Advanced School in High Performance and GRID Computing -
Concepts and Applications

30 November - 11 December, 2009

From Source Code to Executable
(part 3)

A. Kohlmeyer
University of Pennsylvania
Philadelphia
USA
From Source Code to Executable
Part 3: Makefiles

Advanced School in High Performance
and GRID Computing

Axel Kohlmeyer

Institute for Computational Molecular Science
Overview / Make

- The idea behind make
- General Syntax
- Rules Examples
- Variables Examples
- Pattern Rules
- Special Rules
- Dependencies
- Conventions
Makefiles: Concepts

- Simplify building large code projects
- Speed up re-compile on small changes
- Consistent build command: make
- Platform specific configuration via Variable definitions
Makefiles: Syntax

- Rules:
  target: prerequisites
  command
  ^this must be a 'Tab' (|<- ->|)
- Variables:
  NAME= VALUE1 VALUE2 value3
- Comments:
  # this is a comment
- Special keywords:
  include linux.mk
# first target is default:
all: hello sqrt

hello: hello.c
    cc -o hello hello.c

sqrt: sqrt.o
    f77 -o sqrt sqrt.o

sqrt.o: sqrt.f
    f77 -o sqrt.o -c sqrt.f
Makefiles: Variables Examples

# uncomment as needed
CC= gcc
#CC= icc -i-static
LD=$(CC)
CFLAGS= -O2

hello: hello.o
  $(LD) -o hello hello.o

hello.o: hello.c
  $(CC)-c $(CFLAGS) hello.c
Makefiles: Automatic Variables

CC = gcc
CFLAGS = -O2

howdy: hello.o yall.o
   $(CC) -o $@ $^

hello.o: hello.c
   $(CC) -c $(CFLAGS) <$

yall.o: yall.c
   $(CC) -c $(CFLAGS) <$
Makefiles: Pattern Rules

OBJECTS=hello.o yall.o

howdy: $(OBJECTS)
   $(CC) -o $@ $^ 

hello.o: hello.c
yall.o: yall.c

.c.o:
   $(CC) -o $@ -c $(CFLAGS) $<
Makefiles: Special Targets

.SUFFIXES:
.SUFFIXES: .o .F

.PHONY: clean install

.F.o:

$(CPP) $(CPPFLAGS) $< -o $*.f
$(FC) -o $@ -c $(FFLAGS) $*.f

clean:

rm -f *.f *.o
Makefiles: Calling make

- Override Variables:
  - `make CC=icc CFLAGS=©-O2 -unroll©`
- Dry run (don't execute):
  - `make -n`
- Don't stop at errors (dangerous):
  - `make -i`
- Parallel make (requires careful design)
  - `make -j8`
- Alternative Makefile
  - `make -f make.pgi`
ARFLAGS= rcsv
LIBOBJ= tom.o dick.o harry.o

helloguys: hello.o libguys.a
 $(CC) -o $@ $< -L -lguys

libguys.a: $(LIBOBJ)
ar $(ARFLAGS) $@ $?

tom.o: tom.c guys.h
dick.o: dick.c guys.h
harry.o: harry.c guys.h
hello.o: hello.c guys.h
LIBSRC= tom.c dick.c harry.c
LIBOBJ= $(LIBSRC:.c=.o)
LIBDEP= $(LIBSRC:.c=.d)
.c.d:
   $(CC) $(CFLAGS) -MM $< > $@
   include $(LIBDEP)

alternatively (note, some makes require .depend to exist):
.depend dep:
   $(CC) $(CFLAGS) -MM $(LIBSRC) > .depend
   include .depend
Makefile Portability Caveats

- Always set the SHELL variable:
  
  ```bash
  SHELL=/bin/sh
  ```

  (make creates shell scripts from rules).

- GNU make has many features, that other make programs don't have and vice versa.

- Use only a minimal set of Unix commands: `cp, ls, ln, rm, mv, mkdir, touch, echo,...`

- Implement some standard 'phony' targets: `all, clean, install`