Software Productivity Tools

Purdue School on HPC

Sept. 4-5, 2008

Dave Seaman

(dseaman@purdue.edu)

Rosen Center for Advanced Computing

A Selection of Tools

- Tar
- Make
- PBS
- Version control systems
  - CVS
  - subversion

Tar

- Simple Archives
- Compressed Archives (*gzip*)
- Compressed Archives (*bzip2*)

Simple Archives

- Creating an Archive
- Exracting from an Archive
- Looking before You Leap

Creating an Archive

- tar cf myarchive.tar *files*
- Use *c* option to create an archive
- Use *f* option to specify a disk file (default is tape)

Extracting from an Archive

- tar xf myarchive.tar *[files…]*
- Use *x* option to extract files.
- Default is to extract all files in archive.

Looking before You Leap

- Many archives will extract into a subdirectory, but some write files in current directory.
- To be safe, extract into an empty directory.
- Use *t* option to see what's there:

tar tf myarchive.tar

Compressed Archives (gzip)

- We may use gzip to compress an archive:

tar cf mystuff.tar mystuff

gzip mystuff.tar

- A shortcut is available when using GNU tar:

tar zcf mystuff.tar.gz mystuff

Uncompressing with gunzip

- A compressed archive name may end in .tar.gz, .tar.Z, or .tgz.
- We may extract files by using a two-step process:

gunzip mystuff.tar.gz

tar xf mystuff.tar

Uncompressing Shortcuts

- We may use zcat to unzip and pipe the result into tar:

```
zcat mystuff.tar.gz | tar xf -
```

- GNU tar has the z option for zip/unzip:

```
tar zxf mystuff.tar.gz
```

Bzip2 Compression

- Bzip2 is used much like gzip, but tends to have better compression ratios at the cost of slightly more time.

bzip2 mystuff.tar

bunzip2 mystuff.tar.bz2

bzcat mystuff.tar.bz2 | tar xf -

GNU tar and bzip2

- GNU tar uses the *j* option to invoke bzip2/bunzip2:

tar jcf mystuff.tar.bz2 mystuff

tar jxf mystuff.tar.bz2

Make

- Automates maintaining files that depend on other files.
- May be used for programs, documents, or arbitrary development projects.
- Instructions contained in *Makefile* control steps needed to keep files up to date.

Simple Makefile


```
# variable definitions
PROG    = graph
SRC     = main.c graph.c
OBJ     = main.o graph.o
CC      = icc
CFLAGS = -O2
LOADLIBES = -lm
```

Simple Makefile (cont.)


```
all:        $(PROG)

$(PROG):    $(OBJ)

   $(LINK.c) -o $@ $(OBJ) $(LOADLIBES)

# additional dependency rule

main.o graph.o:    graph.h
```

Rules

target: prereq1 prereq2 …

command1

command2

[…]

Each command must be preceded by a tab.

Invoking make

- make [-f *makefile*] [*target*] [*var=defs*]…
- Default *makefile* names (in order):
  - GNUmakefile
  - makefile
  - Makefile
- Default *target* is first one in *makefile.*
- Variable definitions on command line override values given in *makefile.*

Simple Makefile


```
# variable definitions
PROG    = graph
SRC     = main.c graph.c
OBJ     = main.o graph.o
CC      = icc
CFLAGS = -O2
LOADLIBES = -lm
```

Simple Rule

- Our simple *makefile* contains as its first rule:

```
all:    $(PROG)
```

- This means `make` is equivalent to `make all`.
- There is one prerequisite: the target *all* depends on $(PROG), which involves a variable substitution:

```
PROG = graph
```

Variable Substitution

- Variable name is preceded by $ and is enclosed in parentheses: $(PROG)
- Parentheses may be omitted only if the name is a single character. Some have standard meanings, such as $@ for the target currently being updated.
- Braces may be used instead of parentheses: ${PROG}

Default Rules

- The Makefile says $(PROG) depends on $(OBJ) and gives a command to use for creating $(PROG):

```
$(PROG):    $(OBJ)
    $(LINK.c) -o $@ $(OBJ) $(LOADLIBES)
```

- However, no rules are stated for producing the $(OBJ) files.

The Default Rules

- Since no commands were found for making the object files, *make* used the default rule:

```
%.o:  %.c

# built-in commands to make a .o file

   $(COMPILE.c) $(OUTPUT_OPTION) $<
```

- The rule beginning with %.o: %.c is an example of a *pattern rule*.

Default Compilation Rule

- $(COMPILE.c) is predefined by *make*:

```
COMPILE.c = $(CC) $(CFLAGS) $(CPPFLAGS) \
    $(TARGET_ARCH) -c
```

- You may define the variables (CC, CFLAGS, …) as needed in order to customize the command.

Some Standard Variables

$(COMPILE.c)	Command for compiling C programs
$(LINK.c)	Command for linking C programs
$<	First prerequisite
$@	Target being updated
$(CC)	C compiler
$(CFLAGS)	Flags for C compiler
$(CPPFLAGS)	Flags for C preprocessor
$(CXX)	C++ compiler

Commands Produced

- Running *make* with these rules may produce several commands:

```
icc -O2 -c -o main.o main.c

icc -O2 -c -o graph.o graph.c

icc -O2 -o graph main.o graph.o -lm
```

- Only the necessary steps are actually performed, depending on which files are older than their prerequisites.

Specifying Dependencies

- The files *main.c* and *graph.c* both #include "graph.h". If we make changes in *graph.h*, then both *main.o* and *graph.o* need to be updated (recompiled).
- No commands are needed here. We only need the dependency to force recompilation using the built-in rule.

Other Useful Flags with *make*

-n	Display the commands that will be generated, but don't execute.
-v	Show verbose output.
-p	Show complete database of rules, including built-in rules.

For More Information

- http://www.gnu.org/software/make/manual/make.html

Portable Batch System (PBS)

- Submits jobs to run on a cluster.
- Schedules jobs to run when resources become available.
- Manages running of jobs and returns output to user.
- Allows interactive execution.

Useful PBS Commands

qsub	Submit PBS job
qdel	Delete PBS job
qstat	Display status of PBS jobs, queues, or servers
pbsnodes	Query PBS host status

The *qsub* Command

- To submit the script *myscript* as a batch job, use:

qsub *[options] myscript*

- Options may be embedded in a script file or specified on the command line.
- To submit an interactive job, use:

qsub -I *[options]*

- Use man qsub to learn more.

Using Options

- Command line example:

qsub -q standby -l select=2:ncpus=4:mpiprocs=4 myscript

- Batch file example:

#PBS -q standby

#PBS -l select=2:ncpus=4:mpiprocs=4

#PBS -l walltime=4:00:00

- The command line takes precedence.

Some Useful *qsub* Flags

-l *resource_list*	Specify resources required for job.
-q queue_name	Specify queue in which to run job.
-m *mail_options*	Specify mail to be sent at specific times.
-S *shell_path*	Specify shell to use (default is login shell).

Resources

- Request two nodes with 4 processors per node for running an MPI program:

-l select=2:ncpus=4:mpiprocs=4

- Request one node with 4 processors and 2 GB memory for running an OpenMP program:

-l select=1:ncpus=4:ompthreads=4:mem=2g

- Request an extended time limit:

-l walltime=24:00:00

Queues

- Use qstat -a to see available queues.
- Not all queues are available to every user.
- Different queues generally have different resource limits.
- Specify -q queue_name when submitting job.

Mail Options

-mn	No mail is sent.
-ma	Mail is sent if job is aborted.
-mb	Mail is sent when job begins execution.
-me	Mail is sent when job ends execution.

A Simple Job Script


```
#PBS -S /bin/csh -l nodes=4:ppn=1

# Change to directory where job was submitted.
cd $PBS_O_WORKDIR

# Make sure mpich2 is loaded.
module load mpich2-intel

# Run the examples.
make test
```

Mail Options

-mn	No mail is sent.
-ma	Mail is sent if job is aborted.
-mb	Mail is sent when job begins execution.
-me	Mail is sent when job ends execution.