{C}

Tools of the Trade

make

Building Software

gcc is our *compiler*
- Turns C code into machine code

ar is our *librarian*
- Gathers machine code files into groups called libraries

But calling these over and over is tedious!

```
$ gcc -std=c99 -c list.c
$ ar rcu liblist.a list.o
$ gcc -std=c99 -o trends trends.c -L. -llist
```

Building Software

4

Luckily, the process of building (and rebuilding) software can be automated!

make

- Automates the software build process

make

5

The basics:

- When you type `make` on the command line, `make` looks for a file named `Makefile`
- `Makefile` describes how your sources are converted into programs
- `make` can also take arguments, to change how the program is built:

```
make CFLAGS="-g"
```

make is a very common way to build software, but there are others

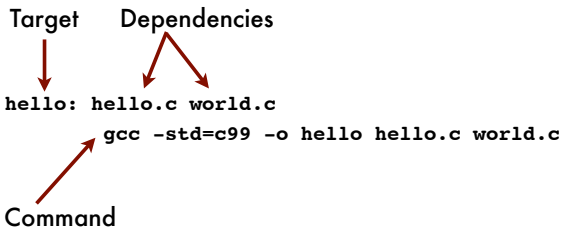
- autoconf, cmake, scons, Xcode, Visual Studio, complicated mess of scripts, …

Makefile

6

A `Makefile` is a list of *targets*, *dependencies*, and *commands*

A simple `Makefile`:



Makefile

7

A simple `Makefile`:

Capital M
Yes, case does matter!

```
hello: hello.c world.c
	gcc -std=c99 -o hello hello.c world.c
```

Tab (NOT spaces!)
Some editors will put spaces even if you ask for a tab, so be careful!

Building with make

8

When you **make**, if the *target* does not exist, or *dependencies* have changed, it builds

```
$ ls
Makefile    hello.c    world.c
$ make
gcc -std=c99 -o hello hello.c world.c
$ ls
Makefile    hello    hello.c    world.c
$ make
make: `hello' is up to date
$ vi hello.c
$ make
gcc -std=c99 -o hello hello.c world.c
```

Multiple targets

9

If your Makefile lists multiple targets, only the first is default

- Makefile:

```
hello: hello.c world.c
	gcc -std=c99 -o hello hello.c world.c

goodbye: goodbye.c world.c
	gcc -std=c99 -o goodbye goodbye.c world.c
```

- Command line:

```
$ make              # builds hello
$ make goodbye      # builds goodbye
```

Dependencies and targets

10

Dependencies can also be targets!

- Makefile:

```
hello: hello.c libworld.a
	gcc -std=c99 -o hello hello.c -L. -lworld

libworld.a: world.c
	gcc -std=c99 -c world.c
	ar rcu libworld.a world.o
```

- Command line:

```
$ make
gcc -std=c99 -c world.c
ar rcu libworld.a world.o
gcc -std=c99 -o hello hello.c -L. -lworld
```

Pseudo-targets

11

Some targets may not actually be programs or files

There is no program called "all"

```
all: hello goodbye
```

But building "all" builds both hello and goodbye

```
hello: hello.c world.c
	gcc -std=c99 -o hello hello.c world.c

goodbye: goodbye.c world.c
	gcc -std=c99 -o goodbye goodbye.c world.c

clean:
	rm -f hello goodbye
```

"make clean" is a common pseudo-target for cleaning up

Compiling vs linking

12

When a GCC command includes multiple C files, each are compiled, then all are linked into a single program:

```
gcc -std=c99 -o hello hello.c world.c
```

- Builds `hello.c` into `hello.o`
- Builds `world.c` into `world.o`
- Links `hello.o` and `world.o` into `hello`

If some C files don't change, you're wasting time recompiling them. **`make`** to the rescue!

Compiling vs linking

13

```
hello: hello.o world.o
        gcc -std=c99 -o hello hello.o world.o

hello.o: hello.c
        gcc -std=c99 -c hello.c

world.o: world.c
        gcc -std=c99 -c world.c
```

Variables

14

Avoid repetition and be flexible by making variables

- Makefile:

```
CC=gcc
CFLAGS=-std=c99 -O2 -g

hello: hello.c
        $(CC) $(CFLAGS) -o hello hello.c
```

- Command line:

```
$ make
gcc -std=c99 -O2 -g -o hello hello.c
$ rm hello ; make CFLAGS="-g"
gcc -g -o hello hello.c
```

Patterns

15

Very common patterns (such as compiling .c files into .o files) can be grouped

e.g. a target for all .o files:

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