

# ***Software Development Libtools***

*Davide Balzarotti*

Eurecom – Sophia Antipolis, France

# Homework Status

- Deadline: Tuesday 31st at 8am
- 2150 submissions (488 correct)
  - 28 students finished them all (compared with 1 last year at the same time)
  - I'm a bit skeptical...
- You DONT need to solve everything to be able to take the final exam !!
  - But obviously the more the better

# Software Development Tools

1. Writing and managing code
2. Configuring and Building the program
  - ✓ GCC
  - ✓ Makefiles
  - ✓ Autotools
3. Packaging and Distributing the application
4. Debugging and Profiling

# The Problem of Shared Libraries

- The format of shared libraries differs between systems
  - `libhello.so`
  - `libhello.dll`
  - `libhello.sl`
  - `libhello.dylib`
- Also in unix-like systems shared libraries are built, named and managed in different ways
  - Some platforms don't even provide native shared libraries (Ultrix 4.2)
  - Some platforms name their libraries `libXX.so`, while others use just `XX.o`
  - Different compilers require different flags
- How to keep track of all this?

# Libtool

- GNU libtool simplifies the developer's job by
  - Encapsulating the platform-specific dependencies
  - Hiding the complexity of using shared libraries behind a consistent, portable interface
  - Providing to the user a simple, portable way to build libraries
- The libtool solution consists of
  - A new library format that abstracts all the others:  
`libname.la` (libtool archive)
  - A wrapper script for the compiler and linker that translates operations involving `libname.la` into the correct operation for the current system using the real library
- Libtool is usable by itself, but we will see how to integrate it with `autoconf` and `automake`

# First Step: libtoolize

- The `libtoolize` shell script provides a standard way to add libtool support to an autotool package
- `libtoolize prepare configure` to generate a custom version of the libtool script in the project directory. This script is then executed at the appropriate time by the automake-generated makefiles
- Libtoolize adds to the project the following files:
  - `config.guess`: script that try to guess the canonical system name
  - `config.sub`: contains a list of the machine supported by (some of) the GNU software and tries to match them to a canonical name
  - `ltmain.sh`: script that provides generalized library-building support services

# Adding libtool to an Autotool Project

- Add **AC\_PROG\_LIBTOOL** to `configure.ac` to initialize libtool
- Use the **LTLIBRARIES** primary to declare libtool archives in `Makefile.am`

```
lib_LTLIBRARIES = mylib.la
```

- Use the **LDADD** variable on the program target to link against local libtool archives

```
vbin_PROGRAMS = myprog  
myprog_LDADD = mylib.la
```

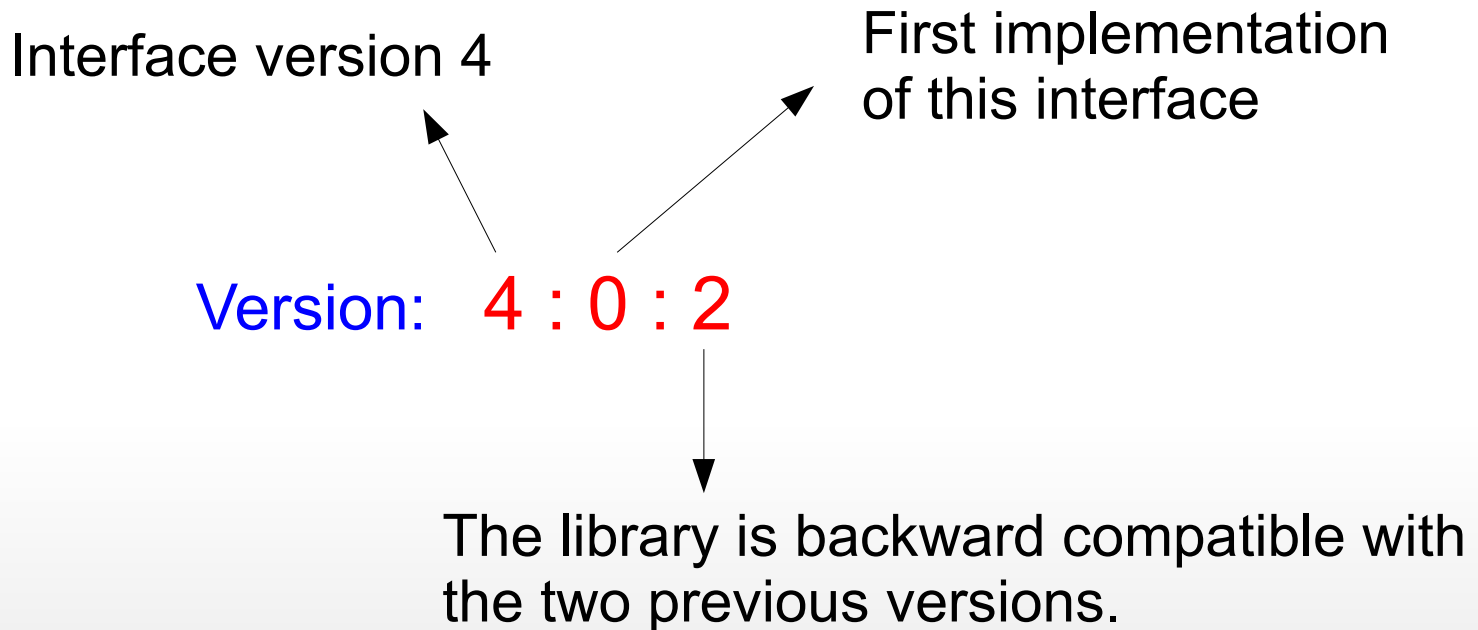
# Controlling the Type of Library

- By default, both static and shared libraries are built
- This default can be changed in a package using two macros in `configure.ac`:
  - `AC_DISABLE_SHARED` do not build shared libraries by default
  - `AC_DISABLE_STATIC` do not build static libraries by default
- The installer can override these settings by using configure options:
  - `--enable-shared` build shared libraries
  - `--disable-shared` don't
  - `--enable-static` build static libraries
  - `--disable-static` don't
- At least one flavor is always built (and in some systems there is no choice)

# Library Names and Versions

- Libtool has its own formal versioning scheme that tracks changes to the library interface
  - An interface is the set of exported entry points into the library
- The library version is composed by three parts:
  - **CURRENT** – The latest interface implemented by the library
  - **REVISION** – The implementation number of the CURRENT interface (e.g., build number or number of bugs fix)
  - **AGE** – The difference between the newest and oldest interfaces that this library implements. In other words, the library implements all the interface numbers in the range from number current - age to current

# Library Names and Versions



It can be linked into executables which were built with a release of this library that exported the current interface number, or any of the previous two interfaces.

# Naming Conventions

- Start with version information '0:0:0' for each libtool library
  - These numbers should be specified using `-version-info` (if not, the default is 0:0:0)

```
lib_LTLIBRARIES = libhello.la  
libhello_la_SOURCES = say.c say.h  
libhello_la_LDFLAGS = -version-info 0:0:0
```

- The version number of a project is a completely different thing of the version number of any libraries shipped with the project
  - It is a very **common mistake** to try to force the libraries to have the same version number as the current release version of the package

# Update the Library Version

- Update the version information only immediately before a public release of your software
- If the library source code has changed since the last update (e.g. for a bugfix) but the interface has not changed, then increment *revision*
- If any interface has been added, removed, or changed since the last update, increment *current*, and set *revision* to 0

$c:r:a \rightarrow c:r+1:a$

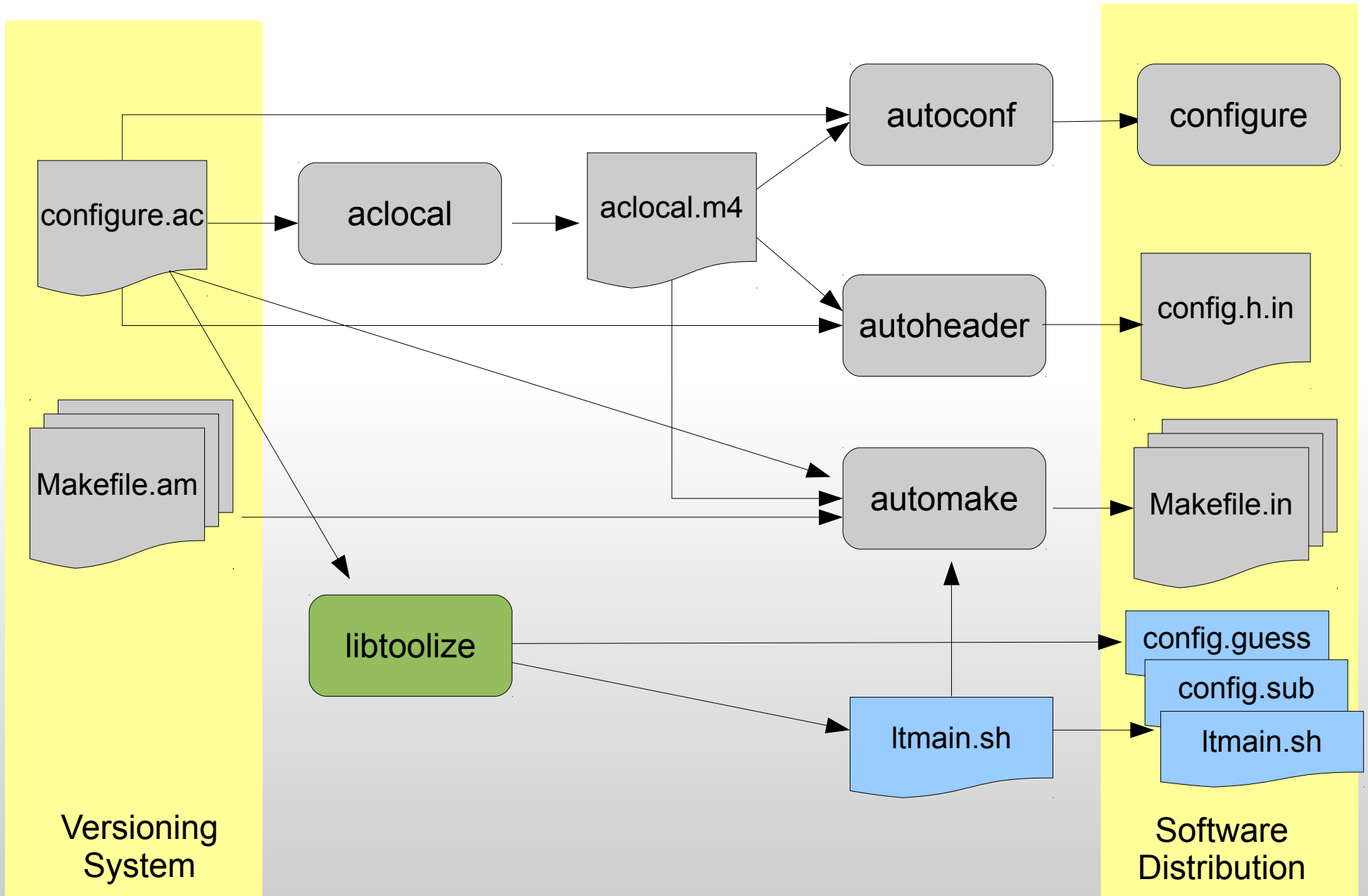
- If the new interface is a superset of the previous interface (that is, if the previous interface has not been broken by the changes in this new release), increment the *age*

$c:r:a \rightarrow c+1:0:a+1$

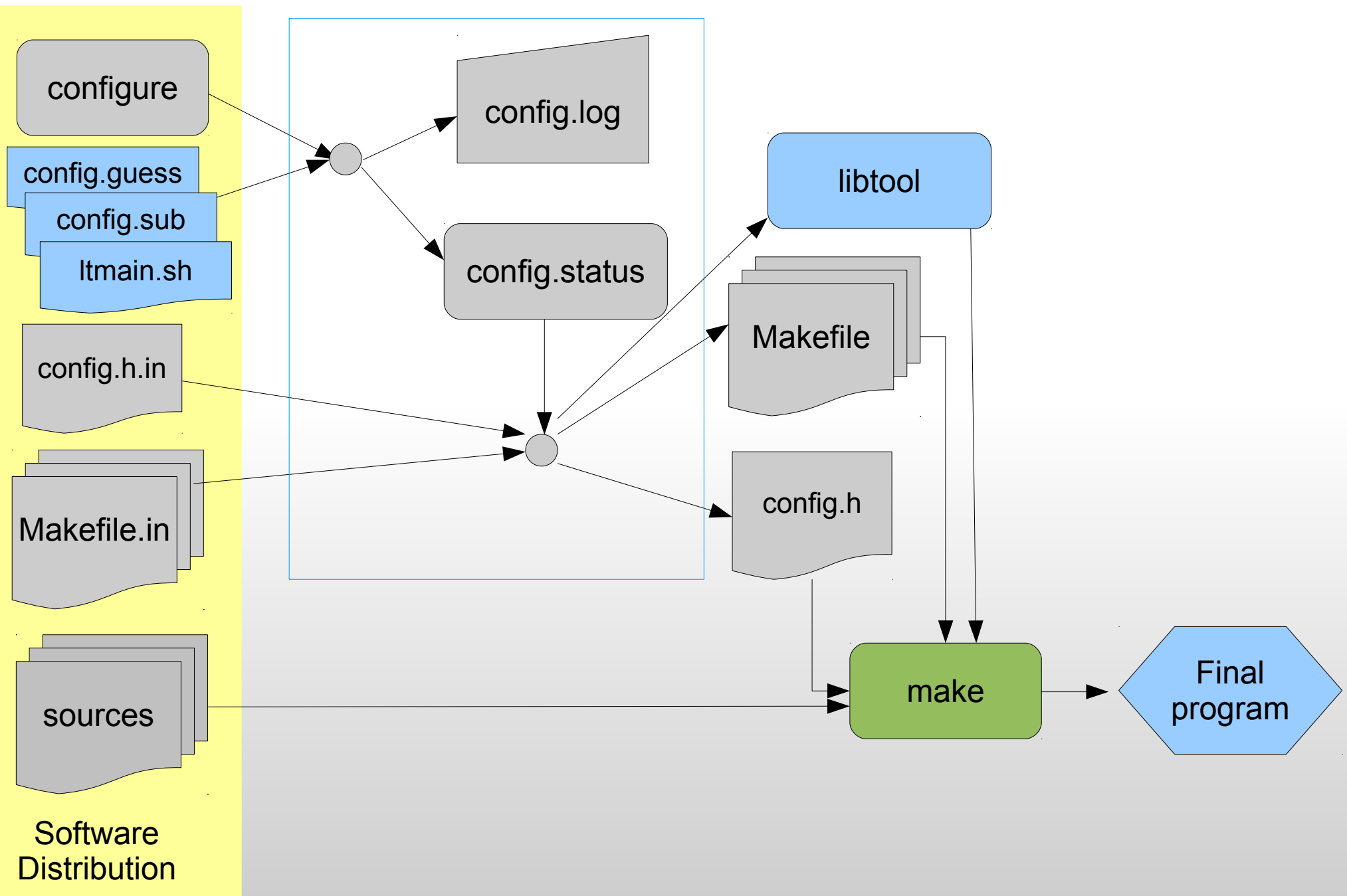
- If the new interface has removed elements with respect to the previous interface, then you have broken backward compatibility, then set *age* to 0

$c:r:a \rightarrow c+1:0:0$

# Developer's View (with libtool)



# Installer's View (with libtool)



# Very Simple Example

src/func.h

```
void f(int x, int y, int z);
```

src/func.c

```
#include <stdio.h>
#include <func.h>

void f(int x, int y, int z){
    printf("%d\n",x+y+z);
}
```

src/main.c

```
#include <func.h>

int main(){
    f(1,2,3);
    return 0;
}
```

Makefile.am

```
SUBDIRS = src
```

src/Makefile.am

```
lib_LTLIBRARIES = libfunc.la
libfunc_la_SOURCES = func.c func.h
libfunc_la_LDFLAGS = -version-info 1:0:0

bin_PROGRAMS = very_simple
very_simple_SOURCES = main.c
very_simple_LDADD = libfunc.la
```

configure.ac

```
AC_INIT([very_simple], [1.0], [davide@foo.bar])
AC_PROG_LIBTOOL
AM_INIT_AUTOMAKE([1.9 foreign])
AC_PROG_CC
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

```
> aclocal
> libtoolize
> autoconf
> automake -a
```

```
configure.ac
configure
Makefile.am
Makefile.in
config.guess
ltmain.sh
config.sub
src/
  Makefile.am
  Makefile.in
  func.c
  func.h
  main.c
autom4te.cache/
aclocal.m4
install-sh
depcomp
missing
config.log
config.status
```

```
> aclocal
> libtoolize
> autoconf
> automake -a
> configure
....
....
>
```

```
configure.ac
configure
Makefile.am
Makefile.in
Makefile
config.guess
ltmain.sh
config.sub
libtool
src/
  Makefile.am
  Makefile.in
  Makefile
  func.c
  func.h
  main.c
autom4te.cache/
aclocal.m4
install-sh
depcomp
missing
config.log
config.status
```

```
> aclocal
> libtoolize
> autoconf
> automake -a
> configure
....
....
> make
....
....
>
```

```
src/
  Makefile.am
  Makefile.in
  Makefile
  func.c
  func.h
  main.c
  very_simple
  libfunc.la
  func.o
  func.lo
  main.o
  .libs/
    very_simple
    libfunc.so.1
    libfunc.la
    func.o
    libfunc.so.1.0.0
    libfunc.a
    libfunc.lai
    libfunc.so
```

```
> make
....
....
> file src/*
func.c:      ASCII C program text
func.lo:    ASCII English text
func.o:     ELF 32-bit LSB ...
libfunc.la: libtool library file
main.o:     ELF 32-bit LSB
very_simple: Bourne-Again shell script text executable
....
```

```
> make
....
....
> file src/*
func.c:      ASCII C program text
func.lo:     ASCII English text
func.o:      ELF 32-bit LSB ...
libfunc.la:  libtool library file
main.o:      ELF 32-bit LSB
very_simple: Bourne-Again shell script text executable
....

> file src/.libs/*
func.o:      ELF 32-bit LSB relocatable...
libfunc.a:   current ar archive
libfunc.la:  symbolic link to `../libfunc.la'
libfunc.lai: libtool library file
libfunc.so:  symlink to `libfunc.so.1.0.0'
libfunc.so.1: symlink to `libfunc.so.1.0.0'
libfunc.so.1.0.0: ELF 32-bit LSB shared object...
very_simple: ELF 32-bit LSB executable,
              dynamically linked (uses shared libs), not stripped
```

# Uninstalled Binaries

(after you run make but before you run make install)

- The program (`very_simple`) is not actually a binary, but a **shell script** which sets up the environment so that when the real binary is called it finds its the shared libraries in the correct locations
  - The real binary is built too, but it is stored in an hidden subdirectory
- If you need to look at the binary with another program (to debug it, for example) you have to use the `libtool` script to run the program

```
> libtool gdb very_simple
```

# Installed Binaries

```
> ./configure --prefix /tmp/test
....
....
> make & make install
....
> cd /tmp/test
> ls
  bin/  lib/

> find . -exec file {} \;
./lib/libfunc.so: symbolic link to `libfunc.so.1.0.0'
./lib/libfunc.so.1: symbolic link to `libfunc.so.1.0.0'
./lib/libfunc.la: libtool library file
./lib/libfunc.a: current ar archive
./lib/libfunc.so.1.0.0: ELF 32-bit LSB shared object..

./bin/very_simple: ELF 32-bit LSB executable, dynamically
                    linked (uses shared libs)
```