Makefiles and the GNU Build System

Tristan Miller

February 2025

Makefiles

Compiling C programs

Two steps:

- C compiler converts each C file (`.c`) to an *object module* (`.o`)
  - Preprocessor: `#includes` other files, expands macros, conditional compilation
- Linker combines multiple object modules into an executable or shared library

```
gcc -c main.c -o main.o
gcc -c parser.c -o parser.o
gcc -c network.c -o network.o
gcc main.o parser.o network.o -o myprogram
```

Build automation

- Naïve solution: write a shell script to automate this
  - Rebuild everything on every change

Build automation

- Naïve solution: write a shell script to automate this
  - Rebuild everything on every change
- Check if a file is up to date before building it
  - If a file gets rebuilt, also rebuild everything that depends on it

make

- Created (just like everything else worthwhile) in Bell Labs in the 1970s
- Multiple different popular implementations
  - GNU make
  - BSD make
  - Microsoft nmake
- These are all somewhat different – we will focus on GNU make

`make`

- Configure using a `Makefile`
- List of *rules* – each rule says how to "make" a file
- Rules specify a *target*, some *prerequisites*, and a *recipe*
- `make` will rebuild out-of-date prerequisites before building the target

Variables

- Similar to variables in shell scripts: `$(var)` expands to the contents of `var`
- Set a variable with = (lazy) or := (immediate)
- Fill in an unset variable with ?=
- Append to a variable with +=

Rules

- Phony rules: don't need to actually produce a file
- Pattern rules can be used to reduce repetition
  - `%.o: %.c`
- Wildcards (`*.h`) expand to all matching files
- Builtin implicit rules for common tasks (ex. object file from C)

Rules

- Multiple targets: the rule is copied for each target

```
out1 out2: file
    cp file $@
```

  is the same as

```
out1: file
    cp file $@
out2: file
    cp file $@
```

- Use `&:` instead of `:` to group the targets into one rule

Rules

- Order only prerequisites

  ```
  target: r1 r2 | r3
      ...
  ```

  `r3` will be built before `target`

Rules

- Static pattern rules: pattern rules restricted by a list

  ```
  $(objects): %.o: %.c
      ...
  ```

  Applies the pattern rule only to items in `$(objects)` matching `%.o`, with the prerequisite `%.c`

Functions

- Use a function: `$(function arg1,arg2,...)`

Functions

- Use a function: `$(function arg1,arg2,...)`
- `wildcard`: expands the wildcard characters `*`, `?`, and `[...]`
    - `$(wildcard *.cpp)`

Functions

- Use a function: `$(function arg1,arg2,...)`
- `wildcard`: expands the wildcard characters `*`, `?`, and `[...]`
  - `$(wildcard *.cpp)`
- `patsubst`: string replacement according to a pattern
  - `$(patsubst %.c,%.o,$(cfiles))`

Functions

- Use a function: `$(function arg1,arg2,...)`
- `wildcard`: expands the wildcard characters `*`, `?`, and `[...]`
  - `$(wildcard *.cpp)`
- `patsubst`: string replacement according to a pattern
  - `$(patsubst %.c,%.o,$(cfiles))`
- `shell`: run shell commands
  - `$(shell find -name *.html -type f)`

Recipes

- Each line of the recipe is a shell command
- `make` expands variables first before running (use `$$` to escape `$`)
- Rules starting with `@` won't get echoed
- Split a command between lines with backslashes

Recipes

```
target:
    echo "home directory: $$HOME"
    @if [ "$(number)" -eq 5 ]; then \
        echo "it is five" > target \
    else \
        echo "it is not five" > target \
    fi
```

Recursion

- Use `include` to include the rules from other makefiles
- Run other makefiles: `cd subdir && $(MAKE)`
- Remake your own makefile

`make` flags

- `-j/-jobs`: set the number of threads make can use
- `-f/--file/--makefile`: specify a makefile (default `Makefile` or `makefile`)
- `-I/--include-dir`: search this directory for more makefiles
- `-n/--dry-run`: print which rules make would run without running them
- `-q/--question`: check if target is up to date without building
- `--print-data-base`: print rules and variables make loads from reading a makefile

GNU Autotools

Building software is hard

We want to be able to build the same software on multiple different systems

- Different kernels

Building software is hard

We want to be able to build the same software on multiple different systems

- Different kernels
- Different CPU architectures

Building software is hard

We want to be able to build the same software on multiple different systems

- Different kernels
- Different CPU architectures
- Different versions of an architecture

Building software is hard

We want to be able to build the same software on multiple different systems

- Different kernels
- Different CPU architectures
- Different versions of an architecture
- Different C standard libraries

Building software is hard

We want to be able to build the same software on multiple different systems

- Different kernels
- Different CPU architectures
- Different versions of an architecture
- Different C standard libraries
- Different file system layouts

Building software is hard

We want to be able to build the same software on multiple different systems

- Different kernels
- Different CPU architectures
- Different versions of an architecture
- Different C standard libraries
- Different file system layouts
- Different executable formats

Building software is hard

We want to be able to build the same software on multiple different systems

- Different kernels
- Different CPU architectures
- Different versions of an architecture
- Different C standard libraries
- Different file system layouts
- Different executable formats
- Different compilation options

Building software is hard

We want to be able to build the same software on multiple different systems

- Different kernels
- Different CPU architectures
- Different versions of an architecture
- Different C standard libraries
- Different file system layouts
- Different executable formats
- Different compilation options
- Also, people don't like writing makefiles

GNU Autoconf

- Before building software, we should check that the environment is good
  - Dependencies are installed
  - Tools support the features we need
- GNU Autoconf generates a `configure` script that does this
- Configuration given in the `configure.ac` file (m4)

GNU Automake

- Used with Autoconf to generate makefiles
- Automatically determines dependencies and updates them when files change
- Configured with `Makefile.am` files

Alternatives

- CMake
- Meson
- I have never used these but people seem to prefer them to Autotools