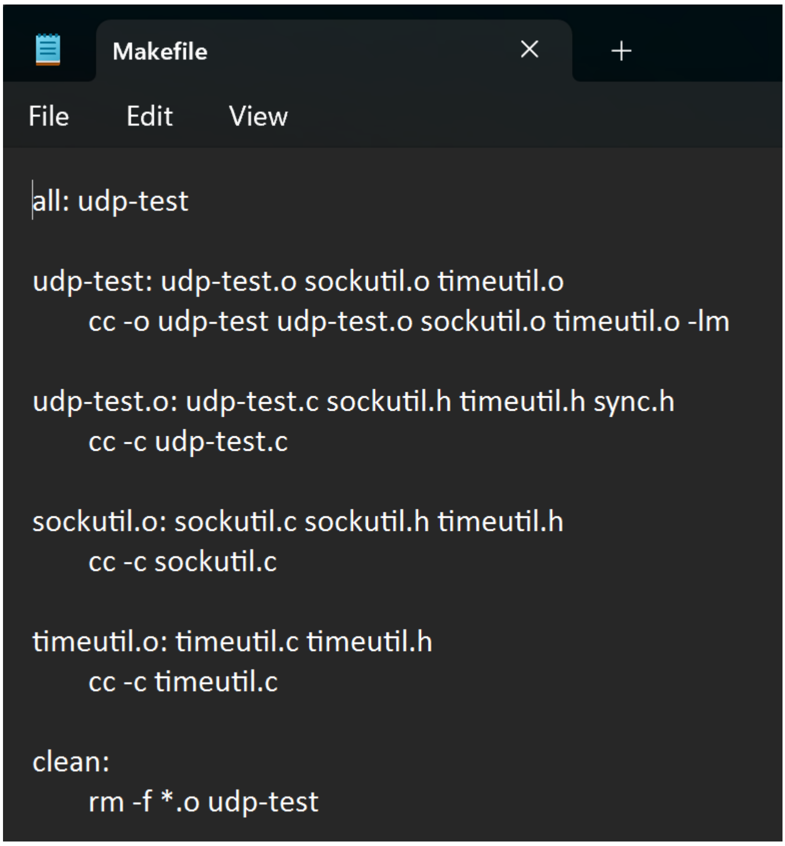
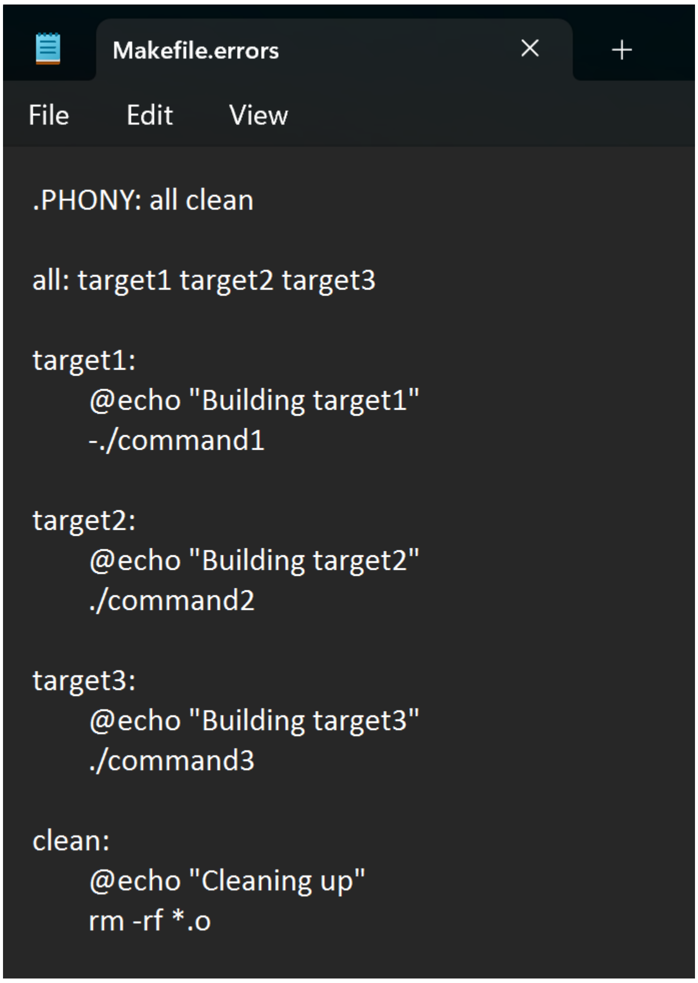
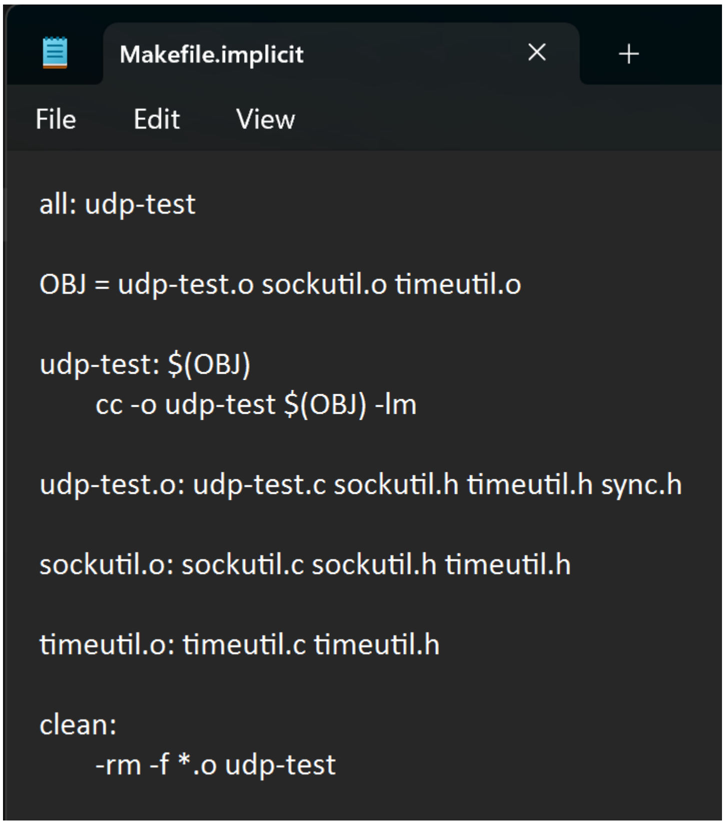


```
all: udp-test

udp-test: udp-test.o sockutil.o timeutil.o
	cc -o udp-test udp-test.o sockutil.o timeutil.o -lm

udp-test.o: udp-test.c sockutil.h timeutil.h sync.h
	cc -c udp-test.c

sockutil.o: sockutil.c sockutil.h timeutil.h
	cc -c sockutil.c

timeutil.o: timeutil.c timeutil.h
	cc -c timeutil.c

clean:
	rm -f *.o udp-test
```

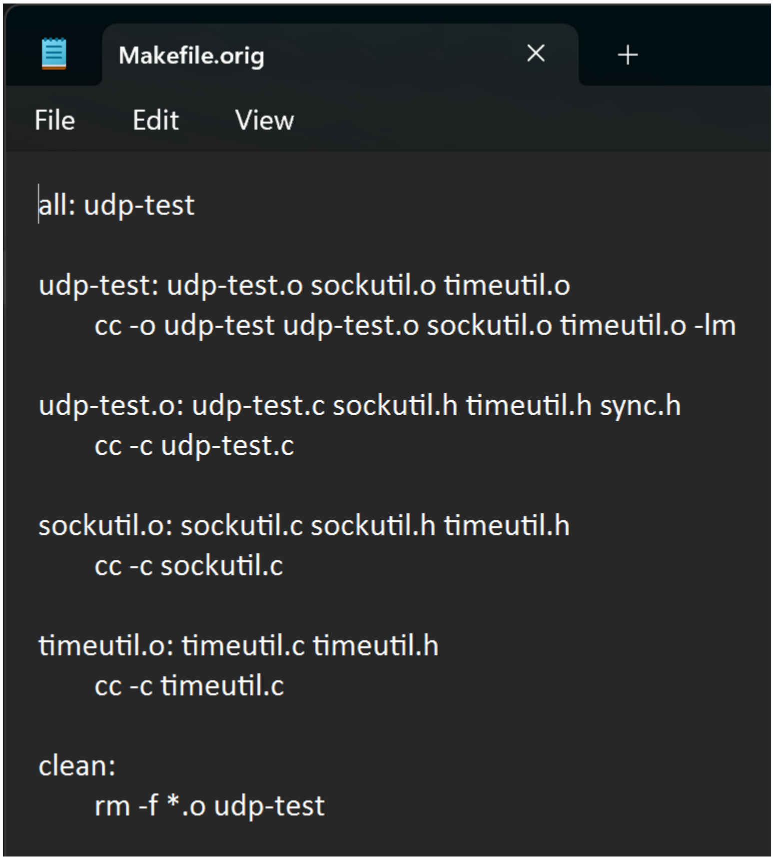
`rr



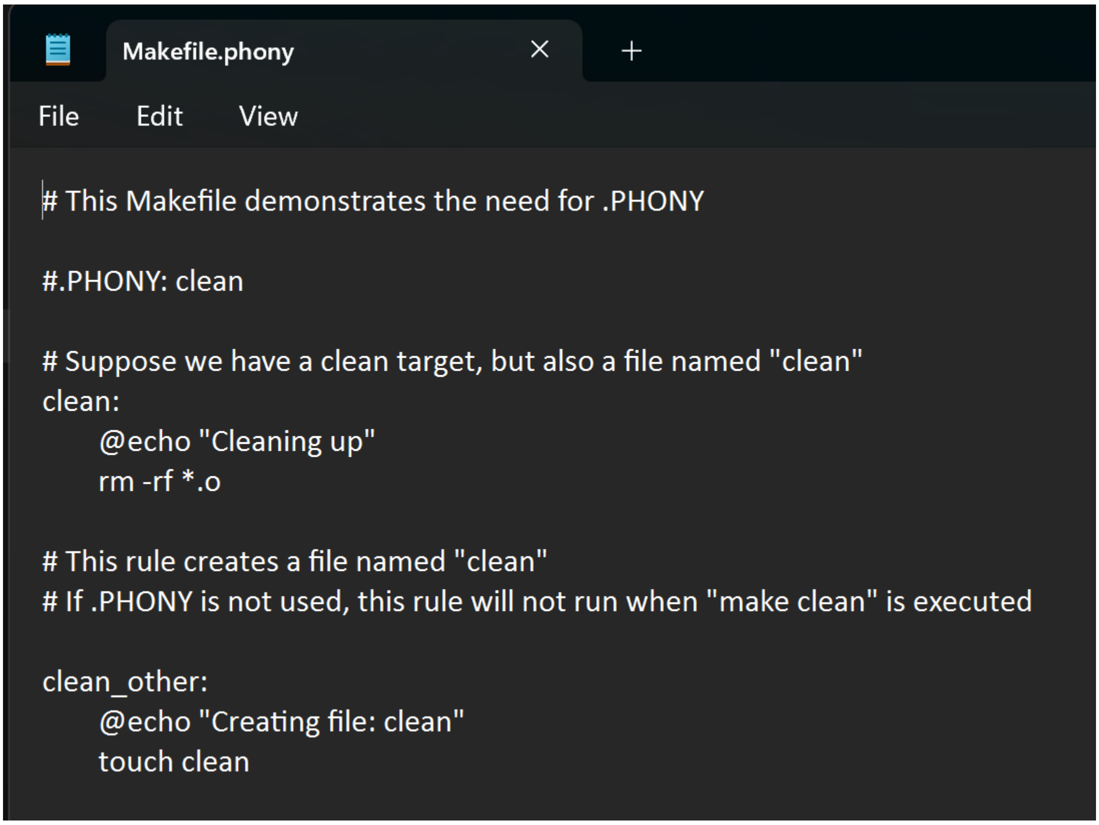
```
.PHONY: all clean

all: target1 target2 target3

target1:
	@echo "Building target1"
	-./command1

target2:
	@echo "Building target2"
	./command2

target3:
	@echo "Building target3"
	./command3

clean:
	@echo "Cleaning up"
	rm -rf *.o
```



```
all: udp-test

OBJ = udp-test.o sockutil.o timeutil.o

udp-test: $(OBJ)
	cc -o udp-test $(OBJ) -lm

udp-test.o: udp-test.c sockutil.h timeutil.h sync.h

sockutil.o: sockutil.c sockutil.h timeutil.h

timeutil.o: timeutil.c timeutil.h

clean:
	-rm -f *.o udp-test
```



```
all: udp-test

udp-test: udp-test.o sockutil.o timeutil.o
	cc -o udp-test udp-test.o sockutil.o timeutil.o -lm

udp-test.o: udp-test.c sockutil.h timeutil.h sync.h
	cc -c udp-test.c

sockutil.o: sockutil.c sockutil.h timeutil.h
	cc -c sockutil.c

timeutil.o: timeutil.c timeutil.h
	cc -c timeutil.c

clean:
	rm -f *.o udp-test
```



```
# This Makefile demonstrates the need for .PHONY

#.PHONY: clean

# Suppose we have a clean target, but also a file named "clean"
clean:
	@echo "Cleaning up"
	rm -rf *.o

# This rule creates a file named "clean"
# If .PHONY is not used, this rule will not run when "make clean" is executed

clean_other:
	@echo "Creating file: clean"
	touch clean
```

rm -f is to force the removal without asking for permission

rm -r is to recursively remove the files from a directory

if there is a file named “clean”, make clean will execute only if some pre requisite of clean was changed and it might ignore the clean that is used to remove all files. So to make sure that make clean removes the files, we add .PHONY: clean

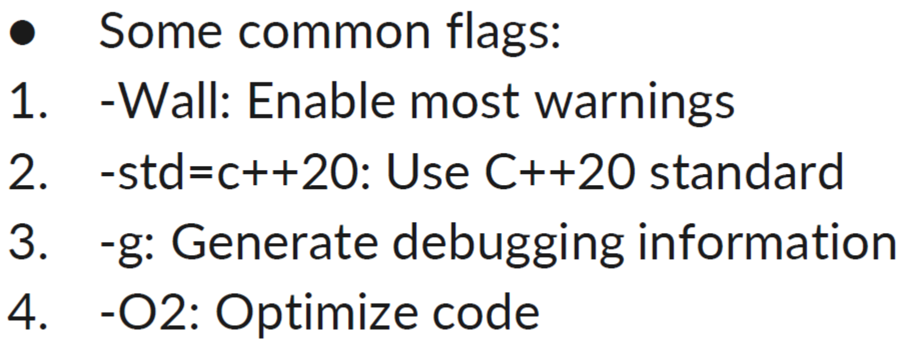
add “-“ infront of commands to suppress any error that arises and continue with the other commands.

$@ - Target

$^ - All Prerequisites

$< - First Prerequisite

use g++ -c if you are making an object file and don’t use -c if you are making an executable file



- Some common flags:

1. -Wall: Enable most warnings
2. -std=c++20: Use C++20 standard
3. -g: Generate debugging information
4. -O2: Optimize code

- The important variables used by implicit rules are:

1. CC: Program for compiling C programs, default cc
2. CXX: Program for compiling C++ programs, default g++
3. CFLAGS: Extra flags to give to the C compiler.
4. CXXFLAGS: Extra flags to the C++ compiler.
5. CPPFLAGS: Extra flags to give to the C preprocessor.
6. LDFLAGS: Extra flags to give to compilers when they are supposed to invoke the linker.

`n.o` is made automatically from `n.cc` or `n.cpp` with a command of the form `$(CXX) -c $(CPPFLAGS) $(CXXFLAGS) $^ -o $@`