CS 501
Make and Other Build Tools

Nate Nystrom

September 13, 1999

Make

Make is a utility to help compile large programs.

It automatically determines which pieces of the program need recompilation.

- Recompilation is decided usually by last-modified time.

- Example: if foo.o is compiled from foo.c, make will rebuild foo.o only if foo.c is newer than the foo.o or if foo.o does not exist.

- There are build tools which recompile based on other criteria, for example when a function is changed.

Many variants on the theme

- GNUmake

  Has extensions for building on multiple platforms

- Microsoft nmake

  Built into Visual Studio, which creates the makefiles for you, but can also be used on the command line

- CM

  Compilation Manager for ML programs—does automatic dependency analysis, but only works with ML

- `java -cs`

  Recompiles the out-of-date source files while running the Java program—also does dependency analysis for you

How to use make

1. Create a makefile
   - Describes the relationships among files in the program
   - Specifies the sets of commands used to build the files that make up the program

2. Run `make` *target*
   - Builds a specific target specified in the makefile
   - Leaving out the target argument will build the first target in the makefile

A Simple Makefile

```
prog: main.o graph.o queue.o
	cc -o prog main.o graph.o queue.o

main.o: main.c
	cc -c -O main.c

graph.o: graph.c
	cc -c -O graph.c

queue.o: queue.c
	cc -c -O queue.c
```

Files and other targets on which a target is dependent are placed after the ":".

Commands to build the target are on successive lines.

Each line must begin with a tab.

Make will complain about a missing "separator character" if the tab is missing.

A Slightly More Complicated Simple Makefile

```
OBJ = main.o graph.o queue.o

prog: $(OBJ)
        cc -o $@ $(OBJ)

.c.o:
        cc -c -O $<

clean:
        -rm -f $(OBJ) prog
```

The `.c.o:` rule specifies how to build a .o file from a .c file.

`$@` is a special variable for the target file name.

`$<` is a special variable for the first dependency (the .c file in this case).

The `clean:` rule is a rule to delete the object files and prog. Run textttmake clean to execute the rule.

The "−" in front of `rm` tells make to ignore the exit status of the command. The rule will always succeed.

Makedepend

`makedepend` is a utility to analyze the dependencies in a set of C source files and append the rules to the end of a makefile.

Example: for the following makefile:

```
p: sort.o
        cc -o p0 $<
```

The command `makedepend sort.c` produces

```
p: sort.o
        cc -o p0 $<

# DO NOT DELETE

sort.o: /usr/include/stdio.h
sort.o: /usr/include/features.h
sort.o: /usr/include/gnu/stubs.h
sort.o: /usr/include/gnu/types.h
...
```

Makedepend (continued)

The makefile is edited in-place. A safer solution is:

```
makedepend -f deps.mk sort.c
```

and to use the following makefile:

```
p: sort.o
	cc -o p0 $<

include defs.mk
```

Portability

If building on multiple platforms, you might want to use a `configure` script.

- Search for the location of standard include and library files
- Tests if certain macros are defined
- Can check word-size, endianness, alignment
- `configure` scripts are built with `autoconf`.

GNU make also has some extensions for building for multiple targets:

```
ifeq ($(ARCH), sun4-solaris)
  CFLAGS += -xO4
else
  ifeq ($(ARCH), hppa1.1-hpux)
    CFLAGS += +O4
  endif
endif
```

Finally

This slide set was built with make:

```
TEXDIR = /usr/share/texmf/tex/latex

LATEX = export TEXINPUTS;\
        TEXINPUTS=.;\
        TEXINPUTS=$$TEXINPUTS:$(TEXDIR)/base;\
        TEXINPUTS=$$TEXINPUTS:$(TEXDIR)/cite;\
        TEXINPUTS=$$TEXINPUTS:$(TEXDIR)/graphics;\
        TEXINPUTS=$$TEXINPUTS:$(TEXDIR)/psnfss;\
        TEXINPUTS=$$TEXINPUTS:$(TEXDIR)/misc;\
        TEXINPUTS=$$TEXINPUTS:$(TEXDIR)/tools;\
        latex

.SUFFIXES: .tex .dvi .ps .aux .log

all: make.ps

.tex.dvi:
        $(LATEX) $<

.dvi.ps:
        dvips -f > $@ < $<

clean:
        rm -f *.ps *.log *.aux *.dvi *~
```