

Makefiles

<https://www.cs.colostate.edu/~cs157/LectureMakefile.pdf>

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/7ab3e7de-106e-4e33-a10a-033bc12205a2/LectureMakefile.pdf>

1. How does Makefile work?
 - a. Makefile will look for the first rule and then check for the dependencies. It will then look if those dependencies have other dependencies. Once this is done, it will map it out as a tree and verify if the executable is in place. If no, it will compile the c files and produce the o files and final executable. If executable is already present, it will check if there are any updates to header or source files.
2. Why do I need to use the -c flag when giving the rules to compile an object file in Makefile?
 - a. This will tell the compiler to compile the c files.
3. The make file below compiled the program successfully with the -c flag (I was getting an error before that). But when I removed the flag, and run make, it says nothing to be done for all. Why?
 - a. Because I didn't make any changes to the source files.

```
1 cc = gcc
2 CFLAGS = -Wall -Wextra -Werror
3 DEPS = test.h
4 OBJFILES = hello.o sum.o div.o
5
6 all: libft
7
8 libft: $(OBJFILES)
9     $(cc) $(OBJFILES) -o libft
10
11 hello.o: hello.c $(DEPS)
12     $(cc) -c hello.c $(CFLAGS)
13
14 sum.o: sum.c $(DEPS)
15     $(cc) -c sum.c $(CFLAGS)
16
17 div.o: div.c $(DEPS)
18     $(cc) -c div.c $(CFLAGS)
```

```
ekeer@LAPTOP-SBKCNEP:~/hello$ make
gcc -c hello.c -Wall -Wextra -Werror
gcc -c sum.c -Wall -Wextra -Werror
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeer@LAPTOP-SBKCNEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o libft sum.c sum.o test.h
ekeer@LAPTOP-SBKCNEP:~/hello$ ./libft
3
2
ekeer@LAPTOP-SBKCNEP:~/hello$ make
make: Nothing to be done for 'all'.
ekeer@LAPTOP-SBKCNEP:~/hello$
```

4. It's important to include the header files into makefile so that whenever there are changes to the header files, the c files will be recompiled. Else it wouldn't be.
5. What are the advantages of having several dependencies? It makes it quicker to compile because it will only recompile the file that has been changed. If I put everything in one rule, it will rerun all files regardless of how many files changed.

The screenshot shows a VS Code editor with a Makefile open. The Makefile defines variables for compiler, flags, dependencies, and object files, and includes a rule for building a library (libft) from object files. The terminal on the right shows the output of running 'make' multiple times, demonstrating how the build system updates the library and then the final executable.

```

C hello.c x Makefile x C sum.c C div.c C test.h
Makefile
1 cc = gcc
2 CFLAGS = -Wall -Wextra -Werror
3 DEPS = test.h
4 OBJFILES = hello.o sum.o div.o
5
6 all: libft
7
8 libft: $(OBJFILES)
9 | $(cc) $(OBJFILES) $(CFLAGS) -o libft
10
11

```

```

ekeen@LAPTOP-BRHC3NEP:~/hello$ make
gcc -c hello.c -Wall -Wextra -Werror
gcc -c sum.c -Wall -Wextra -Werror
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-BRHC3NEP:~/hello$ ls
Makefile div.c div.o hello.c libft sum.c sum.o
ekeen@LAPTOP-BRHC3NEP:~/hello$ ./libft
3
2
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
make: Nothing to be done for 'all'.
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
make: Nothing to be done for 'all'.
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-BRHC3NEP:~/hello$ ./libft
3
some changes
2
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
gcc -c hello.c -Wall -Wextra -Werror
gcc -c sum.c -Wall -Wextra -Werror
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
ekeen@LAPTOP-BRHC3NEP:~/hello$ rm libft
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
gcc hello.c sum.c div.c -Wall -Wextra -Werror -o libft
ekeen@LAPTOP-BRHC3NEP:~/hello$ ./libft
3
some changes
2
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
gcc hello.c sum.c div.c -Wall -Wextra -Werror -o libft
ekeen@LAPTOP-BRHC3NEP:~/hello$

```

The screenshot shows a VS Code editor with a C source file (div.c) open. The source file contains a function 'sum' that prints the sum of two integers. The terminal on the right shows the output of running 'make' multiple times, demonstrating how the build system updates the final executable when the source file is modified.

```

C hello.c x Makefile C sum.c C div.c x C test.h
C div.c > sum(int,int)
1 #include "test.h"
2
3 int sum(int x, int y)
4 {
5     printf("%d\n", x + y);
6     printf("%s\n", "some changes here now");
7     return (0);
8 }
9

```

```

ekeen@LAPTOP-BRHC3NEP:~/hello$ make
gcc -c hello.c -Wall -Wextra -Werror
gcc -c sum.c -Wall -Wextra -Werror
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-BRHC3NEP:~/hello$ ls
Makefile div.c div.o hello.c libft sum.c sum.o
ekeen@LAPTOP-BRHC3NEP:~/hello$ ./libft
3
2
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
make: Nothing to be done for 'all'.
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
make: Nothing to be done for 'all'.
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-BRHC3NEP:~/hello$ ./libft
3
some changes
2
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
gcc -c hello.c -Wall -Wextra -Werror
gcc -c sum.c -Wall -Wextra -Werror
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
ekeen@LAPTOP-BRHC3NEP:~/hello$ rm libft
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
gcc hello.c sum.c div.c -Wall -Wextra -Werror -o libft
ekeen@LAPTOP-BRHC3NEP:~/hello$ ./libft
3
some changes
2
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
gcc hello.c sum.c div.c -Wall -Wextra -Werror -o libft
ekeen@LAPTOP-BRHC3NEP:~/hello$ make
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-BRHC3NEP:~/hello$

```

6. Even using the way this Makefile is written, it still only update the changed div.c file before relinking everything.


```

C sum.c > div(int, int)
1 #include "test.h"
2
3 int div(int x, int y)
4 {
5     printf("%d\n", (y / x));
6     return (0);
7 }
8

make: Nothing to be done for 'all'.
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-88HC3NEP:~/hello$ ./libft
3
some changes !
2
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c hello.c -Wall -Wextra -Werror
gcc -c sum.c -Wall -Wextra -Werror
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-88HC3NEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o libft sum.c sum.o test.h
ekeen@LAPTOP-88HC3NEP:~/hello$ rm libft
ekeen@LAPTOP-88HC3NEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o sum.c sum.o test.h
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-88HC3NEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o libft sum.c sum.o test.h
ekeen@LAPTOP-88HC3NEP:~/hello$ ./libft
3
some changes !
2
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c %.c -Wall -Wextra -Werror
gcc: error: %.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
make: *** [Makefile:12: sum.o] Error 1
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c -Wall -Wextra -Werror
gcc: fatal error: no input files
compilation terminated.
make: *** [Makefile:12: sum.o] Error 1
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c -Wall -Wextra -Werror
gcc: fatal error: no input files
compilation terminated.
make: *** [Makefile:13: sum.o] Error 1
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-88HC3NEP:~/hello$ rm libft
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-88HC3NEP:~/hello$ ./libft
3
new changes!
2ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-88HC3NEP:~/hello$

```

```

Makefile
1 cc = gcc
2 CFLAGS = -Wall -Wextra -Werror
3 DEPS = test.h
4 OBJFILES = hello.o sum.o div.o
5 SRCSFILES = hello.c sum.c div.c
6
7 all: libft
8
9 libft: $(OBJFILES)
10     $(cc) $(OBJFILES) -o libft
11
12 %.o: %.c $(DEPS)
13     $(cc) -c $(SRCSFILES) $(CFLAGS)
14
15

make: Nothing to be done for 'all'.
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-88HC3NEP:~/hello$ ./libft
3
some changes !
2
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c hello.c -Wall -Wextra -Werror
gcc -c sum.c -Wall -Wextra -Werror
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-88HC3NEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o libft sum.c sum.o test.h
ekeen@LAPTOP-88HC3NEP:~/hello$ rm libft
ekeen@LAPTOP-88HC3NEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o sum.c sum.o test.h
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-88HC3NEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o libft sum.c sum.o test.h
ekeen@LAPTOP-88HC3NEP:~/hello$ ./libft
3
some changes !
2
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c %.c -Wall -Wextra -Werror
gcc: error: %.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
make: *** [Makefile:12: sum.o] Error 1
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c -Wall -Wextra -Werror
gcc: fatal error: no input files
compilation terminated.
make: *** [Makefile:12: sum.o] Error 1
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c -Wall -Wextra -Werror
gcc: fatal error: no input files
compilation terminated.
make: *** [Makefile:13: sum.o] Error 1
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-88HC3NEP:~/hello$ rm libft
ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-88HC3NEP:~/hello$ ./libft
3
new changes!
2ekeen@LAPTOP-88HC3NEP:~/hello$ make
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-88HC3NEP:~/hello$

```

9. How make clean works (with `rm -f`) and what happens when I recompile? Clean removed the `.o` files forcibly and without `rm` prompting for permission with the `-f` flag

(if file doesn't permit writing, -f will ignore this and remove the file), -f also ignores non-existent files. When I used make, it will use the c files (gcc -c directive) to generate the output files.

The screenshot shows a VS Code editor with a Makefile open. The Makefile contains the following content:


```
1 cc = gcc
2 CFLAGS = -Wall -Wextra -Werror
3 NAME = libft
4 DEPS = test.h
5 SRC = hello.c sum.c div.c
6 OBJ = $(SRC:.c=.o) #substitution references transforms the contents of the src variable, changing all file suffix
7
8 all: $(NAME)
9
10 $(NAME): $(OBJ)
11     $(cc) $(OBJ) -o libft
12
13 %.o: %.c $(DEPS)
14     $(cc) -c $(SRC) $(CFLAGS)
15
16 fclean: clean
17     rm -f $(NAME)
18
19 clean:
20     rm -f $(OBJ)
21
22 re: fclean all
23
24 .PHONY: all clean fclean re #to tell make that a target is not a real file. Because without this, if a target the
```

The terminal on the right shows the execution of the Makefile:

```
ekleen@LAPTOP-888K3NEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o libft sum.c sum.o test.h
ekleen@LAPTOP-888K3NEP:~/hello$ make clean
rm -f hello.o sum.o div.o
ekleen@LAPTOP-888K3NEP:~/hello$ ls
Makefile div.c hello.c libft sum.c test.h
ekleen@LAPTOP-888K3NEP:~/hello$ make
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekleen@LAPTOP-888K3NEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o libft sum.c sum.o test.h
ekleen@LAPTOP-888K3NEP:~/hello$ make re
rm -f libft
rm -f libft
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekleen@LAPTOP-888K3NEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o libft sum.c sum.o test.h
ekleen@LAPTOP-888K3NEP:~/hello$
```

Makefile : fclean gives error if program is not yet built

Thanks for contributing an answer to Stack Overflow! Please be sure to answer the question. Provide details and share your research! Asking for help, clarification, or responding to other

 <https://stackoverflow.com/questions/30576357/makefile-fclean-gives-error-if-program-is-not-yet-built>



What Actually "rm -rf" Command Do in Linux?

The rm command is a UNIX and Linux command line utility for removing files or directories on a Linux system. In this article, we will clearly explain what actually "rm -rf" command can do in Linux.

 <https://www.tecmint.com/linux-rm-command-examples/>

What Actually "rm -rf" Command Do in Linux?

```
tecmint@tecmint: ~/tecmint_files $ touch tecmint.txt foasmint.txt
tecmint@tecmint: ~/tecmint_files $ ls -l
total 0
-rw-r--r-- 1 tecmint tecmint 0 Jul 6 12:21 foasmint.txt
-rw-r--r-- 1 tecmint tecmint 0 Jul 6 12:21 tecmint.txt
tecmint@tecmint: ~/tecmint_files $ rm -rf tecmint.txt
tecmint@tecmint: ~/tecmint_files $ ls
foasmint.txt tecmint
tecmint@tecmint: ~/tecmint_files $ rm -rf foasmint.txt
tecmint@tecmint: ~/tecmint_files $ ls
tecmint@tecmint: ~/tecmint_files $
```

 TecMint

- Once I removed the .o files, does it mean that the libft executable is corrupted?
Nope, the libft still works because it's a separate file used in a separate process.

```

1 cc = gcc
2 CFLAGS = -Wall -Wextra -Werror
3 NAME = libft
4 DEPS = test.h
5 SRC = hello.c sum.c div.c
6 OBJ = $(SRC:.c=.o) #substitution references transforms the contents of the src variable, changing all file suffix
7
8 all: $(NAME)
9
10 $(NAME): $(OBJ)
11     $(cc) $(OBJ) -o libft
12
13 %.o: %.c $(DEPS)
14     $(cc) -c $(SRC) $(CFLAGS)
15
16 fclean: clean
17     rm -f $(NAME)
18
19 clean:
20     rm -f $(OBJ)
21
22 re: fclean all
23
24 .PHONY: all clean fclean re #to tell make that a target is not a real file. Because without this, if a target tha
25

```

```

ekeen@LAPTOP-889K3NEP:~/hello$ ./libft
3
new changes tests tada tada!
2
ekeen@LAPTOP-889K3NEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o libft sum.c sum.o test.h
ekeen@LAPTOP-889K3NEP:~/hello$ make clean
rm -f hello.o sum.o div.o
ekeen@LAPTOP-889K3NEP:~/hello$ ls
Makefile div.c hello.c libft sum.c test.h
ekeen@LAPTOP-889K3NEP:~/hello$ ./libft
3
new changes tests tada tada!
2
ekeen@LAPTOP-889K3NEP:~/hello$

```

11. After fclean, make will generate object files from the c files.

```

1 cc = gcc
2 CFLAGS = -Wall -Wextra -Werror
3 NAME = libft
4 DEPS = test.h
5 SRC = hello.c sum.c div.c
6 OBJ = $(SRC:.c=.o) #substitution references transforms the contents of the src variable, changing all file suffix
7
8 all: $(NAME)
9
10 $(NAME): $(OBJ)
11     $(cc) $(OBJ) -o libft
12
13 %.o: %.c $(DEPS)
14     $(cc) -c $(SRC) $(CFLAGS)
15
16 fclean: clean
17     rm -f $(NAME)
18
19 clean:
20     rm -f $(OBJ)
21
22 re: fclean all
23
24 .PHONY: all clean fclean re #to tell make that a target is not a real file. Because without this, if a target tha
25

```

```

ekeen@LAPTOP-889K3NEP:~/hello$ ./libft
3
new changes tests tada tada!
2
ekeen@LAPTOP-889K3NEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o libft sum.c sum.o test.h
ekeen@LAPTOP-889K3NEP:~/hello$ make clean
rm -f hello.o sum.o div.o
ekeen@LAPTOP-889K3NEP:~/hello$ ls
Makefile div.c hello.c libft sum.c test.h
ekeen@LAPTOP-889K3NEP:~/hello$ ./libft
3
new changes tests tada tada!
2
ekeen@LAPTOP-889K3NEP:~/hello$ ls
Makefile div.c hello.c libft sum.c test.h
ekeen@LAPTOP-889K3NEP:~/hello$ make
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-889K3NEP:~/hello$ make
make: Nothing to be done for 'all'.
ekeen@LAPTOP-889K3NEP:~/hello$ ls
Makefile div.c div.o hello.c hello.o libft sum.c sum.o test.h
ekeen@LAPTOP-889K3NEP:~/hello$ make fclean
rm -f hello.o sum.o div.o
rm -f libft
ekeen@LAPTOP-889K3NEP:~/hello$ ls
Makefile div.c hello.c sum.c test.h
ekeen@LAPTOP-889K3NEP:~/hello$ make
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-889K3NEP:~/hello$

```

12. Why do I need to include the libft.a together when compiling it with my something.c which is using the header test.h (but something.c isn't part of the Makefile)?

The screenshot shows a Visual Studio Code editor with a C program named `something.c` and its compilation process in the terminal. The program is a simple function `div(2, 4)` that returns 2. The terminal shows the compilation steps: creating object files (`hello.o`, `sum.o`, `div.o`) and linking them into a library (`libft.a`) using the `ar` command. The `Makefile` is also visible in the Explorer pane.

```

1 #include "test.h"
2
3 int main(void)
4 {
5     div(2, 4);
6 }
7

```

```

div.c  hello.c  libft.a  sum.o
ekeen@LAPTOP-8RHC3NEP:~/hello$ ar -t libft.a
hello.o
sum.o
div.o
ekeen@LAPTOP-8RHC3NEP:~/hello$ vim something.c
ekeen@LAPTOP-8RHC3NEP:~/hello$ gcc something.c
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./a.out
ekeen@LAPTOP-8RHC3NEP:~/hello$ ls
Makefile  div.c  hello.c  libft.a  sum.c  test.h
a.out  div.o  hello.o  something.c  sum.o
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./a.out
ekeen@LAPTOP-8RHC3NEP:~/hello$ rm a.out
ekeen@LAPTOP-8RHC3NEP:~/hello$ make fclean
rm -f hello.o sum.o div.o
rm -f libft.a
ekeen@LAPTOP-8RHC3NEP:~/hello$ ls
Makefile  div.c  hello.c  something.c  sum.c  test.h
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
ar -csr libft.a hello.o sum.o div.o
ekeen@LAPTOP-8RHC3NEP:~/hello$ gcc something.c
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./a.out
ekeen@LAPTOP-8RHC3NEP:~/hello$ rm a.out
ekeen@LAPTOP-8RHC3NEP:~/hello$ make fclean
rm -f hello.o sum.o div.o
rm -f libft.a
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
ar -csr libft.a hello.o sum.o div.o
ekeen@LAPTOP-8RHC3NEP:~/hello$ gcc something.c
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./a.out
ekeen@LAPTOP-8RHC3NEP:~/hello$ gcc something.c libft.a
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./a.out
2
ekeen@LAPTOP-8RHC3NEP:~/hello$

```

- When using `ar` without the `%.o: %.c` rule, I can just modify any file and it will recompile only the modified file (in this example, it's `sum.c`). So how does `ar` actually work? How substitution reference produce the `.o` files?

The screenshot shows the `Makefile` and its execution in the terminal. The `Makefile` defines the compilation rules for the program. The terminal shows the execution of `make`, which compiles the source files and links them into a library (`libft.a`) using the `ar` command. The `Makefile` also defines the `fclean` and `clean` targets to remove the generated files.

```

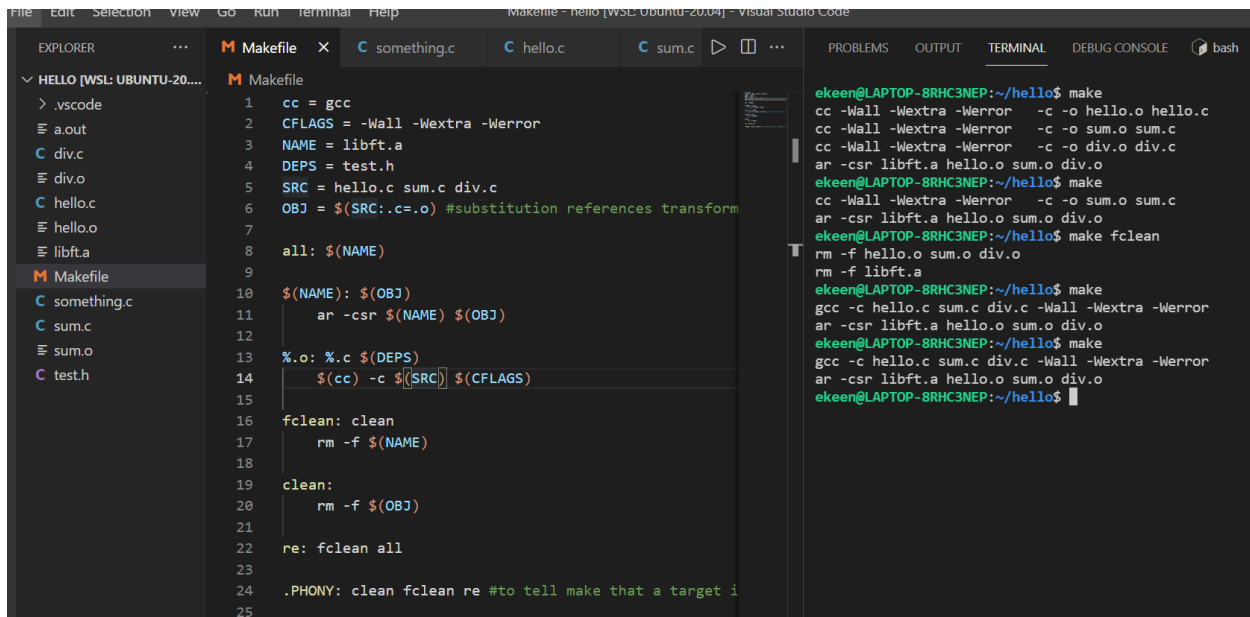
1 cc = gcc
2 CFLAGS = -Wall -Wextra -Werror
3 NAME = libft.a
4 DEPS = test.h
5 SRC = hello.c sum.c div.c
6 OBJ = $(SRC:.c=.o) #substitution references transform
7
8 all: $(NAME)
9
10 $(NAME): $(OBJ)
11     ar -csr $(NAME) $(OBJ)
12
13 fclean: clean
14     rm -f $(NAME)
15
16 clean:
17     rm -f $(OBJ)
18
19 re: fclean all
20
21 .PHONY: clean fclean re #to tell make that a target is
22

```

```

ekeen@LAPTOP-8RHC3NEP:~/hello$ make
cc -Wall -Wextra -Werror -c -o hello.o hello.c
cc -Wall -Wextra -Werror -c -o sum.o sum.c
cc -Wall -Wextra -Werror -c -o div.o div.c
ar -csr libft.a hello.o sum.o div.o
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
cc -Wall -Wextra -Werror -c -o sum.o sum.c
ar -csr libft.a hello.o sum.o div.o
ekeen@LAPTOP-8RHC3NEP:~/hello$

```

The screenshot shows the Visual Studio Code editor with a project named 'HELLO [WSL: UBUNTU-20...]' open. The Explorer sidebar on the left shows files: .vscode, a.out, div.c, div.o, hello.c, hello.o, libft.a, Makefile, something.c, sum.c, sum.o, and test.h. The Makefile editor is active, showing the following content:

```
1 cc = gcc
2 CFLAGS = -Wall -Wextra -Werror
3 NAME = libft.a
4 DEPS = test.h
5 SRC = hello.c sum.c div.c
6 OBJ = $(SRC:.c=.o) #substitution references transform
7
8 all: $(NAME)
9
10 $(NAME): $(OBJ)
11     ar -csr $(NAME) $(OBJ)
12
13 %.o: %.c $(DEPS)
14     $(cc) -c $(SRC) $(CFLAGS)
15
16 fclean: clean
17     rm -f $(NAME)
18
19 clean:
20     rm -f $(OBJ)
21
22 re: fclean all
23
24 .PHONY: clean fclean re #to tell make that a target i
25
```


The terminal on the right shows the execution of the Makefile:

```
keen@LAPTOP-8RHC3NEP:~/hello$ make
cc -Wall -Wextra -Werror -c -o hello.o hello.c
cc -Wall -Wextra -Werror -c -o sum.o sum.c
cc -Wall -Wextra -Werror -c -o div.o div.c
ar -csr libft.a hello.o sum.o div.o
keen@LAPTOP-8RHC3NEP:~/hello$ make
cc -Wall -Wextra -Werror -c -o sum.o sum.c
ar -csr libft.a hello.o sum.o div.o
keen@LAPTOP-8RHC3NEP:~/hello$ make fclean
rm -f hello.o sum.o div.o
rm -f libft.a
keen@LAPTOP-8RHC3NEP:~/hello$ make
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
ar -csr libft.a hello.o sum.o div.o
keen@LAPTOP-8RHC3NEP:~/hello$ make
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
ar -csr libft.a hello.o sum.o div.o
keen@LAPTOP-8RHC3NEP:~/hello$
```

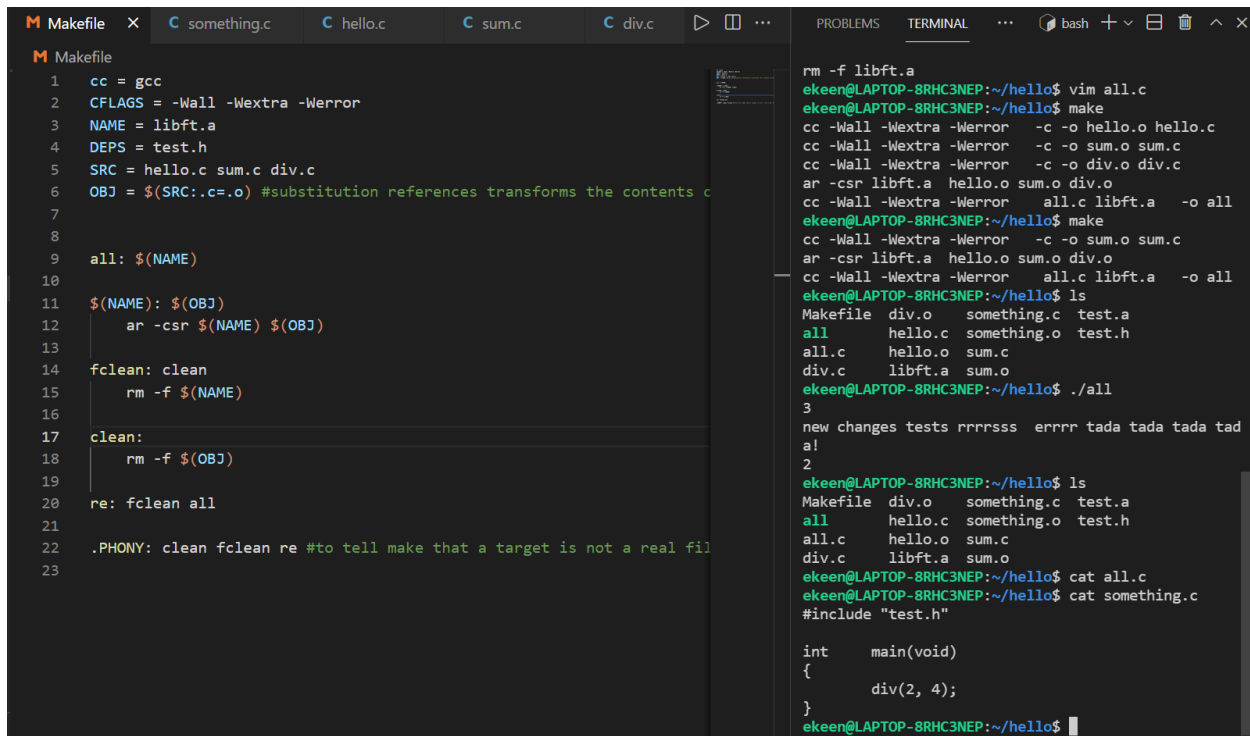
14. Here, I excluded "all", a pseudo target (doesn't exist as a file) out of .PHONY and create an actual all.c file. What happened was that the all.c file got compiled with libft.a to produce an executable file all (but all.c is an empty file, only something.c has included test.h header). A good practice is to include all targets that aren't dependent on actual files in .PHONY to avoid unintended behavior (Make is assuming that it's a file, hence it will always recompile the file even if nothing has changed) and save time during compilation.

How do I make Makefile to recompile only changed files?

Not sure if this is causing your specific problem but the two lines:
a_functions.c: a.h main.c: main.h are definitely wrong, because
there's generally no command to re-create a C file based on a

 <https://stackoverflow.com/questions/7815400/how-do-i-make-makefile-to-recompile-only-changed-files>



The image shows a code editor with a Makefile on the left and a terminal on the right. The Makefile defines variables for compiler, flags, name, dependencies, source files, and object files. It includes rules for building the library, cleaning, and running tests. The terminal shows the execution of these rules, including compilation commands and file listings.

```
M Makefile x something.c hello.c sum.c div.c
M Makefile
1 cc = gcc
2 CFLAGS = -Wall -Wextra -Werror
3 NAME = libft.a
4 DEPS = test.h
5 SRC = hello.c sum.c div.c
6 OBJ = $(SRC:.c=.o) #substitution references transforms the contents of
7
8
9 all: $(NAME)
10
11 $(NAME): $(OBJ)
12     ar -csr $(NAME) $(OBJ)
13
14 fclean: clean
15     rm -f $(NAME)
16
17 clean:
18     rm -f $(OBJ)
19
20 re: fclean all
21
22 .PHONY: clean fclean re #to tell make that a target is not a real file
23

rm -f libft.a
ekeen@LAPTOP-8RHC3NEP:~/hello$ vim all.c
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
cc -Wall -Wextra -Werror -c -o hello.o hello.c
cc -Wall -Wextra -Werror -c -o sum.o sum.c
cc -Wall -Wextra -Werror -c -o div.o div.c
ar -csr libft.a hello.o sum.o div.o
cc -Wall -Wextra -Werror all.c libft.a -o all
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
cc -Wall -Wextra -Werror -c -o sum.o sum.c
ar -csr libft.a hello.o sum.o div.o
cc -Wall -Wextra -Werror all.c libft.a -o all
ekeen@LAPTOP-8RHC3NEP:~/hello$ ls
Makefile  div.o  something.c  test.a
all       hello.c  something.o  test.h
all.c     hello.o  sum.c
div.c     libft.a  sum.o
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./all
3
new changes tests rrrrrss errrr tada tada tada tad
a!
2
ekeen@LAPTOP-8RHC3NEP:~/hello$ ls
Makefile  div.o  something.c  test.a
all       hello.c  something.o  test.h
all.c     hello.o  sum.c
div.c     libft.a  sum.o
ekeen@LAPTOP-8RHC3NEP:~/hello$ cat all.c
ekeen@LAPTOP-8RHC3NEP:~/hello$ cat something.c
#include "test.h"

int main(void)
{
    div(2, 4);
}
```

15. Looks like CFLAGS operate using some sort of implicit rule because when I commented it out, the compilation flags wasn't included. CFLAGS is one of Make's predefined macro. Similarly, I can just specify the dependencies without the recipe and make will use implicit rule to compile it.

```

Makefile
1 cc = gcc
2 #CFLAGS = -Wall -Wextra -Werror
3 NAME = libft.a
4 DEPS = test.h
5 SRC = hello.c sum.c div.c
6 OBJ = $(SRC:.c=.o) #substitution references transforms the contents c
7
8
9 all: $(NAME)
10
11 $(NAME): $(OBJ)
12     ar -csr $(NAME) $(OBJ)
13
14 fclean: clean
15     rm -f $(NAME)
16
17 clean:
18     rm -f $(OBJ)
19
20 re: fclean all
21
22 .PHONY: all clean fclean re #to tell make that a target is not a real
23

```

```

ekeen@LAPTOP-8RHC3NEP:~/hello$ make fclean
rm -f hello.o sum.o div.o
rm -f libft.a
ekeen@LAPTOP-8RHC3NEP:~/hello$ make all
cc -Wall -Wextra -Werror -c -o hello.o hello.c
cc -Wall -Wextra -Werror -c -o sum.o sum.c
cc -Wall -Wextra -Werror -c -o div.o div.c
ar -csr libft.a hello.o sum.o div.o
cc -Wall -Wextra -Werror all.c libft.a -o all
ekeen@LAPTOP-8RHC3NEP:~/hello$ make all
cc -Wall -Wextra -Werror -c -o sum.o sum.c
ar -csr libft.a hello.o sum.o div.o
cc -Wall -Wextra -Werror all.c libft.a -o all
ekeen@LAPTOP-8RHC3NEP:~/hello$ make all
make: 'all' is up to date.
ekeen@LAPTOP-8RHC3NEP:~/hello$ make fclean
rm -f hello.o sum.o div.o
rm -f libft.a
ekeen@LAPTOP-8RHC3NEP:~/hello$ rm all
all all.c
ekeen@LAPTOP-8RHC3NEP:~/hello$ rm alls
rm: cannot remove 'alls': No such file or director
y
ekeen@LAPTOP-8RHC3NEP:~/hello$ ls
Makefile all.c hello.c sum.c
all div.c something.c test.h
ekeen@LAPTOP-8RHC3NEP:~/hello$ rm all
ekeen@LAPTOP-8RHC3NEP:~/hello$ ls
Makefile div.c something.c test.h
all.c hello.c sum.c
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
cc -c -o hello.o hello.c
cc -c -o sum.o sum.c
cc -c -o div.o div.c
ar -csr libft.a hello.o sum.o div.o
ekeen@LAPTOP-8RHC3NEP:~/hello$

```

```

Makefile
5 #
6 # By: ekeen-wy <ekeen-wy@student.42kl.edu.my>
7 #
8 # Created: 2021/10/14 15:20:32 by ekeen-wy
9 # Updated: 2021/10/14 15:54:18 by ekeen-wy
10 #
11 # *****
12
13 cc = gcc
14 CFLAGS = -Wall -Wextra -Werror
15 NAME = libft.a
16 DEPS = test.h
17 SRC = hello.c sum.c div.c
18 OBJ = $(SRC:.c=.o) #substitution references transforms the contents c
19
20 all: $(NAME)
21
22 $(NAME): $(OBJ)
23     ar -csr $(NAME) $(OBJ)
24
25 $(OBJ): $(DEPS)
26
27 fclean: clean
28     rm -f $(NAME)
29
30 clean:
31     rm -f $(OBJ)
32
33 re: fclean all
34

```

```

ar -csr libft.a hello.o sum.o div.o
ekeen@LAPTOP-8RHC3NEP:~/hello$ make fclean
rm -f hello.o sum.o div.o
rm -f libft.a
ekeen@LAPTOP-8RHC3NEP:~/hello$ ls
Makefile hello.c sum.c
div.c something.c test.h
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
cc -Wall -Wextra -Werror -c -o hello.o hello.c
cc -Wall -Wextra -Werror -c -o sum.o sum.c
cc -Wall -Wextra -Werror -c -o div.o div.c
ar -csr libft.a hello.o sum.o div.o
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
cc -Wall -Wextra -Werror -c -o sum.o sum.c
ar -csr libft.a hello.o sum.o div.o
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
make: Nothing to be done for 'all'.
ekeen@LAPTOP-8RHC3NEP:~/hello$ make fclean
rm -f hello.o sum.o div.o
rm -f libft.a
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
cc -Wall -Wextra -Werror -c -o hello.o hello.c
cc -Wall -Wextra -Werror -c -o sum.o sum.c
cc -Wall -Wextra -Werror -c -o div.o div.c
ar -csr libft.a hello.o sum.o div.o
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
cc -Wall -Wextra -Werror -c -o hello.o hello.c
cc -Wall -Wextra -Werror -c -o sum.o sum.c
cc -Wall -Wextra -Werror -c -o div.o div.c
ar -csr libft.a hello.o sum.o div.o
ekeen@LAPTOP-8RHC3NEP:~/hello$

```

<https://unix.stackexchange.com/questions/26409/how-to-avoid-make-redoing-a-library>

Check the link above on using ar utility

[Makefiles — EPITECH 2022 - Technical Documentation 1.3.38 documentation](#)