Review

- What are the benefits of bash scripts?
- How do we run bash scripts?
- What can we do in bash so far?

Using commands in commands

- Examples from my scripts

```
jarfiles=`ls $TURNINDIR/$STUDENT/$LAB/*.jar`
for jarfile in $jarfiles
do
    echo "Jar file: $jarfile"
    numJavaFiles=`jar tf $jarfile | grep -c ".java"`
    if [ $numJavaFiles = 0 ]; then
       echo "No Java Files submitted by $STUDENT"
    fi
done
```


BUILD TOOLS

Distributing Software

- Pieces typically distributed:
  - Binaries/Bytecode
  - Required libraries
  - Data files
  - Documentation

  artifacts
- Typically packaged in an archive:
  - e.g., tgz, jar, zip, rpm
- May need all of these or some subset of them

Related Tools: make

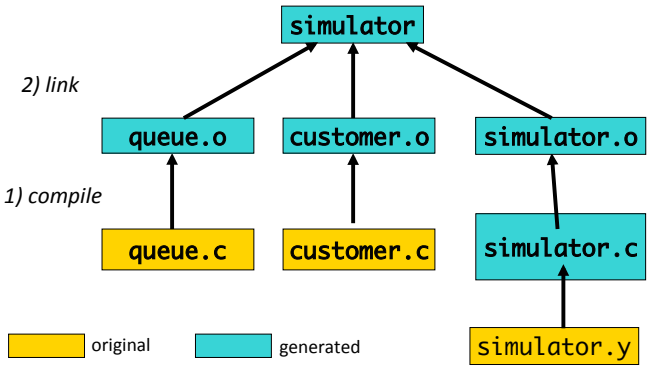
- **make**: A program for building and maintaining computer programs
  - Developed at Bell Labs around 1978 by Stu Feldman
    - Now, head of Schmidt Sciences at the foundation Schmidt Philanthropies
    - Past President of ACM
- Instructions stored in a special format file called a **makefile**

http://www.gnu.org/software/make/

Typical Use Cases

- Installing software from source
  - README describes how to make the software
- Developing in C, C++, … and want to reduce the time in typing/running compilation commands
  - Example: C, C++ program → executable
    1. Compilation: source code → machine code (object files)
    2. Linking: multiple object files → one executable file

Example of Typical Compilation



make Features

- Contains the build instructions for a project
  - Automatically updates files based on a series of *dependency rules*
  - Supports multiple configurations for a project
  - Language-independent
- Only re-compiles necessary files after a change (conditional compilation)
  - Major timesaver for large projects
  - Uses timestamps of the intermediate files
- Typical usage: executable is updated from object files which are in turn compiled from source files

Example Makefile

```
# Example Makefile
CC=g++                                  Variables
CFLAGS=-g -Wall -DDEBUG
OBJECTS=customer.o simulator.o queue.o

simulator: $(OBJECTS)                   Dependencies
        $(CC) $(CFLAGS) -o simulator $(OBJECTS)
simulator.o: simulator.c
        $(CC) $(CFLAGS) -c simulator.c

customer.o: customer.c
        $(CC) $(CFLAGS) -c customer.c
…
clean:                                  Commands
        rm $(OBJECTS) simulator
```

Rules/ Targets

Must be a tab

Running:

```
$ make
$ make clean
$ make -f other_makefile
```

By default looks for `makefile`

Related Tools: Apache Ant

- Java-based build tool
- Similar to make

Make	Ant
Shell-based, makefile	Java, XML config files

`http://ant.apache.org/`

XML: eXtensible Markup Language

- Looks similar to HTML
  - HTML's stricter sibling
- Designed to structure, store, and transport data
  - Text file → PORTABLE!
- Made up of *nested* elements
  - Hierarchy of data
- Schema
  - *Define* your own tags, tag nesting, tag attributes

XML Example

```
<email>
   <to>you@somewhere.org</to>
   <from>me@here.org</from>
   <subject>Reminder</subject>
   <message>Don't forget me this
weekend!</message>
</email>
```

XML Example

Root element

child elements

```
<email>
   <to>you@somewhere.org</to>
   <from>me@here.org</from>
   <subject>Reminder</subject>
   <message>Don't forget me this
weekend!</message>
</email>
```

Close every element you open

XML Example

attribute

```
<imdb>
        <movie category="comedy">
                <title lang="en">Juno</title>
                <title lang="es">La joven vida de Juno</title>
        </movie>
        <movie category="comedy">
                <title lang="en">Chicken Run</title>
                <title lang="de">Hennen Rennen</title>
        </movie>
</imdb>
```


Ant buildfile: `build.xml`

- Starts with XML version:

```
<?xml version="1.0" encoding="UTF-8"?>
```

- Root element: `project`
  - `name` attribute: name of the project
  - `default` attribute: default *target*
  - `basedir` attribute: directory to run from

```
<project name="Hello World"
default="Hello" basedir=".">
```


Ant `target`

- Target: has a name, set of tasks to execute
- Can specify which targets to execute
  - If no target given, use project's default
- Can depend on other targets
- Examples:
  - Compile
  - Distribute
    - Needs compile

Closes open tag

```
<target name="compile"/>
<target name="jar"
depends="compile"/>
```

Example Ant target

What does this do?

```
<target name="compile"
      description="Compile the source code">
   <mkdir dir="build/classes"/>
   <javac srcdir="src"
          destdir="build/classes"
          debug="on">
      <include name="**/*.java"/>
      <classpath refid="build.class.path"/>
   </javac>
</target>
```

build-replay.xml

Ant `property`

- Like a variable: defines a name and its value:
  - `<property name="vname" value="vvalue" />`
- To use property, use `${name}`
- In `build.xml` in Eclipse, add two properties:
  - HelloText=Hello
  - WorldText=World

Looking Ahead

- Assignment 3 due Monday