CSE 220: Systems Programming

Ethan Blanton

Department of Computer Science and Engineering
University at Buffalo
Make

Make is a general-purpose dependency resolver.

That’s a fancy term that means:

- You provide it with a list of rules.
- The rules say “If you have A, this is how you get B”.
- You say “I want Z”.
- It figures out how to get Z from what you have.

The canonical use of Make is building software.
GNU Make

The flavor of Make we will use is GNU Make.

Some CS department machines are FreeBSD.

Be aware that, on those machines, GNU make is gmake.
Building Software

Make has several features for building software.

In particular, it can detect changed files.

Suppose that:

- fileA is built from fileB
- fileA has changed

If you ask Make to create fileB, it will rebuild it.

On the other hand, if no dependencies have changed, Make does nothing.
Setting Variables

Make can set variables that can be used later.

Make variables are actually rather complicated.

The simplest syntax is:

```make
VARNAME := value
```

This sets the variable VARNAME to the value value.
Using Variables

Variables can be dereferenced after they are set.

Dereferences use the syntax `$(VARNAME)`

The value of the variable will be inserted where it is dereferenced.

It is helpful to think of most Make constructs as strings.

If you need an actual $, use $$.
Make Rules

Make *rules* are what it uses to build dependency chains.

A rule expresses:

- The item to be created
- What it requires
- How to build it from what it requires

Rules can be expressed transitively:

A requires B which requires C;
I want A, so build C then B then A.
Make is very fussy about syntax.

In particular, it ascribes meaning to the tab character.

A make rule is:

```
target: dependencies
   recipe
```

The blank space before recipe must be a tab.

This rule says “target is created from dependencies, and recipe is how you create it.”

- The target is typically a filename
- The dependencies are typically files
- The recipe is a Unix shell script
Example Recipe

Let’s see an example, from PA0:

CC := gcc

calc: calc.o
   $(CC) -o calc $^

This says:

- calc is built from calc.o
- The command to build calc is gcc -o calc calc.o

Don’t worry too much about $^; it means “all of this rule’s dependencies.”
Optional Readings

License

Copyright 2019 Ethan Blanton, All Rights Reserved.

Reproduction of this material without written consent of the author is prohibited.

To retrieve a copy of this material, or related materials, see https://www.cse.buffalo.edu/~eblanton/.