
# ANTLR

## a short introduction

Andrea Polini, Luca Tesei

Formal Languages and Compilers
MSc in Computer Science
University of Camerino

# What's that?

ANTLR v.4 is a powerful parser generator that you can use to read, process, execute, or translate structured text or binary files.

From a grammar as a formal language description, ANTLR generates a parser for that language that can automatically build parse trees. ANTLR also automatically generates tree walkers that you can use to visit the nodes of those trees to execute application-specific code.

# What's that?

ANTLR v.4 is a powerful parser generator that you can use to read, process, execute, or translate structured text or binary files.

From a grammar as a formal language description, ANTLR generates a parser for that language that can automatically build parse trees. ANTLR also automatically generates tree walkers that you can use to visit the nodes of those trees to execute application-specific code.

# How can I get it?

- Download last complete jar from `http://www.antlr.org/download.html`
- Put it in an appropriate folder, e.g. `/usr/local/lib`
- The jar contains:
  - all dependencies necessary to run the ANTLR tool
  - the runtime library needed to compile and execute recognizers generated by ANTLR
  - a sophisticated tree layout support library: `http://code.google.com/p/treelayout`
  - a template engine useful for generating code and other structured text: `http://www.stringtemplate.org`

# How can I install it?

- Set the CLASSPATH environment variable to include "." and the jar:

```
> export
CLASSPATH=".:/usr/local/bin/antlr-4.7.1-complete.jar:$CLASSPATH"
```

- You can do it every time you start a session in a shell or you can edit the `.bash_profile` file
- To run the ANTLR4 Tool:

```
> java -jar /usr/local/lib/antlr-4.0-complete.jar
```

or directly:

```
> java org.antlr.v4.Tool
```

- To save typing:

```
> alias antlr4='java -jar /usr/local/lib/antlr-4.0-complete.jar'
```

# How should I use it?

**File `Hello.g4`**

```
grammar Hello; // Define a grammar called Hello
r :  'hello' ID ; // Match the word 'hello' followed by an identifier
ID : [a-z]+ ; // Match lower-case identifiers
WS : [\t \r \n]+ -> skip ; // skip spaces, tabs, newlines, \r (Windows)
```

```
> antlr4 Hello.g4
```

produces:

```
Hello.g4 HelloLexer.java HelloParser.java
Hello.tokens HelloLexer.tokens
HelloBaseListener.java HelloListener.java
```

Then:

```
> javac *.java
```

# Testing Hello

- ANTLR4 generates an executable recognizer embodied by `HelloParser.java` and `HelloLexer.java`
- There is not (yet) a main program to trigger language recognition
- ANTLR4 provides a a flexible testing tool in the runtime library called `TestRig`
- `> alias grun='java org.antlr.v4.runtime.misc.TestRig'`
- The test rig takes:
  - a grammar name
  - a starting rule name
  - various options for the desired output

# Testing Hello

```
> grun Hello r -tokens  # start the TestRig on grammar Hello at rule r
hello parrt             # input for the recognizer that you type
<eof>                   # type ctrl+D on Unix or ctrl+Z on Windows}\\
```

Outputs a detailed description of the tokens:

```
[@0,0:4='hello',<1>,1:0]
[@1,6:10='parrt',<2>,1:6]
[@2,12:11='<EOF>',<-1>,2:0]
```

# Testing Hello

```
> grun Hello r -tree
hello parrt
<eof>
```

Outputs the parse tree in LISP-style text:

```
(r hello parrt)
```

# Testing Hello

```
> grun Hello r -gui
hello pippo
<eof>
```

Opens a graphical representation of the parse tree: