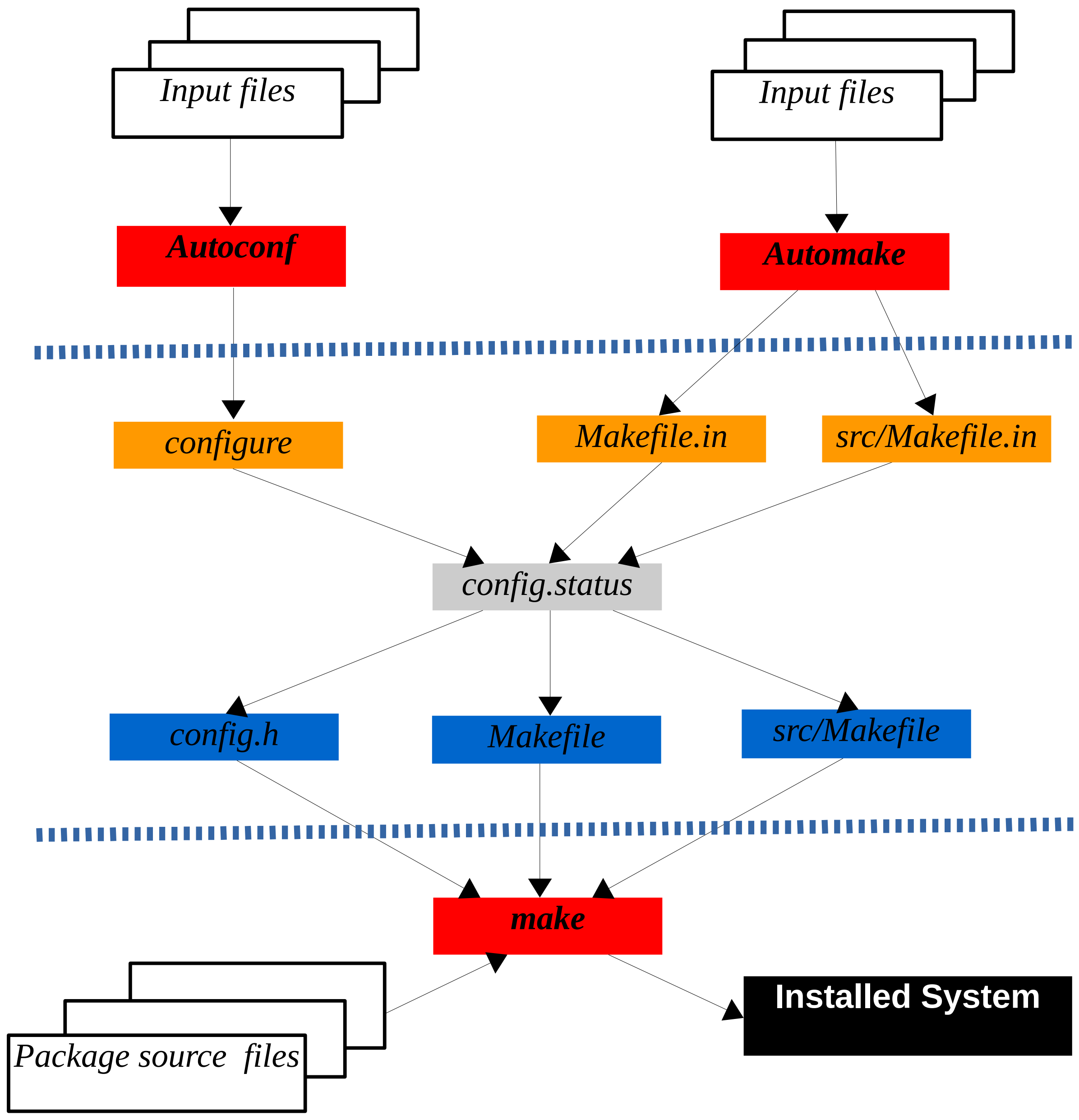
Summary of the GNU Autotools

Overview

Packages are compiled and installed on Linux/Unix platforms using the ubiquitous 'make' utility. 'make' is driven by one or more 'Makefiles' that specify what the 'make' program has to do. Thus each combination of a package and Linux/Unix platform requires 'Makefiles' that are specific to that combination of package and platform.

It is useful to have tools that support the creation of the 'Makefiles'. There are 2 main GNU Autotools to do this. One is called ***Automake*** and its purpose is to support the creation of 'Makefiles' to suit the package. The other is called ***Autoconf*** and its purpose is to support the tailoring of the 'Makefiles' to fit the platform. In practice their usage is intertwined (although in principle they can be used independently of each other). ***Automake*** uses the tool ***Libtool*** to deal with software libraries. There are other subsidiary Autotools as well.

The following diagram summarises how the 2 tools are used together :-



The colour code is as follows :

- A 'white' box represents a file that is an input to the Autotool Build System.
- A 'red' box represents a program in the Autotool Build System.
- A 'yellow' box represents a script file created by a 'red' program.
- A 'grey' box represents a script file created by the execution of a 'yellow' script file.
- A 'blue' box represents a file that is to be input to the 'make' utility program.
- A 'black' box represents the installed system.

The 'blue' dotted lines are used to show that the whole process comprises 3 parts :

1. The ***Autoconf*** and ***Automake*** programs and their input files are located on the platform of the creator/maintainer of the package; they are run there by the package developer/maintainer to create the 'yellow' script files. The tools are not downloaded to the installation platform.
2. The 'yellow' script files are downloaded, with the package source code files, to the user's platform. (Typically all the files are collected and compressed into a **`.tar.gz`** file, which is downloaded and unzipped on the user’s platform). The user executes the **`./`configure** command in order to execute the 'yellow' script files so as to create the 'blue' input files for 'make'. No other user action is required.
3. The user then runs the 'make' utility to compile and install the package on their platform. The 'Makefiles' produced conform to GNU Coding Standards, which among other things, allow the user to :
   - compile the program;
   - clean up, i.e. remove files resulting from the compilation;
   - install the program in the required directory;
   - uninstall the program from where it was installed;
   - create a source distribution archive, i.e. a tarball.

   So 'make' is executed with whatever options are needed to install the package as required.

The diagram assumes that there can be 'Makefiles' in sub-directories as well as in the main directory. The tools support a GNU standard file system hierarchy, implemented via defaults; *src* is a standard sub-directory name for source files.

The tools are generalised so that they can cope with the installation of library code in addition to applications.

This overview omits a number of files that are often involved in practice, and may end up on the user's platform.

Automake

The purpose of ***Automake*** is to enable a programmer to write a high-level specification of a package's build requirements – i.e. what needs to be built and where it should it be installed. It makes the programmer more productive by relieving them of the tedious and error-prone detail involved in writing a 'Makefile' *per se*.

Automake is written in Perl; it uses ***Libtool*** to handle both static and shared libraries.

The specification for each required 'Makefile' is written in 'make' syntax. It is a series of 'make' variable definitions, with additional rules as required. The specification is held in a file called *Makefile.am*, which is input to ***Automake***, which outputs a shell script held in a file called *Makefile.in*.

Each *Makefile.am* contains a series of 'make' variable definitions, supplemented by Makefile-type rules where necessary. At its most basic, a *Makefile.am* contains :

- a line declaring the name of the program to build;
- a list of source code files;
- a list of command line options to be passed to the compiler (e.g. the directories containing the relevant header files);
- a list of command line options to be passed to the linker (e.g. the libraries needed and the directories containing them).

From the information in a *Makefile.am* file, ***Automake*** generates a *Makefile.in* file.

More sophisticated functionality is also provided. For example, ***Automake*** handles dependency information, so that if a source file is modified, the next invocation of 'make' will know which files to re-compile and link.

Autoconf

The main aim of ***Autoconf*** is to make it easy for a user to install a package on a platform. Supporting the developer/maintainer is a secondary goal.

The input to ***Autoconf*** is a file called *configure.ac*[1] which lists the platform-specific build and run-time features that the package needs or can use. The platform must be checked to ensure that they are all available; if they are, when the 'Makefiles' are created, they are configured to take account of the platform's specific features.

From its *configure.ac* input file, ***Autoconf*** creates a shell script file called *configure*. It is the *configure* file that is run on the installation platform, and which for each *Makefile.in* file, generates a 'Makefile' to be used with 'make' on the installation platform.

configure.ac is a shell script written in terms of macros; indeed it may contain only macros. Each macro encapsulates the searching for, checking, and/or testing of a specific feature, i.e. it executes one specific function. Apart from an initialisation macro, which must come first, and an output macro, which must come last, the macros appear in a 'partial sequence'; i.e. the sequence is flexible, requiring only that a macro appears after any other(s) that produce necessary input for it.

So ***Autoconf*** may be viewed as essentially a macro processor that converts *configure.ac* into *configure*, a fully-fledged shell script.

Libtool

Because many Linux/Unix systems have their own format for shared libraries, and compiling and linking can require different build flags in many cases, ***Libtool*** is used to avoid the complication this can cause by providing a single, simple new library format that abstracts from all the different variants, includes static libraries, and can be used instead for all libraries.

The new format is referenced by a call to *lib**name**.la*, where ***name*** is the name of the library and *la* stands for 'libtool archive'.

Libtool consists of a wrapper script that translates operations involving *lib**name**.la* into the correct operations for the current platform's real library. In a *Makefile.am*, the link is to **.la* files.

1 The file *configure.ac* is sometimes known as *configure.in*. They are both names for the same file. *configure.ac* is now the preferred name, although the name *configure.in* still works. It is recommended that the **.in* suffix is now only used for files processed by the *config.status* script.

configure.ac and the various *Makefile.am* files need to be adjusted accordingly to cope with the relevant macros, etc.

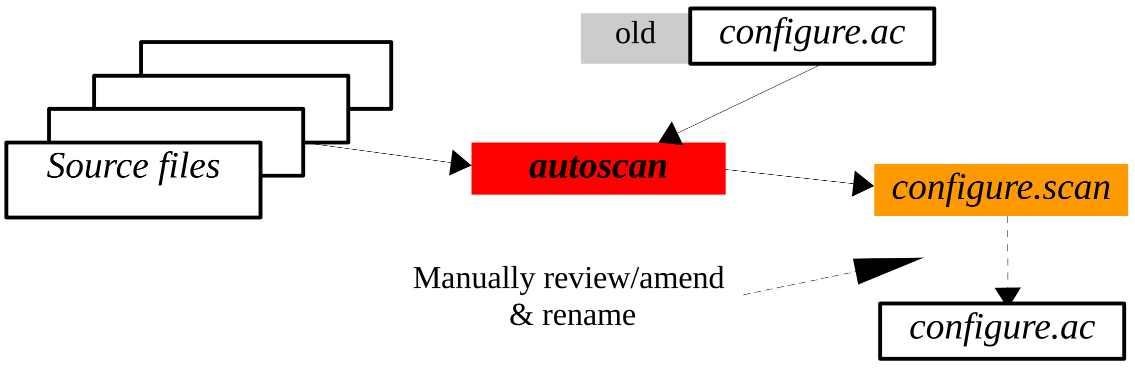
The Usage of Automake & Autoconf

The overview of ***Automake*** and ***Autoconf*** used together omits a number of other files and subsidiary Autotools that are, at least optionally, involved in their operation. Therefore a more detailed description is now given.

Some of these other files are script files, and end up on the user's platform; so it is useful to know what they are, and whether they should or could be removed.

Creating File '*configure.ac*'

The *configure.ac* file can be written manually. Alternatively the ***Autoscan*** program can be used to examine package source files in the directory tree for common portability problems; it creates a file called *configure.scan*, which is a preliminary *configure.ac* file. After manually checking it, and amending it if necessary, it must be renamed *configure.ac* in order for it to be used thereafter. ***Autoscan*** can also check any pre-existing *configure.ac* for completeness. The following diagram summarises this :-



Since *configure.ac* is read by several Autotools, it is worthwhile to make the consequent macro processing efficient. The actual macro processing is carried out by the general-purpose GNU M4 macro processor included in all Linux/Unix systems. However instead of running M4 directly, the Autotools invoke it via ***autom4te***. The Autotools have a number of common needs when applying M4, and these are factored into a layer above M4, namely ***autom4te***, which supports them via macro processing extensions designed to support the Autotools.

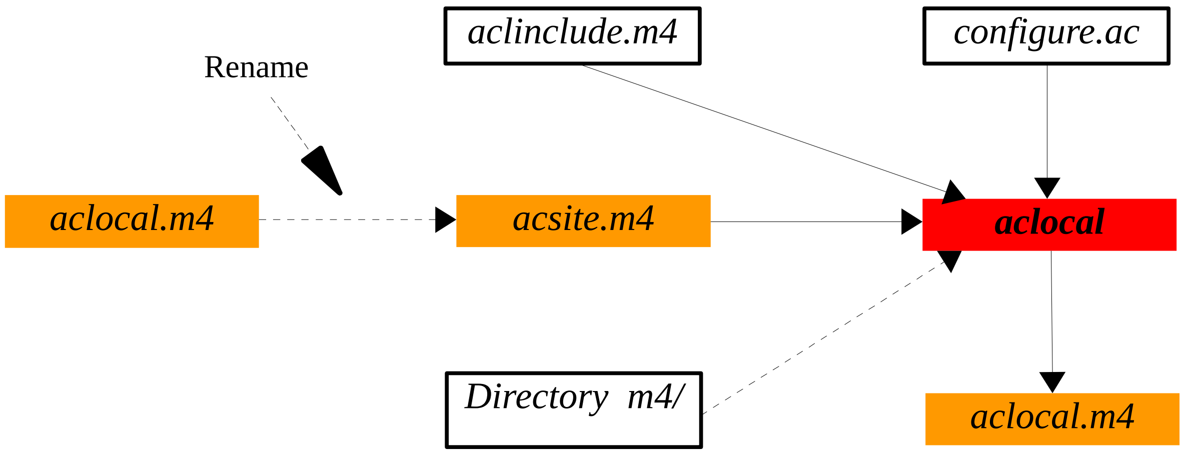
One common need is that for storing files of information derived from M4 macros when one Autotool runs, for use when a subsequent Autotool runs; this can save a lot of subsequent re-processing. The information is stored in files in directory *autom4te.cache*. (Since *autom4te.cache* is a cache, it can be removed later if necessary).

Creating File '*aclocal.m4*'

In order for ***Autoconf*** to process its input *configure.ac* file, it needs all the macros relevant to the processing collected together into one file, to be called *aclocal.m4*. The subsidiary Autotool ***aclocal*** does this.

All the stock macros, required for common configuration checks, are installed with ***Autoconf***, and held in the file *aclocal.m4*. This set of macros may need to be expanded to include others needed to install the package. To enlarge the set of macros, *aclocal.m4* is renamed *acsite.m4*, and then ***aclocal*** reads the contents of *acsite.m4*, *acinclude.m4* (containing local macros) as well as the input file *configure.ac* to create a new *aclocal.m4* file, which now contains the expanded set.

This can be portrayed schematically as :-

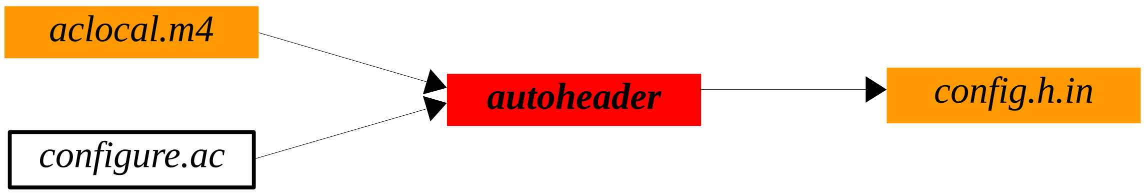


A new alternative to using *acinclude.m4* is instead to collect additional macros in individual *.m4* files in a directory called *m4*, and have ***aclocal*** read and add all those macros to those in *aclocal.m4*.

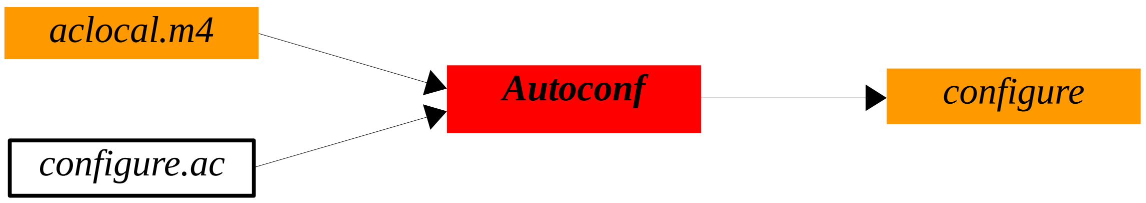
In due course, the functionality of ***aclocal*** is to be incorporated into ***Autoconf***.

Creating File '*config.h.in*'

The utility ***autoheader*** reads *configure.ac* and *aclocal.m4* to create a file called *config.h.in* containing macro definitions for the package's *.cpp* files :-



Running *Autoconf*

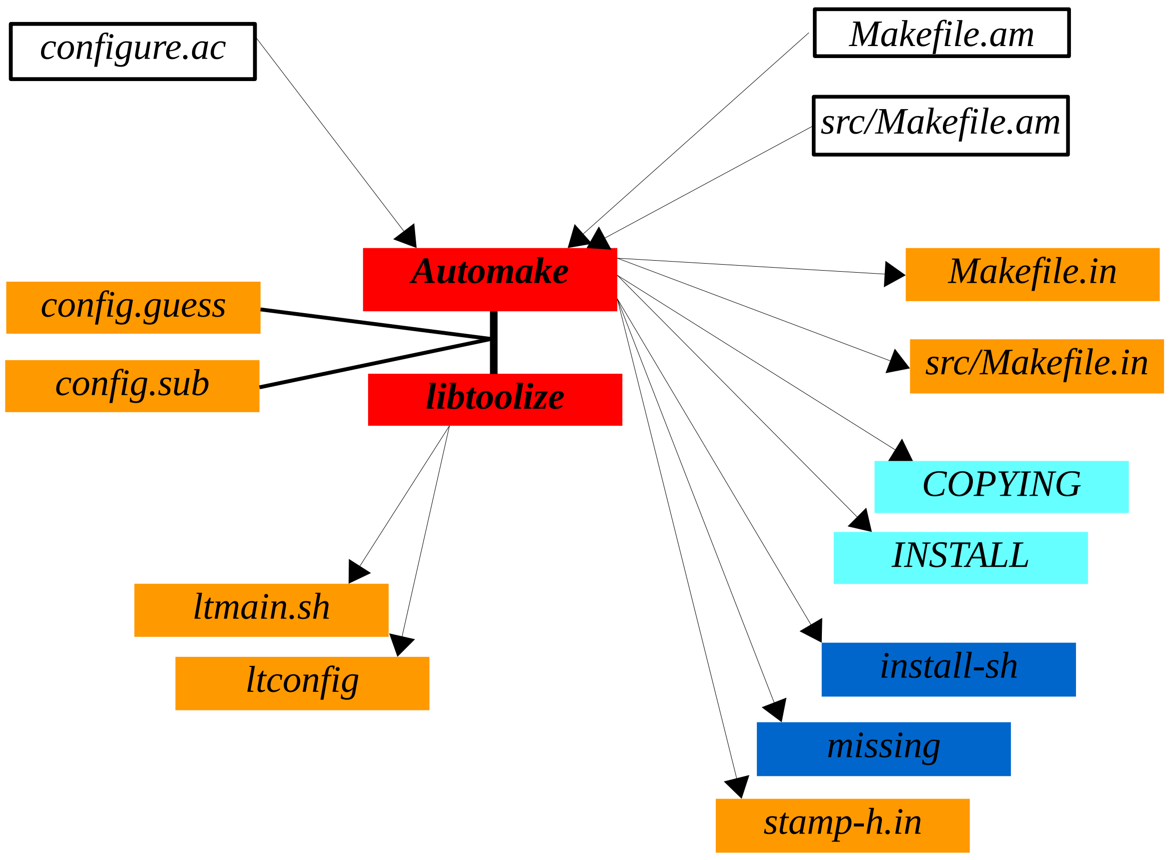


When ***Autoconf*** is run with those same 2 file inputs, it creates the *configure* script file. It is not necessary to understand the detailed, technical contents of *configure*.

Running *Automake*

When ***Automake*** is run, it generates a *Makefile.in* shell script for each *Makefile.am* file that is input. It is not necessary to understand the detailed, technical contents of *Makefile.in* files*.*

Automake also generates other script files to be used by 'make', together with other standard files specified by the GNU Programming Standard for package installations :-



Note the following :

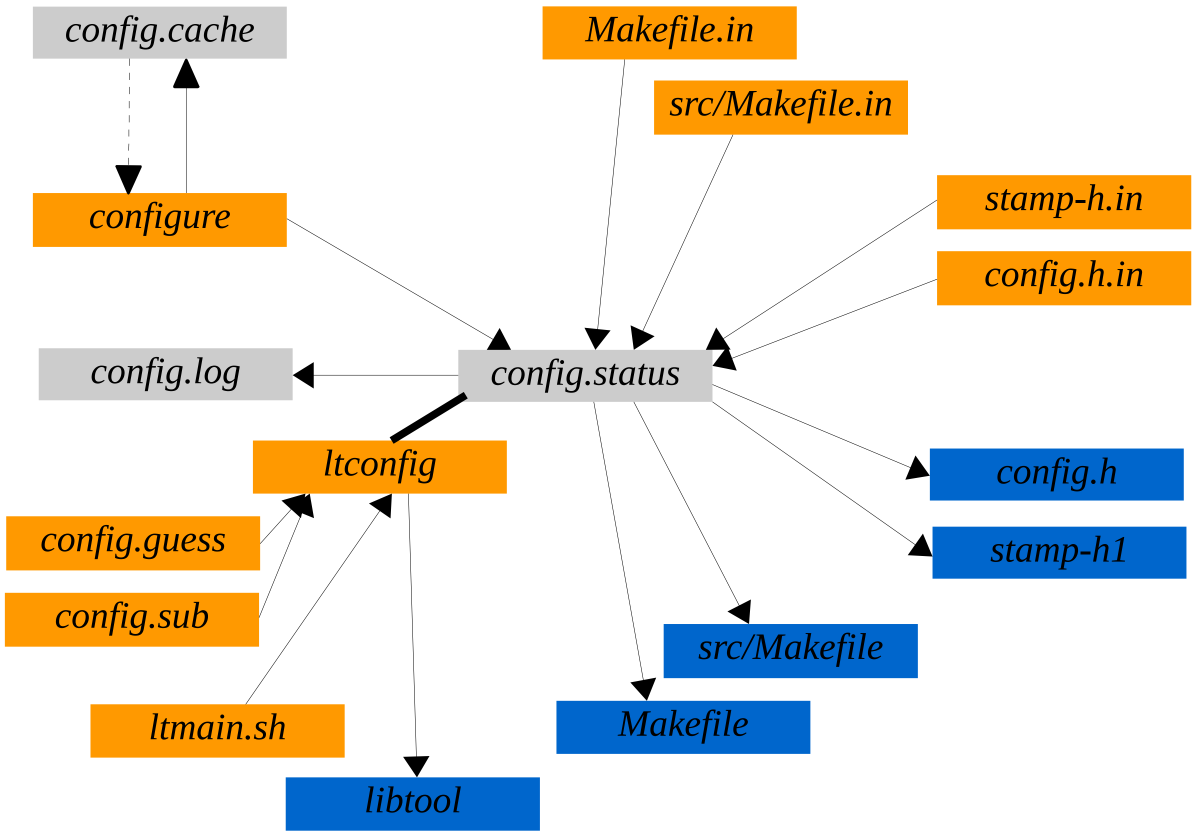
- The input files are one *configure.ac* file and as many *Makefile.am* files as is required by the installation of the package on the platform.
- ***Automake*** uses the ***libtoolize*** program where necessary to handle static and shared libraries. The latter generates the script files *ltmain.sh* and *ltconfig* which are the means by which the libraries will actually be set up on the installation platform.
- The script files *config.guess* and *config.sub* are used by ***Automake*** and ***libtoolize*** to derive information about the system on which the installation will be built. *config.guess* determines the type of the computing system; it returns a 'configuration name' of the form “CPU-Manufacturer-Kernel-Operating System”, or an abbreviated version of this. *config.sub* translates abbreviated names into the full canonical form.
- A *Makefile.in* is generated and output for each *Makefile.am* input. *Makefile.in* files are complex script files, just as the *configure* file is, which will be used in the next phase to generate the actual 'Makefiles' required to install the package on the installation platform.
- The output files *COPYIN*G and *INSTALL* are standard 'boilerplate' files created to conform to the GNU package installation standards. (Hence the ‘cyan’ colour). Depending on the parameters input to ***Automake***, other such files in the GNU standard can be output as well, e.g. *AUTHORS*, *NEWS*, *README*, and *ChangeLog*. Note that these files are created to be part of the package installation, and so will appear on the platform as an ancillary part of the installation.
- The script file *install-sh* should come with ***Automake*** for use with the expansion of the AC_PROG_INSTALL macro. The Linux ‘Install’ utility copies files from source

locations to destination locations and sets their permissions. Since this utility can vary between platforms, *install-sh* acts as a wrapper round the local 'Install', so that a 'Makefile' can always install files as required on the user's platform.

- The script file *missing* is a wrapper for use around several of the Autotools. If a 'Makefile' triggers a re-build that requires the use of an Autotool, then as that Autotool is not (normally) on the user's platform, the *missing* script will output a helpful and meaningful error message to the user; should the tool actually be on the user's platform, it will allow it to be executed as normal.
- *stamp-h.in* is output as a timestamp file to indicate whether *config.h.in* is up-to-date; it contains the time expressed as a 'timestamp' string. This allows *config.h.in* to be marked as up-to-date without actually modifying it, which can be useful since *config.h.in* depends upon *configure.ac*, but it is easy to change *configure.ac* in a way which does not affect *config.h.in*.

Running the *configure* File

The previous steps have created all the 'yellow' build system files. The final Autotool step is to run *configure*, by executing **`./configure`**. Executing *configure* probes the target system for functions, libraries and tools on the platform that the package needs. It then creates the script file *config.status*, and executes it. It is *config.status* that generates all the files required by 'make' from the input files provided :-



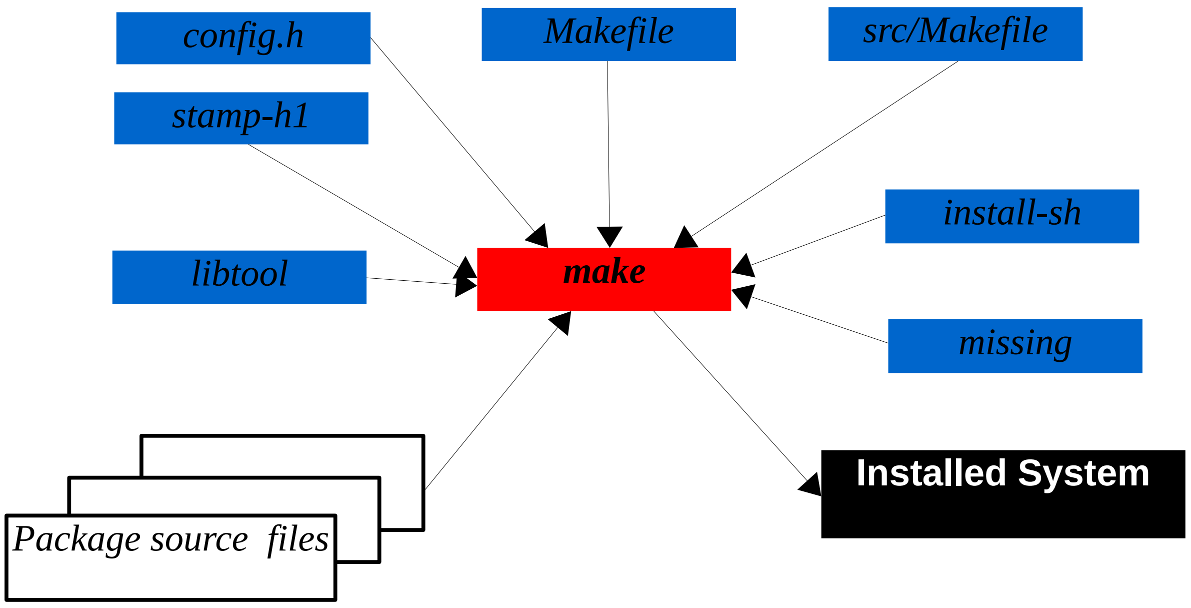
Note the following :

- The 'grey' and 'blue' files are created during the process and both will exist at the end of it, although only the 'blue' files are needed by 'make'.
- *configure* can take configuration 'variables' as input parameters to control how it performs. Run **`./configure –help`** to get a full list of them.

- *configure* generates a *config.cache* file which contains the results of the system probing. *config.cache* can be used to speed up any future re-configurations, so *configure* may also read in any pre-existing *config.cache* file.
- The *config.status* script creates the 'Makefiles' from the *Makefile.in* files, the *config.h* file from the *config.h.in* file, and the *stamp-h1* file from the *stamp-h.in* file. In doing this, *config.status* applies GNU coding standards, which include such things as :
  - program behaviour (e.g. error reporting, standard command line options),
  - coding style
  - configuration,
  - Makefile conventions.
- The *config.h* file contains all the '#define' statements required by the compiler. *stamp-h1* enables 'make' to consider *config.h* as up-to-date and avoid re-compilation even when the latter's modification time indicates otherwise.
- *config.status* also creates a *config.log* file in which is recorded messages produced by compilers (to help with debugging if *configure* makes a mistake).
- *config.status* can be run on its own to regenerate files without re-running the system probing.
- If one or more libraries are to be set up, *config.status uses the ltconfig* shell script file to do the work. *ltconfig* takes input from the *ltmain.sh* file and outputs the *libtool* file. *libtool* will be used by 'make' to actually handle the libraries; it is invoked as a wrapper script around compilers, linkers, install and cleanup programs.
- *ltconfig* uses the script files *config.guess* and *config.sub* for the same purposes as before.

Running 'make'

It is the execution of **`make`** commands after executing **`./configure`** that actually installs the package. The following diagram summarises the files involved in executing **`make`** commands (excluding 'Boilerplate' files) :-



Use of *Autoreconf*

If the package is updated, and/or the package's build system is updated, and/or any Autotools are updated, the ***Autoreconf*** program is used to update everything, re-generate all the input files required by **`./configure`** command and **`make`**.

Autoreconf takes the *configure.ac, Makefile.am* and any *src/Makefile.am* files as input, and runs ***Autoconf***, ***Autoheader***, ***aclocal***, ***Automake***, ***libtoolize*** and ***autopoint*** repeatedly, as appropriate, to output an updated package's build system. By default, ***Autoreconf*** only remakes those files that are older than their sources, but it can be given a parameter that forces it to update all files regardless.

Sequence of File Usage

1. **User-Provided Input Files**

   configure.ac
   Write manually, or apply ***autoscan*** to the package source files and amend the result if necessary.

   Makefile.am
   Write manually for each directory in which a 'Makefile' is required.

2. **User-Generated Input Files**

   aclocal.m4
   Generate using ***aclocal***, with possible manual input.

   config.h.in
   Generate using **autoheader**.

3. **System-Provided Support Files**

   install-sh, ***missing***, ***config.guess***, ***config.sub***.
   Normally provided with ***Automake*** and ***libtoolize***.

4. **Execute Autoconf**

   This generates ***configure***.

5. **Execute Automake**

   This generates a ***Makefile.in*** for each ***Makefile.am***, ***stamp-h.in***, if libraries are to be created, ***ltmain.sh*** and ***ltconfig***, and the 'Boilerplate' files.

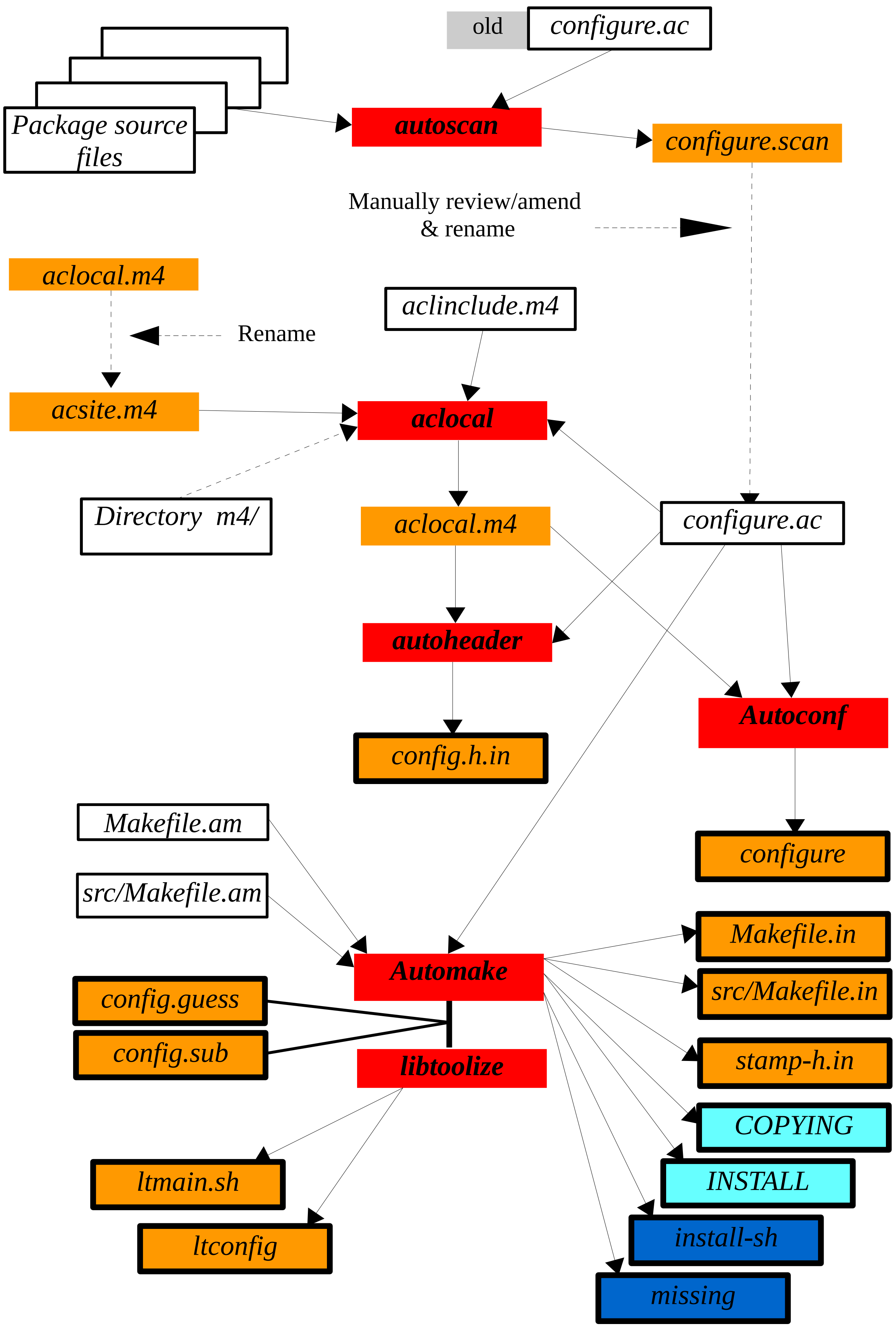
6. **Include the following files in the 'tarball' which contains the package's source files.**

   All the ***Makefile.in*** files, ***stamp-h.in***, ***ltmain.sh*** and ***ltconfig*** if libraries are to be created, the 'Boilerplate' files, ***configure***, ***install-sh***, ***missing***, ***config.guess***, ***config.sub***, and ***config.h.in***.

7. **Download the 'tarball' to the installation platform.**
   Unpack the 'tarball'.
   Execute './configure'.
   Execute the 'make' command to build the package.
   If required, execute 'make install' to copy the software to the right location with the right file permissions.

Generating the Autotool Files for the 'Tarball'

The following diagram combines the previous diagrams for the stages required to create the build system for the package.



Coloured boxes with black outlines represent files to be downloaded to the user’s platform.