DICo

Edit-Make-Debug: lo sviluppo di programmi ai tempi di UNIX[1]

Mattia Monga

Dip. di Informatica e Comunicazione
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

Milano – 24 novembre 2006

Lo sviluppo ai tempi di UNIX

Il ciclo classico che ha accompagnato generazioni di programmatori:

1. Edit
2. Make
3. Debug

Lo sviluppo ai tempi di UNIX

Il ciclo classico che ha accompagnato generazioni di programmatori:

1. Edit
   - `vi`
   - `emacs`
2. Make
   - `autoconf`
   - `make`
   - `cc`
3. Debug
   - `gdb`
   - `gcov`

Lo sviluppo ai tempi di UNIX

DICo

Il ciclo classico che ha accompagnato generazioni di programmatori:

1. Edit
   - `vi`
   - `emacs`
2. Make
   - `autoconf`
   - `make`
   - `cc`
3. Debug
   - `gdb`
   - `gcov`
4. (Evolve)
   - `diff, patch`
   - `rcs`

DICo

Disclaimer

- Chi parla è un fedele della *Church of Emacs*
- Proveniente da precedenti esperienze di concubinato con `vi` (peccato in cui ogni tanto torna ad indulgere)
- Il seguente seminario
  - **non** è un tentativo di convertire alcuno all'*emacsesimo*
  - vorrebbe presentare le buone idee presenti in entrambi, in maniera per quanto possibile equilibrata.
  - fornire una breve prospettiva storica
  - Si parlerà di
    - `vi` nella versione più vicina all'originale (`nvi` e non `vim`)
    - `emacs` nella versione GNU (e non `xemacs`)

vi

- Bill Joy (co-fondatore della SUN), 1976, per BSD UNIX
- dal comando `visual` dell'editor di linea `ex`
- nella Church of Emacs: *vicious interface*
- `vi vi vi` $\mapsto$ 666

Le buone idee di `vi`

DICo

- *Modal editor*
  - modo input
  - modo comandi
- I comandi di movimento e modifica sono sostanzialmente ortogonali *ortogonali*
- small and fast
- fa parte dello standard POSIX

vi in una slide

- Movimento:
  - `h,j,k,l` (o frecce)
  - `0`, beginning of line, `$`, end of line
  - `w`, beginning of word, `e`, end of word
  - `(num)G`, goto line num, `/`,
  - `(,)`, sentence
- Modifica:
  - `i,a` insert before/after
  - `o,O` add a line
  - `d,c,r` delete, change, replace
  - `y,p` “to yank” and paste
  - `u` undo `.` redo
  - `s/reg/rep/[g]` search and replace

emacs

DICo

- Richard Stallman, Guy Steele (fondatore della FSF), 1975,
- macro per TECO (line editor)
- GNU Emacs, 1984
- plurale *emacsen*
- Emacs Makes A Computer Slow

Example

TECO session

```
*EBhello.c$$
*P$$
*SHello$0TT$$
    printf("Hello world!\n");
*-5DIGoodbye$0TT$$
    printf("Goodbye world!\n");
*EX$$
```

Le buone idee di `emacs`

- Il modo comandi è ottenuto con tasti meta (o con il metodo dei *prefix*)
- Interprete Lisp
- I comandi possono essere dati in tre modi:
  1. Tasto
  2. Menu
  3. Nome del comando
- Frame, Window, Buffer, Minibuffer, Point-Mark, Region
- Major e minor mode per ogni buffer, che ridefiniscono tasti e comandi
- Può essere usato in `vi` mode!

emacs in una slide

DICo

Command	Keystroke	Description
forward-word	M-f	Move forward past one word.
search-word	C-s	Search a word in the buffer.
undo	C-/	Undo last change, and prior changes if pressed repeatedly.
keyboard-quit	C-g	Abort the current command.
fill-paragraph	M-q	Wrap text in ("fill") a paragraph.
find-file	C-x C-f	Visit a file (you specify the name) in its own editor buffer.
save-buffer	C-x C-s	Save the current editor buffer in its visited file.
save-with-newname	C-x C-w	Save the current editor buffer as a file with the name you sp
save-buffers-kill-emacs	C-x C-c	Offer to save changes, then exit Emacs.
set-marker	C-[space]/C-@	Set a marker from where you want to cut or copy.
cut	C-w	Cut all text between the marker and the cursor.
copy	M-w	Copy all text between the marker and the cursor.
paste	C-y	Paste text from the emacs clipboard
kill buffer	C-x k	Kill the current buffer

Emacs' circus

- Interprete Lisp
- Eliza (a Rogerian psychotherapist) by Weizenbaum
- Kill ring, Macro, Regexp search and replace
- Dynamic abbrev
- Diff, merge mode
- Compressed, encrypted files
- Programming modes, IDE

Digressione: TAGS

- Una *tag table* è un indice che elenca come le definizioni di entità di un programma complesso sono disperse fra tanti file
- `etags` (Emacs) e `ctags` (Vi)
- `M-x visit-tags-table`
- `M-x find-tag`

Problema

Il programmatore non scrive mai *tutto* un programma, nel senso di tutte le istruzioni necessarie a controllare la macchina per un determinato scopo. Ciò che scrive deve *adattarsi* a:

- chiamate di sistema
- librerie
- "layout" delle risorse di sistema
- ...

Come garantire la portabilità in ambienti diversi?

Variability management

Il problema di portare uno stesso programma in contesti differenti è detto *Variability management* (VM) e in alcuni casi cambiano anche le feature dell'applicazioni (*product families*)
Open source e VM sono fortemente legati: *from my itch to everyone's itch*. Anche le distro sono soprattutto uno sforzo di VM.
Il progetto GNU ha sviluppato fin dagli anni '90 strumenti di VM a *building time:* nome collettivo autotools

In principio era il caos...

DICo

A physicist, an engineer, and a computer scientist were discussing the nature of God. Surely a Physicist, said the physicist, because early in the Creation, God made Light; and you know, Maxwell's equations, the dual nature of electromagnetic waves, the relativistic consequences... An Engineer!, said the engineer, because before making Light, God split the Chaos into Land and Water; it takes a hell of an engineer to handle that big amount of mud, and orderly separation of solids from liquids... The computer scientist shouted: And the Chaos, where do you think it was coming from, hmm?

Autotools

- Autoconf produce un programma (`configure`) in grado di fare test per capire di cosa dispone l'ambiente target
- Automake produce il `makefile` adatto a costruire il programma in un determinato ambiente target
- Libtool semplifica la gestione di librerie dinamiche

Storia

Anni '90, *source code portability*, per gestire le (centinaia) di varianti di UNIX esistenti:

Metaconfig by Larry Wall, Harlan Stenn, and Raphael Manfredi.

Cygnus 'configure' script , by K. Richard Pixley, praticamente identico all'originale GCC 'configure' script, by Richard Stallman.

GNU Autoconf by David MacKenzie.

Imake part of the X Window system.

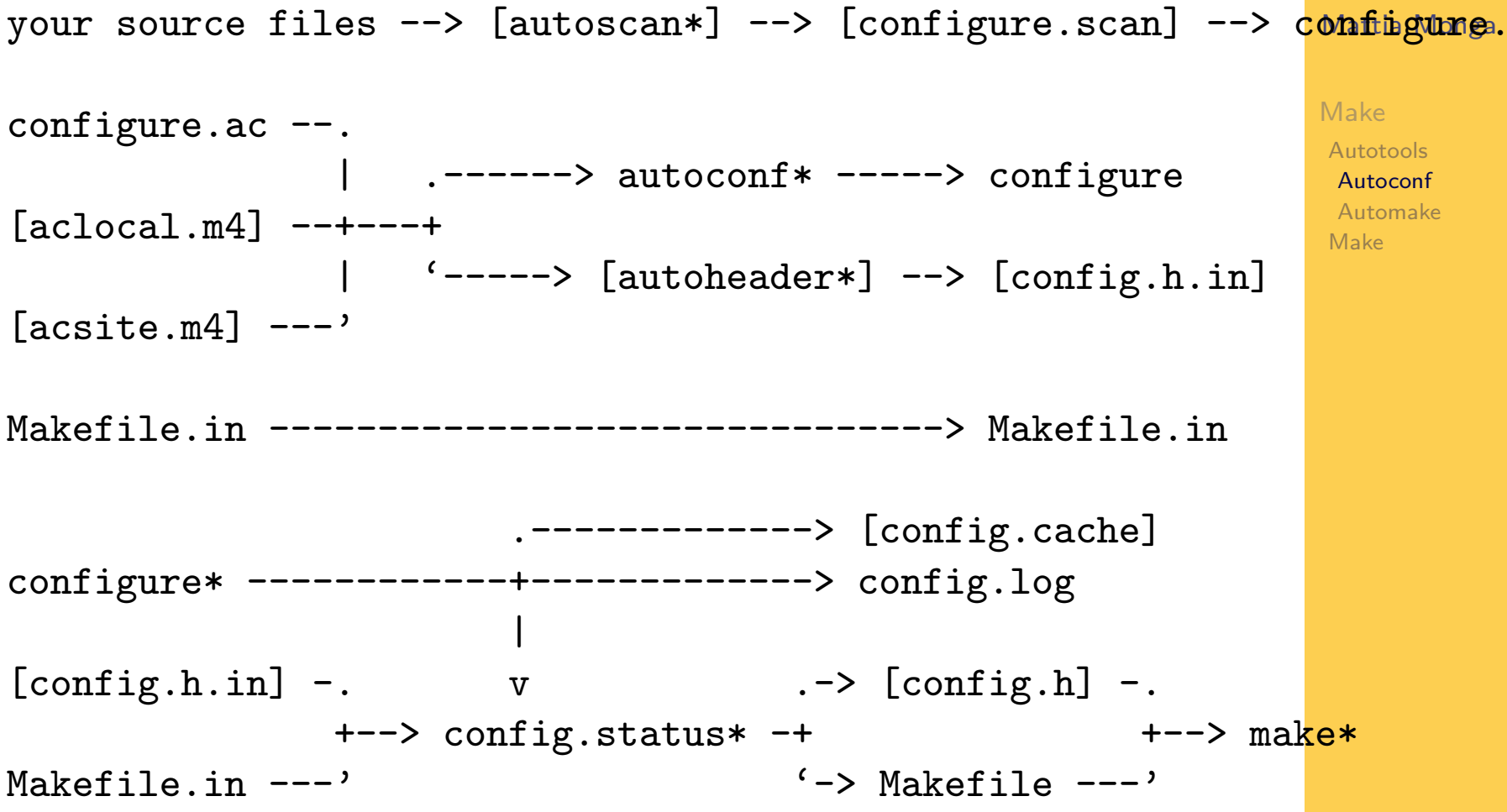
Molte delle feature sono attualmente confluite in Autoconf.

Obiettivo

L'obiettivo di Autoconf è produrre uno script `configure` che faccia i test utili a identificare le peculiarità dell'ambiente.

- Per i programmi C, tipicamente viene prodotto un `config.h` contenente una serie di `#define` e inclusioni condizionali e il `Makefile`
- La configurazione può essere guidata (layout, ecc.)
- Al termine il programma può essere compilato e installato (tipicamente: `make all && make install`)

Overview


your source files --> [autoscan*] --> [configure.scan] --> configure.
configure.ac --.
| .------> autoconf* -----> configure
[aclocal.m4] --+---+
| '-----> [autoheader*] --> [config.h.in]
[acsite.m4] ---'
Makefile.in ------------------------------> Makefile.in
.-------------> [config.cache]
configure* ------------+-------------> config.log
|
[config.h.in] -. v .-> [config.h] -.
+--> config.status* -+ +--> make*
Makefile.in ---' '-> Makefile ---'

configure.ac


```
dnl Process this file with autoconf to produce a configure script.
AC_INIT(main.c)
AM_INIT_AUTOMAKE(foonly, 1.0)
AC_PROG_CC
AM_PROG_LEX
AC_PROG_YACC
AC_OUTPUT(Makefile)
```

Una serie di macro M4

Macro predefinite

Ci sono già moltissime macro predefinite per le evenienze piú comuni:

- Programs. AC_CHECK_PROG AC_PATH_TOOL
- Libraries. AC_CHECK_LIB AC_SEARCH_LIBS
- Headers. AC_CHECK_HEADERS([sys/socket.h])
- Typedefs and structures. AC_CHECK_MEMBERS([**struct** stat.st_blksize])
- Functions. AC_CHECK_FUNCS(inet_aton inet_addr, **break**)
- System services. AC_SYS_LONG_FILE_NAMES
- Output. AC_OUTPUT
- Options.

Le macro a volte producono un programma “oracolo” che viene eseguito durante la configurazione.

M4

DICo

Ogni tanto bisogna scriversele, però...

AC_DEFUN(MACRO−NAME, MACRO−BODY)

```
AC_CACHE_CHECK([for EMX OS/2 environment],
                [ac_cv_emxos2],
                [AC_COMPILE_IFELSE([AC_LANG_PROGRAM([],
                [return __EMX__;])],
                [ac_cv_emxos2=yes],
                [ac_cv_emxos2=no])])
```

Automake

`configure` genera il `Makefile` a partire da un `Makefile.in`.
Automake serve a generare `Makefile.in` partendo da un `Makefile.am`
Vantaggi:

- Vengono creati automaticamente i target standard
- Le dipendenze fra target possono a loro volta dipendere dall'ambiente di building
- Automake processa `configure.ac` e riutilizza e incorpora le variabili definite da configure

Makefile.am

DICo

Sostanzialmente si definiscono solo le parti variabili. Le dipendenze standard vengono generate

bin_PROGRAMS = tgif

tgif_SOURCES = main.c

Make

DICo

Stuart Feldman, 1977 at Bell Labs.
Permette di specificare dipendenze fra processi di generazione.
Dipendenze: se cambia questo file, allora il processo di generazione deve essere ripetuto.

```
helloworld: helloworld.o
        gcc -o $@ $<

helloworld.o: helloworld.c
        gcc -c -o $@ $<

.PHONY: clean
clean:
        rm helloworld.o helloworld
```

Fare le pulci al programma

1. Riconoscere l'esistenza di un malfunzionamento (bug)
2. Isolare il difetto
3. Identificare la catena causale
4. Ideare una soluzione
5. Convalidare la soluzione

GDB

- Scritto per il progetto GNU, nel 1986, da R. Stallman
- Debugger simbolico
- Modellato su `dbx` di BSD Unix, by Mark Linton
- Ora alla versione 6.0
- Come il GCC, gestisce decine di architetture diverse

Concetti fondamentali

Breakpoint

Un punto del programma in cui l'esecuzione deve essere bloccata, tipicamente per esaminare lo stato in quell'istante.

Stepping

Eseguire il programma *passo a passo*. La granularità del passo può arrivare fino all'istruzione macchina.

Stato

Lo stato del programma può essere analizzato come:

- forma simbolica: secondo i simboli definiti nel linguaggio di alto livello e conservati come *simboli di debugging*
- memoria virtuale: stream di byte suddiviso in segmenti
  - Text: contiene le istruzioni (spesso read only)
  - Initialized Data Segment: variabili globali inizializzate
  - Uninitialized Data Segment (bss): variabili globali non inizializzate
  - Stack: collezione di *stack frame* per le chiamate di procedura. Cresce verso il basso.
  - Heap: Strutture dati create dinamicamente. Cresce verso l'alto.

Uso del debugger

- `break ...`
- `run ...`
- `p(rint) ...`
- `n(ext)`
- `s(tep)`
- `backtrace`
- `set ...`

gcov

Un tool per misurare quante volte viene eseguita ogni istruzione (o branch) durante l'attività di test.

- Il programma deve essere opportunamente instrumentato e viene salvato il grafo di flusso
- Ogni esecuzione produce dei dati `.gcda`
- Che poi vengono esaminati da `gcov`