

Make: hands on

Make documentation

The official GNU Make manual is the authoritative reference for GNU Make. Every feature we cover in this module is also described in the manual:

https://www.gnu.org/software/make/manual/html_node/index.html

Make

Reads the following from a file called `Makefile`:

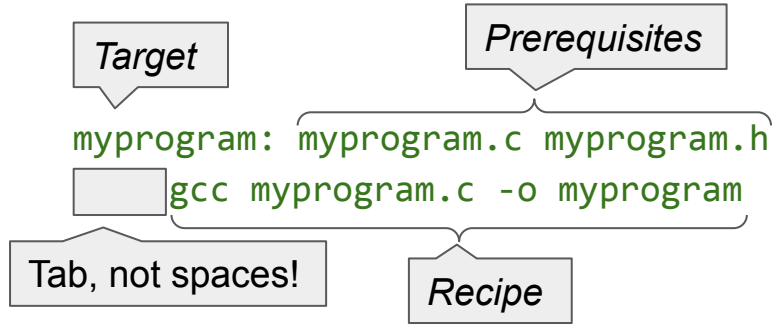
- Rules
 - Target
 - Prerequisites
 - Recipe
- Variable definitions
- Special targets

Anatomy of a Make rule

Target: A file that the rule produces. Make will check the timestamp of it to see if the rule needs rebuilding. A rule cannot have multiple targets.

Prerequisites: Files that the rule depends on. If any prerequisite is newer than the target, the recipe runs. If a prerequisite has its own rule, Make will run that rule first if necessary. *It's up to you to get this right.*


Recipe: A list of shell commands to run to generate the target. Each line runs in its own shell, so `cd` and setting shell variables won't persist across multiple lines.



What questions do
you have?

demo time!

Simple Makefile without dependencies



Target

myprogram:

```
gcc myprogram.c -o myprogram
```



Rules

"Build
myprogram using
gcc"

Uh oh... it doesn't rebuild

```
$ make myprogram
gcc myprogram.c -o myprogram
$ vim myprogram.c
...
$ make myprogram
make: myprogram is up to date.
$
```


Sample Makefile

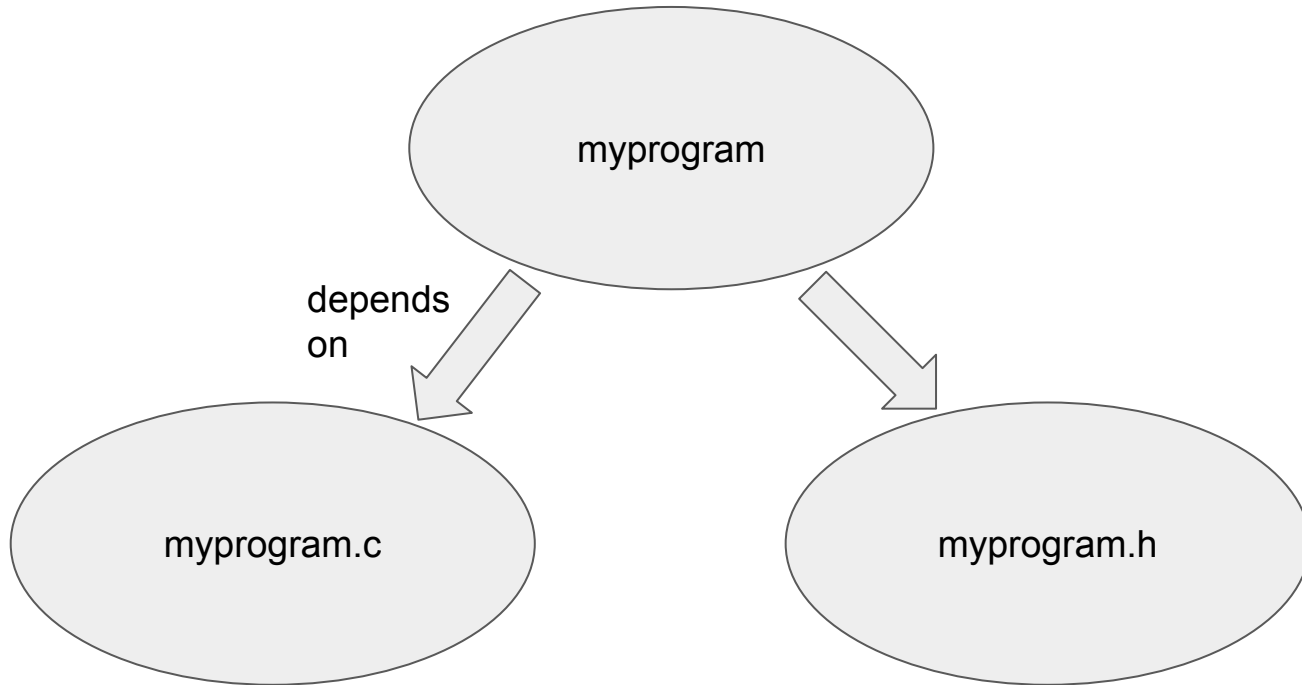
Target

Dependencies

```
myprogram: myprogram.c myprogram.h  
gcc myprogram.c -o myprogram
```

Rules

"(Re)build
myprogram if
either
myprogram.c or
myprogram.h
changes"



Use gcc to build myprogram.c

Not passed to gcc, but still affects the program!

make

Run rules from Makefile to build the given target(s).

By default, builds the first target in the file.

-j[n]: Runs up to n rules simultaneously.

-f <file>: Reads rules from <file> instead of Makefile.

--silent (-s): Don't print commands.

```
$ make log.o
gcc -c log.c

$ make main
gcc -c main.c
gcc main.o log.o -o main

$ make # Same as "make main"
make: 'main' is up to date.

$ make clean
rm -rf *.o main
$
```

Split compilation

- It's possible to compile C files to .o files of machine code and then link them together
- Make makes this more useful by only rebuilding the .o files whose corresponding .c files changed

.PHONY

"A phony target is one that is not really the name of a file; rather it is just a name for a recipe to be executed when you make an explicit request. There are two reasons to use a phony target: to avoid a conflict with a file of the same name, and to improve performance.

If you write a rule whose recipe will not create the target file, the recipe will be executed every time the target comes up for remaking."

https://www.gnu.org/software/make/manual/html_node/Phony-Targets.html

demo time!

Make vs shell script

- Shell script *can* do all the things Make can do
 - BUT there are first-class features of Make that make your life easier
- Shell scripting *is* already familiar to you
 - BUT may not be the right hammer for this nail
- Shell script *might* be shorter at first
 - BUT as your build system feature needs get longer, Make will be shorter in the end
- You can always do shell scripting from within Make