D A S U N A N K A
R A B B I N A K A
F A R A K O M A I

Make & Makefiles

S. AbdusSalam

Department of Physics,
Shahid Beheshti University,
Islamic Republic of Iran

ICTP SMR 3107, 2017-March-06 (1395-12-17)

Make: A (Software) Build Automation Tool

- Automatic dependency-tracking build utility
- Uses Makefiles
- Examples
- Hands-on exercise

Build Dependency & Dependency-tracking

- Target dependencies

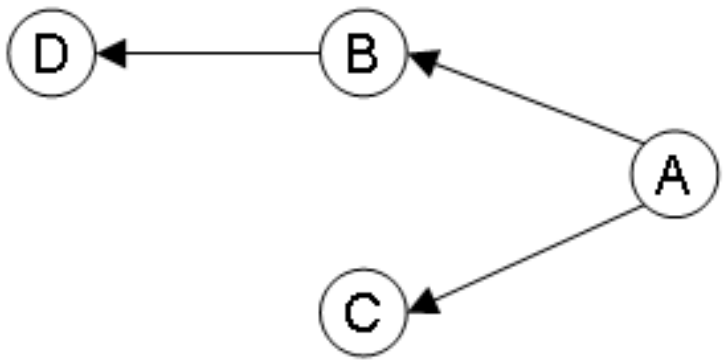

Figure : A picture showing a target to be built, file A, and its dependencies. A depends on B and C. B depends on D.

Build Dependency & Dependency-tracking

- Target dependencies, a practical example

```
1   echo "This is file D." > D
2
3   echo "File C is just an other file." > C
4
5   echo "File B depends on file D." > B
6   cat D >> B
7
8   echo "File A is our target, to be built." > A
9   echo "But it depends on file B and C." >> A
10  echo >> A
11  cat B C >> A
12  echo >> A
13
14  echo "Done. The target A is now built." >> A
```

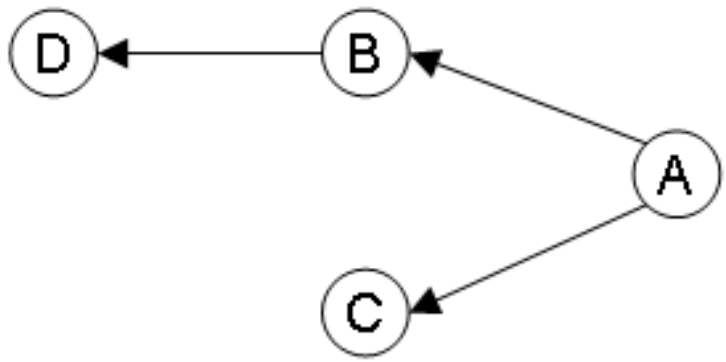

Figure : A picture showing a target to be built, file A, and its dependencies. A depends on B and C. B depends on D.

Build Dependency & Dependency-tracking

- Target dependencies, a practical example

```
1   echo "This is file D." > D
2
3   echo "File C is just an other file." > C
4
5   echo "File B depends on file D." > B
6   cat D >> B
7
8   echo "File A is our target, to be built." > A
9   echo "But it depends on file B and C." >> A
10  echo >> A
11  cat B C >> A
12  echo >> A
13
14  echo "Done. The target A is now built." >> A
```

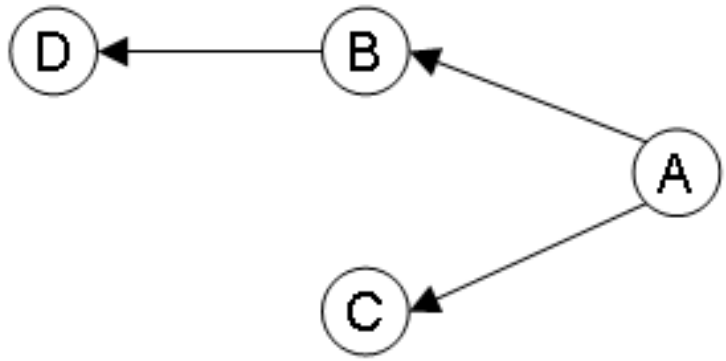

Figure : A picture showing a target to be built, file A, and its dependencies. A depends on B and C. B depends on D.

- Target examples: analysis plots, library/binary compilation
- Dependency-tracking: modification time-stamps
- make: track of targets, dependencies, and time-stamp relations.
- Makefile: file with description of how make should do its work.

Make &Makefile

make manages the compilation, linking, and/or execution of commands for buiding targets within a project.

The instructions of how make should perform the management are written in a text file usually called makefile.

The targets to be made within a project can be executable binaries, library files, publication-quality plots for an analysis, et cetera.

make compress the chains of commands for compilations, linkings, running an analysis, production of plots, etc to a single one.

Updating a target: only changed files and the targets that depend on the changed files are re-compiled or made again.

Make &Makefile

make looks for a makefile in the directory where it is called:

```
make
```

Otherwise use the "-f" flag:

```
make -f [nameOffile]
```

Inside a makefile, any things following the character "#" are meant to be comments. That is, make will not execute it.

```
# a comment line
```

Apart from comments, a makefile is made of rules. A rule is made of (1) dependency-line, and (2) a set of commands, in the form:

```
target [target ...]: [dependencie/prerequisite list ...]
    [command 1]      # tab is required before command
    .
    .
    .
    [command n]
```

Makefile: Rules

- Save the following in a file with name, rulesExample

```
1   # makefile dependency rules
2
3   A: B C
4           echo "File A is our target, to be built." > A
5           echo "But it depends on file B and C." >> A
6           echo >> A
7           cat B C >> A
8           echo >> A
9           echo "Done. The target A is now built." >> A
10
11  B: D
12          echo ‘‘File B depends on File D.’’ > B
13          cat D >> B
14
15  D:
16          echo ‘‘This is file D.’’ > D
17
18  C:
19          echo ‘‘File C is just another file.’’ > C
```

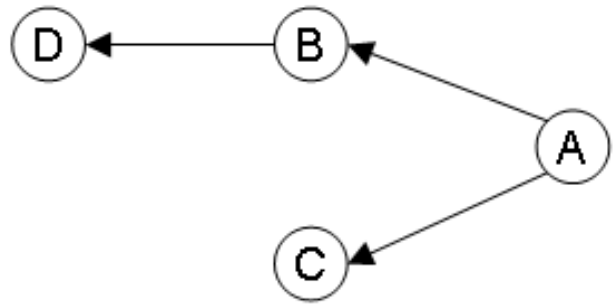

Figure : A picture showing a target to be built, file A, and its dependencies. A depends on B and C. B depends on D.

- Any of the following shell commands will seek to build A

```
make -f rulesExample

make -f rulesExample A
```

- For building C only, run the following command

```
make -f rulesExample C
```

Makefile: Example 1

```
1   # An example makefile
2
3   all: A
4
5   D:
6           echo ‘‘This is file D.’’ > D
7
8   C:
9           echo ‘‘File C is just another file.’’ > C
10
11  B: D
12          echo ‘‘File B depends on File D.’’ > B
13
14  A: B C
15          cat B C > A
16          echo >> A
17
18  clean:
19          rm A B C D
```

If you save the makefile with the name makefile then:

```
make        # builds A

make D      # builds D only
```

Makefile: Macros

Consider the following section of a makefile with one rule.

```
# a comment line
CXX = g++
salam.x: salam.cpp
        $(CXX) salam.cpp -o salam.x
```

Here "=" makes CXX a macro. The "$" and parenthesis in "$(CXX)" expands the macro; that is, replaces CXX with g++.

Macros containing simple strings as in the above example are also called "variables".

Built-in macros examples: (0) $@ refers to the target. (1) $< refers to the first prerequisite. (2) $^ means all prerequisites.

```
# a comment line
CXX = g++
salam.x: salam.cpp
        $(CXX) $< -o $@
```

Makefile: Pattern Rules

Pattern rule uses the character '%' for relating target and prerequisite names. For instance, the following will build object files with same name as the corresponding ".cpp" source files.

```
# a comment line
CXX = g++
%.o: %.cpp
        $(CXX) $< -o $@
```

Here, each file ending with ".cpp" is a prerequisite to a corresponding target with the same name as the prerequisite but ending with ".o".

Comment related to next example: The "back-slash" character is used for continuing long lines.

Reference:
www.gnu.org/software/make/manual/make.html

Makefile: Example 2

The source files are at:
http://facultymembers.sbu.ac.ir/abdussalam/gslpp.tgz

```
1   CXX = g++
2   CXXFLAGS = -c -I./include
3   SRC = src
4   objects = gslpp_complex.o gslpp_matrix_double.o\
5           gslpp_vector_double.o gslpp_matrix_complex.o\
6           gslpp_vector_complex.o
7   AR = ar r
8   GSL_INC = -I/usr/local/include
9   GSL_LINK = -L/usr/local/lib -lgsl -lgslcblas
10
11  all: libgslpp.a test.x
12
13  %.o: $(SRC)/%.cpp
14          $(CXX) $(CXXFLAGS) $< -o $@
15
16  libgslpp.a: $(objects)
17          $(AR) $@ $^
18  # $@ target name
19  # $< 1st prerequisite or dependent source
20  # $^ all prerequisites
21  # see https://www.gnu.org/software/make/manual/html_node/Automatic-Variables.html
22
23  test.x: tests/newsimpletest.cpp libgslpp.a
24          $(CXX) $(GSL_INC) $(CXXFLAGS) $< -o test.o
25          $(CXX) test.o -L./ -lgslpp $(GSL_LINK) -o $@
26          chmod +x $@
27
28  clean:
29          rm *.o test.x libgslpp.a
```

Time for Hands-on Excersice