Tools di autoconfigurazione
parte 1^ (base)

mobytrick

15 settembre 2014

In ambiente Linux spesso la distribuzione di un prodotto avviene fornendo il sorgente dei programmi più un file di (auto)configurazione `configure`. Per poter fruire del prodotto la trafila da seguire è la seguente:

1. `./configure`
   lo script esegue una serie di controlli per assicurarsi che tutto il software necessario sia presente. Alla fine viene approntato il Makefile, tarato specificatamente per l'ambiente in cui si opera

2. `make`
   vengono generati l'eseguibile ed in generale tutti i file di contorno. Potrebbero esserci, ad esempio, librerie, file di configurazione, man pages, e via discorrendo

3. `make install`
   l'eseguibile e gli eventuali file di contorno vengono spostati nelle locazioni di utilizzo. Per questa fase potrebbero essere necessari privilegi da superuser

In tutto questo procedimento il punto focale è lo script configure. Tale script assicura che tutto l'occorrente per un determinato prodotto/pacchetto sia presente. È evidente che progetti differenti abbiano controlli diversi ma non è pensabile di scriverlo manualmente. È prassi comune generarlo in maniera abbastanza automatizzata. Tra le varie fasi, una in particolare si incarica di leggere i sorgenti e, in base a quanto trova, predispone gli opportuni controlli (predisporre è differente da effettuare!).

Passando ai prerequisiti software, è necessaria la presenza dei pacchetti `grep`, `automake`, `make`, `m4` e `perl`. Inoltre, visto che il linguaggio di programmazione

è il C, è necessaria la presenza del relativo compilatore. Ma si potrebbe optare anche per la presenza di quello del C++.

Per comodità assumiamo la seguente situazione di partenza. I sorgenti sono costituiti da due file C, `main.c` e `get_ora.c`. Si trovano entrambi nella directory di lavoro `esempio` e concorrono a formare l'eseguibile `esempio`. Non c'è nessun nesso tra nome dell'eseguibile e directory, solo una voluta coincidenza. Per ottenere l'eseguibile il comando è[1]:

```
# gcc -ansi -O2 -pedantic -s -Wextra -Wall -o esempio
main.c get_ora.c
```

Per generare lo script di configurazione configure seguiremo la seguente *roadmap*:

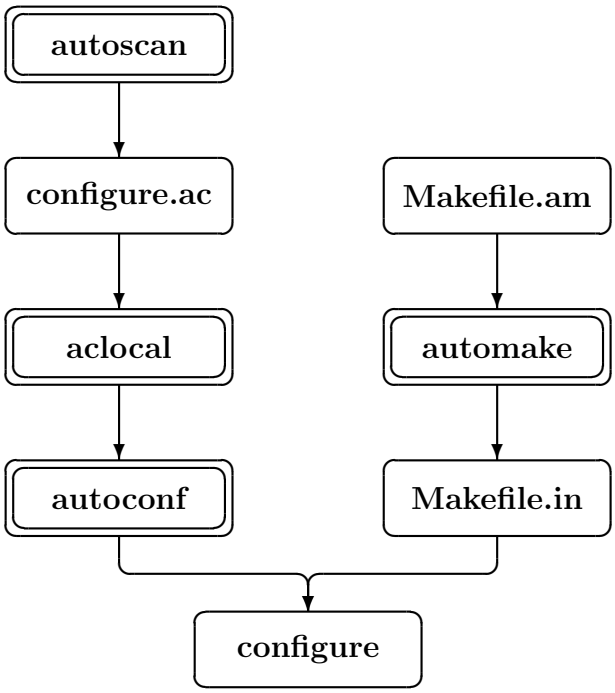

Figura 1: Roadmap

Devono essere allestiti i file iniziali `Makefile.am` e `configure.ac`. Tramite le tre utility `aclocal`, `autoconf` e `automake` vengono generati lo script di configurazione ed i file di contorno.

[1]per motivi tipografici i comando appare su due linee

La sequenza delle operazioni si basa sul fatto che i due programmi C, il cui source è riportato a pag. 7, sono tutto sommato abbastanza semplici. Nel caso in cui ci siano di mezzo degli header file locali e/o delle librerie, il percorso è leggermente più complicato ma la complessità dei file da cui partire è nettamente superiore, come spiegato nell'apposito paragrafo. Per prima cosa i sorgenti vengono collocati in un'apposita cartella -che supporremo chiamarsi `src`-, figlia della directory di lavoro. Servono poi un nome -che supporremo essere `esempio`- con cui individuare il pacchetto ed un numero di versione. Senza scendere in dettagli, comunque spiegati più avanti, usiamo `1.0`. Sistemati i sorgenti, ci si dedica ai file di configurazione `Makefile.am` e `configure.ac`. Come verrà spiegato tra poco di `Makefile.am` ce ne devono essere tanti quante sono le cartelle coinvolte. In questo caso specifico 2: la directory di lavoro e quella dei sorgenti. In ogni caso si tratta di file minimi, 2 righe ciascuno. Nella cartella di lavoro viene creato il seguente file `Makefile.am` (attenzione, Unix è case sensitive):

```
AUTOMAKE_OPTIONS = foreign
SUBDIRS = src
```

La prima riga indica che il pacchetto non è conforme allo standard GNU. La seconda riga serve ad indicare la cartella dei sorgenti. In questa viene creato un altro file `Makefile.am`, composto da due righe:

```
bin_PROGRAMS = esempio
esempio_SOURCES = main.c get_ora.c
```

la prima riga indica il nome dell'eseguibile, quella successiva i sorgenti di partenza con i quali costruirlo. I nomi sono separati da uno spazio. Dal grafico riportato a pag. 4 si evince come `esempio` e `src` possano essere cambiati a piacere. Le righe rosse indicano le mutue dipendenze.

Ora si passa al ben più impegnativo e fondamentale file `configure.ac`. La costruzione a mano è improponibile, perché impervia. Si tratta di sapere cosa e come controllare. Decisamente più conveniente generare un *template* in maniera automatica tramite l'utility:

```
# autoscan
```

che spazzola la directory di lavoro e grazie anche al `Makefile.am` prende in considerazione la cartella dei sorgenti. Crea due file: `autoscan.log`, di nessun interesse, e `configure.scan`. È il template di cui sopra. È un file di testo. Questo il suo contenuto, al netto dei commenti e delle righe vuote:

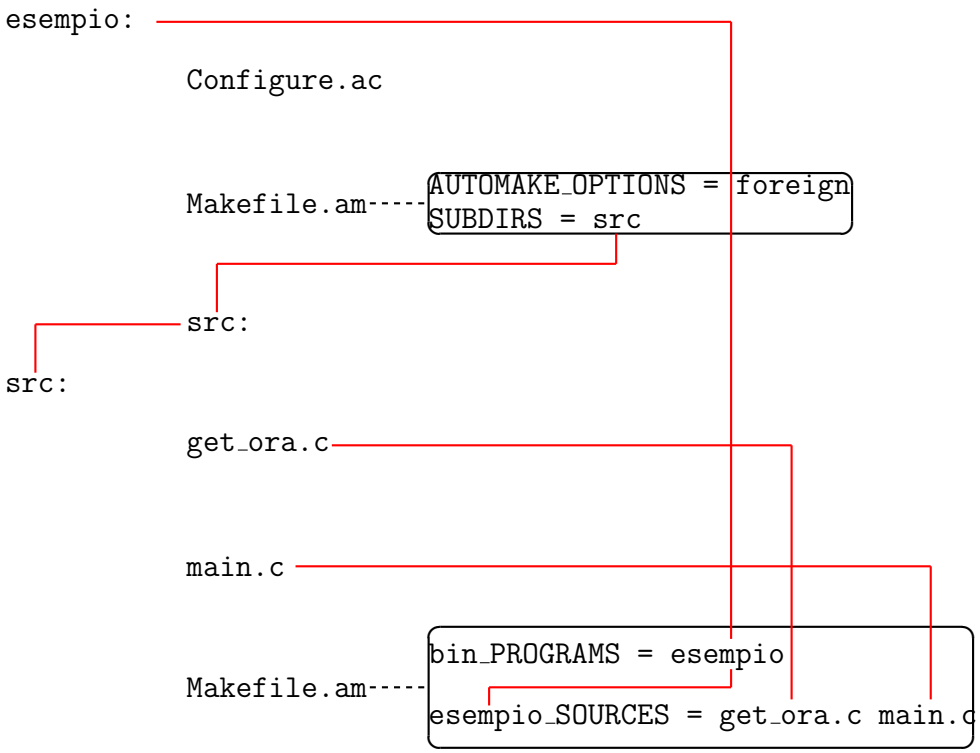

Figura 2: Grafico

```
AC_PREREQ(2.61)
AC_INIT(FULL-PACKAGE-NAME,VERSION,BUG-REPORT-ADDRESS)
AC_CONFIG_SRCDIR([esempio/get_ora.c])
AC_CONFIG_HEADER([config.h])
AC_PROG_CC
AC_HEADER_STDC
AC_CHECK_HEADERS([stdlib.h])
AC_STRUCT_TM
AC_CONFIG_FILES([Makefile
                 esempio/Makefile])
AC_OUTPUT
```

Il file risulta costituito da una serie di macro, seguite opzionalmente da dei parametri specifici. Le uniche due macro che obbligatoriamente devono esserci sono `AC_INIT` e `AC_OUTPUT` (ove `AC` sta per **a**uto **c**onfigurazione). Le altre eventuali vanno poste in mezzo. I controlli in genere riguardano le librerie e le

rispettive funzioni, gli header, le strutture, le caratteristiche del compilatore e le system services. Sempre che ce ne sia il bisogno, ovviamente.

Scendendo in dettaglio:

1. `AC_PREREQ(xxx)` è l'unica macro che può essere posta esternamente ad `AC_INIT` ed `AC_OUTPUT`. Stabilisce che la versione del tool autoconf non deve essere inferiore a quanto indicato dal parametro. Gli strumenti di autoconfigurazione sono (molto) farraginosi e con (molti) lati oscuri. Tra una versione e la successiva esistono delle discontinuità. Questa macro serve ad ottenere una certa omogeneità tra i vari tools

2. `AC_CONFIG_SRCDIR` il nome è alquanto ambiguo. Non serve a definire il nome della cartella ove ci sono i sorgenti, bensì l'esistenza di un determinato file

3. `AC_PROG_CC` serve ad individuare il nome del compilatore C. Ciò è dovuto al fatto che i sorgenti sono scritti in quel linguaggio. Se fossero stati scritti in C++, allora la macro sarebbe stata `AC_PROG_CXX`

4. `AC_HEADER_STDC` e `AC_CHECK_HEADERS` controllano la congruità degli header files. Tali macro, su installazioni recenti, sono del tutto superflue, in quanto già da tempo le versioni del compilatore C sono uniformate a quanto stabilito dallo standard. In passato non era così, da cui i controlli

5. `AC_STRUCT_TM` genera i controlli da effettuare su typedefs e strutture

6. `AC_CONFIG_FILES` indica i file da configurare. Ma per i Makefile è necessaria anche la presenza della macro `AM_INIT_AUTOMAKE`

Le macro a disposizione sono molte di più di quelle qui riportate. Come al solito, in rete si trovano esaurienti materiali con cui approfondire l'aspetto.

Il template, in quanto tale, non può essere usato così come confezionato da autoscan. Queste le modifiche da apportare:

- tre interventi nella riga che inizia con `AC_INIT`

    1. la stringa `FULL-PACKAGE_NAME` va sostituita col nome del progetto
    2. la stringa `VERSION` va sostituita col numero pattuito. Il *numero* è formato così: `major.minor` (per major, rispettivamente, minor *release*, separate dal carattere `.` [punto]). Si intuisce che collateralmente è possibile far entrare in gioco un qualche strumento per la gestione del *versioning*

3. i caratteri `,BUG-REPORT-ADDRESS` vanno eliminati, in quanto ininfluenti

- Dopo la riga `AC_INIT` bisogna inserire la macro `AM_INIT_AUTOMAKE`. Attenzione! Le prime due lettere sono proprio `AM` (per **a**uto **m**ake) e non AC.

- La riga contenente `config.h`. In questo caso specifico, ove non sono utilizzati header file dell'utente, la macro o va tolta del tutto oppure commentata (basta premettere il carattere `#`)

Nel prosieguo delle operazioni è prevista la presenza di alcuni file. Il loro contenuto, esclusivamente testuale, non interessa, ma devono essere presenti. Tanto vale crearli in precedenza:

```
# touch AUTHORS ChangeLog NEWS README
```

In ogni caso i nomi sembrano abbastanza indicativi circa l'eventuale contenuto. La fase preparatoria è finalmente conclusa. Come da roadmap, verranno eseguite in serie tre fasi. In ciascuna verrà lanciato in esecuzione un comando. Si inizia con:

```
# aclocal
```

tool propedeutico ad `autoconf`. Raccoglie in un file tutte le macro di cui ha bisogno. Poi si lancia in esecuzione il secondo di 3 comandi:

```
# autoconf
```

È di importanza fondamentale, perché è lui che in base al file di configurazione `configure.ac` provvede alla generazione dello script di autoconfigurazione `configure`. Infine si lancia in esecuzione l'ultimo dei comandi:

```
# automake --add-missing
```

Partendo dai file di configurazione `Makefile.am` vengono creati i rispettivi file `Makefile.in`. Si tratta di file propedeutici. Quando lo script `configure` verrà fatto girare, userà anche questi file per creare i relativi Makefile. Il parametro `add-missing` istruisce `automake` a provvedere gli script per la segnalazione di errori, l'installazione, etc.

È fatta.

Poiché il procedimento è stato un po' laborioso, è consigliabile fare dei controlli. Ma seguire la solita trafila (./configure; make; make install) non rende giustizia alla potenza dei tools usati. Si può creare un vero e proprio pacchetto di distribuzione. Sono sufficienti i seguenti 2 comandi:

```
# ./configure
# make dist
```

Si lancia in esecuzione lo script di configurazione in modo da ottenere la creazione dei Makefile. Il secondo comando serve a creare un file per la distribuzione del prodotto. Tutti e i soli file utili alla costruzione del pacchetto vengono riuniti in un file tramite utility tar. Il nome del file viene costruito utilizzando il nome del pacchetto (in questo caso `esempio`) e la versione (traendola da `AC_INIT`). Date le premesse: `esempio-1.0.tar.gz` (per ridurre la dimensione del file, viene fatto intervenire il compressore `gzip`).

Questo file può essere trasportato sul pc su cui si intende usare il prodotto. Lì viene decompresso e scomposto nei suoi costituenti. A questo punto è possibile finalmente procedere nel modo canonico (./configure; make; make install). Lo script `configure` gira sull'installazione su cui si intende installare il prodotto. Quindi esegue i controlli in quel particolare ambiente, indicando eventuali mancanze. La potenzialità del prodotto sta tutta qui: la predisposizione dei controlli da effettuare in modo da essere sicuri che non ci siano intoppi.

Di seguito il source dei due programmi C

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

int main (void)
{
  int           ora;
  extern int    get_ora (void);

  ora = get_ora ();
  if (ora < 4 || ora > 22)
    (void) printf ("buona notte\n");
  else
    if (ora <= 12)
      (void) printf ("buon giorno\n");
```

```
    else
      if (ora < 18)
        (void) printf ("buon pomeriggio\n");
      else
        (void) printf ("buona sera\n");
  exit (EXIT_SUCCESS);
}

#include <stdlib.h>
#include <time.h>

int get_ora (void)
{
  time_t        nowbin;
  struct tm     *nowstr;

  if (time (&nowbin) < 0)
    exit (EXIT_FAILURE);
  nowstr = localtime (&nowbin);
  return (nowstr->tm_hour);
}
```