Build Tools

Joseph Hallett

January 13, 2023

University of
BRISTOL

So what's all this about?

An *awful* lot of the things we do with a computer are about *format shifting*
We do this when we compile code:

- `cc -c library.c -o library.o`
- `cc hello.c library.o -o hello`

When we archive files:

- `zip -r coursework.zip coursework`

When we draw figures:

- `dot -Tpdf flowchart.dot -O flowchart.pdf`

Can we automate this?

YES!

We *could* write a shellscript and stick all the tasks in one place...

```
#! /usr/bin/env bash
cc -c library.c -o library.o
cc hello.c library.o -o hello
zip -r coursework.zip coursework
dot -Tpdf flowchart.dot -O flowchart.pdf
```

But can we do better than this?

- ▶ Do we really need to recompile the C program if only our flowchart has changed?
- ▶ Can we generalise build patterns?

Make

Make in an *ancient* tool for automating builds.

- ▶ Developed by *Stuart Feldman* in 1976
- ▶ Takes *rules* which tell you how to build files
- ▶ Then follows them to build the things you need!

Two main dialects of it (nowadays):

BSD Make More old fashioned, POSIX

GNU Make More featureful, default on Linux

In practice, unless your developing a BSD *every one* uses GNU Make

- ▶ If you're on a Mac, or BSD box install GNU Make and try `gmake` if things don't work

Makefiles

Rules for Make are placed into a *Makefile* and look like the following:

```
hello: hello.c library.o
        cc -o hello hello.c library.o

library.o: library.c
        cc -c -o library.o library.c

coursework.zip: coursework
        zip -r coursework.zip coursework

flowchart.pdf: flowchart.dot
        dot -Tpdf flowchart.dot -O flowchart.pdf
```

If you ask make to build hello it will figure out what it needs to do:

```
$ make hello
cc -c -o library.o library.c
cc -o hello hello.c library.o
```

Making changes

If you alter files... Make is smart enough to only rerun the steps you need:
For example if you edit `hello.c` and rebuild:

```
$ make hello
cc -o hello hello.c library.o
```

But if you edit `library.c` it can figure out it needs to rebuild *everything*

```
$ make hello
cc -c -o library.o library.c
cc -o hello hello.c library.o
```

Phony targets

As well as rules for how to make files you can have *phony* targets that don't depend on files but just tell make what to do when they're run

Often a `Makefile` will include a phony:

all	typically first rule in a file (or marked `.default`): depends on everything you'd like to build
clean	deletes all generated files
install	installs the program

```
$ make
cc -c -o library.o library.c
cc -o hello.o hello.c library.o
zip -r coursework.zip coursework
dot -Tpdf flowchart.dot -O flowchart.pdf
```

```
.PHONY: all clean

all: hello coursework.zip flowchart.pdf

clean:
        git clean -dfx

hello: hello.c library.o
        cc -o hello hello.c library.o

library.o: library.c
        cc -c -o library.o library.c

coursework.zip: coursework
        zip -r coursework.zip coursework

flowchart.pdf: flowchart.dot
        dot -Tpdf flowchart.dot -O flowchart.pdf
```

Pattern rules

(So far, everything *should* have worked in GNU and BSD Make... here on out we're in GNU land)
What if we wanted to add an extra library to our `hello` programs? We could go and update the `Makefile` but its better to generalise!

```
CC=clang
CFLAGS=-Wall -O3

.PHONY: all clean

all: hello coursework.zip flowchart.pdf
clean:
        git clean -dfx

hello: hello.c library.o extra-library.o

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

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

%.zip: %
        zip -r $@ $<

%.pdf: %.dot
        dot -Tpdf $< -O $@
```

Implicit pattern rules

Actually because Make is so old, it knows about compiling C (and Fortran/Pascal...) code already:

```
.PHONY: all clean

all: hello coursework.zip flowchart.pdf
clean:
        git clean -dfx

hello: hello.c library.o extra-library.o

%.zip: %
        zip -r $@ $<

%.pdf: %.dot
        dot -Tpdf $< -O $@
```

Lets get even more general!

Suppose we wanted to add more figures... we could add dependencies on `all` to build them or...

```
.PHONY: all clean
figures=$(patsubst .dot,.pdf,$(wildcard *.dot))

all: hello coursework.zip ${figures}
clean:
        git clean -dfx

hello: hello.c library.o extra-library.o

%.zip: %
        zip -r $@ $<

%.pdf: %.dot
        dot -Tpdf $< -O $@
```

Make is crazy powerful

I love Make...
- ▶ I abuse it for compiling everything
- ▶ For distributing reproducible science studies
- ▶ For building and deploying websites

Pattern rules and the advanced stuff is neat...
- ▶ ...but if you never use it I won't be offended
- ▶ Make is one of those tools that you'll come back to *again and again* over your careers.
- ▶ ...and there's a *bunch* of tricks I haven't shown you `;-)`

Go and read the *GNU Make Manual*
- ▶ Its pretty good for a technical document

In conclusion

When you get a bit of software... and you find a `Makefile` in there...
Just type `make`!

- ▶ (and make sure your projects build in the same way!)

(Actually often you'll have to type `./configure` then `make` for reasons we'll come to *next time*.)

- ▶ No I'm not going to teach you *autotools* don't worry!