Outline

- Maven
- NuGet
- Gradle
- GNU build system
- CMake
Maven

- Project management and building tool
  - mainly for Java

- Typical usage scenarios made simpler for users

- Encourages best-practices and conventions
  - Directory layout
  - Naming of tests

Best-practice guidelines

- Directory tree (layout)

```
  my-app
  -- pom.xml
  -- src
      -- main
          -- java
              `-- com
                  `-- mycompany
                      `-- app
                          -- App.java
                      `-- resources
                  `-- test
                      -- java
                          `-- com
                              `-- mycompany
                                  `-- app
                                      -- AppTest.java
      `-- target
          `-- classes
```

- Test case names

  **/*Test.java, **/Test*.java

Key concepts

- **Goal**
  - Single action to be executed
    - Construction of directory layout
    - Compilation of Java sources
  - Similar to **task** in Ant

- **Phase**
  - Step in the build lifecycle
    - generate-sources, compile, deploy
  - Sequence of goals
  - Similar to **target** in Ant

- **Build lifecycle**
  - Ordered sequence of phases
  - Similar to **dependencies between targets** in Ant
Typical build lifecycle

1. validate
2. compile
3. test
4. package
5. integration-test
6. verify
7. install to local repository
8. deploy
Project Object Model (POM)

- Project’s configuration (build script)
  - Stored in the `pom.xml` file

```xml
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Usage

• Project setup
  
  `mvn archetype:generate \
  -DarchetypeArtifactId=maven-archetype-quickstart \ 
  -DgroupId=com.mycompany.app -DartifactId=my-app`

• Build lifecycle: `mvn <name of a phase>`
  ▪ Compilation: `mvn compile`
  ▪ Packaging: `mvn package`
  ▪ Web-site generation: `mvn site`
  ▪ Rebuild into local repository: `mvn clean install`

• Default remote repository (central)
  ▪ `http://repo1.maven.org/maven2/`
Advanced features

- Creating local repositories
- Creating packages with metadata
  - To be stored into repository

- Modifications of standard workflow

- Project inheritance (modules)

- Extensibility via plugins
  - Plugin implements a set of related goals
Example

- [Link](http://d3s.mff.cuni.cz/teaching/software_development_tools/files/maven-ex.tgz)
  - DSI Utilities: original sources, build.xml, pom.xml
  - Project home page: http://dsutils.di.unimi.it/
Want to know more about Maven?

- Read the guide

- Try it yourself
  - Create new project
  - Add source files
  - Run compilation
NuGet

• Package manager for .NET

• Similar concepts to Maven

• Integration to Visual Studio

• Web: https://www.nuget.org/

• Docs: https://docs.microsoft.com/en-us/nuget/
Gradle

- Another popular general-purpose build tool
  - Java, Scala, C, C++, Android
- Encourages best practices (like Maven)
- Script language (DSL) based on Groovy

- Web: [https://gradle.org/](https://gradle.org/)

- Examples
  - [https://docs.gradle.org/current/userguide/tutorial_java_projects.html](https://docs.gradle.org/current/userguide/tutorial_java_projects.html)
  - [https://docs.gradle.org/current/userguide/tutorial_using_tasks.html](https://docs.gradle.org/current/userguide/tutorial_using_tasks.html)
  - Running: `gradle build`
Motivation for GNU build system

- Portability of programs
  - over different UNIX-like systems
  - existing standards (C, POSIX) define only core aspects

- System-specific configuration
  - e.g., use of KDE instead of Gnome

- Complexity of Make files
  - unreadable, hard to maintain
  - writing all the rules is tedious

- Portability of Make files
  - Make is standardized by POSIX, but not all UNIX-like systems are 100% compliant
Selected portability and compatibility issues

- Programs in C
  - `exit()`: may return `void` or `int` (error code)
  - `free(NULL)`: sometimes does nothing
  - `malloc(0)`: returns `NULL` or valid pointer
  - (and many more)

- Functions in different headers and libraries

- Shell and utilities: Awk, Grep, Sed, ...
  - Multiple implementations (not all compatible)
Solutions for portability and compatibility

- Virtualized environment (Java, C#/.NET)

- GNU build system (Autotools)
  - De-facto standard in Unix/Linux world
  - Explicit support for different flavors
  - Database of known portability issues
  - Resolves issues during configuration
  - Uses only features available everywhere
GNU build system (Autotools)

- Autoconf
  - Configuration detector

- Automake
  - Makefile generator

- Libtool
  - Abstracts creation of libraries

- Gettext
  - Support for localization
End user’s perspective

1. Download the source code
2. “./configure”
   - Automatically tests the target system
   - e.g. for presence of required libraries
   - Detects system configuration (OS, HW)
   - Automatically generates Make files
3. “make”
4. “make install”
End user’s perspective – configuration

- Installation root directory
  - “configure --prefix=/opt”

- Cross-compilation
  - “configure --host”

- Optional features of the software
  - “configure --enable-FEATURE”
  - “configure --disable-FEATURE”

- Optional packages (libraries) to build with
  - “configure --with-PACKAGE”
  - “configure --without-PACKAGE”
What is behind the scenes

shell scripts with many system tests

configure

config.cache

config.log

autoheader

config.h.in

config.h

Makefile.in

Makefile

automake

Makefile.am

script which generates make files, taking test results from the cache

cache for test results between restarts
Autoconf & the “configure” script

- Very portable shell script
  - Uses features in the lowest-common-denominator of known shells (no functions, ...)
  - Generated from a template (*configure.ac*)
  - Based on a library of tests of well-known portability and compatibility issues
AC_INIT(package, version, bug-report-address)

information about the package
checks for programs
checks for libraries
checks for header files
checks for types
checks for structures
checks for compiler characteristics
checks for library functions
checks for system services

AC_CONFIG_FILES([output file, ...])
AC_OUTPUT
AC_INIT([GNU cflow], [1.2], [bug-cflow@gnu.org])
AC_CONFIG_HEADER([config.h])

# Checks for programs.
AC_PROG_CC
AC_PROG_LEX

# Checks for header files.
AC_HEADER_STDC
AC_CHECK_HEADERS([stdlib.h string.h unistd.h locale.h])

AC_OUTPUT

a single argument to a macro
multiple arguments to a macro
macros
“configure” – another example

  - configure.ac
  - src/config.h.in
  - src/client.cpp

- Achieving support for multiple platforms
Generating “configure.ac”

- Autoscan
  - Inspects source files (C/C++) to detect common portability issues
  - Generates skeleton of `configure.ac`

- Ifnames
  - Reports variables used in preprocessor conditionals
  - Often used to solve platform dependency issues
    - Example: `#if HAVE_LOCALE_H`
Automake – creating portable Makefiles

- configure.ac
- Makefile.am
- Makefile.in
- config.status
- Makefile
Supported targets

- install, install-exec, install-data
- uninstall
- clean
- distclean
  - clean to what is distributed
  - removes also files generated by configure
- check
  - run test of compiled binaries
- installcheck
  - run test of installed program
- dist
  - creates source code distribution package (tarball)
“Makefile.am” template

Makefile.am

SUBDIRS = src

dist_doc_DATA = README

install README into docdir and put it into distribution

directories to be processed before this directory

src/Makefile.am

bin_PROGRAMS = hello
hello_SOURCES = main.c
“Makefile.am” template

Makefile.am

SUBDIRS = src
dist_doc_DATA = README

“hello” is a program to be installed into bindir

src/Makefile.am

bin_PROGRAMS = hello
hello_SOURCES = main.c

program “hello” can be built from source “main.c”
since “main.c” is a source file, it will also be put into distribution (by “make dist”)
Autotools – the whole picture again

- configure.ac
- config.log
- config.cache
- config.status
- config.h
- Makefile

Dependencies:
- configure
- autoconf
- automake
- autoheader
- ifnames
- autoscan
- Makefile.am

Files:
- Makefile.in
- config.h.in
- Makefile
Further reading

- http://www.sourceware.org/autobook/
- https://www.gnu.org/software/autoconf/
- https://www.gnu.org/software/automake/
Optional task: Autotools

- Use it on the MyDB program (homework 4)
  - Replace all the Makefiles you created by hand
CMake

- Cross-platform free and open-source build management application

- Compiler-independent tool
  - Supports various native build systems (make, Xcode, MS Visual Studio)


- Two phases of the build process
  - Generate native build scripts from platform-independent configuration (CMakeLists.txt)
  - Run target platform’s native tool for the actual build
Other build tools

- **Ivy**
  - [https://ant.apache.org/ivy/](https://ant.apache.org/ivy/)

- **Scons**

- **Bazel**
  - [http://bazel.io/](http://bazel.io/)