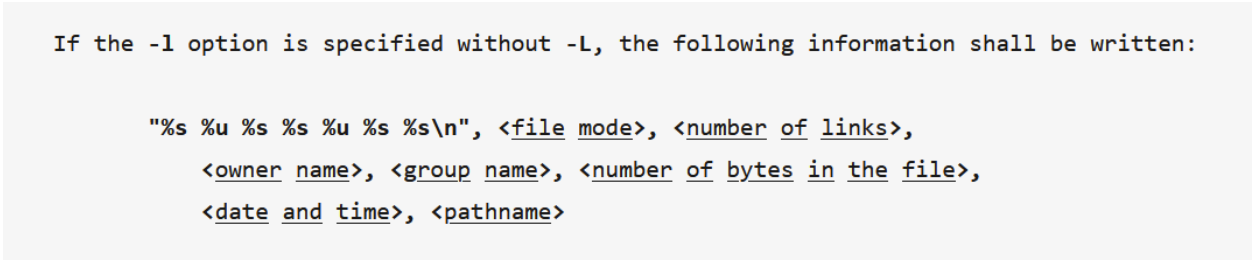


**Figure 3:** ubuntu docx explaining the output formatting

The file `a.out` is 15.96 KB, or 15,960 bytes. The file `hello.c` is 0.27 KB, or 269 bytes. The fourth column of the output lists the number of bytes in each file. [3]

[1] Capture a 'screenshot' of the command(s) to print out the file.
[2] What files do you see? Include a 'screenshot' of the listing output.
[3] How big is each file? How did you determine the size(s)?

`-o file`

Place the primary output in file *file*. This applies to whatever sort of output is being produced, whether it be an executable file, an object file, an assembler file or preprocessed C code.

If `-o` is not specified, the default is to put an executable file in `a.out`, the object file for *`source.suffix`* in *`source.o`*, its assembler file in *`source.s`*, a precompiled header file in *`source.suffix.gch`*, and all preprocessed C source on standard output.

**Figure 4:** -o command description from the GCC 14.2 manual

The compiler gcc is the GNU Compiler Collection. It was origiginally written as the compiler for the GNU operating system. The compiler default is to put the executable file in `a.out`, so the file `a.out` contains the executable generated from compiling `hello.c`. [4]


**Figure 5:** Generated compiler output execution

[5]

[4] Research the compiler gcc, what file contains the executable generated from compiling hello.c?

[5] Now run the generated compiler output. Include a 'screenshot' of the output in your submission.

6

```
goetschm@AAD-PF50KM51:~/cpe2600$ gcc hello.c -S
goetschm@AAD-PF50KM51:~/cpe2600$ ls -l
total 24
-rwxr-xr-x 1 goetschm goetschm 15960 Sep  4 16:02 a.out
-rw-r--r-- 1 goetschm goetschm   269 Sep  4 16:01 hello.c
-rw-r--r-- 1 goetschm goetschm   720 Sep  5 10:20 hello.s
goetschm@AAD-PF50KM51:~/cpe2600$ cat hello.s
        .file   "hello.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello world!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movq    %rsi, -16(%rbp)
        leaq    .LC0(%rip), %rax
        movq    %rax, %rdi
        call    puts@PLT
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long   1f - 0f
        .long   4f - 1f
        .long   5
0:
        .string "GNU"
1:
        .align 8
        .long   0xc0000002
        .long   3f - 2f
2:
        .long   0x3
3:
        .align 8
4:
```

**Figure 6:** Result of -S option

[6] Some options you should try: -S -c -Wall -Wextra -std=c89. Try each of these options and by observation and research determine how each option changes the behavior of gcc.

`-S`

Stop after the stage of compilation proper; do not assemble. The output is in the form of an assembler code file for each non-assembler input file specified.

By default, the assembler file name for a source file is made by replacing the suffix '`.c`', '`.i`', etc., with '`.s`'.

Input files that don't require compilation are ignored.

`-S` | This option stops the compiler before assembling. It creates a source file containing the assembly code for the program.

```
goetschm@AAD-PF50KM51:~/cpe2600$ gcc hello.c -c
goetschm@AAD-PF50KM51:~/cpe2600$ ls -l
total 28
-rwxr-xr-x 1 goetschm goetschm 15960 Sep  4 16:02 a.out
-rw-r--r-- 1 goetschm goetschm   269 Sep  4 16:01 hello.c
-rw-r--r-- 1 goetschm goetschm  1512 Sep  9 09:21 hello.o
-rw-r--r-- 1 goetschm goetschm   720 Sep  5 10:20 hello.s
goetschm@AAD-PF50KM51:~/cpe2600$ cat hello.o


◆◆UH◆◆H◆◆◆}◆H◆u◆H◆H◆◆◆◆◆◆Hello world!GCC: (Ubuntu 11.4.0-
1ubuntu1~22.04) 11.4.0GNU◆zRx
`                    )E◆C
◆◆      )hello.cmainputs◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆ .symtab.strtab.s
hstrtab.rela.text.data.bss.rodata.comment.note.GNU-stack.
note.gnu.property.rela.eh_fra◆e @)◆0
```

**Figure 7:** Result of -c option

`-c`

Compile or assemble the source files, but do not link. The linking stage simply is not done. The ultimate output is in the form of an object file for each source file.

By default, the object file name for a source file is made by replacing the suffix '`.c`', '`.i`', '`.s`', etc., with '`.o`'.

Unrecognized input files, not requiring compilation or assembly, are ignored.

`-c` | This option stops the compiler before linking. The result is an object file for the source file.

```
goetschm@AAD-PF50KM51:~/cpe2600$ gcc hello.c -Wall
goetschm@AAD-PF50KM51:~/cpe2600$
```

**Figure 8:** Result of -Wall option

`-Wall`

This enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros. This also enables some language-specific warnings described in Options Controlling C++ Dialect and Options Controlling Objective-C and Objective-C++ Dialects.

`-Wall`| This enables all warnings about constructions. The result from `hello.c` was nothing flagged.

```
goetschm@AAD-PF50KM51:~/cpe2600$ gcc hello.c -Wextra
goetschm@AAD-PF50KM51:~/cpe2600$
```

**Figure 9:** Result of -Wextra option

`-Wextra`

This enables some extra warning flags that are not enabled by `-Wall`. (This option used to be called `-W`. The older name is still supported, but the newer name is more descriptive.)

`-Wextra`| This enables additional warnings not enabled by `-Wall`. The result from `hello.c` was nothing flagged.

```
goetschm@AAD-PF50KM51:~/cpe2600$ gcc hello.c -std=c89
hello.c: In function ‘main’:
hello.c:9:1: error: C++ style comments are not allowed in
 ISO C90
    9 | // Print a message to the user
      | ^
hello.c:9:1: note: (this will be reported only once per i
nput file)
goetschm@AAD-PF50KM51:~/cpe2600$
```

**Figure 10:** Result of -std=c89 option

`-std=`

Determine the language standard. See Language Standards Supported by GCC, for details of these standard versions. This option is currently only supported when compiling C or C++.

‘`c90`’
‘`c89`’
‘`iso9899:1990`’

Support all ISO C90 programs (certain GNU extensions that conflict with ISO C90 are disabled). Same as `-ansi` for C code.

`-std=c89|` This sets the language standard to c89. The result of `hello.c` was not compiled because a comment conflicted with the set language standard.

*-o file*

Place the primary output in file *file*. This applies to whatever sort of output is being produced, whether it be an executable file, an object file, an assembler file or preprocessed C code.

If `-o` is not specified, the default is to put an executable file in `a.out`, the object file for *`source.suffix`* in *`source`*`.o`, its assembler file in *`source`*`.s`, a precompiled header file in *`source.suffix`*`.gch`, and all preprocessed C source on standard output.

Though `-o` names only the primary output, it also affects the naming of auxiliary and dump outputs. See the examples below. Unless overridden, both auxiliary outputs and dump outputs are placed in the same directory as the primary output. In auxiliary outputs, the suffix of the input file is replaced with that of the auxiliary output file type; in dump outputs, the suffix of the dump file is appended to the input file suffix. In compilation commands, the base name of both auxiliary and dump outputs is that of the primary output; in compile and link commands, the primary output name, minus the executable suffix, is combined with the input file name. If both share the same base name, disregarding the suffix, the result of the combination is that base name, otherwise, they are concatenated, separated by a dash.

```
gcc -c foo.c ...
```

will use `foo.o` as the primary output, and place aux outputs and dumps next to it, e.g., aux file `foo.dwo` for `-gsplit-dwarf`, and dump file `foo.c.???r.final` for `-fdump-rtl-final`.

If a non-linker output file is explicitly specified, aux and dump files by default take the same base name:

```
gcc -c foo.c -o dir/foobar.o ...
```

will name aux outputs `dir/foobar.*` and dump outputs `dir/foobar.c.*`.

A linker output will instead prefix aux and dump outputs:

```
gcc foo.c bar.c -o dir/foobar ...
```

will generally name aux outputs `dir/foobar-foo.*` and `dir/foobar-bar.*`, and dump outputs `dir/foobar-foo.c.*` and `dir/foobar-bar.c.*`.

The one exception to the above is when the executable shares the base name with the single input:

```
gcc foo.c -o dir/foo ...
```

in which case aux outputs are named `dir/foo.*` and dump outputs named `dir/foo.c.*`.

The location and the names of auxiliary and dump outputs can be adjusted by the options `-dumpbase`, `-dumpbase-ext`, `-dumpdir`, `-save-temps=cwd`, and `-save-temps=obj`.

**Figure 11:** docx entry for -o command

You can change the behaviour of the output of the executable by using the `-o` option. [7]

# Sources

- Ubuntu Docx
- GCC Manual 14.2.0

[7] By now you should know that the name of the executable gcc builds is a.out by default... How can you change this behavior?