

# COMP 2400 UNIX Tools

Christian Grothoff

`christian@grothoff.org`

`http://grothoff.org/christian/`

# Makefile Basics

'make' reads instructions from a file 'Makefile'. A Makefile is essentially a list of rules. Each rule has the form:

```
TARGET: DEPENDENCIES
```

```
TAB COMMAND
```

```
TAB . . . . .
```

```
TAB . . . . .
```

The first target is the default target.

# Makefile Example (1/2)

```
hello: hello.c
```

```
TAB gcc -o hello hello.c
```

```
clean:
```

```
TAB rm -f hello hello.o
```

```
install: hello
```

```
TAB mkdir -p /usr/local/bin
```

```
TAB rm -f /usr/local/bin/hello
```

```
TAB cp hello /usr/local/bin/hello
```

## Makefile Example (2/2)

```
hello: hello.o  
TAB gcc -o hello hello.o
```

```
hello.o: hello.c  
TAB gcc -c hello.c
```

# Makefile Variables

```
variable1=value
```

```
variable2=value
```

```
target:
```

```
TAB gcc -o $(variable1) $(variable2).c
```

# Dependencies

Sometimes the source file  $X$  needs to be recompiled because the source file  $Y$  has changed.

- We say that  $X$  depends on  $Y$
- Cyclic dependencies are possible
- Makefiles can be used to specify dependencies
- For C/C++, the compiler can determine dependencies
- For Java, no efficient way to determine dependencies exists

# Beyond Makefiles

- Writing Makefiles can become messy
  - Manually determining dependencies is error-prone
  - Re-compiling everything all the time is not feasible for large projects
- ⇒ autotools generate Makefiles that use gcc's dependency information

# Makefile.am Syntax

```
bin_PROGRAMS = myapp  
myapp_SOURCES = app1.c app2.c app3.c app4.c app...
```

Makefile.am will be compiled by autoconf to Makefile.in. Makefile.in will be translated to Makefile by configure.

Makefile will support targets all, clean, install, dist.



# Hello World with Autotools (1/4)

```
#include <stdio.h>
int main(int argc, char * const * argv) {
    printf("Howdy world!\n");
    return 0;
}
```

Makefile.am:

```
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```

# Hello World with Autotools (2/4)

configure.ac:

```
AC_INIT(hello.c)
```

```
AM_INIT_AUTOMAKE(hello,0.1)
```

```
AC_PROG_CC
```

```
AC_PROG_INSTALL
```

```
AC_OUTPUT(Makefile)
```

# Hello World with Autotools (3/4)

```
$ autoreconf -fi
```

```
$ ./configure --prefix=$HOME
```

```
$ make
```

```
$ make install
```

```
$ make dist
```

# Hello World with Autotools (4/4)

```
$ tar xvfz hello-0.1.tar.gz  
$ cd hello-0.1  
$ configure  
$ make  
$ make install  
$ ~/bin/hello
```

# Other things to do

- Organize project in subdirectories
- Add testcases
- Install non-binary files (for example, images)
- Check for dependencies
- Generate C headers to include (`config.h`)
- Conditional targets

# Common directory structure

**src** source code

**doc** documentation

**m4** macros required by autoconf

**po** message translations

**contrib** auxiliary files (for example, images)

# Top-level files

**README** description of the software

**AUTHORS** summary list of all contributors

**ChangeLog** detailed list of changes for developers

**COPYING** copyright (alternatively called LICENSE)

**INSTALL** general installation instructions

**NEWS** list of recent changes for end-users

# Less-common Directories

**src/include** installed headers

**intl/** GNU gettext

**libltdl/** GNU libtool's libltdl (for plugins)

**debian/** files for building a Debian package

**doc/man/** man pages



# Autotools Summary

- Automatic configuration for different platforms
- Automatic Makefile generation with dependencies
- Support for test suite (make check)
- Support for source distribution (make dist)
- Support for creating shared libraries
- User does **not** need to have autoconf/automake

# Writing Testcases

```
check_PROGRAMS = test1 test2 test3 ...
TESTS          = $(check_PROGRAMS)
test1_SOURCES = test1.c
```

# Creating Libraries

```
lib_LTLIBRARIES = libfoo.so
libfoo_la_SOURCES = foo.c
libfoo_la_LDFLAGS = -version-info 1:4:1
libfoo_la_LIBADD = libbar.la
noinst_LTLIBRARIES = libbar.so
libbar_la_SOURCES = bar.c
```

# Linking against Libraries

```
bin_PROGRAMS = prog
prog_SOURCES = prog.c
prog_LDFLAGS = -lz
prog_LDADD = $(top_builddir)/src/bar/libbar.la
```

# Installing Headers

```
include_HEADERS = foo.h
```

# Adding Files to Package

```
EXTRA_DIST = ABOUT-NLS
```

# Adding Manpages

```
man_MANS = foo.1 libbar.3  
EXTRA_DIST = $(man_MANS)
```

# Processing Subdirectories

```
SUBDIRS = m4 src doc po
```



# Examples

- <http://www.gnu.org/software/hello/>
- <https://gnunet.org/svn/libmicrohttpd/>
- <https://gnunet.org/svn/gnunet-gtk/>

# Inspecting Binaries

- nm
- ldd
- file

# Questions



# Task

Setup an autotools-based build-system for your project. Begin with a simple directory structure (`src/` and `doc/man/`) and a minimal C application.

Generate a draft of your `configure.ac` file with the `autoscan` tool (the generated file will be called `configure.scan` and must be manually renamed).

Test your build system using the minimal code. Then write a `test-testcase` before writing any actual code.