Autotools

La soluzione GNU al problema della configurazione del software[1]

di Ivan Venuti (ivanvenuti@yahoo.it)

Il problema di compilare e installare software su macchine diverse (dove per "diverse" si intende macchine che differiscono sia in termini di harware presente che di software installato) è molto sentito, soprattutto in ambiente Linux, dove la quasi totalità dei programmi viene distribuita in formato sorgente. È necessario che la compilazione del programma tenga conto delle peculiarità del sistema su cui avviene, pena la non "portabilità" del codice. È auspicabile che le configurazioni necessarie avvengano in automatico, senza dover pretendere che l'utente che esegue l'installazione intervenga a configurare manualmente i sorgenti che, spesso, richiedono conoscenze sia tecniche che sistemistiche non banali.
Non a caso Java, per fare un esempio, deve gran parte della sua popolarità alle caratteristiche intrinseche di portabilità del codice realizzato con tale linguaggio. Per farlo però impone a priori che certi dati siano di un certo tipo (per esempio specifica la dimensione di tutti i tipi di dato primitivi). Questo porta a delle inefficienze in quanto non si sfruttano le caratteristiche peculiari della macchina in cui girerà tale applicazione. Per certi compiti questa è una limitazione troppo onerosa.
Ma anche programmando, per esempio, in C/C++ è possibile scrivere del codice realmente portabile, a patto di personalizzare l'installazione.
Quindi il problema basilare è: che strumenti esistono per automatizzare la configurazione del software?
GNU propone un insieme di tool, che sono: Autoconf, Automake e Libtool. Nel loro insieme questi tool vengono spesso indicati con il termine *"GNU Autotools"* ([1]). In questo articolo vedremo le loro caratteristiche salienti e quali possono essere i vantaggi nell'adottarli.

Il problema di fondo

Il problema principale nella configurazione del software è quello di "controllare" le risorse a disposizione e solo successivamente, in base a questo "controllo", compilare e installare il proprio progetto in modo che si "conformi" a tali risorse.
Anche senza voler scrivere del software multi-piattaforma possono sorgere dei problemi.: nel mondo Unix, per ragioni storiche, esistono una serie di "dialetti", ognuno con proprie caratteristiche. Inoltre è un sistema operativo ampiamente configurabile. Questa caratteristica, per i suoi indubbi vantaggi, è stata mantenuta in Linux; ma d'altra parte essa aggiunge una ulteriore complicazione a chi sviluppa: pacchetti software e non sa che ambiente trova al momento dell'installazione (tanto per fare un esempio librerie installate possono differire notevolmente in due installazioni distinte dello stesso sistema operativo: in alcuni sistemi possono essere presenti con certe caratteristiche, in altri con altre, in altri ancora mancare del tutto!).
Insomma, la situazione sarebbe davvero tragica per chi deve installare programmi. Per fortuna

1 Questo articolo è stato pubblicato su *Linux Journal Edizione Italiana* N. 25, Aprile 2002 edito dalla Duke Italia (http://www.duke.it); quest'ultima detiene i diritti di pubblicazione e utilizzazione economica di quest'articolo; l'autore è stato autorizzato a renderlo pubblico sul proprio sito Web personale.

esistono gli Autotool che non solo semplificano l'intero processo di installazione e configurazione, ma aiutano a scrivere del codice sorgente che sia maggiormente portabile e configurabile e, cosa assai importante, gran parte del processo di configurazione è reso automatico.

I "protagonisti" e la loro storia

In estrema sintesi, ecco le caratteristiche dei tool che prendiamo in esame ([1]):
Autoconf permette di rendere i programmi portabili, in quanto permette di automatizzare una serie di test per scoprire le caratteristiche del sistema in cui si vuole installare il programma. In particolare i test avvengono prima della compilazione del progetto, permettendo "adattare" la compilazione in base alle caratteristiche riscontrate.
Automake permette di generare dei semplificare la descrizione dei componenti software, e dell'organizzazione degli stessi, che compongono il programma.
Infine *Libtool* è un comando che si interfaccia sia con il compilatore che con il linker, e rende portabile la creazione di librerie dinamiche indipendentemente dalla piattaforma su si esegue il programma.
Già presi a se stanti questi tool risolvono problemi specifici non banali, se poi vengono usati insieme, ecco che si hanno a disposizione strumenti evoluti che permettono di creare programmi realmente portabili e configurabili.
Storicamente i vari tool sono nati in modo autonomo, ognuno per risolvere specifici problemi. Poi essi sono evoluti fino ad integrarsi perfettamente.

Maggiori dettagli

Lo script `configure` testa certe caratteristiche sul sistema (come certi tipi di dato o la presenza di certe feature) e mette a disposizione del compilatore del programma sorgente il risultato di tali test. In questo modo, per esempio, è possibile costruire un programma C/C++ con direttive `#ifdef` e far sì che in base alle caratteristiche testate vengano compilate certe parti di programma e non altre (compilazione condizionale).
Automake permette di tener traccia delle dipendenze tra i sorgenti e di generare Makefile appropriati: tali Makefile sono dei file che automatizzano la compilazione e l'installazione e vengono usati dal comando `make` (essi contengono una serie di "regole", interpretate dal comando `make`).. In parole povere, senza l'uso di "make", sarebbe necessario compilare uno a uno i sorgenti e poi procedere agli altri passi per completare l'installazione. Inoltre, supponendo di cambiare solo una parte dei sorgenti, sarebbe necessario ricompilare solo questi ultimi (e i sorgenti che dipendono da essi), senza perdere tempo a ricompilare gli altri: farlo a "mano" sarebbe impensabile: `make` automatizza anche questo tipo di compilazione. Per maggiori informazioni su `make` si veda [2].
Automake riconosce una sintassi simile a make. Le differenze principali sono che Automake permette:

- l'uso di direttive di inclusione (include);
- supporto di condizionali (utili nel caso si voglia includere o meno certe parti in base, per esempio, ai risultati di `configure`);
- supporta la cosiddetta "macro assignment".

Il processo di configurazione/installazione con gli Autools

Volendo usate tutte le funzionalità degli Autotools (non è detto che ciò accade sempre: spesso li si può usare solo in parte o solo per risolvere specifici problemi) ecco il processo da seguire (schematizzato in **Figura 1**):

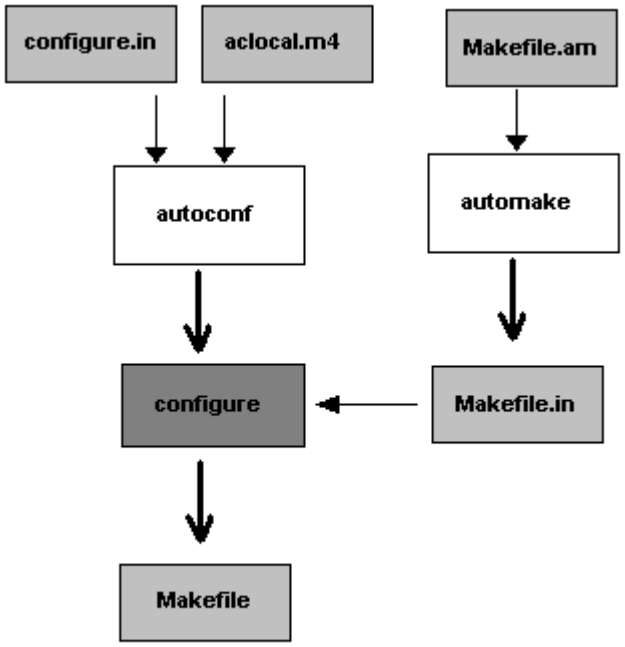

Figura 1: Il processo di generazione di un Makefile attraverso l'uso degli autotools.

Lo sviluppatore scrive un file chiamato `Makefile.am`: esso ha una sintassi più semplice rispetto ai Makefile "normali", ma come essi vi si descrive la "struttura" del programma.
Il file Makefile.am viene letto dal tool Automake e il risultato è un file che si chiama `Makefile.in`, e rispetta le norme definite dai "*GNU Makefile Standards*" (si veda [3]).
Questo file dev'essere trasformato nel Makefile "finale" (chiamato semplicemente `Makefile`).
Per farlo si lo script `configure` che viene generato dal tool Autoconf, a partire da un file chiamato `configure.in`. In particolare `configure` esegue i test sulle caratteristiche del sistema, determinandole in automatico. Se qualcosa non è in grado di determinare, allora sarà necessario editare gli header file che ha generato (intervento fatto dallo sviluppatore).
Il file configure.in contiene chiamate a Macro e e frammenti di codice (specifico per la shell di sistema, normalmente la Bourne Shell). Se le Macro non sono macro standard, è necessario ricorrere al tool aclocal (che genererà il file `aclocal.m4`).
Esistono poi altre utilità per semplificare la scrittura dei vari file iniziali (come sempre si fa riferimento a [1] per i dettagli).
Per quanto riguarda Libtool, esso nasce per permettere la generazione di librerie dinamiche.
Le librerei statiche sono delle librerie di funzionalità che vengono incluse nei programmi (pertanto vengono "copiate", o meglio "accodate" ad ogni programma che ne fa uso).
Questa inclusione ha degli svantaggi: in primo luogo occupa maggior spazio, sia su disco fisso, sia in memoria. In secondo luogo se c'è la necessità di aggiornare tali librerie (per esempio per mettervi

una versione con le stesse caratteristiche ma più efficiente) è necessario ricompilare tutti i programmi che la includono.
Le librerie dinamiche invece, non vengono incluse nei programmi che ne fanno uso, ma in essi viene messo un “riferimento” ad esse. Esse vengono caricate in memoria quando un programma ne fa uso. Se esistono più programmi che usano la stessa libreria è necessario caricarla in memoria una volta soltanto, rendendola condivisa ai programmi.
Si capisce bene che l’uso di librerie dinamiche è preferibile per la maggior parte dei casi. È verosimile che anche i programmi da installare vogliano a loro volta installare e far uso di librerie dinamiche. La complessità (dovuta alla diversità tra i vari sistemi operativi) viene ridotta facendo uso di Libtool.

A chi sono utili

Chi dovrebbe/potrebbe avvantaggiarsi di questi tool? Gli utenti, in primo luogo: nella maggior parte dei pacchetti che usano gli Autotool per configurare la loro compilazione, i passi necessari per completare l’installazione sono davvero minimi (e ampiamente spiegati nella documentazione che li accompagna), e possono risolversi nei seguenti (e soli) comandi:

```
$ configure
$ make
$ make install
```

Ma i vantaggi sono evidenti anche per chi produce/realizza o distribuisce il software. Possiamo identificare diverse figure professionali che si possono avvantaggiare dall’uso degli Autotool, anche se ogni categoria li userà con fini diversi e con livelli di conoscenza differenti:
Gli utenti: nella maggior parte dei pacchetti che usano gli Autotool per configurare la loro compilazione, i passi necessari per completare l’installazione sono davvero minimi (e ampiamente spiegati nella documentazione che li accompagna).
I capi progetto dovrebbero preoccuparsi da subito della problematica di rendere portabile il codice del progetto che stanno sviluppando: normalmente è molto costoso (sia in termini di tempi che di soldi) accorgersi in fase avanzata del progetto che questa è una caratteristica necessaria. Formare dei gruppi di lavoro competenti con questi tool, o indirizzarli ad un loro approfondimento fin dall’inizio può essere molto vantaggioso.
I programmatori: sapere che esistono questi tool e utilizzarli da subito (soprattutto in progetti complessi) aiuta a strutturare meglio il codice (per esempio costruire i Makefile solo quando si è già scritto un bel po’ di file sorgente, può portare a dimenticarsi delle dipendenze) e semplifica la fase di installazione/configurazione finale (si pensi ad accorgersi che il proprio programma non funziona perché un tipo di dato usato non è del formato di come si presumeva lo fosse: è necessario rivedere tutto il progetto dove lo si usa, e rifare il test delle parti modificate; se invece si prevede fin dall’inizio le parti di programma che cambiano in base al formato di quel dato, si presume che l’intero progetto sia più stabile e meno soggetto ad errori).
Infine, anche i sistemisti sono aiutati nella conoscenza di questi tool. È tipico (anzi: consigliato!) che i programmi GNU usino questi tool. Una loro conoscenza migliora la conoscenza delle fasi di installazione e configurazione in atto nel momento che si installano questi programmi.

Conclusioni

Per forza di cosa l'articolo non è sceso troppo nei dettagli "operativi" dei tool presentati, e allo stesso tempo è molto "denso"di contenuti.
Vi consiglio di approfondire gli argomenti trattati con il libro [1] (anche nella versione consultabile online).
Un ottimo tutorial ("*Learning the GNU development tools*") può essere trovato all'indirizzo http://www.st-andrews.ac.uk/~iam/docs/tutorial.html, oppure si veda "*Developing software with GNU*", presente alla pagina http://www.codelearn.com/cpp/gnutools/toolsmanual.html.
Alcuni riferimenti per progetti che hanno fatto uso dei tool descritti: all'indirizzo http://grid-data-management.web.cern.ch/grid-data-management/docs/gdmp/GDMP_Autotool.pdf è scaricabile il documento "*Using GNU Autotools for the GDMP package*" (formato PDF), mentre alla pagina http://www.cs.columbia.edu/~ezk/research/autotools/index.html
trovate "*Overhauling Amd for the '00s: A Case Study of GNU Autotools* " di Erez Zadok;
quest'ultimo documento illustra l'analisi di un caso di conversione di un progetto esistente, il cui sviluppo aveva portato a tener presente il problema della compatibilità. Si vede come, convertendo il progetto per fargli usare gli Autotool, ha portato ad una diminuzione del 33% del suo codice, nonché al miglioramento della sua portabilità.
Purtroppo, allo stato attuale, per padroneggiare gli Autotool sono necessarie notevoli conoscenze tecniche e, cosa ancor più problematica, è necessario imparare un insieme disomogeneo di linguaggi (essi sono linguaggi di programmazione di shell, formalismi per esprimere Macro etc.). Questo fatto è dovuto principalmente alla mancanza di un progetto di fondo uniformante. Si spera che questa limitazione possa essere presto risolta, anche se essa richiede una revisione completa del loro funzionamento. Inoltre rimangono dei problemi nell'uso di questi tool al di fuori dei sistemi operativi Unix-like (per una serie di osservazioni al riguardo consiglio di leggere i thread *"*Auto-tools & Win32 & Borland C++ Builder*"* sulla lista di discussiona di automake, reperibile alla pagina http://sources.redhat.com/ml/automake/2001-05/msg00359.html).
Come osservato poc'anzi, imparare ad usare questi tool, correttamente e in modo ottimale,non è certo banale: come tutte le cose richiede tempo e migliora con l'uso e l'esperienza.
Se si è disposti a sacrificare un po' di risorse per la loro comprensione, sarà tutto a vantaggio di progetti maggiormente portabili e uno skill che di sicuro ripagherà nel medio-lungo termine.

Bibliografia

[1] "*GNU Autoconf, Automake And Libtool*", AAVV, New Riders 2000, (http://sources.redhat.com/autobook/)
[2] "*GNU Make*", http://www.gnu.org/manual/make/
[3] "*GNU Coding Standards*", http://www.gnu.org/prep/standards_toc.html