

### 3. MAKE UTILITY

Dave Gill  
gill@ucar.edu

### 3.1 Really Big Projects with make

Let us **make** man in our image, in our likeness, and let them rule over the fish of the sea and the birds of the air, over the livestock, over all the earth, and over all the creatures that move along the ground.

Gen 1:26

NCAR M<sup>3</sup>

### 3.1 UNIX make Utility

- Two-fold purpose: 1) overview of UNIX make command, and 2) use within MM5 system
- As programming complexity increases from a single source file to multiple includes, dependencies and conditional compilation, make becomes a necessity

NCAR M<sup>3</sup>

### 3.1 UNIX make Utility

- Only re-compiles what is required, recognizes tree-like structure of multiple source files for single executable

NCAR M<sup>3</sup>

### 3.2 make Functionality

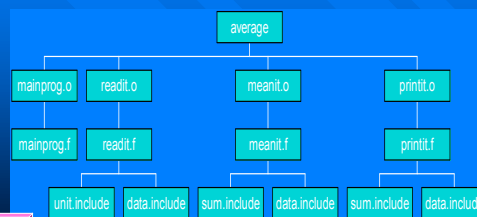
- Dependency is the underlying relationship between two files
- myprog.f → myprog.o → myprog.exe

myprog.f is a dependency file for the target myprog.o, and myprog.o is a dependency for the target myprog.exe

NCAR M<sup>3</sup>

### 3.2 make Functionality

See section 3.8, hierarchical tree dependency structure



NCAR M<sup>3</sup>

### 3.2 make Functionality

- Date and time of last modification used to determine whether dependency is out of date wrt target
- When improper time relationship exists, make uses rules to restore the target
- Hierarchy of include files, source, object and executable follows this sequential time dependency, leading to natural association of dependency timestamps

NCAR M<sup>3</sup>

### 3.3 The Makefile

- Makefile, makefile (make -f make.file)
- File read by make utility which contains **dependency relationships** and rules for updating targets (**generation commands**)
- Dependency relations – determine when a file must be regenerated
- Generation commands – how do you build out of date files

NCAR M<sup>3</sup>

### 3.4 Sample make Syntax

```
targetfile: dependencies
<tab>    command1
<tab>    command2

myprog.exe: mysource1.f mysource2.f
<tab>    f77 -o myprog.exe mysource1.f \
        mysource2.f
```

NCAR M<sup>3</sup>

### 3.4 Sample make Syntax

- Rule – begins in the first position of a line, with the following format  
`target : dependencies`
- If the files to the right are NEWER than the files to the left of the colon, a new target is rebuilt

NCAR M<sup>3</sup>

### 3.4 Sample make Syntax

- Dependency rule **MAY** be followed by one or more commands
- Commands must begin with a <tab> character to be recognized, otherwise they are seen as rules or macros, and then you are toast
- Commands are passed to the shell to execute (note this is **sh**, not **csh**)

NCAR M<sup>3</sup>

### 3.5 Macros

- Similar to shell variables, syntactically and semantically  
`MyFlags = -a -b -c -d`
- Usage of \$(MyFlags) expands to:  
`-a -b -c -d`
- The () may be omitted if the macro name is only a single character
- () are not required as in csh for an array

NCAR M<sup>3</sup>

### 3.6 Internal Macros

- Built in cool, short-cuts, sure to impress members of the digerati
- `$@` name of the current target
- `$<` dependency file, as if from implicit rule
- `$?` list of all dependencies newer than target
- `$*` basename of current target

NCAR M<sup>3</sup>

### 3.7 Default Suffixes and Rules

- Typical default rules for FORTRAN, shut off with “make -r” (“make -p” for the brave and curious)

```
.f.o:
<tab> $(FC) $(FFLAGS) -c $<

.f:
<tab> $(FC) $(FFLAGS) $(LDFLAGS) \
      $< -o $@
```

NCAR M<sup>3</sup>

### 3.7 Default Suffixes and Rules

- Typical default suffixes, typically at the beginning of a Makefile (or included near the top)

```
.SUFFIXES: .o .c .f
```

NCAR M<sup>3</sup>

### 3.7 Default Suffixes and Rules

- All of the MM5 system Fortran codes are set up to be processed by cpp.
- Not all Fortran compilers handle this in the same way.
- Gain uniformity through explicit rules:

```
.F.o:
<tab> $(RM) $@
<tab> $(CPP) $(CPPFLAGS) $*.F >! $*.f
<tab> $(FC) -c $(FFLAGS) $*.f
```

NCAR M<sup>3</sup>

### 3.8 Program Dependency Chart

- Head to that other slide, Dave

NCAR M<sup>3</sup>

### 3.9 Program Components

```
PROGRAM mainprog
CALL readit
CALL meanit
CALL printit
STOP 99999
END
```

NCAR M<sup>3</sup>

### 3.9 Program Components

```
SUBROUTINE readit
  Include 'unit.include'
  INCLUDE 'data.include'
  OPEN(iunit, file='input.data', ACCESS = &
    'sequential', FORM='FORMATTED')
  READ(iunit,FMT='(F10.4)') data
  RETURN
END
```

NCAR M<sup>3</sup>

### 3.9 Program Components

```
SUBROUTINE meanit
  INCLUDE 'data.include'
  INCLUDE 'sum.include'
  DO L=1,length
    sum = sum + data(L)
  END DO
  sum = sum / FLOAT(length)
END
```

NCAR M<sup>3</sup>

### 3.9 Program Components

```
SUBROUTINE printit
  INCLUDE 'data.include'
  INCLUDE 'sum.include'
  PRINT *,data(1:length)
  PRINT *, 'average = ',sum
END
```

NCAR M<sup>3</sup>

### 3.9 Program Components

- unit.include  
PARAMETER ( iunit=7 )
- sum.include  
COMMON /avg/ sum
- data.include  
PARAMETER ( length = 10 )  
COMMON /space/ data(length)

NCAR M<sup>3</sup>

### 3.10 makefile Example 1

```
average: mainprog.o readit.o meanit.o printit.o
  f77 -o average mainprog.o readit.o meanit.o printit.o
mainprog.o : mainprog.f
  f77 -c mainprog.f
readit.o : readit.f unit.include data.include
  f77 -c readit.f
meanit.o : meanit.f data.include sum.include
  f77 -c meanit.f
printit.o : printit.f data.include sum.include
  f77 -c printit.f
```

NCAR M<sup>3</sup>

### 3.10 makefile Example 2

```
average: mainprog.o readit.o meanit.o printit.o
  f77 -o $@ mainprog.o readit.o meanit.o printit.o
mainprog.o : mainprog.f
  f77 -c $<
readit.o : readit.f unit.include data.include
  f77 -c $<
meanit.o : meanit.f data.include sum.include
  f77 -c $*.f
printit.o : printit.f data.include sum.include
  f77 -c $*.f
```

NCAR M<sup>3</sup>

### 3.10 makefile Example 3

```

OBS = mainprog.o readit.o meanit.o printit.o
average: $(OBS)
    f77 -o $@ $(OBS)
readit.o : readit.f unit.include data.include
    f77 -c $<
meanit.o : meanit.f data.include sum.include
    f77 -c $*.f
printit.o : printit.f data.include sum.include
    f77 -c $*.f

```



NCAR M³

### 3.10 makefile Example 4

```

.o:
    rm -f $@
    f77 -c $*.f
OBS = mainprog.o readit.o meanit.o printit.o
average: $(OBS)
    f77 -o $@ $(OBS)

readit.o : unit.include data.include
meanit.o : data.include sum.include
printit.o : data.include sum.include

```



NCAR M³

### 3.11 MM5 make Commands

- Directly put macro definitions into the make command
- Precedence over values initialized as macros inside the makefile

```
make "FC=f90" "FFLAGS=-g"
```

NCAR M³

### 3.12 Top-level Makefile

- Example from TERRAIN, so just 2 levels: top and lower
- **.IGNORE**: same as **-i**
- **AR = ar ru** macros
- **default**: first target is default, any name
- **uname -a > .tmpfile** if test for vendor

NCAR M³

### 3.12 Top-level Makefile

- **grep CRAY .tmpfile**
- **if [ \$\$? = 0 ] ; then blah**
- **\$(MAKE) all** "all" is the low-level target
- Note **CPPFLAGS** includes NCARGGRAPHICS macro: **NCARG** or **NONCARG**

NCAR M³

### 3.12 Top-level Makefile

- **??? fi ; \** end of each if ; then block
- Second target is **terrain.deck**
- Must specifically name any target (other than first) to activate it
- **make terrain.deck**
- **clean**: typical target to zap detritus

NCAR M³

### 3.13 Low-level Makefile

`.IGNORE:` unnecessary with `$(MAKE)`  
`.SUFFIXES:` `.F .f .i .o` pseudo target, expl suffixes  
  
`.F.f:`  
`<tab> $(CPP) $(CPPFLAGS) $* F > $@`  
  
`OBS = ia.o ...` macro definition  
`SRC = $(OBS:.o=.f)` list of source files  
`cray dec hp ibm sgi sun default:` first target

NCAR M<sup>3</sup>

### 3.13 Low-level Makefile

- `@echo` "you need ..." easy error trapping
- `all:` target specified in top-level Makefile
- `terrain.exe data_area.exe rdem.exe` three dependency files
- `$(FC)` defined in top-level Makefile
- `anal2.o:` just an overall great name
- Note `crld.o` is listed more than once

NCAR M<sup>3</sup>