

Buildtools

Lets never manually compile things, eh?

Joseph Hallett

October 16, 2024



Format shifting

So much of what we do with a computer is about *format shifting*

- ▶ Convert these JPEGs to PNGs
- ▶ Summarise this spreadsheets worth of data
- ▶ Build an AI model from these observations
- ▶ **Convert this code into a binary**

Convert this code into a binary

For programmers this is a really common one

This lecture we're going to be talking about tools to do this:

- ▶ in general (make)
- ▶ for Java code specifically (maven)

The key point I want you to take away:

- ▶ You shouldn't have to remember a bunch of random shell commands to compile your code
- ▶ You should *just. type. make.*
- ▶ (and it should be somewhat smart about it)

Confessions of a rabbid fan...

I am scarily obsessed with make

- ▶ I will actively abuse it to do more than it was ever designed to do
- ▶ I have written non-trivial programs in it to do *horrific* things
- ▶ Take the bits that are useful to you ;-)

Lets work from an example...

I have some C code.

`greeter.c` which is a program for greeting people (`main()` is here)

`library.c` which is a set of library code used by `greeter.c`

`library.h` which is the library's header file

I want to build these into a single program

► How do I do that?

First attempt

```
gcc -c library.c  
gcc greeter.c library.o -o greeter
```

Could stick into a shellscript...

This will work...

- ▶ But it sucks

Why does this suck?

```
gcc -c library.c  
gcc greeter.c library.o -o greeter
```

What if library.c fails to compile?

The compilation will continue anyway...

```
gcc -c library.c || exit $?  
gcc greeter.c library.o -o greeter
```

This also sucks (unless you like Go...)

What if we need to use *Clang* or a set of optimizations

Could type them all out but that adds duplication...

```
CC=clang  
CFLAGS=-O2  
$CC $CFLAGS -c library.c  
$CC $CFLAGS greeter.c library.o -o greeter
```

What if we update greeter.c only?

It will still recompile library.o

► Seems wasteful

```
if [ ! library.o -nt library.c ]; then
    gcc -c library.c
fi

if [ ! greeter -nt greeter.c -a ! library.o -nt greeter.c ]; then
    gcc greeter.c library.o -o greeter
fi
```

This sucks.

And if you put them all together...

```
CC=clang
CFLAGS=-O2
if [ ! greeter -nt greeter.c -a ! library.o -nt library.c ]; then
    $CC $CFLAGS -c library.c || exit $?
fi

if [ ! library.o -nt greeter.c ]; then
    $CC $CFLAGS greeter.c library.o -o greeter
fi
```

And then you add two more libraries (one of whom library.c depends on)

```
CC=clang
CFLAGS=-O2

if [ !library1.o -nt library1.c ]; then
    $CC $CFLAGS -c library1.c || exit $?
fi

if [ ! library2.o -nt library2.c ]; then
    $CC $CFLAGS -c library2.c || exit $?
fi

if [ ! library.o -nt library.c -a ! library2.o -nt library.c ]; then
    $CC $CFLAGS -c library.c library2.o || exit $?
fi

if [ ! greeter -nt greeter.c -a ! library.o -nt greeter.c -a ! library1.o -nt greeter.c ]; then
    $CC $CFLAGS greeter.c library.o library1.o -o greeter
fi
```

And now I want it to run in parallel...

```
CC=clang
CFLAGS=-O2

if [ !library1.o -nt library1.c ]; then
    ($CC $CFLAGS -c library1.c || exit $?) &
fi

if [ ! library2.o -nt library2.c ]; then
    ($CC $CFLAGS -c library2.c || exit $?) &
fi

wait

if [ ! library.o -nt library.c -a ! library2.o -nt library.c ]; then
    ($CC $CFLAGS -c library.c library2.o || exit $?) &
fi

wait

if [ ! greeter -nt greeter.c -a ! library.o -nt greeter.c -a ! library1.o -nt greeter.c ]; then
    $CC $CFLAGS greeter.c library.o library1.o -o greeter
fi
```

Oh and I'm guessing that is broken...

I suspect that the `|| exit $? won't do what I want and I need to send a semaphore and handle that now.`

- ▶ I'm not going to test this
- ▶ It's a hypothetical example
- ▶ Either way this is gross and it sucks and its repetitious and *I HATE IT*

Luckilly we have better tools

make is a tool for shifting files between formats

- ▶ It fixes all the bugbears with the shell version automatically
- ▶ It dates back to the dawn of computers
- ▶ If you see a Makefile you compile it by typing make

Unfortunately, since it is so old...

- ▶ There are competing implementations
 - ▶ POSIX Make is the standard one (BSD/Macs use versions of this)
 - ▶ GNU Make is the one everyone uses (Linux uses this... BSD/Mac users install gnumake and call it as gmake)
- ▶ The syntax is a bit weird

What does it look like?

Create a file called Makefile (or GNUmakefile)

```
CC=gcc
CFLAGS=-O2

.default: greeter

greeter: greeter.c library.o library1.o
    $CC $CFLAGS greeter.c library.o library1.o

library.o: library.c library2.o
    $CC $CFLAGS -c library.c library2.o -o library.o

library1.o: library1.c
    $CC $CFLAGS -c library1.c -o library1.o

library.2: library2.c
    $CC $CFLAGS -c library2.c -o library2.o
```


And what does that mean?

```
greeter: greeter.c library.o library1.o  
        $CC $CFLAGS greeter.c library.o library1.o
```

To build greeter you'll need the files:

- ▶ greeter.c, library.o and library1.o
- ▶ Rebuild any of them if their source is newer
- ▶ And rebuild greeter if any of these are newer

To do the build you need to run

- ▶ `$CC $CFLAGS greeter.c library.o library1.o`
- ▶ This line is indented with a **tab**!

And the CC bit?

```
CC=gcc  
CFLAGS=-O2
```

Set variables to these values

- ▶ No spaces around the =

.default?

```
.default greeter
```

If you run make with no arguments...

- ▶ It'll run the first rule by default
- ▶ or it'll run the one you mark as `.default`
 - ▶ Explicit beats implicit!

And if you run make

It will build the *default* rule

- ▶ or the first one if not specified

It will see if theres a way to make all the dependencies

- ▶ And check if it needs to build them at all

It will do it in parallel

- ▶ ...if you use `-j4` (or however many processes you want)

Can we do better?

If you look at our rules for building object files there's a *pattern*

- ▶ To build *something* .o you compile *something* .c (and all its other dependencies) with the -c flag

Can we abstract this?

```
%O: %.C  
$CC $CFLAGS -c $^ -o $@
```

%O: %.C To get /something/=o= you need a /something/=c=

\$^ The entire dependency list

\$@ The target output

Patternrule GNUmakefile

```
CC=gcc
CFLAGS=-O2

.default: greeter

%.o: %.c
    $CC $CFLAGS -c $^ -o $@

%: %.c
    $CC $CFLAGS -o $@ $^

greeter: greeter.o library.o library1.o
library.o: library.c library2.o
```

But make is old

It has builtin rules for C (C++ and Pascal and a few others)...

```
CC=gcc
CFLAGS=-O2
.default: greeter
greeter: greeter.c library.o library1.o
library.o: library.c library2.o
```

Thats make, folks!

That is 90% of everything you'll ever need with make.

- ▶ But I said this is my *favourite* tool
- ▶ I should show you some more advanced tricks

Some more general good practices

- ▶ You should add a rule called `all` that builds everything
- ▶ You should add a rule called `install` that installs your into `$PREFIX/bin`
- ▶ You should add a rule called `clean` that removes all build artefacts
- ▶ You should declare targets that build things that aren't output files as `.phony`

```
CC=gcc
CFLAGS=-O2
.default: all
.phony: all clean install
all: greeter

clean:
    $RM -rf $(git ls-files --others --exclude-standard)

install: greeter
    install -m 0755 -o root -g root -s greeter "$PREFIX/greeter"

greeter: greeter.c library.o library1.o
library.o: library.c library2.o
```


Bonus tricks

We set `CC` to be `gcc`... but what if the user wants to override it?

- What if they want to do a build with `-O3` or `-g` in their `CFLAGS`?

```
CC?=gcc  
CFLAGS?=-O2
```

Now the user can override them with an environment variable

```
CC=clang make all
```

What if you don't want to list all your files

Say I'm writing a paper with figures

- ▶ If any of my figures change I will need to recompile my paper

```
paper.pdf: paper.tex figures/figure1.png figures/figure2.png figures/figure3.png  
pdflatex paper
```

Seems tedious to keep updating the dependencies as I add figures?

```
paper.pdf: paper.tex $(wildcard figures/*.png)  
pdflatex paper
```

Say I need to convert a bunch of files...

As part of my paper I have a bunch of flowcharts written in *GraphViz*

- I convert these to PNGs with the `dot` command

```
%.png: %.dot
    dot -Tpng $< -o $@

flowcharts=$(patsubst .dot,.png,$(wildcard figures/*.dot))
paper.pdf: paper.tex $(wildcard figures/*.png) ${flowcharts}
    pdflatex paper
```

Yer what now?

```
% .png: %.dot
dot -Tpng $< -o $@

flowcharts=$(patsubst .dot,.png,$(wildcard figures/*.dot))
paper.pdf: paper.tex $(wildcard figures/*.png) ${flowcharts}
pdflatex paper
```

Lets say I have 3 files:

- ▶ figures/diagram.dot
- ▶ figures/flowchart.dot
- ▶ figures/chart.dot
- ▶ \$(wildcard figures/*.dot)
 - ▶ Expands to figures/diagram.dot figures/flowchart.dot figures/chart.dot
- ▶ \$(patsubst .dot,.png,\$(wildcard figures/*.dot))
 - ▶ Expands to figures/diagram.png figures/flowchart.png figures/chart.png

If diagram.dot changes...

- ▶ Then make will rebuild diagram.png and paper.pdf

There is more

A whole bunch more!

- ▶ Read the manual... its not that bad for a technical document
- ▶ Forcing ordering in dependencies
- ▶ Forcing randomization of dependencies
- ▶ More variables and functions than you can shake a stick at

But there is one thing that make doesn't do particularly well...

(Library) Dependencies

Make is really good about knowing how to shift one file to another

- ▶ But it doesn't know anything about the code its compiling
- ▶ It's just pattern matching on extensions and access times

Modern languages have libraries

- ▶ We don't normally compile everything from scratch anymore
- ▶ ...usually.
- ▶ We'd like our build tools to fetch them automatically

Library-aware buildtools

Every language has their own tooling!

Commonlisp ASDF and Quicklisp

Go Gobuild

Haskell Cabal

Java Ant, Maven, Gradle...

JavaScript NPM

Perl CPAN

Python Distutils and requirements.txt

R CRAN

Ruby Gem

Rust Cargo

L^AT_EX CTAN and TeXlive

...and *many* more.

And they're all different

Very little similarity between *any* of them.

- ▶ You need to learn the ones you use.
- ▶ We'll play in the labs with *Maven* for Java a little bit

Maven

Build tool for Java (mostly)

- ▶ Others exist (gradle and ant)
- ▶ Configured in XML
- ▶ Fairly standard and available everywhere
- ▶ *Needlessly* verbose

(I *dislike* it and generally use Make and manage things myself but YMMV...)

Lets create a new project

```
mkdir /tmp/src
cd /tmp/src
mvn archetype:generate \
  -DgroupId=uk.ac.bristol.cs \
  -DartifactId=hello \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DinteractiveMode=false
```

(Plus a lot of downloads I've omitted...)

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----< org.apache.maven:standalone-pom >-----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----[ pom ]-----
[INFO]
[INFO] >>> archetype:3.2.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< archetype:3.2.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO] --- archetype:3.2.1:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO]
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO]
[INFO] Parameter: basedir, Value: /tmp/src
[INFO] Parameter: package, Value: uk.ac.bristol.cs
[INFO] Parameter: groupId, Value: uk.ac.bristol.cs
[INFO] Parameter: artifactId, Value: hello
[INFO] Parameter: packageName, Value: uk.ac.bristol.cs
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /tmp/src/hello
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
```

So whats that done?

```
find . -type f
```

```
./hello/pom.xml  
./hello/src/main/java/uk/ac/bristol/cs/App.java  
./hello/src/test/java/uk/ac/bristol/cs/AppTest.java
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>uk.ac.bristol.cs</groupId>
  <artifactId>hello</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>hello</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

This is xml!

XML primer

Format for writing trees that can be parsed by a computer *and* a human

- ▶ Basically a generalized form of HTML
- ▶ schema defines what all the tags mean

```
<!-- This is a comment -->  
<tag attribute=value>  
  <innerTag>Hello</innerTag>  
  <innerTag>World!</innerTag>  
</tag>
```

You can't stick stuff wherever... the tags define relationships between what they are and what they contain.

To add a library

If I want to add a library... Go find it on a Maven repository and pick the version you want:

The screenshot shows the Maven Repository website for the artifact `org.antlr:antlr4-runtime`. The page includes a search bar, navigation links, and a detailed view of the artifact. A yellow arrow points to the version `4.13.1` in the version list table.

MVN REPOSITORY Search for groups, artifacts, categories Search Categories | Popular | Contact Us

Home » org.antlr » antlr4-runtime

ANTLR 4 Runtime
The ANTLR 4 Runtime

License: [BSD 3-clause](#)

Categories: [Parser Generators](#)

Tags: [antlr](#) [generator](#) [compiler](#) [parser](#) [runtime](#)

Ranking: #289 in MvnRepository (See Top Artifacts)
#1 in Parser Generators

Used By: 1,749 artifacts

Indexed Artifacts (39.8M)

Indexed Repositories (2103)

Popular Categories

Popular Tags

Version	Vulnerabilities	Repository	Usages	Date
4.13.x		Central	256	Sep 04, 2023
4.13.1		Central	68	May 21, 2023
4.12.x		Central	98	Feb 19, 2023
4.12.0		Central	178	Sep 04, 2022
4.11.x		Central	2	Sep 04, 2022
4.11.1		Central	130	Apr 15, 2022
4.11.0		Central	14	Apr 11, 2022
4.10.x		Central	258	Nov 06, 2021
4.10.1		Central	192	Mar 11, 2021
4.10.0		Central	90	Jan 05, 2021
4.9.x		Central	88	Nov 04, 2020
4.9.3		Central		
4.9.2		Central		
4.9.1		Central		

Get the <dependency>...

Maven Repository: org.antlr » antlr4-runtime » 4.13.1 — Mozilla Firefox

File Edit View History Bookmarks Tools Help

Session Expired x Maven Repository: org.x +


https://mvnrepository.com/artifact/org.antlr/antlr4-runtime/4.13.1

MVN REPOSITORY

Search for groups, artifacts, categories

Categories | Popular | Contact Us

Indexed Artifacts (39.8M)



Popular Categories

- Testing Frameworks & Android Packages
- Logging Frameworks
- Java Specifications
- JVM Languages
- JSON Libraries
- Language Runtime
- Core Utilities
- Mocking
- Web Assets
- Annotation Libraries
- Logging Bridges
- HTTP Clients
- Dependency Injection
- XML Processing
- Web Frameworks
- I/O Utilities

Home » org.antlr » antlr4-runtime » 4.13.1

ANTLR 4 Runtime » 4.13.1

The ANTLR 4 Runtime

License	BSD 3-clause
Categories	Parser Generators
Tags	antlr generator compiler parser runtime
Date	Sep 04, 2023
Files	pom (3 KB) jar (318 KB) View All
Repositories	Central Mulesoft
Ranking	#289 in MvnRepository (See Top Artifacts) #1 in Parser Generators
Used By	1,749 artifacts

Maven | Gradle | Gradle (Short) | Gradle (Kotlin) | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/org.antlr/antlr4-runtime -->
<dependency>
  <groupId>org.antlr</groupId>
  <artifactId>antlr4-runtime</artifactId>
  <version>4.13.1</version>
</dependency>
```

☒ Include comment with link to declaration

Copied to clipboard!

Indexed Repositories (2103)

- Central
- Atlassian
- WSO2 Releases
- Hortonworks
- JCenter
- Sonatype
- JBossEA
- KtorEAP
- Atlassian Public
- WSO2 Public

Popular Tags

aar android apache api application arm assets build build-system bundle client clojure cloud commons config cran data database eclipse example extension framework github gradle groovy ios javascript jboss kotlin library maven mobile module npm osgi plugin resources slang sdk server service spring sql starter testing tools ui war

And add it to the pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>uk.ac.bristol.cs</groupId>
  <artifactId>hello</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>hello</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.antlr/antlr4-runtime -->
    <dependency>
      <groupId>org.antlr</groupId>
      <artifactId>antlr4-runtime</artifactId>
      <version>4.13.1</version>
    </dependency>
  </dependencies>
</project>
```


And when you want to build...

```
mvn package
```

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----< uk.ac.bristol.cs:hello >-----
[INFO] Building hello 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ hello ---
[WARNING] Using platform encoding (US-ASCII actually) to copy filtered resources
[INFO] skip non existing resourceDirectory /tmp/src/hello/src/main/resources
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ hello ---
[INFO] Recompiling the module because of changed source code.
[WARNING] File encoding has not been set, using platform encoding US-ASCII, i
[INFO] Compiling 1 source file with javac [debug target 1.8] to target/classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ hello ---
[WARNING] Using platform encoding (US-ASCII actually) to copy filtered resources
[INFO] skip non existing resourceDirectory /tmp/src/hello/src/test/resources
[INFO]
```

Other useful commands

`mvn test` run the test suite

`mvn install` install the JAR into your local JAR packages

`mvn clean` delete everything

But these are defined by whatever the archetype you chose at the beginning were.

You can define your own rules...

- ▶ But it isn't as elegant or as easy as *Make*
- ▶ *Not unusual* to find a *Makefile* that calls a whole other build system

```
all:
    mvn build

install:
    mvn install
```

Wrap up

I love make
Maven is really verbose isn't it?