Desenvolvimento de Software para *UNIX*

Rui Carlos A. Gonçalves

29 de Agosto de 2008

Resumo

Neste texto pretende-se descrever formas de desenvolver programas para *UNIX* usando as ferramentas da *GNU*. Não é, no entanto, objectivo deste texto ensinar a programar, mas sim ensinar a criar, a partir do ficheiro que contêm o código do programa, um arquivo de fácil instalação em qualquer plataforma *UNIX*.

Conteúdo

1 Introdução

Quando desenvolvemos um programa em C/C++ habitualmente criamos uma *makefile*, que nos permite automatizar a criação do executável final. Mas uma *makefile* pode não

funcionar correctamente em todos os sistemas *UNIX*, devido a pequenas diferenças entre as várias versões existentes, nomeadamente no nome do compilador de C/C++.

A forma normalmente utilizada para ultrapassar este problema passa por, em vez de uma *makefile*, usarmos uma *script* (`configure`) que gera a *makefile* adaptada ao sistema.

Ao longo deste texto serão descritos os passos para a criação dessa *script*. De referir que este texto é apenas uma breve introdução ao assunto em causa e, como tal, serão indicados os passos a dar para pequenos projectos em que apenas se pretende criar um executável.

2 Makefile.am

O primeiro passo consiste em criar o(s) ficheiro(s) `Makefile.am`. Nestes ficheiros são indicados os directórios que fazem parte do projecto, os programas a serem compilados, as bibliotecas a serem criadas, etc. Quando o nosso projecto tem apenas um directório só precisamos de criar um ficheiro.

Supondo que o nosso executável final se vai chamar `prog-3.1`, e que necessita dos ficheiros `file1.c`, `file2.c` e `file3.c`, deverão ser incluidas no ficheiro as seguintes linhas:

```
bin_PROGRAMS = prog-3.1
prog_3_1_SOURCES = file1.c file2.c file3.c
```

Na primeira linha, através da macro `bin_PROGRAMS` indicamos o nome do executável. Na segunda, através da macro `_SOURCES`, indicamos os ficheiros necessários à criação do executável `prog-3.1`. De notar que, no nome do executável que aparece antes da macro, os caracteres '-' e '.' foram substituídos pelo carácter '_'.

Podemos também criar bibliotecas estáticas ou dinâmicas, no entanto, dada a complexidade associada à construção de bibliotecas dinâmicas, este tópico não será abordado neste texto. Vejamos então como criar uma biblioteca estática. Em primeiro lugar indicamos quais as bibliotecas a serem criadas através da macro `noinst_LIBRARIES`. Depois indicamos quais os ficheiros que serão utilizados por cada uma das bibliotecas indicadas anteriormente, através da macro `_SOURCES`. Para indicarmos que um programa usa uma biblioteca utilizamos a macro `_LDADD`. Ou seja, pegando no exemplo apresentado anteriormente, assumindo que também necessita das bibliotecas `libx.a` e `liby.a`, o ficheiro `Makefile.am` ficaria assim:

```
noinst_LIBRARIES = libx.a liby.a
libx_a_SOURCES = ...
liby_a_SOURCES = ...

bin_PROGRAMS = prog-3.1
prog_3_1_SOURCES = file1.c file2.c file3.c
prog_3_1_LDADD = libx.a liby.a
```

Quando queremos construir bibliotecas é necessário adicionar a macro `AC_PROG_RANLIB` ao ficheiro `configure.ac` (ver secção 3).

Se o nosso projecto for constituído por vários directórios temos que criar um ficheiro `Makefile.am` em cada directório. No ficheiro do directório de topo temos que indicar quais os subdirectórios do nosso projecto. Isso é feito através da macro `SUBDIRS`.

Por vezes podemos ter necessidade de executar operações adicionais, por exemplo, criar um directório durante a instalação, remover certos ficheiro quando fazemos um *clean*, etc. No entanto não é possível acrescentar dependências aos "alvos" criados pelo `automake` sem definir toda a regra, então como podemos resolver este tipo de problemas? Vários alvos suporta uma versão `'-local'`, que podemos criar no `Makefile.am`. Este alvo é executado em conjunto com o que lhe dá origem. Por exemplo, caso queira que durante a instalação seja criada a pasta `'/xpto'` e que a mesma seja removida quando o programa for desinstalado, basta acrescentar as seguintes linhas ao ficheiro `Makefile.am`:

```
install-data-local :
        mkdir /xpto

uninstall-local :
        rm -r /xpto
```

3 Autoscan (configure.ac)

O `autoscan` é uma aplicação que percorre os directórios do projecto e gera um ficheiro `configure.scan`, que servirá de base para o ficheiro `configure.ac`[1]. Depois de executar este comando editamos o ficheiro `configure.scan`, *corrigindo* as informações associadas à macro `AC_INIT` e adicionando na linha seguinte a macro `AM_INIT_AUTOMAKE`. Tal como referido anteriormente, quando são construídas bibliotecas, é necessário incluir também a macro `AC_PROG_RANLIB`. Por fim, alteramos o nome do ficheiro para `configure.ac`.

4 Aclocal

Cria o arquivo `aclocal.m4`. Começa por pesquisar os ficheiros '.m4' encontrados e depois percorre o ficheiro `configure.ac`, copiando as macros encontradas para o ficheiro `aclocal.m4`.

[1] Inicialmente este ficheiro chamava-se `configure.in`; embora, por questões de compatibilidade, esta denominação ainda seja aceite devemos usar `configure.ac`

5 Autoheader

O `autoheader` percorre o ficheiro `configure.ac` para depois criar o ficheiro `config.h.in`, usado pela *script* `configure`. A partir deste será criado o ficheiro `config.h`, que será usado na compilação dos programas.

6 Automake

O `automake` gera um ficheiro `Makefile.in` a partir de cada `Makefile.am`. Os ficheiros criados serão usados pela *script* `configure` para a criação das *makefile's*.

Por *default*[2] verifica a existência de certos arquivos padrão da *GNU* (`INSTALL`, `NEWS`, `README`, `AUTHORS`, `ChangeLog` e um dos seguintes: `COPYING.LIB`, `COPYING.LESSER` ou `COPYING`). Para *desactivar* esta verificação devemos usar a opção `--foreign`.

Podem ainda ser necessários outros ficheiros, que podem ser criados pelo `automake`. Para tal devemos usar a opção `-a`. Se só utilizarmos está opção serão apenas criados *links* para os ficheiros (que estão no directório do *Automake*). Para que os ficheiros sejam efectivamente copiados é necessário usar a opção `-c` (juntamente com a opção `-a`).

7 Autoconf

Por fim o `autoconf` gera a *script* `configure`. Esta *script* permite criar uma *makefile*adaptada ao sistema em causa e onde podemos especificar opções de compilação, directórios de instalação, compilador a ser usado, etc.

8 Exemplo

Nesta secção será apresentado um exemplo para um programa concreto. A construção do exemplo foi feita usando o sistema operativo Mac OS X 10.4.5 (Tiger), o automake 1.9 e o autoconf 2.59.

8.1 Descrição do projecto

Vamos construir o programa `prog` a partir dos ficheiros `main.c` e `array.c` e das bibliotecas (estáticas) `libio.a` e `libutil.a`. Por sua vez a biblioteca `libio.a` é obtida a partir de `read.c` e `print.c` e a biblioteca `libutil.a` a partir de `data.c` e `sort.c`. Na figura 1 é apresentada a árvore que representa o projecto. Os ficheiros com o código encontram-se no subdirectório `src`.

[2] A opção de verificação *default* é `--gnu`; para além desta existem ainda as opções `--foreign`, `--gnits` e `--cygnus`

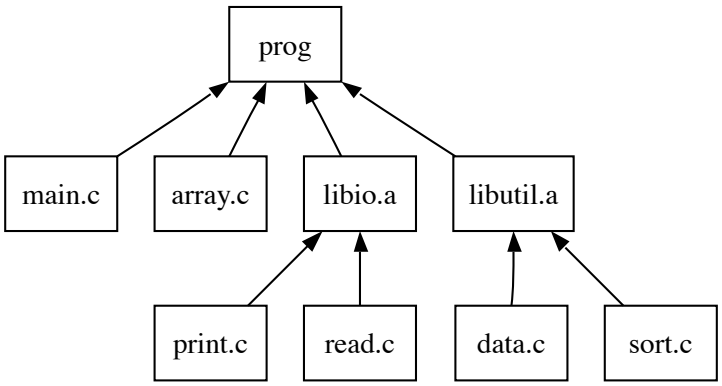

Figura 1: Árvore do projecto.

8.2 Ficheiros `Makefile.am`

Vamos precisar de dois ficheiros: um no directório de topo e outro em `src`.

`./Makefile.am`

```
SUBDIRS = src
```

`./src/Makefile.am`

```
noinst_LIBRARIES = libio.a libutil.a
libio_a_SOURCES = read.c print.c
libutil_a_SOURCES = data.c sort.c

bin_PROGRAMS = prog
prog_SOURCES = main.c array.c
prog_LDADD = libio.a libread.a
```

8.3 Ficheiro `configure.ac`

Para obter este ficheiro começamos por executar o comando `autoscan`. Este programa gerou o ficheiro `configure.scan` com o seguinte conteúdo:

```
1 #                                               -*- Autoconf -*-
2 # Process this file with autoconf to produce a configure script.
3
4 AC_PREREQ(2.59)
5 AC_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)
6 AC_CONFIG_SRCDIR([src/array.c])
```

```
7  AC_CONFIG_HEADER([config.h])
8
9  # Checks for programs.
10 AC_PROG_CC
11
12 # Checks for libraries.
13
14 # Checks for header files.
15
16 # Checks for typedefs, structures, and compiler characteristics.
17
18 # Checks for library functions.
19
20 AC_CONFIG_FILES([Makefile
21                  src/Makefile])
22 AC_OUTPUT
```

De seguida fazemos as seguintes alterações ao ficheiro:

- Actualizamos a linha 5, colocando lá as informações do nosso projecto;
- Alteramos a linha 6 para `AC_CONFIG_SRCDIR([src/main.c])`
- Depois da linha 5 adicionamos a linha `AM_INIT_AUTOMAKE`;
- Depois da linha 10 adicionamos a linha `AC_PROG_RANLIB` (necessário pois vamos criar bibliotecas);

No final obtemos o seguinte ficheiro:

```
1  #                                               -*- Autoconf -*-
2  # Process this file with autoconf to produce a configure script.
3
4  AC_PREREQ(2.59)
5  AC_INIT(main, 1.1, email@email.pt)
6  AM_INIT_AUTOMAKE
7  AC_CONFIG_SRCDIR([src/main.c])
8  AC_CONFIG_HEADER([config.h])
9
10 # Checks for programs.
11 AC_PROG_CC
12 AC_PROG_RANLIB
13
14 # Checks for libraries.
```

```
15
16 # Checks for header files.
17
18 # Checks for typedefs, structures, and compiler characteristics.
19
20 # Checks for library functions.
21
22 AC_CONFIG_FILES([Makefile
23                  src/Makefile])
24 AC_OUTPUT
```

Alteramos então o nome do ficheiro para `configure.ac`.

8.4 Script `configure`

Agora é só executar os seguintes comandos:

```
aclocal
autoheader
automake -a -c --foreign
autoconf
```

Podemos ainda executar o comando `automake -a -c`. O único efeito que este comando terá, será copiar para o directório do projecto os ficheiros `INSTALL` e `COPYING`. O ficheiro `INSTALL` é particularmente importante pois contém indicações de como instalar o programa.

No final já temos a *script* `configure`, a partir da qual podemos gerar a *makefile*que permite, entre outras coisas, compilar e instalar o programa.

Agora para instalar o programa é só executar os comandos:

```
./configure
make
make install
```