

`make` is a system utility for managing the build process (compilation/linking/etc).

There are various versions of `make`; these notes discuss the GNU `make` utility included on Linux systems.

As the GNU Make manual* says:

The `make` utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them.

Using `make` yields a number of benefits, including:

- faster builds for large systems, since only modules that must be recompiled will be
- the ability to provide a simple way to distribute build instructions for a project
- the ability to provide automated cleanup instructions

*<http://www.gnu.org/software/make/manual/make.pdf>

The following presentation is based upon the following collection of C source files:

<code>driver.c</code>	the main “driver”
<code>CSet.h</code>	the "public" interface of the CSet type
<code>CSet.c</code>	the implementation of the CSet type
<code>gradeCSet.h</code>	the "public" interface of the test harness
<code>gradeCSet.c</code>	the implementation of the test harness

The example is derived from an assignment that is occasionally used in CS 2506.

Here's a minimal makefile for the given source base:

```
# CSet minimal makefile
#
SHELL=/bin/bash
#
# Specify compiler and compiler switches:
CC=gcc
CFLAGS=-std=c11 -Wall -W -O0 -ggdb3
#
# Build executable for testing:
driver: driver.c CSet.c gradeCSet.c
    $(CC) $(CFLAGS) -o driver driver.c CSet.c gradeCSet.c
#
# Remove object files:
clean:
    rm -f *.o driver
#
# Archive source and makefile:
package:
    tar cvf CSetCode.tar *.c *.h makefile
```

The given makefile provides:

- a way to create an executable from the given source files: `make driver`

```
. . .  
#  
# Build executable for testing:  
driver: driver.c CSet.c gradeCSet.c  
    $(CC) $(CFLAGS) -o driver driver.c CSet.c gradeCSet.c  
. . .
```

- a way to clear the directory of stale files: `make clean`

```
#  
# Remove object files:  
clean:  
    rm -f *.o driver
```

- a way to package the source files: `make package`

```
#  
# Archive source and makefile:  
package:  
    tar cvf CSetCode.tar *.c *.h makefile
```

The given makefile does not take advantage of the most interesting feature of `make`:

- the ability to only recompile files that are affected by changes

The following presentation is based upon the following collection of C source files:

<code>driver.c</code>	the main “driver”
<code>Polynomial.h</code>	the "public" interface of the Polynomial type
<code>Polynomial.c</code>	the implementation of the Polynomial type
<code>PolyTester.h</code>	the "public" interface of the test harness
<code>PolyTester.c</code>	the implementation of the test harness

The example is derived from an assignment that is occasionally used in CS 2506.

The C source files use the following `include` directives related to files in the project:

```
driver.c:  
  Polynomial.h  
  PolyTester.h
```

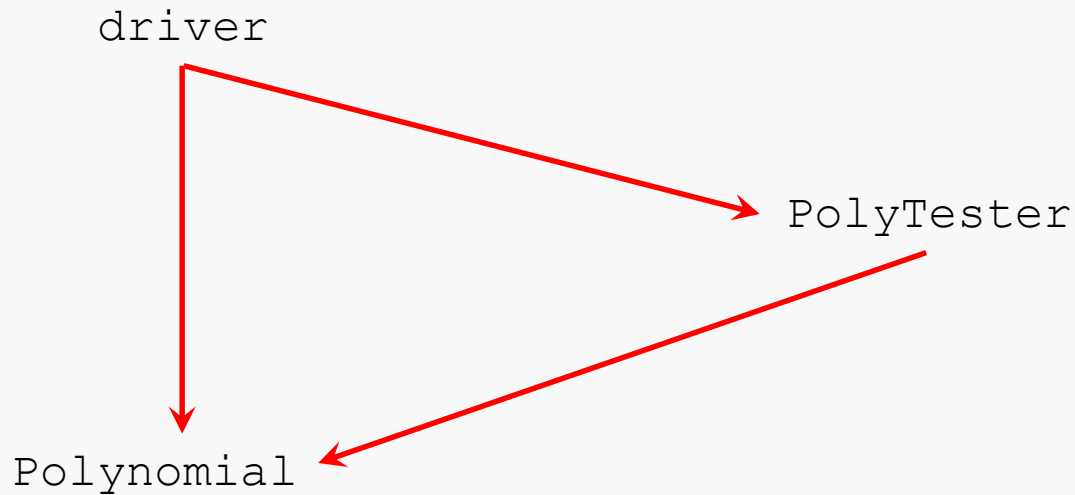
```
Polynomial.c:  
  Polynomial.h
```

```
PolyTester.h:  
  Polynomial.h
```

```
PolyTester.c:  
  PolyTester.h
```

We need to understand how the inclusions affect compilation...

The C source files exhibit the following dependencies (due to include directives):



Source file	Recompile if changes are made to:
driver.c	driver.c or PolyTester.* or Polynomial.*
PolyTester.c	PolyTester.c, PolyTester.h or Polynomial.*
Polynomial.c	Polynomial.c Polynomial.h

You use a kind of script called a *makefile* to tell `make` what to do.

A simple makefile is just a list of rules of the form:

```
target ... : prerequisites ...  
    recipe  
    ...
```

Prerequisites are the files that are used as input to create the target.

A *recipe* specifies an action that `make` carries out.

Here is a simple rule for compiling `PolyTester.c` (and so producing `PolyTester.o`):

```
PolyTester.o: Polynomial.c Polynomial.h PolyTester.c PolyTester.h
    gcc -c -std=c99 -Wall PolyTester.c Polynomial.c
```

Now, we have some issues:

- This doesn't save us any rebuilding... every C file that `PolyTester.o` depends on will be recompiled every time we invoke the rule for that target.
- There is a lot of redundancy in the statement of the rule... too much typing!
- What if we wanted to build for debugging? We'd need to add something (for instance, `-ggdb3`) to the recipe in every rule. That's inefficient.

We can specify targets as prerequisites, as well as C source files:

```
PolyTester.o: Polynomial.o PolyTester.c PolyTester.h  
gcc -c -std=c99 -Wall PolyTester.c
```

Now, if we invoke `make` on the target `PolyTester.o`:

- `make` examines the modification time for each direct (and indirect) prerequisite for `PolyTester.o`
- each involved target is rebuilt, by invoking its recipe, iff that target has a prerequisite, that has changed since that target was last built

We can define variables in our makefile and use them in recipes:

```
CC=gcc
CFLAGS=-O0 -m64 -std=c99 -Wall -W -ggdb3
```

```
PolyTester.o: Polynomial.o PolyTester.c
    $(CC) $(CFLAGS) -c PolyTester.c
```

This would make it easier to alter the compiler options for all targets (or to change compilers).

Syntax note: no spaces around '='.

We can also define a rule with no prerequisites; the most common use is probably to define a cleanup rule:

```
clean:  
    rm -f *.o *.stackdump
```

Invoking `make` on this target would cause the removal of all object and stackdump files from the directory.

Here is a complete makefile for the example project:

```
# Specify shell to execute recipes
SHELL=/bin/bash

# Set compilation options:
#
# -O0          no optimizations; remove after debugging
# -std=c99     use C99 Standard features
# -Wall       show "all" warnings
# -W          show even more warnings (annoyingly informative)
# -ggdb3      add extra debug info; remove after debugging
#
#
CC=gcc
CFLAGS=-O0 -std=c99 -m32 -Wall -W -ggdb3

. . .
```

Here is a complete makefile for the example project:

```
. . .

driver: Polynomial.o PolyTester.o
    $(CC) $(CFLAGS) -o driver driver.c Polynomial.o PolyTester.o

PolyTester.o: Polynomial.o PolyTester.c
    $(CC) $(CFLAGS) -c PolyTester.c

Polynomial.o: Polynomial.c Polynomial.h
    $(CC) $(CFLAGS) -c Polynomial.c

clean:
    rm *.o
```


make can be invoked in several ways, including:

```
make
make <target>
make -f <makefile name> <target>
```

In the first two cases, make looks for a makefile, in the current directory, with a default name. GNU make looks for the following names, in this order:

```
GNUmakefile
makefile
Makefile
```

If no target is specified, make will process the first rule in the makefile.

Using the makefile shown above, and the source files indicated earlier:

```
centos > ll
total 64
-rw-rw-r--. 1 wdm wdm  1197 Feb 15 21:07 driver.c
-rw-rw-r--. 1 wdm wdm   350 Feb 15 21:18 makefile
-rw-rw-r--. 1 wdm wdm 10824 Feb 15 21:07 Polynomial.c
-rw-rw-r--. 1 wdm wdm  5501 Feb 15 21:07 Polynomial.h
-rw-rw-r--. 1 wdm wdm 28914 Feb 15 21:07 PolyTester.c
-rw-rw-r--. 1 wdm wdm   886 Feb 15 21:07 PolyTester.h

centos > make driver
gcc -O0 -std=c99 -m32 -Wall -ggdb3 -W -c Polynomial.c
gcc -O0 -std=c99 -m32 -Wall -ggdb3 -W -c PolyTester.c
gcc -O0 -std=c99 -m32 -Wall -ggdb3 -W -o driver driver.c Polynomial.o
PolyTester.o

centos >
```

Since I hadn't compiled anything yet, `make` invoked all of the rules in `makefile`.

Now, I'll modify one of the C files and run `make` again:

```
centos > touch PolyTester.c

centos > make driver
gcc -O0 -std=c99 -m32 -Wall -ggdb3 -W -c PolyTester.c
gcc -O0 -std=c99 -m32 -Wall -ggdb3 -W -o driver driver.c Polynomial.o
PolyTester.o

centos >
```

The only recipes that were invoked were those for the targets that depend on `PolyTester.c`.

Now, I'll modify a "deeper" C file and run make again:

```
centos > touch Polynomial.c

centos > make driver
gcc -O0 -std=c99 -m32 -Wall -ggdb3 -W -c Polynomial.c
gcc -O0 -std=c99 -m32 -Wall -ggdb3 -W -c PolyTester.c
gcc -O0 -std=c99 -m32 -Wall -ggdb3 -W -o driver driver.c Polynomial.o
PolyTester.o

centos >
```

Again, the only files that were recompiled were the ones depending on the changed file.

Of course, we can also build “secondary” targets:

```
total 64
-rw-rw-r--. 1 wdm wdm  1197 Feb 15 21:07 driver.c
-rw-rw-r--. 1 wdm wdm   350 Feb 15 21:18 makefile
-rw-rw-r--. 1 wdm wdm 10824 Feb 15 21:29 Polynomial.c
-rw-rw-r--. 1 wdm wdm  5501 Feb 15 21:07 Polynomial.h
-rw-rw-r--. 1 wdm wdm 28914 Feb 15 21:27 PolyTester.c
-rw-rw-r--. 1 wdm wdm   886 Feb 15 21:07 PolyTester.h

centos > make PolyTester.o
gcc -O0 -std=c99 -m32 -Wall -ggdb3 -W -c Polynomial.c
gcc -O0 -std=c99 -m32 -Wall -ggdb3 -W -c PolyTester.c

centos >
```

The only files that were compiled were the ones on which the specified target depends.