Make / Makefile

Stanley Yao
Computer Science Department
University of Arizona

Difficulty

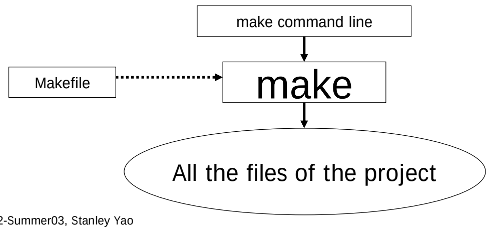
- A large C project
  - 30 source directories
  - 50 files in each directory on average
- Difficulty 1: Please compile all the files in all the directory and then link all object files into an executable
- Difficulty 2: I change only 10 files scattered in 5 directories. Then re-compile the files and re-link the object files.
- Difficulty 3: Change a .h file. Only the .c files that includes the .h file needs to be re-compiled and re-linking is also needed.

Difficulty (cont.)

- Difficulty 4: Installing a big software requires copying 30 files of different categories (executables, libraries, header files, man pages, docs, etc.) into several different locations
- Difficulty 5: After installing, you suddenly want to install the software to a different location in the file system.
- Difficulty 6: Uninstalling a big software requires deleting lots of files in different locations too.
- Etc.
- **We need to make all these automatic!!!**

make

- Control the build of large projects automatically
- Minimize the set of files to be compiled
- With the aid of make, you can handle compile, link, install, uninstall, test, etc.



Basic construct of Makefile

- **Makefile consists of a collection of make rules:**

```
target: src1 src2 ...
<tab>command1
<tab>command2
...
```

- If any of the files “src1”, “src2”, … are updated more recently than the current “target”, “target” will be rebuilt by the “command1”, “command2”, …

Basic make command

- make target
  - Only build target defined in the Makefile
- make target1 target2 …
  - Build target1, target2, … defined in the Makefile
- make
  - Build the first target defined in the Makefile
  - We usually write a helper “all” target as the first target depending on a set of targets we really want to build when calling “make”
- make –n [target list]
  - No execution mode.
- make –f file
  - Specify the makfile explicitly

Variables

- No need to declare variables, assignment does the declaration
- Assignment
  - CC = gcc
- Reference
  - $(CC) hello.c

Variables (cont.)

- Built-in Variables
  - $@: current target
  - $<: first prereq
  - $^: all prereq
- Variable Substitution
  - SRCS = $(OBJS:.o=.c)

Old-fashioned Suffix Rules

```
.c.o:
    $(CC) -c $(CFLAGS) -o $@ $<
```

- Suffix Rules are obsolete because pattern rules are more general and clearer .

Pattern

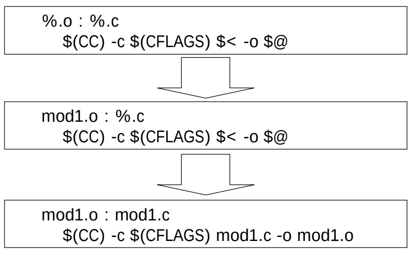
- Pattern
```
%.o : %.c
    $(CC) -c $(CFLAGS) $< -o $@
```
- Build-in Rules: predefined in make
```
%.o : %.c

    $(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@
```
- Variables Used by Built-in Rules
  - CC: Program for compiling C programs; default `cc'.
  - AR: Archive-maintaining program; default `ar'.
- Additional Variables Used by Built-in Rules
  - CFLAGS: Extra flags to give to the C compiler.

How Patterns Match

- A Pattern: <u>prefix</u>+%+<u>suffix</u> (where both <u>prefix</u> and <u>suffix</u> may be empty)
- Pattern matching is similar to wildcard matching (% is like wilecard *)
- The text between <u>prefix</u> and <u>suffix</u> is called <u>stem</u>. (e.g. test.o matchs %.o pattern, test is <u>stem</u>)
- Prereq's are transformed by substituting the <u>stem</u> for %

How Patterns Match (cont.)



```
%.o : %.c
    $(CC) -c $(CFLAGS) $< -o $@
```

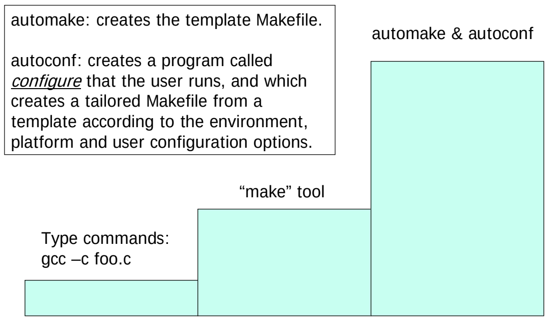
```
mod1.o : %.c
    $(CC) -c $(CFLAGS) $< -o $@
```

```
mod1.o : mod1.c
    $(CC) -c $(CFLAGS) mod1.c -o mod1.o
```

Calculating Dependencies

- Include files
  - Large number of includes
  - Nested includes
- “makedepend” is a program that calculates the header file dependencies for the specified .c files, and appends them to the end of the Makefile.

Further Automation


automake: creates the template Makefile.

autoconf: creates a program called *configure* that the user runs, and which creates a tailored Makefile from a template according to the environment, platform and user configuration options.

Acknowledgement

- John H. Hartman, *Classnotes for Csc352-Spring03*, CS Dept., University of Arizona, 2003
- Free Software Foundation, *GNU make Manual (version 3.80)*, December 25, 2002 http://www.gnu.org/manual/make-3.80/
- Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language (2nd Ed.)*, Prentice Hall, 1988