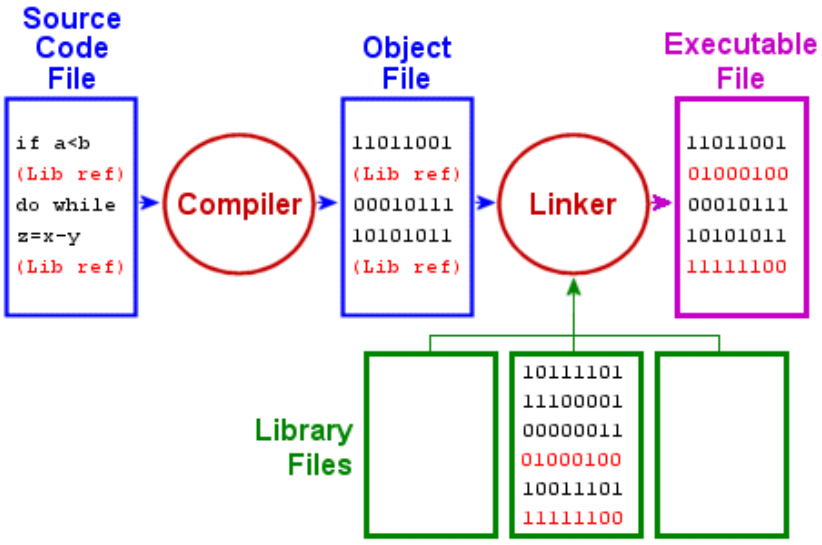
Ian Liu-Johnston
Nov 4 · 6 min read

Files to Find, Libraries to Compile

A guide to finding c source files with Bash and compiling them with gcc



Compilation process

I have a problem: there are all of these directories with all these ".c" files in them, and I have to find specific ones to compile into a static library. However, I'm lazy and I'd rather spend more time now to do less work later. I'm also allergic to tedium, so I'd rather do something that seems more complicated, so I don't get bored with a repetitive task like manually finding files and manually copying them over. That being said, here's how I figured out a way to use Bash commands to do that work for me.

First, I navigate to the root of my repository. `cd ~/holbertonschool-low_level_programming`

```
16:38~ vagrant:holbertonschool-low_level_programming$ls
0x00-hello_world                0x04-pointers_arrays_strings  0x08-static_libraries
0x01-variables_if_else_while    0x05-pointers_arrays_strings  0x09-argc_argv
0x02-functions_nested_loops     0x06-pointers_arrays_strings  README.md
0x03-more_functions_nested_loops  0x07-recursion
16:46~ vagrant:holbertonschool-low_level_programming$
```

I create a directory called “allCfiles” with a child directory called “libToBuild” `mkdir -p allCfiles/libToBuild` . `allCfiles` is created with `libToBuild` nested inside of it.

```
16:46~ vagrant:holbertonschool-low_level_programming$mkdir -p allCfiles/libToBuild
16:48~ vagrant:holbertonschool-low_level_programming$ls
0x00-hello_world                0x04-pointers_arrays_strings  0x08-static_libraries
0x01-variables_if_else_while    0x05-pointers_arrays_strings  0x09-argc_argv
0x02-functions_nested_loops     0x06-pointers_arrays_strings  allCfiles
0x03-more_functions_nested_loops  0x07-recursion              README.md
16:48~ vagrant:holbertonschool-low_level_programming$
```

I look for all files in my current directory and all sub-directories whose name ends in “.c” and execute a copy command from within `find` . I do this with the `-exec cp` flag. The `-exec` flag tells the find command to execute a command on each item found. The copy command copies each object found (denoted as `'{}'`) into the `allCfiles` directory. *note: the `{}` and `;` need to be escaped like this: `'{}'` and `\;` to escape interpretation by the shell.

```
find . -depth -type f -name "*.c" -exec cp '{}' allCfiles \;
```

The output of the command without `-exec cp '{}' allCfiles \;` looks like this:

```
16:53~ vagrant:holbertonschool-low_level_programming$
16:53~ vagrant:holbertonschool-low_level_programming$find . -depth -type f -name "*.c"
./0x00-hello_world/5-printf.c
./0x00-hello_world/test.c
./0x00-hello_world/101-quote.c
./0x00-hello_world/6-size.c
./0x00-hello_world/4-puts.c
./0x06-pointers_arrays_strings/main.9.c
./0x06-pointers_arrays_strings/5-strstr.c
./0x06-pointers_arrays_strings/main.8.c
```

`-exec cp '{}' allCfiles \;` operates on each file found in this manner and copies them into the directory `allCfiles` .

```
16:58~ vagrant:holbertonschool-low_level_programming$ls allCfiles/
0-holberton.c              10-print_comb2.c          4-print_alphabt.c       8-print_base16.c
0-isupper.c                10-print_triangle.c       4-print_most_numbers.c  8-print_diagsums.c
0-memset.c                 11-print_to_98.c          4-print_rev.c           8-print_square.c
0-positive_or_negative.c   1-alphabet.c              4-puts.c                8-rot13.c
0-puts_recursion.c         1-args.c                  4-rev_array.c           9-fizz_buzz.c
0-reset_to_98.c            1-isdigit.c               4-strpbrk.c             9-print_comb.c
0-strcat.c                 1-last_digit.c            5-more_numbers.c        9-set_string.c
0-whatsmyname.c            1-memcpy.c                5-printf.c              9-strcpy.c
100-atoi.c                 1-print_rev_recursion.c   5-print_numbers.c       9-times_table.c
100-change.c               1-strncat.c               5-rev_string.c          libToBuild
100-prime_factor.c         1-swap.c                  5-sign.c                main.0.c
100-print_comb3.c          2-args.c                  5-sqrt_recursion.c      main.1.c
```

List of the allCfiles directory

I copy-paste a list of files to find into an empty text file and call it files_to_find

```
julien@ubuntu:~/0x08. Static Librairies$ ar -t libholberton.a
0-isupper.o
0-memset.o
0-strcat.o
1-isdigit.o
1-memcpy.o
1-strncat.o
100-atoi.o
2-strchr.o
2-strlen.o
2-strncpy.o
3-islower.o
3-puts.o
3-strcmp.o
3-strspn.o
4-isalpha.o
4-strpbrk.o
5-strstr.o
6-abs.o
9-strcpy.o
_putchar.o
```

Because all of the lines in the file `files_to_find` end in `.o`, and all my files end in `.c`, I need to strip the `.o` off of each line. Why I do this is to use `grep` pattern matching. `grep "9-strcpy.o" files_to_find` will not match `9-strcpy.c`, and I will not get any results.

I get around this with sed: `sed -i 's/.o//g' files_to_find`. Sed is a **s**tream **ed**itor, and can manipulate text streams with regular expressions, as exemplified here `'s/.o//g'` This expression translates as “for all strings, if the pattern `.o` is found, replace it with nothing.”

The option `-i` for `sed` changes the output on the file itself, instead of printing it to the standard output. The last argument is the target file name.

The following example is the command `sed 's/.o//g' files_to_find` but it does not have the `-i` flag, so it will not edit the file directly.

```
16:58~ vagrant:holbertonschool-low_level_programming$vi files_to_find
17:02~ vagrant:holbertonschool-low_level_programming$cat files_to_find
0-isupper.o
0-memset.o
0-strcat.o
1-isdigit.o
1-memcpy.o
1-strncat.o
100-atoi.o
2-strchr.o
2-strlen.o
2-strncpy.o
3-islower.o
3-puts.o
3-strcmp.o
3-strspn.o
4-isalpha.o
4-strpbrk.o
5-strstr.o
6-abs.o
9-strcpy.o
_putchar.o
17:02~ vagrant:holbertonschool-low_level_programming$sed 's/.o//g' files_to_find
0-isupper
0-memset
0-strcat
1-isdigit
1-memcpy
1-strncat
100-ai
2-strchr
2-strlen
2-strncpy
3-iswer
3-puts
3-strcmp
3-strspn
4-isalpha
4-strpbrk
5-strstr
6-abs
9-strcpy
_putchar
17:03~ vagrant:holbertonschool-low_level_programming$
```

This enables grep to find `0-isupper.c` by using the pattern:

```
0-isupper
```

At this point, I am still in my repository root at `~/holbertonschool-low_level_programming` . The `files_to_find` file is in here with me, and the new directory I am populating is in a couple directories below me. `~/holbertonschool-low_level_programming/allCfiles/libToBuild`

I navigate into `allCfiles` with `cd allCfiles` and bring `files_to_find` with me: `mv ../files_to_find .` The double dot tells

mv that my file is in the parent directory, and I specify that I want to move it in my current directory with the `.` at the end of the command.

I call the copy command with a shell expansion containing my grep comparison commands to copy all of the matching files into the target directory : `cp $(ls -1 | grep -F -f files_to_find) libToBuild`

The expansion `ls -1 allCfiles | grep -F -f files_to_find` returns the following result

```
19:05~ vagrant:holbertonschool-low_level_programming$ls -1 allCfiles | grep -F -f files_to_find
0-isupper.c
0-memset.c
0-strcat.c
1-isdigit.c
1-memcpy.c
1-strncat.c
2-strchr.c
2-strlen.c
```

When combined with the `cp` command in a shell expansion `$()` , the command moves all of the files into the directory `allCfiles/libToBuild .`

Lastly, I navigate to the directory that contains all of my files:

```
cd libToBuild
```

A summary of the commands:

```
cd ~/holbertonschool-low_level_programming

mkdir -p allCfiles/libToBuild

find . -depth -type f -name "*.c" -exec cp '{}' allCfiles \;

sed -i 's/.o//g' files_to_find

cd allCfiles && mv ../files_to_find .

cp $(ls -1 | grep -F -f files_to_find) libToBuild

cd libToBuild
```

Finding Prototypes

Our next step is to create the header file for source files that require prototypes from other files. In this case, an essential function to my library is `_putchar.c` and some other functions require it.

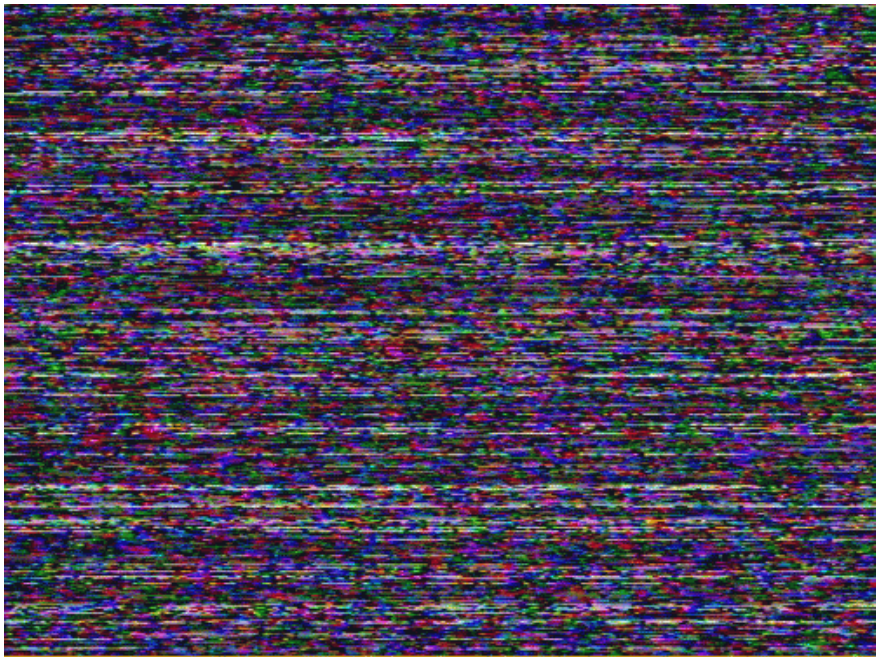
Lucky for me, I was provided with a list of prototypes.

```
int _putchar(char c);
int _islower(int c);
int _isalpha(int c);
int _abs(int n);
int _isupper(int c);
int _isdigit(int c);
int _strlen(char *s);
void _puts(char *s);
char *_strcpy(char *dest, char *src);
int _atoi(char *s);
char *_strcat(char *dest, char *src);
char *_strncat(char *dest, char *src, int n);
char *_strncpy(char *dest, char *src, int n);
int _strcmp(char *s1, char *s2);
char *_memset(char *s, char b, unsigned int n);
char *_memcpy(char *dest, char *src, unsigned int n);
char *_strchr(char *s, char c);
unsigned int _strspn(char *s, char *accept);
char *_strpbrk(char *s, char *accept);
char *_strstr(char *haystack, char *needle);
```

I copy-pasted this list into a text file and named it `holberton.h`

. . .

Compiling a Static Library



Now that we have our required files in our directory, and have generated out header file, it's time to compile them and turn them into a static library. To do that, we need `gcc` and `ar`

(if you want more clarification about these commands, run `man gcc` and `man ar`, though `man gcc` is very dense, as it contains many customization options for compilation.)

The commands are fairly simple, now that our files are in place. First, compile all c source code with `gcc -Wall -Werror -Wextra -pedantic -c *.c` Most of the flag options are for error checking, but the `-c` flag is important. It runs all c files in the current working directory through the preprocessor, then the assembler, and stops at the compiler. The files that are generated here end in `.o` and are binary files that contain machine code.

The final step is to link all of the machine code files together into a library, where all functions from all of the source files can be called from.

We do this with `ar -rc liball.a *.o`

Lets break this down. The command `ar` creates, modifies and

extracts from **ar**chives. The flag `-rc` tells `ar` to create a new archive, and append any new files. `liball.a` is the name for our new library. `*.o` tells `ar` to include all machine code files in the current working directory.

The output is the static library `liball.a` and can be used to compile another program. All custom functions are compiled to object code in one file and are portable, meaning you can move them between systems. To use this library when compiling another source file, use the command,

```
gcc main.c -L. -lall
```

The library is specified as a flag `-lall` This is short for the actual filename, which begins with lib###.a and ends with `.a` . Hence, `l` for lib and `all` for the name of the library.

— Happy Hacking!