Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.035, Fall 2000 — Handout 10 — Infrastructure Tools — Thursday, September 9

---

1. **Make**

   The make utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them. Here's an example of a makefile we use to compile the IR for our compiler.

```
JAVAC   = javac
JAVAI   = java
CLASSES = ../../../classes
JHOME   = $(JAVA_HOME)

CLASSPATH = $(JHOME)/lib/classes.zip:$(CLASSES)

all : code

code:
        $(JAVAC) -d $(CLASSES) -classpath $(CLASSPATH) *.java

clean:
        rm -rf *.class
        rm -rf ../../../classes/java6035/tools/IR/*.class
```

   To use this makefile, we save it as `Makefile` in the directory with our sources. Running the `make` command will now recompile any of the java files that need it. This is confused a bit because the java compiler itself does this as well, but for 6.035 the above sort of makefile is entirely sufficient. You can do lots more with make, see the Info page (`C-h i` in emacs) for more information.

2. **Java on Athena**

   Athena's version of java is now version 1.2.2, which we don't support. To run java on athena, please use the following:

```
athena% add -f java_v1.1.6
athena% java myfile.java
```

   The staff only officially supports Java 1.1.6. Java 1.2.x has some significant differences that may prevent our codebase from working properly with it. Even worse, JLex and javacup may not generate code that works properly with 1.2.x. You're free to try it, and please let us know how it goes. If there are problems with our code (you'll be using it in future stages of the project) we'll try to fix it, but correcting problems with JLex may be more difficult.

One of the things you'll want to set is the `CLASSPATH` environment variable, which tells the java tools where to find the compiled java files. You'll want to add `/mit/6.035/classes/` to `CLASSPATH`, along with whatever directories your own class files are in. If you're unfamiliar with how class files work in Java, talk to your TA.

3. **6.035 locker**

   Not much to say here. The handouts and lectures directories contain the obvious things. You'll find example decaf and espresso programs under examples, and various bits of provided code under provided. Finally, once we set up your groups, every group will be given its own locker inside the groups subdirectory.

4. **CVS**

   CVS is a version control system designed to help you manage a source code base. You'll be writing a lot of code for 6.035, and for many of you this will be the first time that you'll be working on a project of sufficient complexity to require source control. Without it, you'll have a tough time.

   CVS has an excellent tutorial available online at `http://www.loria.fr/~molli/cvs/doc/cvs_toc.html`, as well as lots of information on its info page (again, `C-h i` in emacs). You'll need to understand the concepts of creating a repository (`cvs init`), checking out a working copy of your source tree (`cvs co`), checking in changes (`cvs commit`), getting diffs and logs (`cvs diff` and `cvs log`), and updating your working directory (`cvs update`). You may also find tagging and branching useful.

5. **Other Tools**

   JLex is a scanner generator for Java. It is used for the scanner portion of the class project. Handout 9 describes the basic use of JLex. JLex also has a Web page; see the 6.035 home page, or visit `http://www.cs.princeton.edu/~appel/modern/java/JLex/manual.html`.

   CUP is a parser generator for Java, vaguely reminiscent of the standard *yacc* tool. It is used for the parser portion of the class project. A future handout will describe its use. Like JLex, CUP has a Web page that is linked from the course page. The CUP home page is at `http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html`.