Magic of Makefiles

Guy Nankivell

History of Makefiles

- Born in 1976 (43 years ago)
- Bell Labs Creation
- “Printable, debuggable understandable stuff”
- Designed to replace a bevvy of shell scripts

How it works

This is alarmingly simple in its design;

It leverages the syscall `stat` to check when the output file was created. If there are any edits made on files that are *newer* than this, it recompiles the files.

This gives the user really granular control over the recompilation phase.

Requirements of a Build System

- Portable
- Efficient (Stop unecessary recompilations)
- Language Agnostic (nested projects)
- Fast
- Modular
- Flexible

Little Note...

What I henceforth refer to as ‘make’ is the GNU implementation. This is due to being the most common reference for `make` in the wild.

Cornerstone Features of Make

- Parallel
- Text Replacement
- Ubiquity
- Everyone knows how to use it (Just type `make`!!)

Contextualising my Usage

- C
- Fortran
- LaTeX
- x86-64 Assembly
- Installation Scripts

Have also used it on...

- Java
- F/C#
- Python

Anatomy of a Makefile

```
CONSTANTS = this that
JUNK =  *.aux *.snm *.log *.dvi

target: deps
 recipe to build

clean:
 rm $(JUNK)

.PHONY: clean
```

Okay, but can it work for us?

It is used as the build system for the Linux kernel so it must be doing something right... How about on modern projects?

"Those who do not understand Unix are condemned to reinvent it, poorly."

- Good Readability
- Easy to implement complex logic
- Finding it hard is a symptom of a bad mindset
- MASSIVE community using it for js (more than 3 people)

Node Makefile?

```
PATH  := node_modules/.bin:$(PATH)

source_files     := $(wildcard lib/*.ts)
build_files      := $(source_files:lib/%.ts=dist/%.js)
PACKAGE          := build/my-package-1.0.0.tgz

.PHONY: all

all: $(PACKAGE)

$(build_files): $(source_files) package.json
	npm i
	tsc

$(PACKAGE): $(build_files) .npmignore
	@mkdir -p $(dir $@)
	@cd $(dir $@) && npm pack $(CURDIR)
```

Drawbacks

- Arcane Syntax
- No Dependency Resolution

Compiling this presentation...

```
CC =              pdflatex
SRC =             talk.tex
OUT_DIR =         tmp
CFLAGS =          -output-directory $(OUT_DIR)
FILE_REDIR =      /dev/null
FNAME =           out.pdf

all: $(OUT_DIR)
    @ $(CC) $(CFLAGS) $(SRC) > $(FILE_REDIR)
    @ mv $(OUT_DIR)/*.pdf ./$(FNAME)

clean:
    @ $(RM) -r $(OUT_DIR)/

$(OUT_DIR):
    @ mkdir -p $(OUT_DIR)/
```

Plays nicely with `vim`

From command mode in `vim` all one has to do to action the `Makefile` in the same directory to which your session is running is;

```
:make
```

Which keeps your editing session alive and allows you to view the output of the command and then you need only press enter to drop back into editing mode

Finis.

Questions?