

Unix Makefile

An Advanced Introduction to
Unix/C Programming



Dennis
Ritchie



Ken
Thompson



Linus
Torvalds



Richard
Stallman



Brian
Kernighan

John Dempsey

COMP-232 Programming Languages
California State University, Channel Islands

make/makefile – What is it?

- make is a GNU utility used to maintain groups of programs.
- make can be used to automatically determine which programs from a need to be recompiled.
- For large programs, make automatically determines and compiles one or more source code files used to write large programs.

makefile

- To execute commands in the makefile, you simply type: **make**
- Make will look for the makefile named in the following order:

GNUmakefile, makefile, and Makefile

- You can also specify a makefile by using the `-f` option: **make -f mymake.mk**

makefile Example

```
# Makefile for LAB 2
```

```
CFLAGS=-g
```

```
all: age ratio retire stacks
```

```
age: age.o
```

```
    gcc $(CFLAGS) age.c -o age
```

```
ratio: ratio.o
```

```
    gcc $(CFLAGS) ratio.c -o ratio
```

```
retire: retire.o
```

```
    gcc $(CFLAGS) retire.c -o retire
```

```
stacks: stacks.o
```

```
    gcc $(CFLAGS) stacks.c -o stacks
```

```
clean:
```

```
    rm *.o age ratio retire stacks
```

makefile

```
john@oho:~/LAB2$ ls  
Makefile age.c car.c ratio.c retire.c
```

```
john@oho:~/LAB2$ make car  
cc -g -c -o car.o car.c  
gcc -g car.c -o car
```

← To compile just car.c, use “make car”

```
john@oho:~/LAB2$ make  
cc -g -c -o age.o age.c  
gcc -g age.c -o age  
cc -g -c -o ratio.o ratio.c  
gcc -g ratio.c -o ratio  
cc -g -c -o retire.o retire.c  
gcc -g retire.c -o retire
```

← To compile all programs, use “make” or “make all”

```
john@oho:~/LAB2$ make  
make: Nothing to be done for 'all'.
```

```
john@oho:~/LAB2$ ls  
Makefile age.c ratio.c retire retire.o stacks stacks.o  
age age.o ratio ratio.o retire.c stacks.c
```

makefile Notes

- When writing a makefile, you need to use the <TAB> character.
- Since the TAB is not viewable, many programmers have lost an **infinite** amount of time because they used spaces instead!!!
- To continue to the next line, use \ (without a space).
- I prefer using upper case Makefile instead of makefile because Makefile will usually be listed before your source code files.

makefile – Multiple Source Code Files

- Large programs are divided into multiple source code files with each file containing one or more defined functions.
- A makefile will:
 - Identify all source code files needed to compile the project.
 - Identify compiler options, e.g., `CFLAGS=-m64 -w -g -DORACLE_DB`
 - Identify the location for include files to use, e.g., `-I/project/includes -I.`
 - Identify which libraries to use, e.g., `-lm`
 - Specify how to compile `.c` files into `.o` files.
 - Specify preprocessor commands, e.g., `proc` to compile embedded Oracle code.
 - Specify how to create an executable file to run.
 - Specify additional commands, like clean up or startup commands.

makefile - Multiple Source Code Files

```
john@oho:~/MAKEFILE$ ls
Makefile four.c main.c one.c three.c two.c
```

```
#include <stdio.h>
main()
{
    one();
    two();
    three();
    four();
    printf("All Done.\n");
}
```

```
#include <stdio.h>
one() { printf("One ...\n"); }
```

```
#include <stdio.h>
two() { printf("Two ...\n"); }
```

```
#include <stdio.h>
three() { printf("Three ...\n"); }
```

```
#include <stdio.h>
four() { printf("Four ...\n"); }
```

```
john@oho:~/MAKEFILE$ more Makefile
```

```
SOURCE=\
    main.c\
    four.c\
    one.c\
    three.c\
    two.c
```

```
CFLAGS =-m64 -w -g
DEFINES =-DORACLE
```

```
OBJS =$(SOURCE:.c=.o)
```

```
main: $(OBJS)
    $(CC) $(CFLAGS) $(OBJS) -lm -l. -O -o $@
ls -l main
```

```
clean:
    rm *.o main
```

```
install:
    cp -p main /home/production/main
```

```
john@oho:~/MAKEFILE$ ls
Makefile four.c main.c one.c three.c two.c
```

```
john@oho:~/MAKEFILE$ make
cc -m64 -w -g -c -o main.o main.c
cc -m64 -w -g -c -o four.o four.c
cc -m64 -w -g -c -o one.o one.c
cc -m64 -w -g -c -o three.o three.c
cc -m64 -w -g -c -o two.o two.c
cc -m64 -w -g main.o four.o one.o three.o two.o -lm -l. -O -o main
ls -l main
-rwxr-xr-x 1 john john 24736 Jan  4 11:17 main
```

```
john@oho:~/MAKEFILE$ ls
Makefile four.o main.c one.c three.c two.c
four.c  main  main.o one.o three.o two.o
```

```
john@oho:~/MAKEFILE$ main
One ...
Two ...
Three ...
Four ...
All Done.
```


Makefile Comment

One of the main advantage in using Makefiles in the past was to only compile those modules that changed since the last recompile.

Today, Makefiles help to organize and identify which modules are needed to build executables.

They also identify how to compile a program. E.g., an Oracle myCode.pc will need a preprocessor proc to run first in order to generate the myCode.c file that is compiled by gcc.