PCSE Lab 5

Make!

Cyrus Proctor
Victor Eijkhout

February 19, 2015

A bit about Make

- The *Make* utility helps you manage the building of projects
- *Make* is a *Unix* utility with a long history
- We will be using GNU *Make* – there are other variants
- This lab will focus on C and Fortran languages
- *Make* can be used with virtually any language; even LaTeX and TeX

Why Use *Make*?

- Need to automate compiling large projects
- Could use a batch file
  - All files will be recompiled every time
  - Compliation can take a long time
  - Inefficient when only a few files are changed
- *Make* only recompiles those files which have changed, or have a dependency that has changed

A Basic Fortran Example

foomain.F90

```
program test
  use testmod
  implicit none

  call dummy(1,2)

end program test
```

foomod.F90

```
module testmod

contains

  subroutine dummy(a,b)
    implicit none
    integer a,b
    write(*,*)a,b
  end subroutine dummy

end module
```

A Basic Fortran Example

Basic Fortran Example Makefile

```
fooprog : foomain.o foomod.o
  gfortran -o fooprog foomain.o foomod.o
foomain.o : foomain.F90 foomod.o
  gfortran -c foomain.F90
foomod.o : foomod.F90
  gfortran -c foomod.F90
clean :
  rm -f *.o *.mod fooprog
```

A Basic C Example

foo.c

```
#include "bar.h"

int c=3;
int d=4;

int main(){
  int a=2;
  return
      (bar(a*c*d));
}
```

bar.c

```
#include <math.h>
#include "bar.h"

int bar(int a){
  int b=sqrt(16);
  return(b*a);
}
```

bar.h

```
extern int
    bar(int);
```

A Basic C Example

Basic C Example Makefile

```
fooprog : foo.o bar.o
  gcc -o fooprog foo.o bar.o -lm
foo.o : foo.c
  gcc -c foo.c
bar.o : bar.c
  gcc -c bar.c
clean :
  rm -f *.o fooprog
```

Terminology

- **Target**
  - An output or intermediate file of the build process
- **Dependency**
  - A source or target that is depended upon by another source or target
- **Rule**
  - Description of how to produce a target given its dependencies and commands

Makefile Syntax: Defining Dependencies

- Defines list of dependencies dependencies for a target file
- Syntax

  ⟨target⟩ : ⟨dependencies⟩
- ⟨dependencies⟩ is a space-delimited list

Example

fooprog : foo.o bar.o

Makefile Syntax: Defining Explicit Rules

- Defines a list of dependencies and the commands to perform to produce a target file
- Syntax

  ⟨target⟩ : ⟨dependencies⟩
  ⟨TAB⟩ ⟨commands⟩
- ⟨dependencies⟩ is a space-delimited list
- ⟨commands⟩ MUST start with a **tab** character

Example

```
fooprog : foomain.o foomod.o
	gfortran -o fooprog foomain.o foomod.o
```

Makefile Syntax: Comments, Variables, and Echos

- A comment is preceded by a “#” sign
- A variable is defined by ⟨varname⟩ = ⟨value⟩
  - Once set, a variable must be enclosed by either ${} or $()
- An echo from inside the Makefile is preceded by a “@echo” symbol

Example

```
# I am a comment
myvar = banana
info :
    @echo “Hello World! $(myvar)”
```

Makefile Syntax: Special Automatic Characters

- $\$@$ The target. Use this in the link line for the main program
- $\$\hat{}$ The list of prerequisites. Use this also in the link line for the program
- $\$\langle$ The first prerequisite. Use this in the compile commands for the individual object files
- $\$*$ The stem of the target by stripping any extension

Makefile Syntax: Defining Implict Template Rules

- Defines the commands to perform to produce a type of target file from a type of source file
- Syntax

  ⟨%.target-extension⟩ : ⟨%.dependency-extension⟩
  ⟨TAB⟩ ⟨commands⟩
- ⟨commands⟩ MUST start with a **tab** character

Example 1

```
%.o : %.F90
	$(FC) -c $<
```

Example 2

```
$(THEPROGRAM) : $(FOBJS)
	$(FC) -o $@ $^
```

Makefile Syntax: More!

- There are an incredible amount of possibilities with *Make*
- A few other commonly used structures include
  - Conditionals
  - Wildcards
  - Phony Targets
  - Shell scripting inside the Makefile
  - Automatic Makefile creation with Automake / AutoConf / Libtool

References

- Victor Eijkhout, “Introduction to High Performance Scientific Computing”
- Gabe Cohn, “EE/CS 51 Make Lecture”