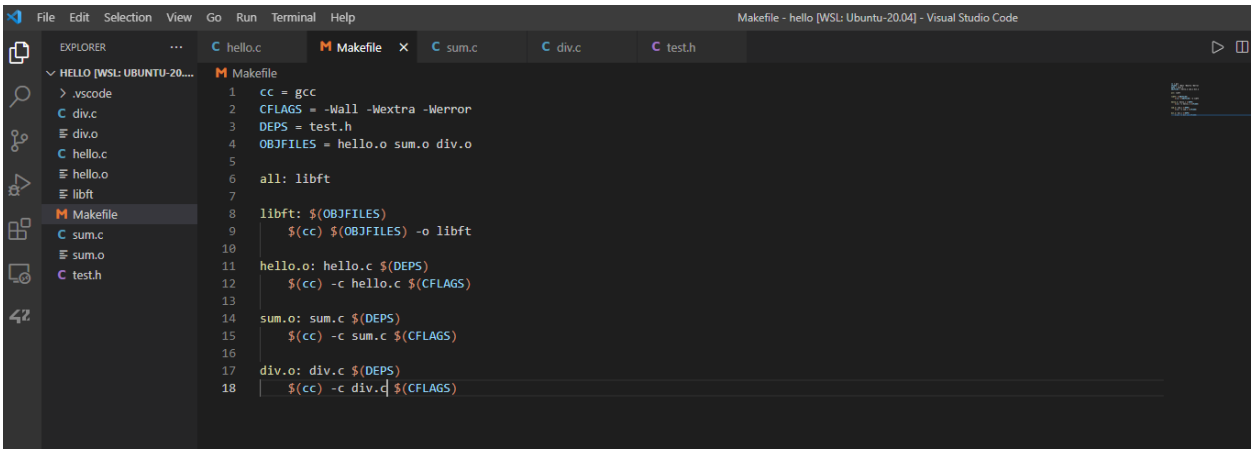
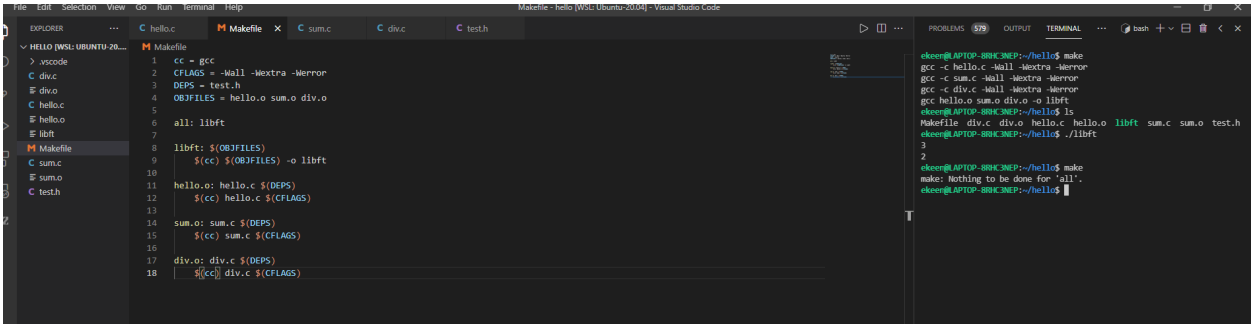
Makefiles

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

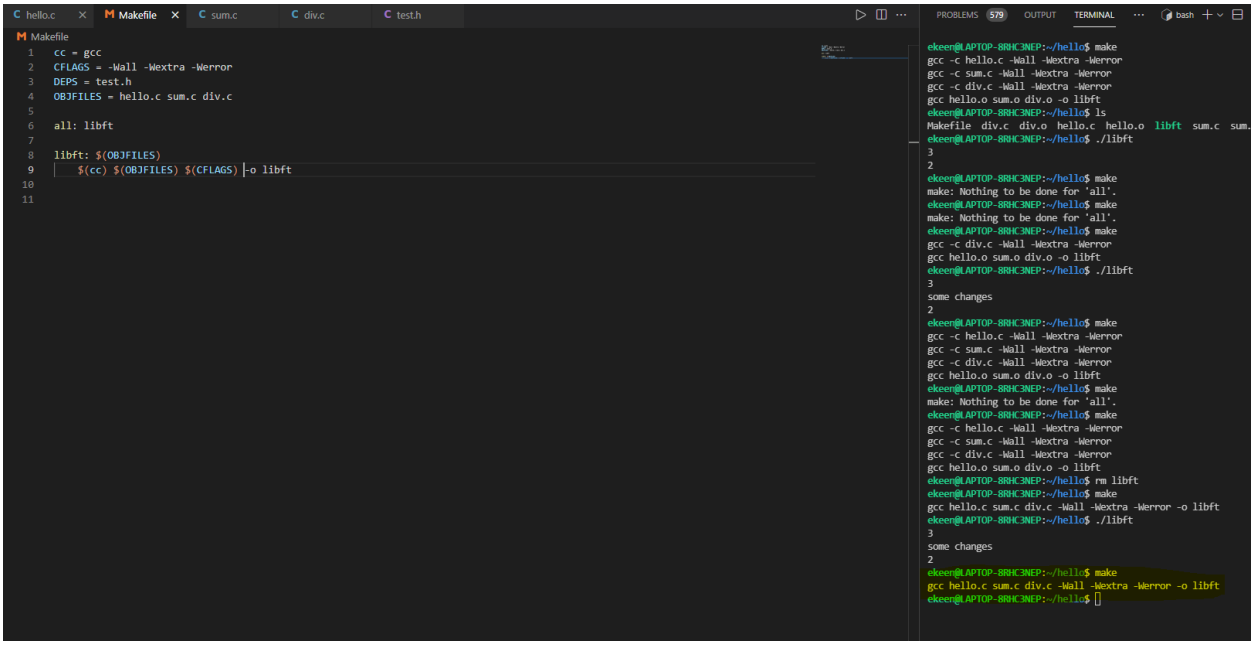
1. How does Makefile work?
    a. Makefile will look for the first rule and then check for the dependencies. It will then look if those dependencies have other dependencies. Once this is done, it will map it out as a tree and verify if the executable is in place. If no, it will compile the c files and 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.

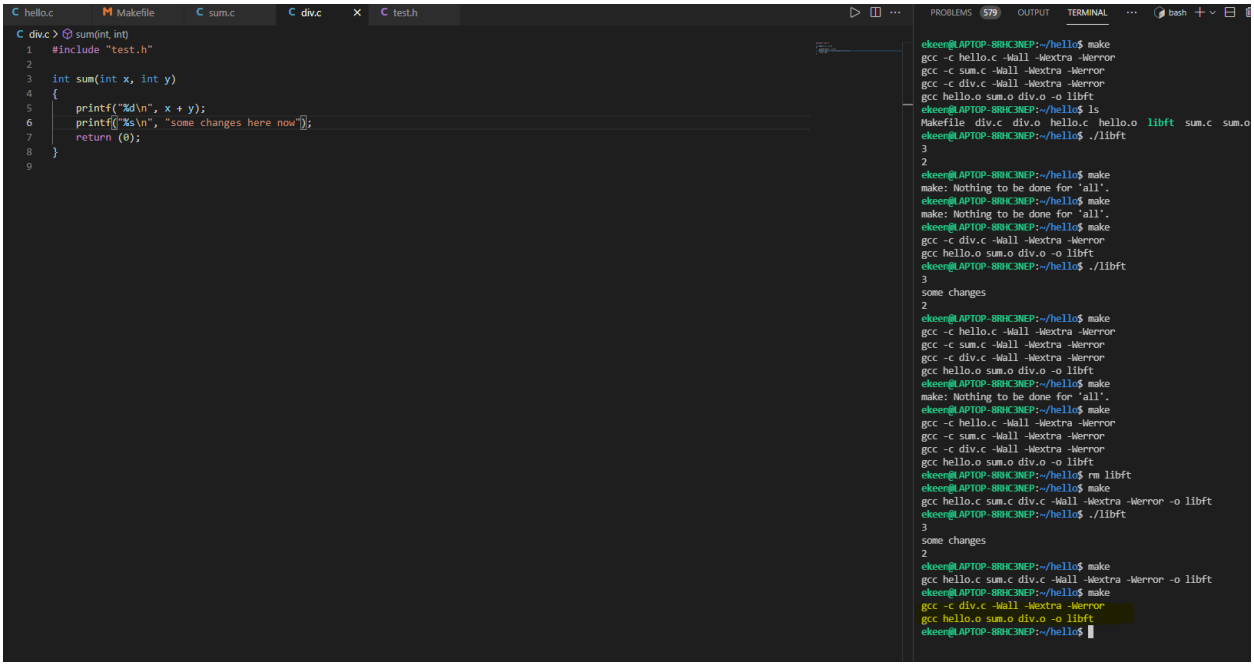




4. It's important to include the header files into makefile so that whenever there are changes to the header files, the c files will be recompiled. Else it wouldn't be.

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.





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

```
cc = gcc
CFLAGS = -Wall -Wextra -Werror
DEPS = test.h
OBJFILES = hello.o sum.o div.o
SRCFILES = hello.c sum.c div.c

all: libft

libft: $(OBJFILES)
    $(cc) $(OBJFILES) -o libft

OBJFILES: $(SRCFILES) $(DEPS)
    $(cc) -c $(SRCFILES) $(CFLAGS)
```

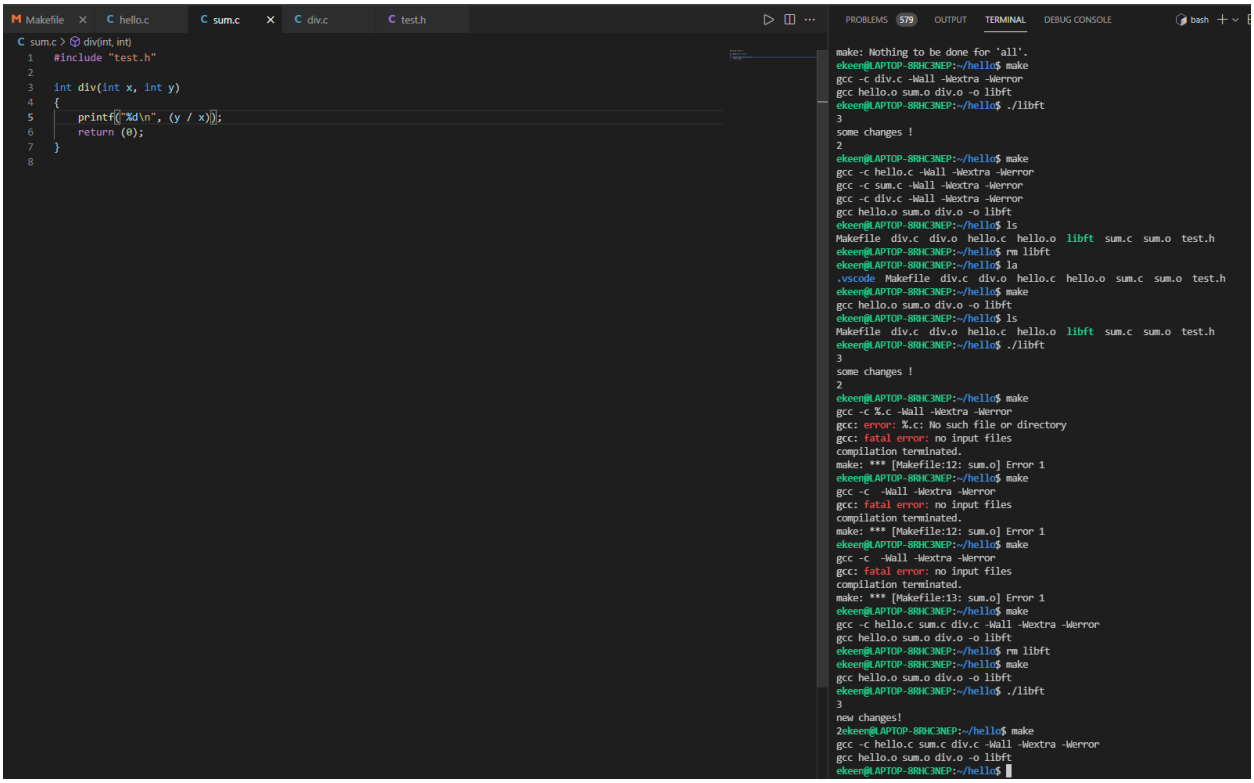
```
ekeen@LAPTOP-8RHC3NEP:~/hello$ ls
Makefile  div.c  div.o  hello.c  hello.o  libft  sum.c  sum
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./libft
3
2
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
make: Nothing to be done for 'all'.
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
make: Nothing to be done for 'all'.
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./libft
3
some changes
2
ekeen@LAPTOP-8RHC3NEP:~/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-8RHC3NEP:~/hello$ make
make: Nothing to be done for 'all'.
ekeen@LAPTOP-8RHC3NEP:~/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-8RHC3NEP:~/hello$ rm libft
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
gcc hello.c sum.c div.c -Wall -Wextra -Werror -o libft
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./libft
3
some changes
2
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
gcc hello.c sum.c div.c -Wall -Wextra -Werror -o libft
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./libft
3
some changes here now
2
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
cc -Wall -Wextra -Werror   -c -o div.o div.c
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./libft
3
some changes here now and again!
2
```

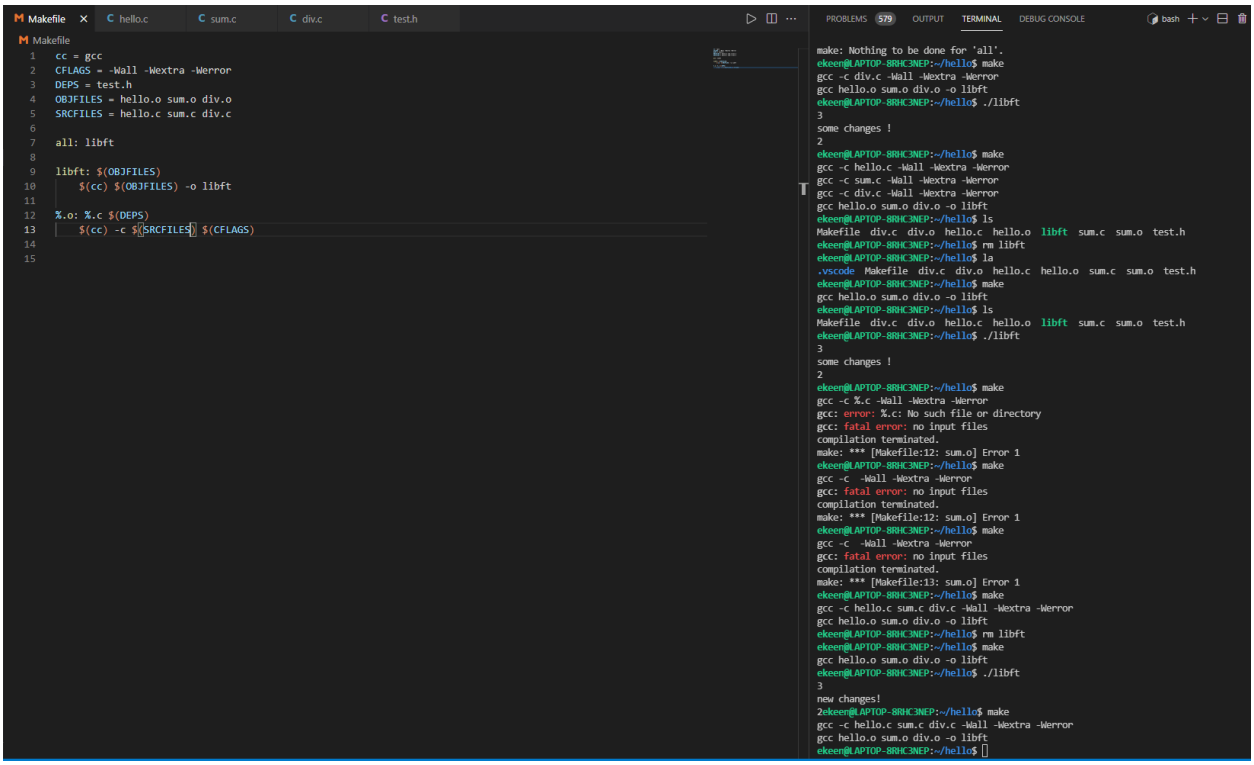
7. Notice that I put %.c in the recipe. This doesn't work as with the target and dependency.

```
cc = gcc
CFLAGS = -Wall -Wextra -Werror
DEPS = test.h
OBJFILES = hello.o sum.o div.o

all: libft

libft: $(OBJFILES)
    $(cc) $(OBJFILES) -o libft

%.o: %.c $(DEPS)
    $(cc) -c %.c $(CFLAGS)
```

```
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
gcc -c hello.c sum.c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./libft
3
some changes here now and again.!
2
ekeen@LAPTOP-8RHC3NEP:~/hello$ ls
Makefile  div.c  div.o  hello.c  hello.o  libft  sum.c  sum.o  test.h
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
make: Nothing to be done for 'all'.
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
gcc -c div.c -Wall -Wextra -Werror
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./libft
3
some changes !
2
ekeen@LAPTOP-8RHC3NEP:~/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-8RHC3NEP:~/hello$ ls
Makefile  div.c  div.o  hello.c  hello.o  libft  sum.c  sum.o  test.h
ekeen@LAPTOP-8RHC3NEP:~/hello$ rm libft
ekeen@LAPTOP-8RHC3NEP:~/hello$ la
.vscode  Makefile  div.c  div.o  hello.c  hello.o  sum.c  sum.o  test.h
ekeen@LAPTOP-8RHC3NEP:~/hello$ make
gcc hello.o sum.o div.o -o libft
ekeen@LAPTOP-8RHC3NEP:~/hello$ ls
Makefile  div.c  div.o  hello.c  hello.o  libft  sum.c  sum.o  test.h
ekeen@LAPTOP-8RHC3NEP:~/hello$ ./libft
3
some changes !
2
ekeen@LAPTOP-8RHC3NEP:~/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-8RHC3NEP:~/hello$
```

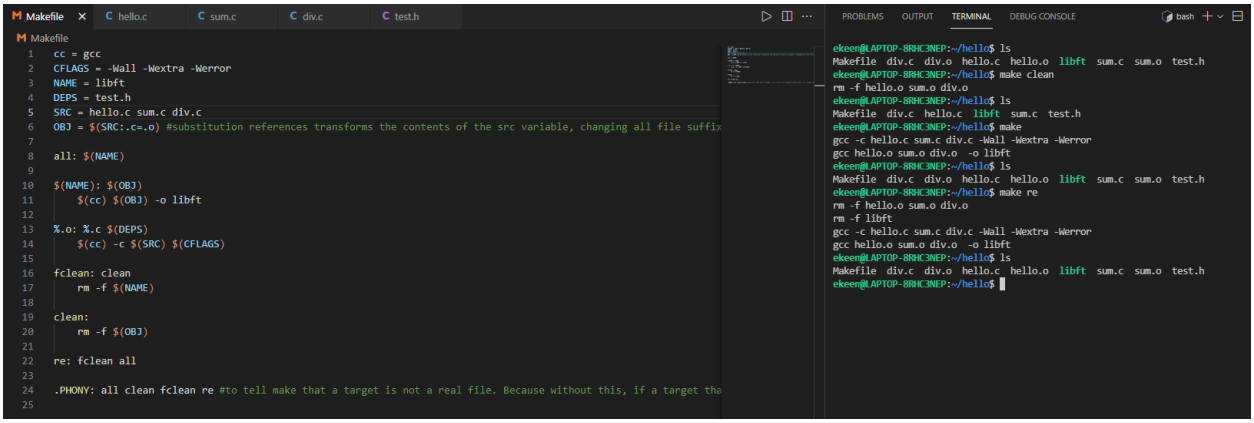
8. The form below recompiles everything.





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.



Makefile : fclean gives error if program is not yet built

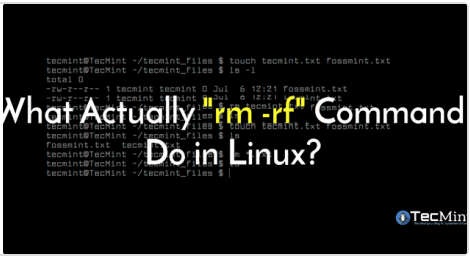
Thanks for contributing an answer to Stack Overflow! Please be sure to answer the question. Provide details and share your research! Asking for help, clarification, or responding to other

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

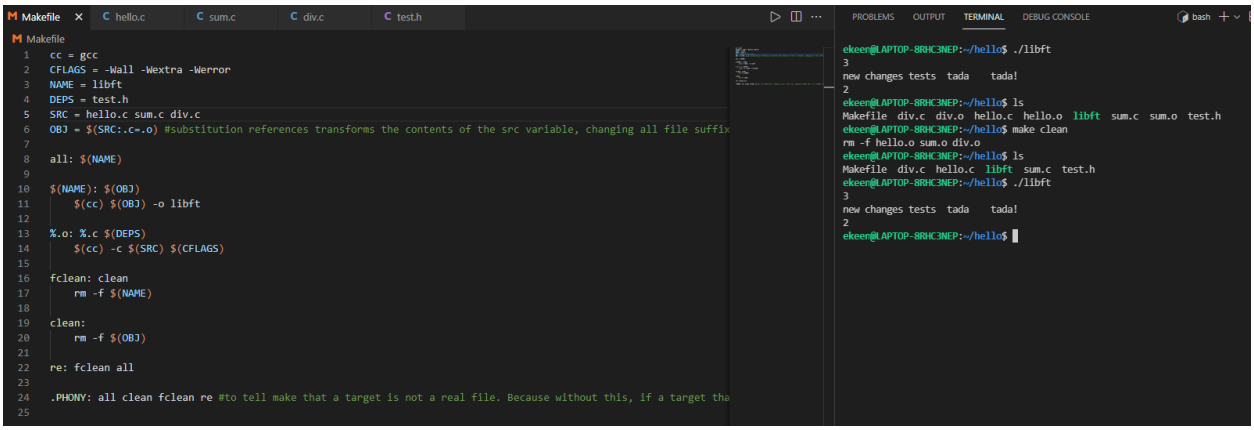
What Actually "rm -rf" Command Do in Linux?

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.

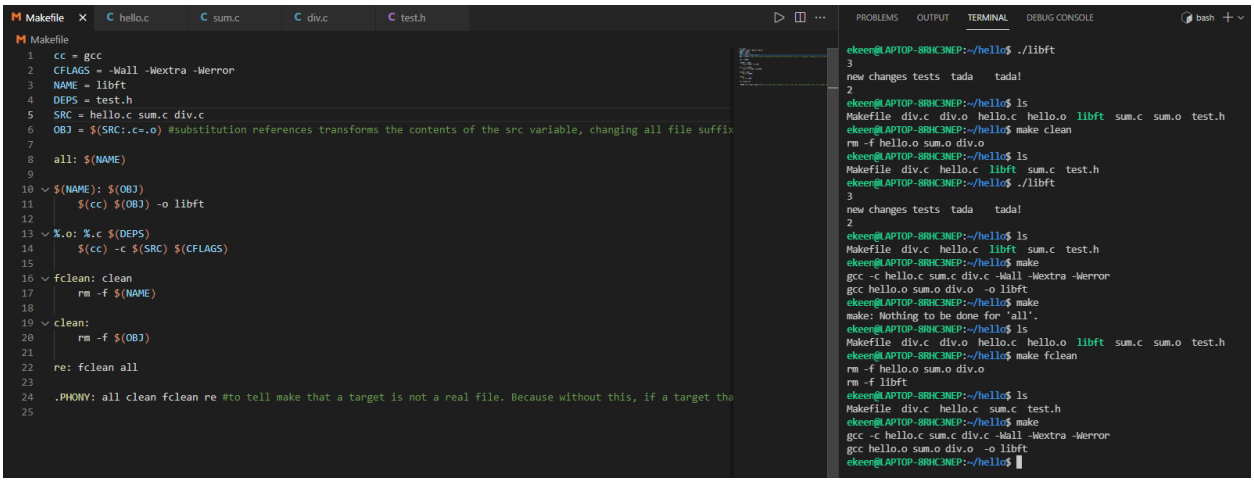
https://www.tecmint.com/linux-rm-command-examples/



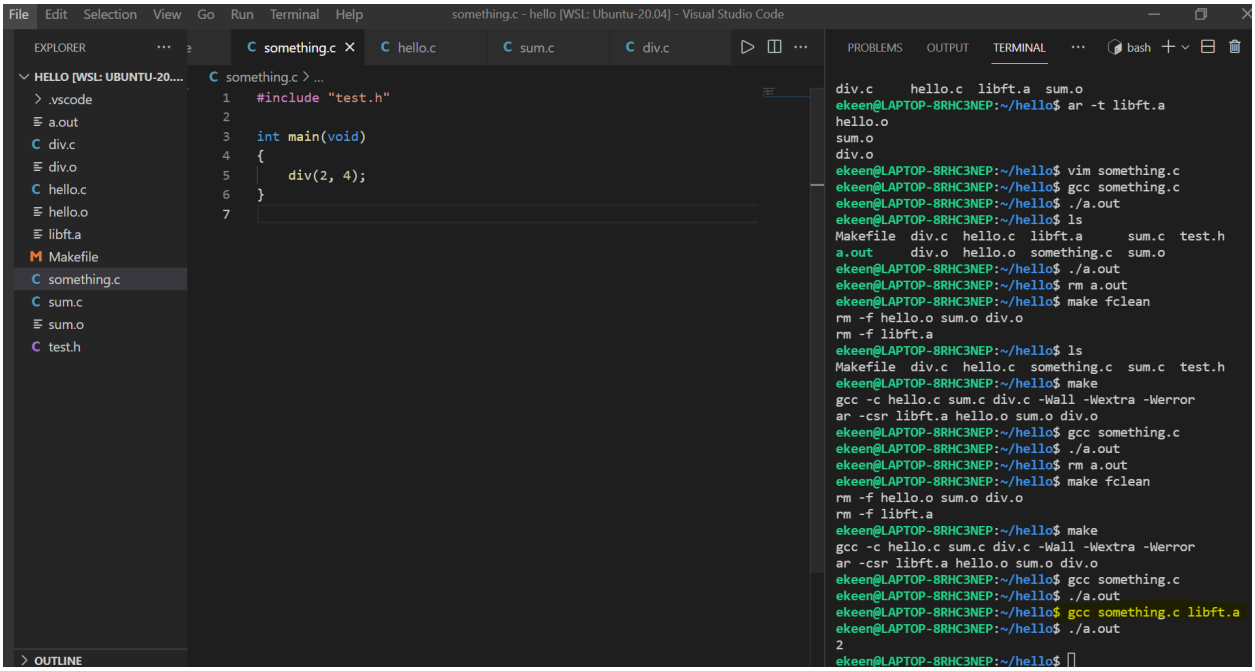
10. Once I removed the .o files, does it mean that the libft executable is corrupted? Nope, the libft still works because it's a separate file used in a separate process.



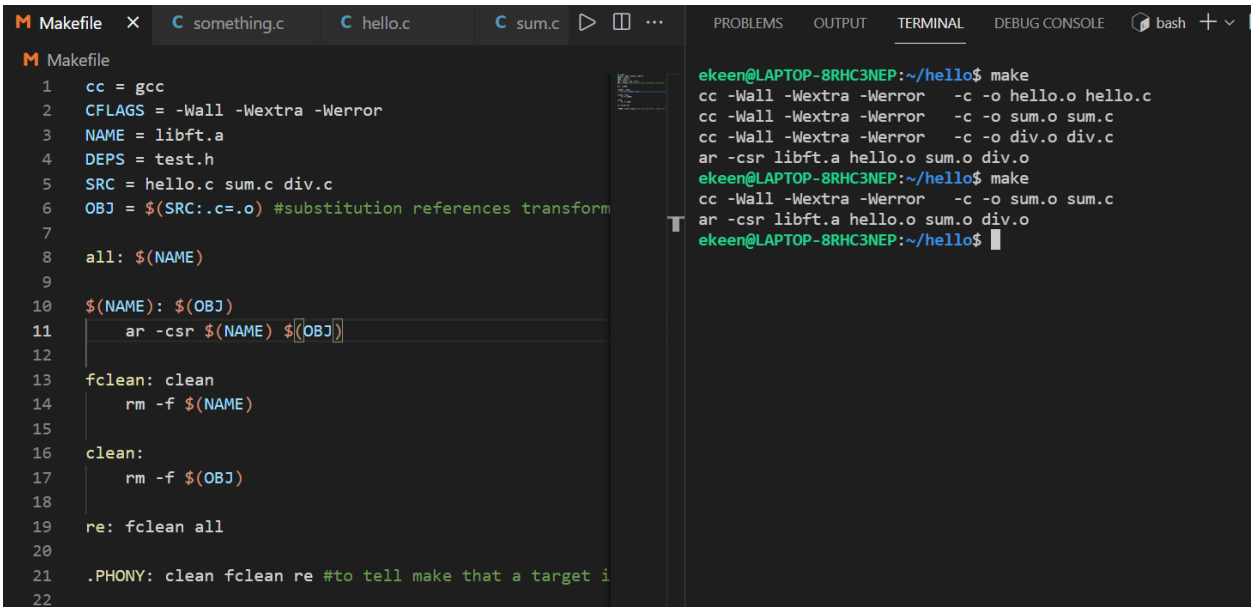
11. After fclean, make will generate object files from the c files.

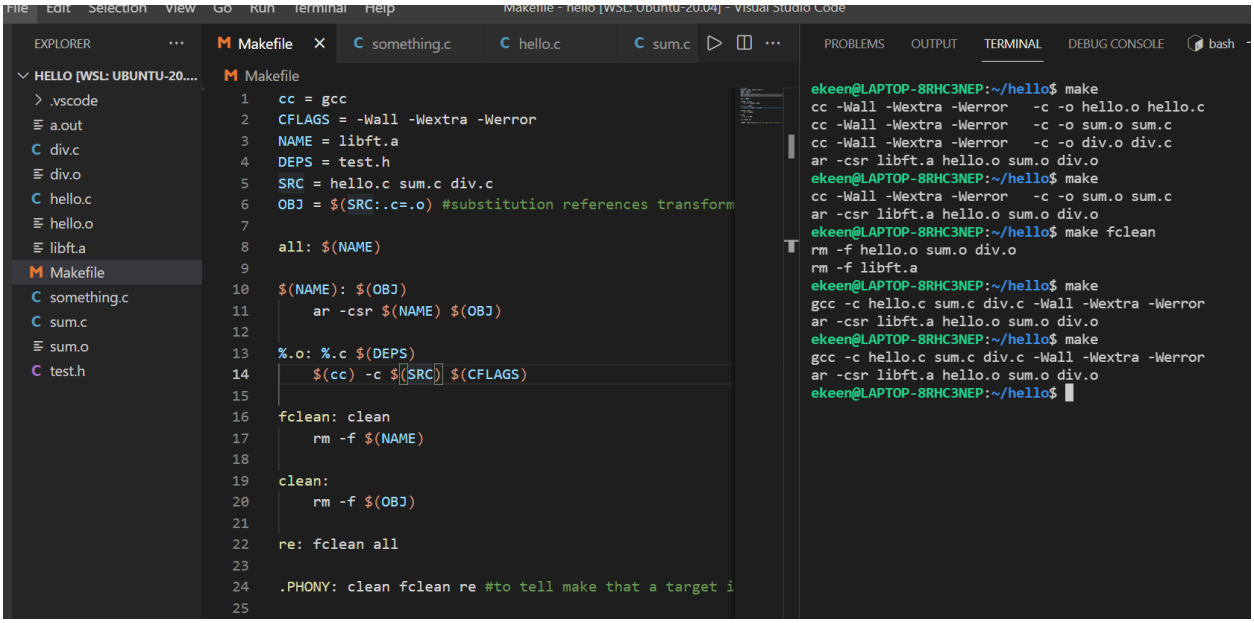


12. Why do I need to include the libft.a together when compiling it with my something.c which is using the header test.h (but something.c isn't part of the Makefile)?



13. 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?



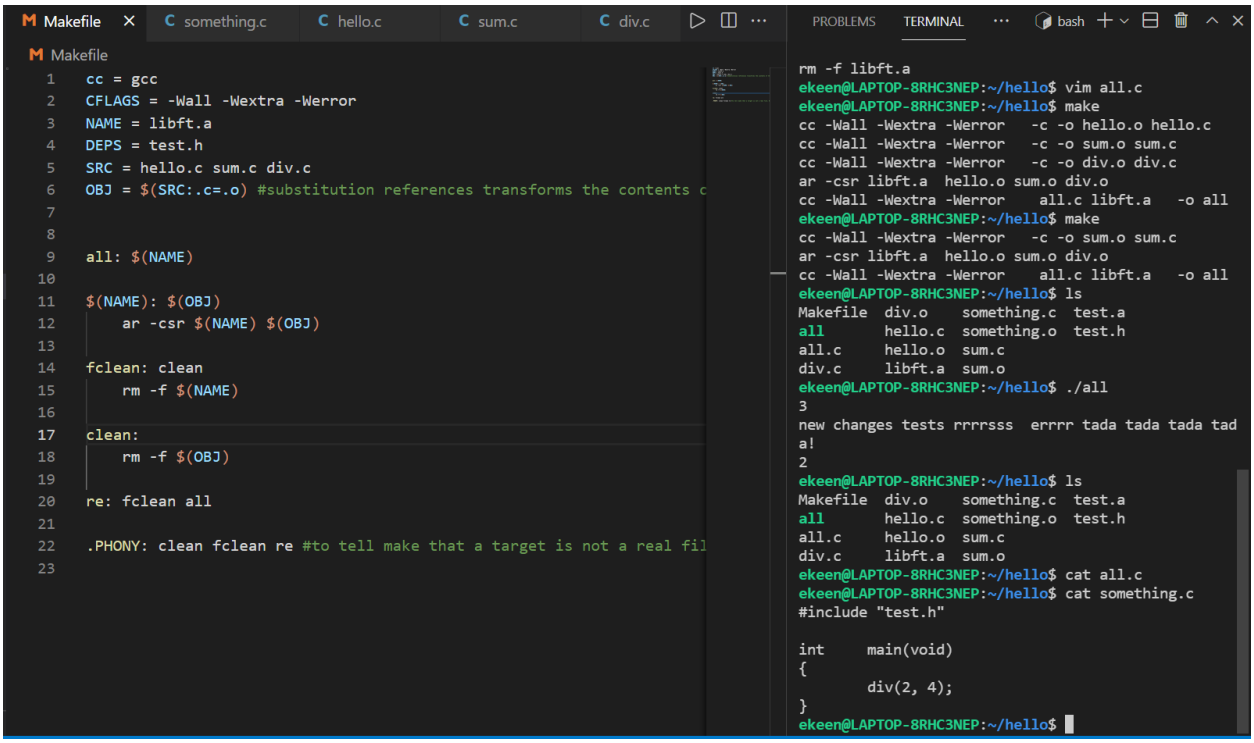


14. Here, I excluded "all", a pseudo target (doesn't exist as a file) out of .PHONY and create an actual all.c file. What happened was that the all.c file got compiled with libft.a to produce an executable file all (but all.c is an empty file, only something.c has included test.h header). A good practice 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



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 | something.c | hello.c | sum.c | div.c

Makefile

```
cc = gcc
#CFLAGS = -Wall -Wextra -Werror
NAME = libft.a
DEPS = test.h
SRC = hello.c sum.c div.c
OBJ = $(SRC:.c=.o) #substitution references transforms the contents c


all: $(NAME)

$(NAME): $(OBJ)
    ar -csr $(NAME) $(OBJ)

fclean: clean
    rm -f $(NAME)

clean:
    rm -f $(OBJ)

re: fclean all

.PHONY: all clean fclean re #to tell make that a target is not a real

```

PROBLEMS TERMINAL bash

```
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 allls
rm: cannot remove 'allls': 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 | sum.c | something.c | hello.c | test.h

Makefile

```
#                                                   +:+ +:+
#    By: ekeen-wy <ekeen-wy@student.42kl.edu.my>    +#+  +:+       +#
#                                                +#+#+#+#+#+   +#+
#    Created: 2021/10/14 15:20:32 by ekeen-wy          #+#    #+#
#    Updated: 2021/10/14 15:54:18 by ekeen-wy         ###   ########.
#
# ******************************************************************

cc = gcc
CFLAGS = -Wall -Wextra -Werror
NAME = libft.a
DEPS = test.h
SRC = hello.c sum.c div.c
OBJ = $(SRC:.c=.o) #substitution references transforms the contents c

all: $(NAME)

$(NAME): $(OBJ)
    ar -csr $(NAME) $(OBJ)

$(OBJ): $(DEPS)

fclean: clean
    rm -f $(NAME)

clean:
    rm -f $(OBJ)

re: fclean all

```

PROBLEMS TERMINAL bash

```
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
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 sum.o sum.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