Make: Managing Compiled Programs

K. Cooper

2012

Types of programs

- Interpreted
  - Line by line
  - Slow
- Compiled
  - Translated to machine code
  - Faster
- Compiled programs very platform-dependent

Process

1. Write the program in some language
2. Compile: translate to *object code*
3. Link: combine machine codes into one *binary*
4. Load: load into memory with libraries and run
5. Shared objects...

Problems

- Platform dependence
- Many files
- Many authors

Make

- Watches file updates
- Allows compiler switches to be maintained in one place
- Provides code maintenance utility

Makefile

- Make looks for `Makefile`
- After that looks for `makefile`
- After that you can use a `-f` command line switch

Target

- target name, starting in first column.
- end the target name with colon
- list of file dependencies follows
- next line starts with `<Tab>`
- `target:  hello.c`
  `<Tab here>/usr/bin/gcc hello.c`
- execute the commands using `make target`

Notes

- The first target is the default
- Can be more than one command
- <Tab>is the *separator*
- Every line executes *in its own shell*
- A backslash (\) at the end of the line allows multi-line commands
- Nice error message if dependencies not met

Related targets

- One target can list other targets as dependencies
- If make sees a target dependency, it goes to that target

Related targets

- One target can list other targets as dependencies
- If make sees a target dependency, it goes to that target

Related targets

- One target can list other targets as dependencies
- If make sees a target dependency, it goes to that target

```
all:  hello.o
    /usr/bin/gcc -o hello hello.o
hello.o:
    /usr/bin/gcc -c hello.c
```

Non-compilation targets

- A target does not necessarily have to compile files.
- Typically, there is a `clean` target - removes object files

Non-compilation targets

- A target does not necessarily have to compile files.
- Typically, there is a `clean` target - removes object files

Non-compilation targets

- A target does not necessarily have to compile files.
- Typically, there is a `clean` target - removes object files

```
clean:
    rm -f *.o
```

Suffixes

- File suffixes typically indicate file function
- Make can use suffixes for generic actions
- `.SUFFIX` target lets you define suffixes
- E.g. `.c.o` target gives generic action for any file with .c or .o suffix
- `$<` denotes the *current* prerequisite

Suffixes

- File suffixes typically indicate file function
- Make can use suffixes for generic actions
- `.SUFFIX` target lets you define suffixes
- E.g. `.c.o` target gives generic action for any file with .c or .o suffix
- `$<` denotes the *current* prerequisite

Suffixes

- File suffixes typically indicate file function
- Make can use suffixes for generic actions
- `.SUFFIX` target lets you define suffixes
- E.g. `.c.o` target gives generic action for any file with .c or .o suffix
- `$<` denotes the *current* prerequisite

Suffixes

- File suffixes typically indicate file function
- Make can use suffixes for generic actions
- `.SUFFIX` target lets you define suffixes
- E.g. `.c.o` target gives generic action for any file with .c or .o suffix
- `$<` denotes the *current* prerequisite

Suffixes

- File suffixes typically indicate file function
- Make can use suffixes for generic actions
- `.SUFFIX` target lets you define suffixes
- E.g. `.c.o` target gives generic action for any file with .c or .o suffix
- `$<` denotes the *current* prerequisite

Suffixes

- File suffixes typically indicate file function
- Make can use suffixes for generic actions
- `.SUFFIX` target lets you define suffixes
- E.g. `.c.o` target gives generic action for any file with .c or .o suffix
- `$<` denotes the *current* prerequisite

```
.SUFFIX: .c .o
.c.o:
    /usr/bin/gcc -c $<
```

Macros

- There are many objects used repeatedly
- Want to change things only once
- Allow changes from command line
- `MACRONAME = some characters`
- refer to this using `${MACRONAME}`

Predefined Macros

- Note that `$@` denotes the current target
- Note that `$?` denotes a list of prerequisites newer than the current target
- Using two dollar signs on these variables makes them suitable for dependency lines