CSC209

Pierre's Guide

Fall 2022

Contents

Chapter 1 **File Systems** Page 2

1.1 Basic Shell Commands 2

1.2 Permissions for files 2

Changing Permissions — 3 • Operations — 4

Chapter 2 **Shell Script and Unix** Page 5

2.1 Basic Unix Symbols 5

2.2 Shell Basics 5

Redirecting Standard Output/Input — 5 • Computing Expressions — 6

Chapter 3 **C Basics** Page 7

3.1 Input Output and Compiling 7

Printing Values — 7 • Compiling from the Command Line — 7 • Reading Input — 8

3.2 Types, Variables and Assignment Statements 8

Double Type — 8

3.3 Character Type 8

Chapter 1

File Systems

1.1 Basic Shell Commands

The basic shell commands are used to navigate around file systems and inspecting files. The basic ones are as follows:

- `man`: Provides a manual for any command followed by it.
- `cd`: Short for *change directory*. Allows navigation between the file system. Following attributes:
  - `cd ..`: Navigate to the parent directory.
- `ls`: Lists the names of files in the given directory.
  - `ls -l`: Gives more information (a *long* list). Includes protection information, file owner, number of characters in file, and date time of last change.
  - `ls -a`: Lists all files including hidden files.
  - `ls -F`: Succeeds all executable files with *.
  - `ls -g`: Like `-l` but does not show owner.
  - `ls -o`: Like `-l` but does not show group.
  - `ls -1`: Includes a line per entry in output.
  - `ls -s`: Includes the size of each entry.

Note:-

The options for `ls` can often be used together

1.2 Permissions for files

The file permissions are indicated at the beginning of a file listing when calling `ls -l`. The file listing follows this format:

```
<type><owner permissions><group permissions><other's permissions>
```
(1.1)

The possible values for each is the following:

- `type:`
  - *-:* indicates a regular file
  - *d:* indicates a directory

- `permissions:` The formatting for each permissions is formatted with three characters. The characters can be `-` or a character indicating the permissions
    1. If the first character is `r` it means that the permission **read** is granted.
    2. If the second character is `w` it means that the permissions **write** is granted.
    3. If the third character is `x` it means that the permission **executable** is granted.

Note:-

Directories are not allowed to have the executable permissions granted

1.2.1 Changing Permissions

The `chmod` command is used to change permissions on files. It follows the following format:

```
chmod <options> <mode> <file>
```
(1.2)

The mode defines the permissions to be changed and to what permission class. It follows the following format in this order:

1. `[ugoa]:` Each letter corresponds to the category of user the permissions should be changed for. Multiple can be used at once. These correlate with the following:
    - `u:` File owner
    - `g:` Users who are members of the group
    - `o:` All other users
    - `a:` All users, equivalent to `ugo`.
2. `[-+=]:` Defines wether the permissions should be removed, added or set. Only one can be used at a time:
    - `-` Removes specified permissions.
    - `+` Grants specified permissions.
    - `=` Changes the current permissions to the specified permissions. If some permissions aren't specified it means those permissions will not be granted. Such as if nothing is after the `=` then that category of user will be granted no permissions.
3. `perms:` These are the permissions which will be changed for the user. These use the same lettering system for permissions when displaying.
4. `[,...]:` The comma can be used to change multiple user classes permissions.

Example 1.2.1 (Using `chmod`)

```
$ chmod u=rwx,g=r,o= examplefilename
```

This would change the permissions for `examplefilename` such that the owner has read, write and executing permissions, the users in the group have read permissions, and the other users have no permissions.

Numeric Method of Permissions

Instead of writing out the permissions, a numeric method can also be used to communicate them. The method follows binary. Instead of a string of letters for permissions we instead convert three bits to a number. The bits correspond to the three main permissions, `rwx`, in their respective order

Example 1.2.2 (Using `chmod` with numeric permissions)

```
$ chmod 600 examplefilename
```

The following command sets the owner's permissions to `6` which is `110` in binary. Equivalent to granting `rw` access. The others are not granted permissions

Note:-

A fourth number at the beginning can be used to set `setui`, `setgid`, and `sticky bit` flags. However, this is more advanced and probably is not used in CSC209

1.2.2 Operations

Using Reference Files

The `--reference` option can be used to copy another files permissions. It follows the following format

```
$ chmod --reference=<reference_file> <file>
```

This code projects the permissions of the `reference_file` onto `file`.

Recursively Changing File Permissions

To operate on all files and directories under a given directory, the `-R` option can be used.

Changing File Permissions in Bulk

This can be done using the find command, which executes a command on different files in a directory depending on the type. This can be useful when some permission is meant to be granted to all directories but not regular files. The command follows this syntax:

```
$ find <path> -type <type> -exec chmod <options> <permissions> {} \;
```

As you can see the entry for file name is ommited from the `chmod` command since it is specified by `find`. The `<type>` can be `d` or `f`, directories and regular files respectively.

Chapter 2

Shell Script and Unix

The unix commands and shell script can be very unintuitive. It does not operate similarly like most programming languages do.

2.1 Basic Unix Symbols

- | Sends the output of a command to the input of the other. If `command1 | command2` is run, the output of `command1` will become the input of `command2`.
- > Redirects standard output.
- < Redirects standard input.
- >> Redirects and appends standard output.
- ; Separate commands on same line.
- () Group commands on the same line.
- / Separator in pathnames.
- ~ Home directory.
- ~<user> The home directory of user.
- . Present working directory.
- .. Parent directory.
- * Wildcard match of any number of characters in filename.
- ? Wildcard match of exactly one character in filename
- [] Wildcard match of any one character enclosed in brackets.
- & Process command in background.

2.2 Shell Basics

2.2.1 Redirecting Standard Output/Input

Redirection is a feature in Unix such that when executing a command, you can change the standard input/output devices.

Output Redirection

The > symbol is used to redirect output (STDOUT).

Example 2.2.1 (Output Redirection)

The line `ls -l > listing` would redirect the output of `ls -l` into a file named listing. This would not output it to the terminal like usual.

Note:-

`>>` can be used to avoid overwriting by adding to the end of the file.

Input Redirection

The `<` symbol is used to redirect input (STDIN).

2.2.2 Computing Expressions

Expressions can be computed using the `expr` command.

Example 2.2.2 (Basic Expressions)

The line, `$ ./<executable> $(expr $num1 + $num2)`, will input the sum of variables `num1` and `num2`

Chapter 3

C Basics

3.1 Input Output and Compiling

3.1.1 Printing Values

To print values we have to use the output stream. One way to do this is through the stdio.h library. This library is included in the code using the following line at the beginning of the file:

```
#include <stdio.h>
```
(3.1)

The function, `printf`, is used to print strings. String formatting is done by including some indicator in the string, usually in the form of a `%` followed by a letter to indicate the type. Then the variable to be inserted is included along the attributes:

- `d`: is used for formatting integer variables, `i`, can also be used. But `d` is more often used since it is *integer decimal*
- `f`: is used for floating-point numbers. Preceding `f` by a decimal point and an integer will indicate the number of values following the decimal point are to be printed. Example being `%.3f` will print whatever value with 3 decimal points.

Note:-

`printf` must be used with the correct number of format specifiers as arguments following the string

3.1.2 Compiling from the Command Line

To run C files we must first compile them. We compile files within the shell. Once we have navigated to the directory with the file we want to compile we run the `gcc` command, followed by the name of the C file we want to compile. This reads the file and creates an executable file, usually called `a.out`. To run `a.out` we enter `./a.out` into the shell. The following options can be used with `gcc`:

- `-Wall` This compiles the file and includes any warnings or errors encountered during compiling. This option should always be used to avoid issues in the future.
- `-o <name>` This defines a name for the executable file outputted by compiling. As explained previously, if this is not specified the outputted file is name `a.out`. The best practice is to always name the `out` file to be the same as the inputted.

If we want to output the data from a ran file into another file we can do the following, `./<executable> > <file to be outputted to> 2>&1`

3.1.3 Reading Input

To read input we can use the `scanf` function. This function requires the same io library as printing, `stdio.h`. `scanf` uses the same attribute list as `printf`, meaning we can also use text formatting with `%` and variables following the string.

Note:-

`scanf` uses `%lf`, long float, instead of `%f` when formatting floats.

In order to apply the inputted value to some variable we must provide the memory address of that variable. This is done with the `&` symbol. This allows `scanf` to place the number inputted into the variable.

Example 3.1.1 (Using `scanf`)

```
scanf("%lf %lf", &num1, &num2)
```

When executed, this line would require the user to provide two floating point numbers, separated by a space. Afterwards, the values inputted by the user would be stored in the variables `num1` and `num2`.

3.2 Types, Variables and Assignment Statements

3.2.1 Double Type

An important thing to note with double types is that it only converts the final result to a double. This opposes Python which also converts outputs of division to float.

Example 3.2.1 (Unclear Division)

The value of `double num` after the following line is executed, `num = 9/4;` is actually `2.0` instead of `2.5`. This is because division between integers does not automatically convert to a float type.

3.3 Character Type

The character type is specified with `char`. To define a character we can use the following:

- `'<character>'`: We can simply define a character by surrounding a single character with single quotations.
- Ascii: We can also simply provide an integer relating to some Ascii symbol:

Example 3.3.1

Running `char ch = 97`, makes `ch` equal to the chracter *a*.