

Managing Projects with Make

Managing compilation and linking in a medium to large programming project with multiple modules can be a daunting task, especially if the compilation and linking stages require complicated command-line arguments.

Fortunately, there is a standard UNIX tool called **make** that automates much of this process.

The **make** command looks for a file call “makefile” (first choice) or “Makefile” (second choice) in the current directory. The **Makefile** contains the steps necessary to make or update certain files.

Versions of make

- There's "old make"
 - Sometimes called `omake`
 - You probably won't find one of these
- There's "new make"
 - Sometimes called `nmake`
 - Normally just called `make`
 - `/usr/ccs/bin/make` on the prime machines
- Then there "gnu make"
 - Usually called `gmake`
 - The standard under Linux; usually just called "make"
 - `/usr/local/bin/gmake` on the prime machines

Gnu Make

- gmake understands all of “old make”
- gmake understands most of “new make”
- gmake is MUCH BETTER
 - So that’s what I’ll be talking about
- gmake first looks for file `GNUmakefile`
 - Then `makefile`, then `Makefile`
- If the file is “truly gmake only”, it should be called `GNUmakefile`

Makefile Syntax

- **Variables:**

VARNAME = string

For Example:

CFLAGS = -g -Wall -Werror -O2

These variables may be referenced by typing

`$(VARNAME)`

or

`${VARNAME}`


- **Dependencies**

file1: file2 file3 file4

This “means” that in order to make file1, file2, file3, and file4 must already exist and must be current.

- **Rules**

file1: file2 file3 file4

 cat file2 file3 file4 >file1

TAB

Using Make

When you issue the command `make`, the `make` command searches the makefile for the first rule and uses this for its target. `Make` first ensures that all the files on which the target depends are up to date, and then it creates the target if it either doesn't exist or if it is older than one of the files on which it depends.

Example

```
CFLAGS = -g -Wall -Werror -O2
CC = gcc
# This is a comment
x: x1.o x2.o x3.o
    $(CC) -o x $(CFLAGS) x1.o x2.o x3.o
x1.o : x1.c project.h
    $(CC) -c $(CFLAGS) x1.c
x2.o : x2.c
    $(CC) -c $(CFLAGS) x2.c
x3.o : x3.c
    $(CC) -c $(CFLAGS) x3.c
```

Make is Smart!!

Because make understands how to do many things on its own, the previous example can be shortened to:

Shorter Example

```
CFLAGS = -g -Wall -Werror -O2
CC = gcc
# This is a comment
x: x1.o x2.o x3.o
    $(CC) -o x $(CFLAGS) x1.o x2.o x3.o
x1.o : x1.c project.h
```

and make will use the CFLAGS and CC that you specified to compile x[123].c without further instructions on your part.

Make Buildin Rules for C

Variable	Default
CC	cc
CFLAGS	
CPPFLAGS	
COMPILE.c	\$(CC) \$(CFLAGS) \$(CPPFLAGS) -c
LINK.c	\$(CC) \$(CFLAGS) \$(CPPFLAGS) \$(LDFLAGS)

Make Buildin Rules for C++

Variable	Default
CCC	CC
CCFLAGS	CFLAGS
CPPFLAGS	
COMPILE.cc	\$(CCC) \$(CCFLAGS) \$(CPPFLAGS) -c
LINK.cc	\$(CCC) \$(CCFLAGS) \$(CPPFLAGS) \$(LDFLAGS)
COMPILE.C	\$(CCC) \$(CCFLAGS) \$(CPPFLAGS) -c
LINK.C	\$(CCC) \$(CCFLAGS) \$(CPPFLAGS) \$(LDFLAGS)

Wildcards

- Here's some fun stuff:

```
SOURCES=${wildcard *.sm}  
BASENAME=${SOURCES:.sm=}  
PSFILES=${SOURCES:.sm=.ps}  
DVIFILES=${SOURCES:.sm=.dvi}  
default: ${DVIFILES} ${PSFILES} maybepdf
```

or

```
HEADERS=${wildcard *.h}  
SOURCES=${wildcard *.cc}  
OBJECTS=${SOURCES:.cc=.o}
```

Optional Stuff

```
.PHONY: pdf maybepdf
EXISTING_PDF_FILES=${wildcard *.pdf}
maybepdf: ${EXISTING_PDF_FILES}
pdf: ${PDFFILES}
```

Ostermann's generic Makefile

- Here's the Makefile that I keep in my source directory:

```
CC=gcc
```

```
CFLAGS=-Wall -Werror -O3 -g
```

```
LDLIBS=-lnsl -lm -lsocket
```

```
default:
```

```
    @echo "Sorry, make doesn't work here!"
```

```
    @exit 1
```

Hints

- If the default rule that you want to run (say `myrule`) isn't at the top, then you just add a first rule that says

`default: myrule`

- Every Makefile should have a “`clean`” rule
 - Remove intermediate files, core dumps, executables, etc

- Read all about it on the web

<http://www.gnu.org/software/make/>

A Really Long Example

See the file

```
"prime:/home/osterman/lib/latex/Makefile.slides.example"
```