Portabilità e distribuzione del codice GNU/Linux

Lezione del 16 Ottobre 2009

GNU Build System

- Obiettivi
  - Portabilità e distribuzione del codice
- Use-cases
  - Punto di vista dell'*utente*
  - Processo (semplificato) di configurazione
- Autotools

Fonti di (non)portabilità

- Consideriamo funzioni in C che possono avere...
  - Nomi differenti su architetture differenti (e.g. `strchr()` vs. `index()`)
  - Prototipi differenti (e.g. `int setpgrp(void)` vs. `int setpgrp(pid_t,pid_t)`)
  - Comportatmenti differenti (cosa succede con `malloc(0)`?)
  - Definizioni diverse/multiple in header differenti (e.g. `string.h` vs. `strings.h`)

Soluzione

- Compilazione condizionale `#if` / `#elif` / `#else`
  - Manuale, complessa da mantenere e revisionare
- Definizione di macro per il preprocessore
  - Preferibile
  - Definizione in un'unica posizione

Esempio: compilazione condizionale

```
#if !defined(CODE_EXECUTABLE)
  static long pagesize = 0;
#if defined(EXECUTABLE_VIA_MMAP_DEVZERO)
  static int zero_fd;
#endif
  if (!pagesize) {
#if defined(HAVE_MACH_VM)
      pagesize = vm_page_size;
#else
      pagesize = getpagesize();
#endif
#if defined(EXECUTABLE_VIA_MMAP_DEVZERO)
      zero_fd = open("/dev/zero", O_RDONLY,0644);
      if (zero_fd < 0) {
        fprintf(stderr, "trampoline: Cannot open /dev/zero!\n");
        abort();
      }
#endif
  }
#endif
```

Esempio: macro di sostituzione

```
#if ! HAVE_FSEEKO && ! defined fseeko
# define fseeko(s, o, w) ((o) == (long) (o) \
                          ? fseek (s, o, w) \
                          : (errno = EOVERFLOW, -1))
#endif
```

- `fseeko()` richiamata ove necessario

Distribuzione

- Per poter distribuire il codice su architetture differenti:
  - Collezione di `#define` per ogni sistema che consideriamo
  - Informare il flusso di compilazione sui path da seguire per trovare le dipendenze necessarie (`-I`, `-D`, `-L`...)
- Manuale
  - Non gestibile
- Automatico
  - Richiede l'uso di tool specifici (*GNU Autotools*)

configure

- Il suo scopo è automatizzare il processo di *configurazione* del codice sull'architettura di riferimento (*target architecture*)
  - Avviene prima dell'effettiva compilazione
  - Permette di gestire i differenti parametri dell'architettura disponibile
  - Esamina il sistema per leggere funzioni, librerie, tool necessari per la compilazione/esecuzione dell'applicativo
- È entrato a far parte di un *qualsiasi* progetto GNU/Linux

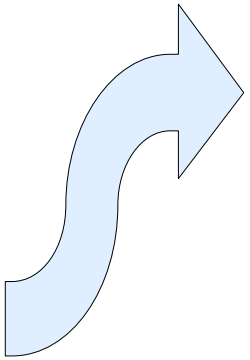
What's next?

- Obiettivi
  - Portabilità e distribuzione del codice
- Use-cases
  - Punto di vista dell'utente
  - Processo (semplificato) di configurazione
- Autotools

Procedura standard di installazione

```
~% tar -zxf amhello-1.0.tar.gz

~% cd amhello-1.0

~/amhello-1.0/ mkdir build && cd build

~/amhello-1.0/build ../configure

~/amhello-1.0/build make

~/amhello-1.0/build ./src/amhello-1.0
```

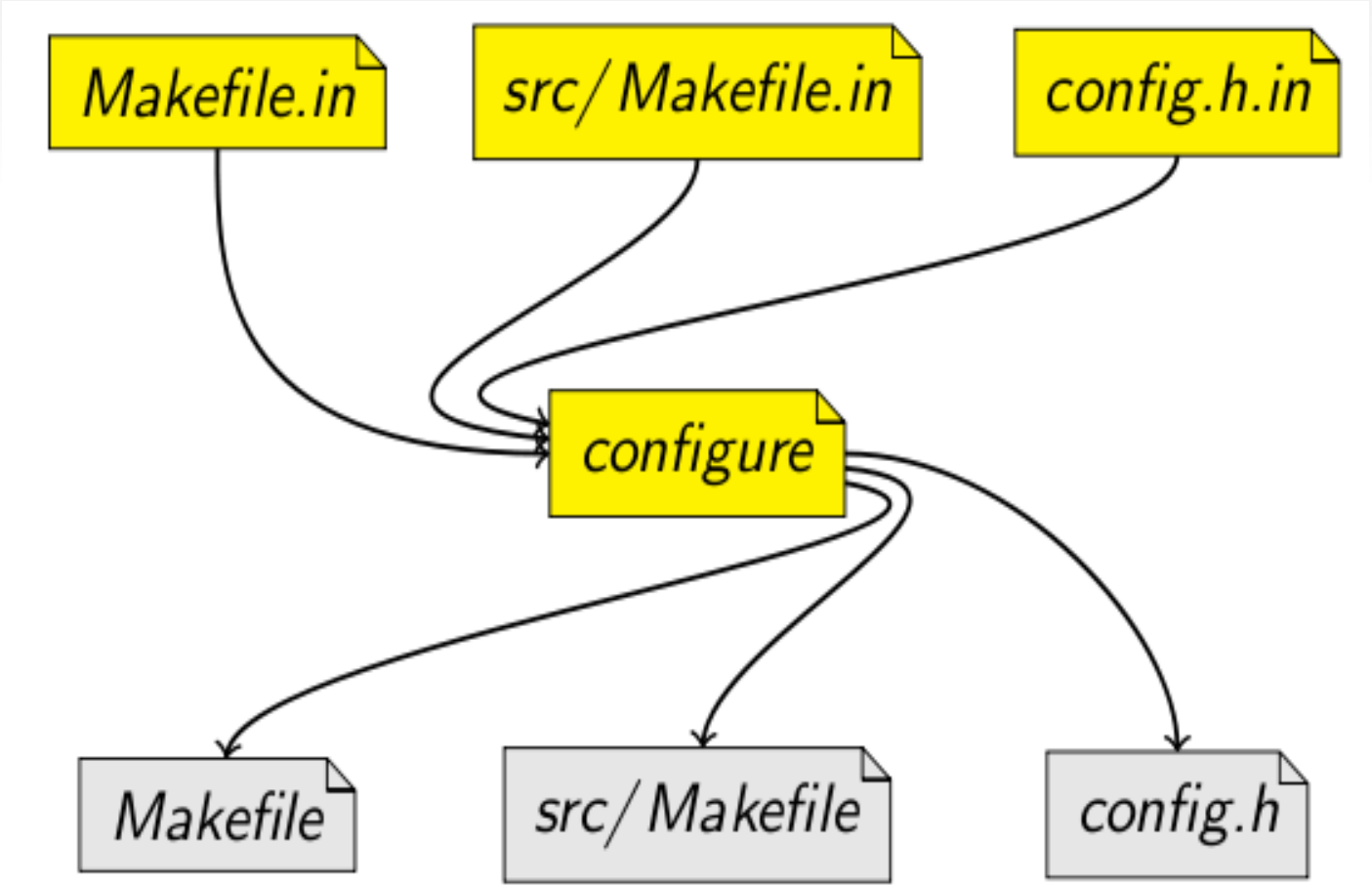
```
amhello-1.0
|-- Makefile.am
|-- Makefile.in
|-- aclocal.m4
|-- config.h.in
|-- configure
|-- configure.ac
|-- depcomp
|-- install-sh
|-- missing
|-- readme
`-- src
    |-- Makefile.am
    |-- Makefile.in
    `-- main.c
```



```
amhello-1.0
|-- Makefile.am
|-- Makefile.in
|-- aclocal.m4
|-- build
|   |-- Makefile
|   |-- config.h
|   |-- config.log
|   |-- config.status
|   |-- readme
|   |-- src
|   |   |-- Makefile
|   |   |-- hello
|   |   `-- main.o
|   `-- stamp-h1
|-- config.h.in
|-- configure
|-- configure.ac
|-- depcomp
|-- install-sh
|-- missing
|-- readme
`-- src
    |-- Makefile.am
    |-- Makefile.in
    `-- main.c
```

Processo di configurazione

- `*.in` file sono utilizzati dallo script per generare i file di configurazione per la compilazione



Supporto

- La configurazione manuale (i.e. richiamare i comandi necessari) può essere complessa
  - Tante caratteristiche da considerare
  - Ricerca esaustiva dei tool, funzioni, header files...
  - Possibili modifiche alle specifiche **GNU Coding Standards** (http://www.gnu.org/prep/standards)
- **GNU Autotools**
  - Semplici istruzioni da seguire per creare un'applicazione GNU-compliant
  - Tiene traccia delle modifiche allo standard

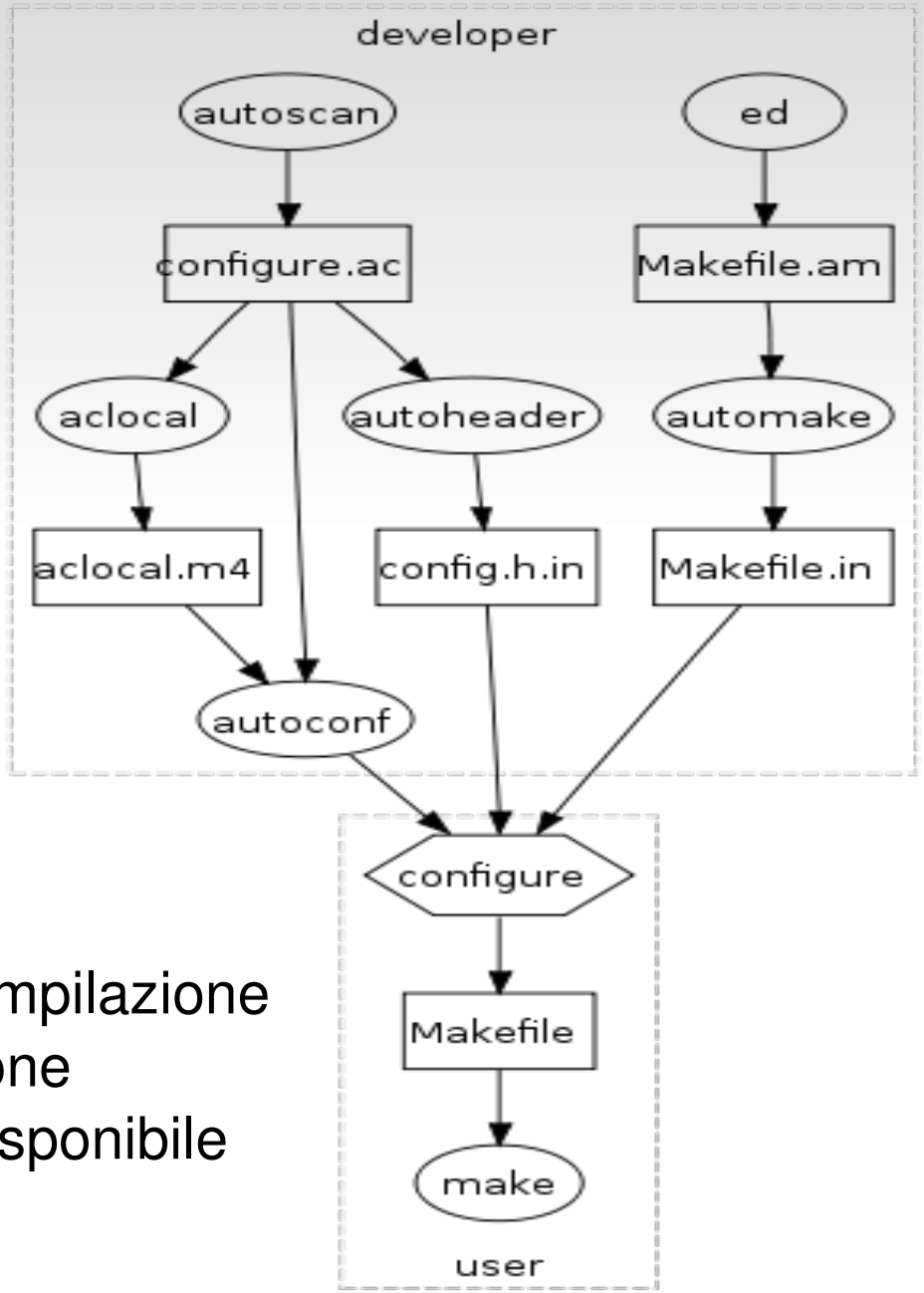
What's next?

- Obiettivi
  - Portabilità e distribuzione del codice
- Use-cases
  - Punto di vista dell'utente
  - Processo (semplificato) di configurazione
- Autotools

Autotools

- Collezione di tool di sviluppo
  - Assistono lo sviluppatore nella creazione di codice portabile (*package*)
- **GNU Autoconf**
  - Genera lo script di configurazione `configure`
- **GNU Automake**
  - Genera `Makefiles` portabili
- **GNU Libtool**
  - Per la creazione di librerie statiche e dinamiche

Diagramma di flusso



Sviluppatore

Configurazione, compilazione
e installazione
sull'architettura disponibile

Autotools by example

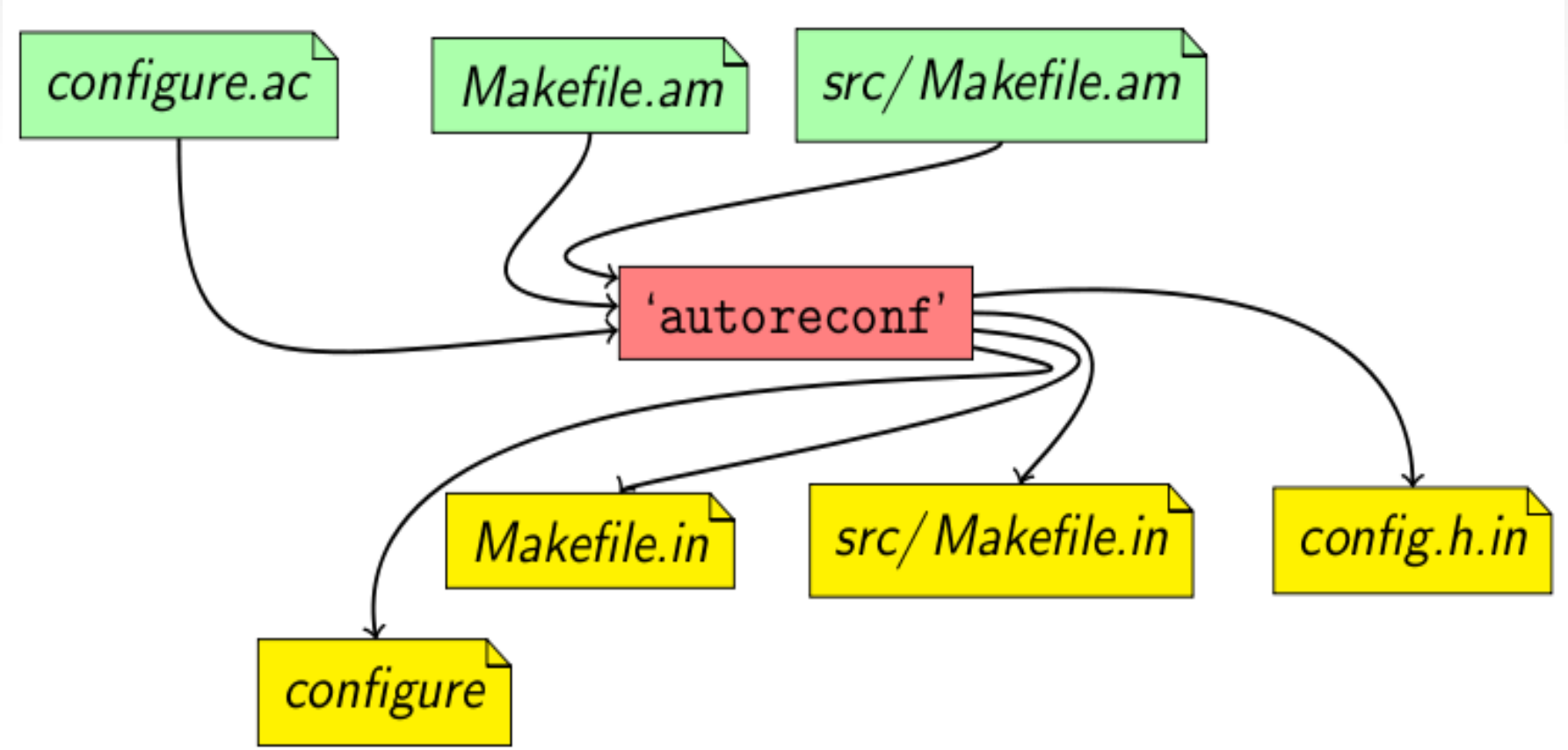
- Consideriamo un semplice "*Hello World!*"

src/main.c

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

int
main (void)
{
  puts ("Hello World!");
  puts ("This is " PACKAGE_STRING ".");
  return 0;
}
```

Primo passo

- Generazione dello script di configurazione e dei `Makefiles`



Input

- Sono necessari tre file
  - `configure.ac`
  - `Makefile.am`
  - `src/Makefile.am`

```
from_scratch/
|-- Makefile.am
|-- configure.ac
`-- src
    |-- Makefile.am
    `-- main.c
```

- Operazioni da eseguire

```
~% autoreconf --install
```

Output

- File di configurazione
  - `Makefile.in`
  - `config.h.in`
  - `configure*`
  - `src/Makefile.in`

File ausiliari
- `aclocal.m4`
- `depcomp*`
- `install-sh*`
- `missing*`

Autotools cache
- `autom4te.cache/*`

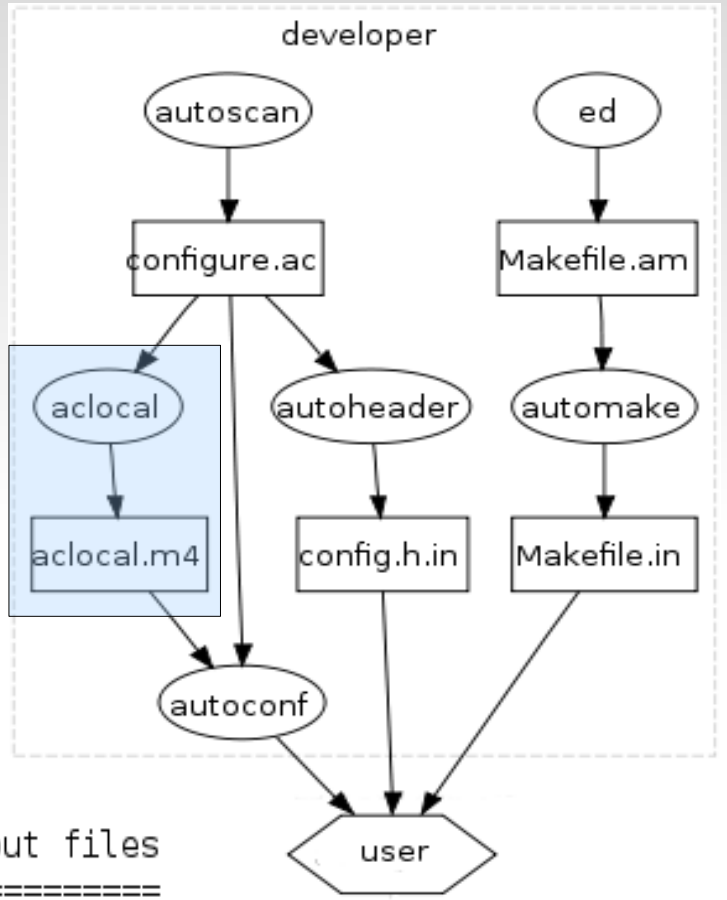
```
from_scratch/
|-- Makefile.am
|-- Makefile.in
|-- aclocal.m4
|-- autom4te.cache
|   |-- output.0
|   |-- output.1
|   |-- requests
|   |-- traces.0
|   `-- traces.1
|-- config.h.in
|-- configure
|-- configure.ac
|-- depcomp
|-- install-sh
|-- missing
`-- src
    |-- Makefile.am
    |-- Makefile.in
    `-- main.c
```

Autoreconf

- `Autoreconf` è la semplificazione dell'intero processo
  - `aclocal`
  - `autoheader`
  - `automake`
  - `autoconf`
- Esegue automaticamente e nell'ordine appropriato tutte le operazioni

aclocal

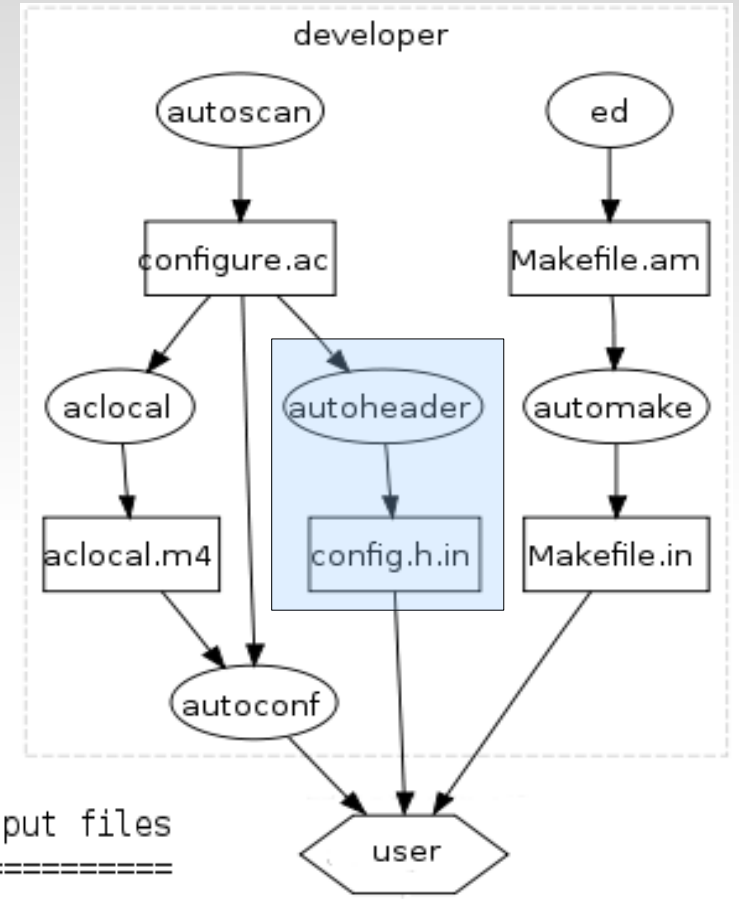
- Combina in una tutte le macro pre-definite e quelle definite dall'utente




```
user input files   optional input     process          output files
===============    ==============     =======          ============

                    acinclude.m4 - - - - -.
                                          V
                                       .-------,
configure.in ------------------------->|aclocal|
                  {user macro files} ->|       |------> aclocal.m4
                                       `-------'
```

autoheader

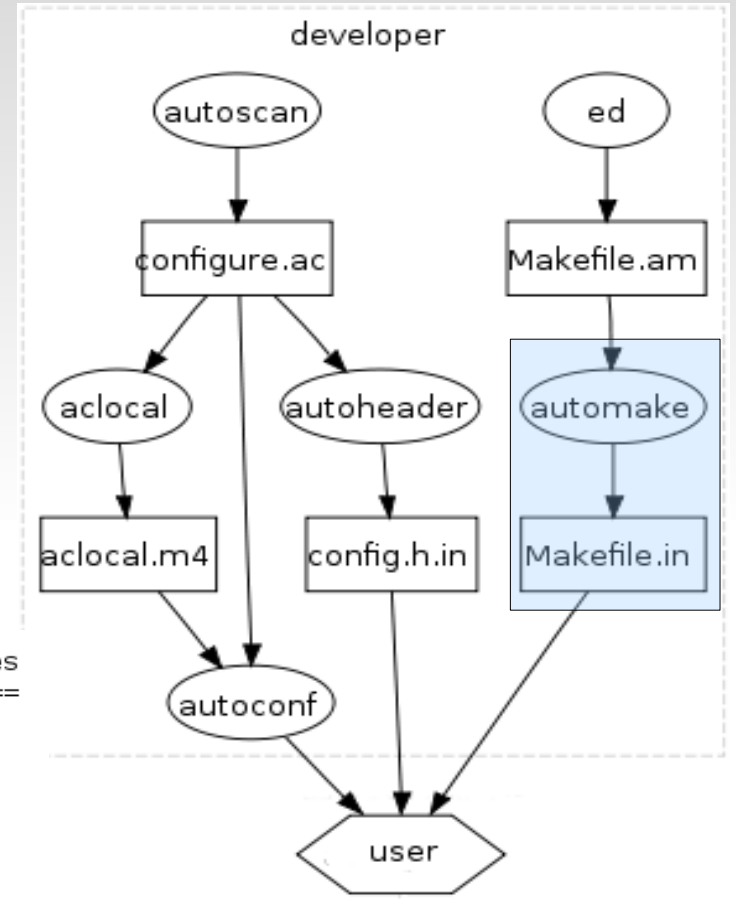
- Prepara il file `config.h.in` contenente definizioni necessari al processo di configurazione sull'architettura target




```
user input files    optional input     process          output files
===============     ==============     =======          ============

                    aclocal.m4 - - - - - - - .
                    (acconfig.h) - - - -.    |
                                        V    V
                                      .----------,
configure.in ----------------------->|autoheader|----> config.h.in
                                      `----------'
```

automake

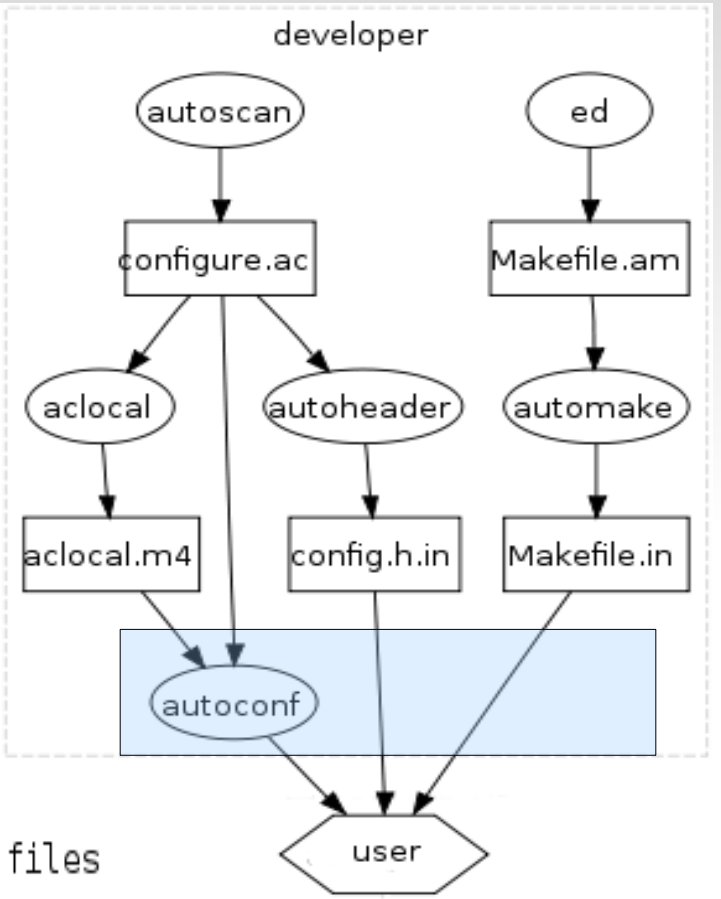
- Prepara i `Makefile.in` per ogni `Makefile.am` presente nel progetto, e alcuni file ausiliari per il rilascio del pacchetto




```
user input files   optional input   processes          output files
================   ==============   =========          ============

                                      .--------,
                                      |        | - - -> COPYING
                                      |        | - - -> INSTALL
                                      |        |------> install-sh
                                      |        |------> missing
                                      |automake|------> mkinstalldirs
configure.in ----------------------->|        |
Makefile.am  ----------------------->|        |------> Makefile.in
                                      |        |------> stamp-h.in
                                  .---+        | - - -> config.guess
                                  |   |        | - - -> config.sub
                                  |   `------+-'
                                  |          | - - - -> config.guess
                                  |libtoolize| - - - -> config.sub
                                  |          |--------> ltmain.sh
                                  |          |--------> ltconfig
                                  `----------'
```

autoconf

- Genera lo script di configurazione `configure.ac`




```
user input files   optional input   processes          output files
===============   ==============   =========          ============

                   aclocal.m4 - - - - - -.
                                         V
                                    .--------,
configure.in ---------------------->|autoconf|------> configure
                                    `--------'
```

configure.ac 1/2

- Inizializzazione di `autoconf`
  - `AC_INIT([amhello],[1.0],[bug-automake@gnu.org])`
    - Nome del package, versione, indirizzo per bug-report
  - `AM_INIT_AUTOMAKE([-Wall -Werror foreign])`
    - Disabilita i warning e li tratta come errori
  - `AC_PROG_CC`
    - Controlla la presenza del compilatore C

configure.ac 2/2

- Configurazione dei file di input e output
  - `AC_CONFIG_HEADERS([config.h])`
    - Dichiara nome del file header di output
  - `AC_CONFIG_FILES(Makefile src/Makefile])`
    - Lista dei Makefile da generare
  - `AC_OUTPUT`
    - Comando per generare l'output

`Makefile.am`

- Top-level `Makefile`
  - Generalmente breve
- `SUBDIRS = src`
  - Cerca (e compila) ricorsivamente nella sotto-directory specificate

src/Makefile.am

- `bin_PROGRAMS = hello`
  - L'output sarà un programma (un eseguibile) e sarà chiamato `hello`
- `hello_SOURCES = main.c`
  - Il programma dipende dal file sorgente `main.c`

GENERALIZZANDO

Struttura di un `configure.ac`

configure.ac

```
# Prelude.
AC_INIT([PACKAGE], [VERSION], [BUG-REPORT-ADDRESS])

# Checks for programs.

# Checks for libraries.
# Checks for header files.
# Checks for typedefs, structures, and compiler characteristics.
# Checks for library functions.
# Output files.

AC_CONFIG_FILES([FILES])
AC_OUTPUT
```

Preambolo

- `AC_INIT(PACKAGE,VERSION,BUG-REPORT-ADDRESS)`
  - Necessario per l'inizializzazione
- `AC_PREREQ(VERSION)`
  - Per richiedere una determinata versione di Autoconf, e.g. `AC_PREREQ([2.61])`
- `AC_CONFIG_AUX_DIR(DIRECTORY)`
  - Script ausiliari andranno in `DIRECTORY`, e.g. `AC_CONFIG_AUX_DIR([build-aux])`

Program-checks

- `AC_PROG_CC, AC_PROG_CXX, AC_PROG_F77`
  - Controllo sui compilatori C, C/C++, Fortran77
- `AC_PROG_YACC, AC_PROG_LEX`
  - Controllo sulla presenza di *FLEX* e *BISON*
- `AC_CHECK_PROGS(VAR,PROGS,`
  `[VAL-IF-NOT-FOUND])`
  - Assegna a `VAR` la prima occorrenza di `PROGS` trovata, altrimenti assegna `VAL-IF-NOT-FOUND`, e.g. `AC_CHECK_PROGS([TAR],[tar gtar],[?])`

Azioni

- Per informare l'utente circa l'eventualità di errori/warning

- `AC_MSG_ERROR(ERROR_DESCRIPTION,[EXIT-STATUS])`
  - Stampa `ERROR_DESCRIPTION` a video e sul file `config.log` e termina
- `AC_MSG_WARN(ERROR_DESCRIPTION)`
  - Stampa a video, ma non termina

Output su `config.h`

- Lo script `configure` può definire della macro che possono essere utilizzate dal nostro programma

- `AC_DEFINE(VARIABLE,VALUE,DESCRIPTION)`
  - Nel file `config.h` troveremo la seguente definizione

```
/* DESCRIPTION */
#define VARIABLE VALUE
```

Header files 1/2

- `AC_CHECK_HEADERS(HEADERS... )`
  - Controlla la presenza dei file passati come argomento
  - Include una direttiva `#define HAVE_HEADER_H` per ogni header trovato

- `AC_CHECK_HEADER(HEADER,[ACT-IF-FOUND],`
  `[ACT-IF-NOT])`
  - Opera su singolo header

Header files 2/2

- Esempio

```
AC_CHECK_HEADERS([sys/param.h unistd.h])
```

```
#define HAVE_SYS_PARAM_H
#define HAVE_UNISTD_H
```

- Come utilizzare questa informazione?

```
#if HAVE_UNISTD_H
#include <unistd.h>
#endif
```

Comandi di output

- `AC_CONFIG_FILES([Makefile src/Makefile])`
  - Crea (*automake*) un `Makefile.in` per ogni `Makefile.am` trovato

Automake input

- I file `Makefile.am` permettono di dare una specifica sul cosa (e quando) compilare, definendo le dipendenze tra i file
  - Automake genera una complessa gerarchia di `Makefiles` leggendo `*.am`

- Occorre dichiarare Automake nel file `configure.ac`
  - `AM_INIT_AUTOMAKE([OPTIONS])`

Directory Layout

- Ci deve essere un `Makefile.am` per ogni directory del progetto
- **OGNI** `Makefile.am` deve essere dichiarato nel file `configure.ac` tramite il comando `AC_CONFIG_FILES(...)`

- Per specificare l'ordine di visita ricorsiva delle directory occorre utilizzare la variabile `SUBDIRS`
- La directory corrente è (*implicitamente*) compilata dopo le altre (a meno di un overriding tramite '`.`')

STRUMENTI PER IL CONTROLLO DI VERSIONE

Sommario

- Introduzione
- Subversion
- Git

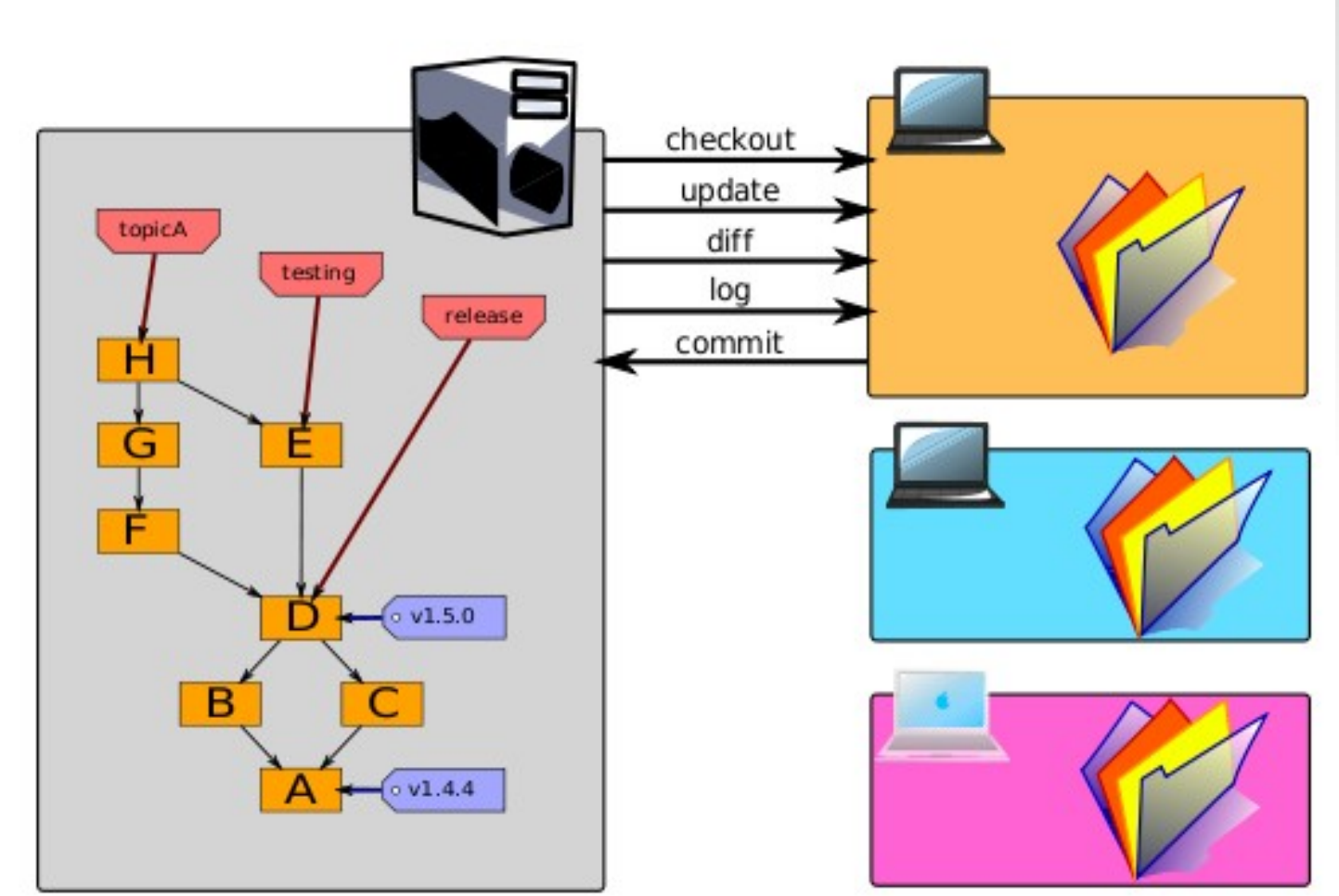
Perchè?

- Più sviluppatori attivi sullo stesso progetto
  - Modifiche alla stessa porzione di codice
  - Modifiche potenzialmente conflittuali
  - Modifiche in parallelo
- Sviluppatori geograficamente distanti
  - Come cooperare?
  - Come scambiare le modifiche al codice in maniera efficiente e sicura?

Soluzione

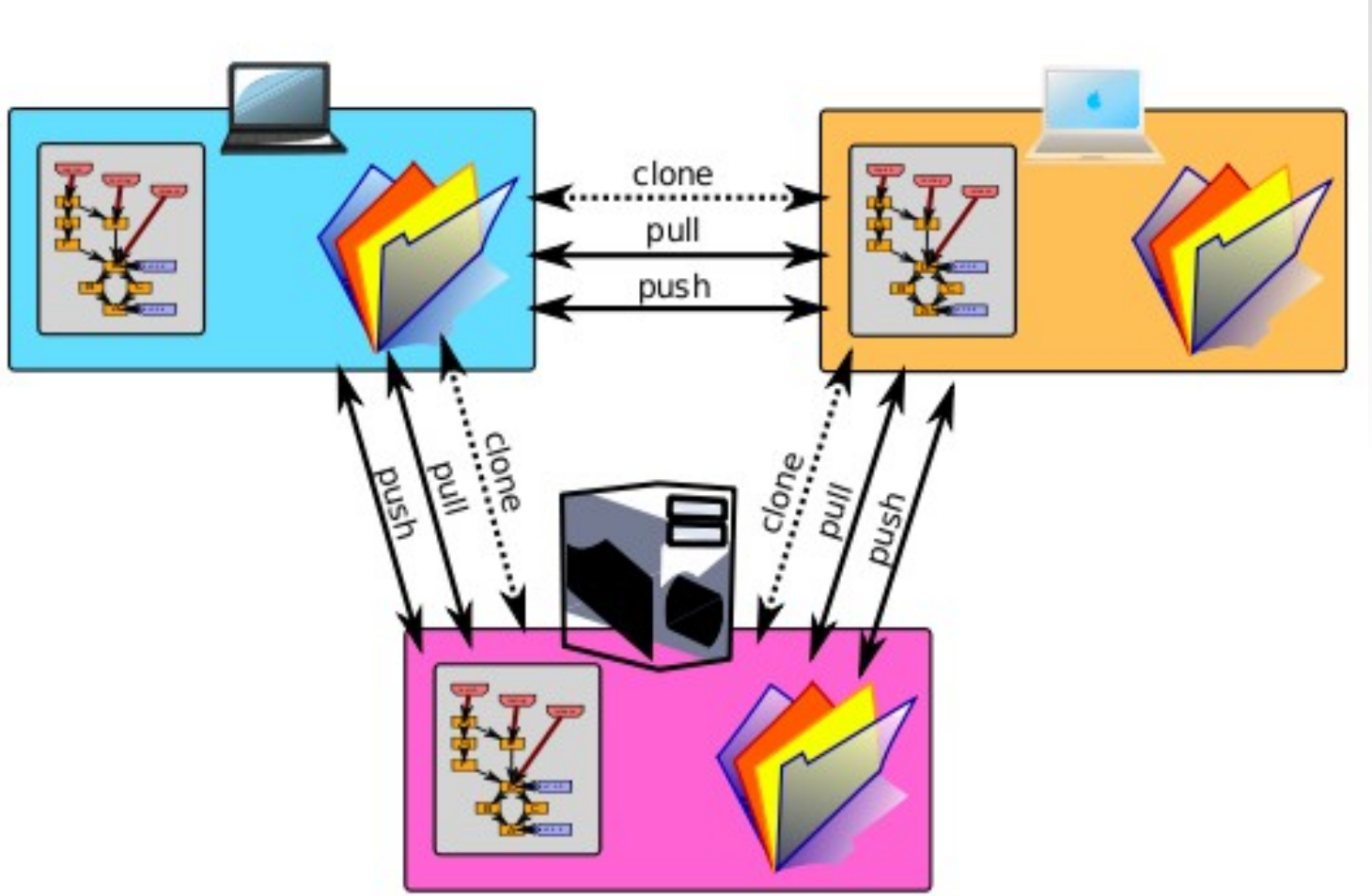
- **Source Control Management**
  - Tener traccia delle modifiche al codice
  - Database delle modifiche (*repository*)
- *Centralizzato*
  - Un singolo database, sul server
  - Il client detiene una propria copia con uno stato
- *Decentralizzato (distribuito)*
  - Ognuno può far la parte del server
  - History completa
  - Operazioni off-line

Soluzione centralizzata



- Single point of failure
- Collo di bottiglia

Soluzione distribuita



- No single poit of failure
- Copie multiple

Subversion: caratteristiche

- Comprende gran parte delle caratteristiche di CVS (obsoleto)
- Un *commit* è un'operazione **atomica**
- Server centralizzato
- Paradigma client/server

Operazioni

- Comandi fondamentali (no amministrazione)
  - **Checkout** [`svn checkout`] permette di creare una copia locale del repository, contenente i file del progetto
  - **Commit** [`svn commit`] permette di applicare in maniera definitiva le modifiche fatte al codice; tali modifiche saranno poi visibili agli altri utenti
  - **Update** [`svn update`] permette di sincronizzarea la copia locale con il repository in remoto, per sincronizzare le modifiche

Git

- Originariamente pensato per supportare lo sviluppo del kernel Linux, ora è utilizzato da *molti* progetti

- Contrariamente a Subversion, Git è **distribuito**
  - Non esiste un repository centrale
  - Ognuno ha un proprio repository locale
  - Sono possibili (e molto importanti) **branches locali**

Git repository

- `.git/config`
  - File di configurazione del repository
- `.git/description`
  - Descrizione del repository
- `.git/info/exclude`
  - Lista dei pattern da ignorare
  - I file corrispondenti ai pattern stabiliti non saranno tracciati

Help dei comandi

- Git comprende oltre 130 comandi
  - L'uso quotidiano però ne richiede un sottoinsieme

- `git help`
  - Lista dei comandi più utili
- `git <comando> -h`
  - Breve descrizione del comando

Clonare un repository

- L'equivalente (~) del *checkout* per SVN è chiamato *cloning* in Git
- Principali differenze
  - Dopo un *checkout* con SVN avrete una copia locale dell'**ultima versione** del repository
  - Effettuando un `git clone` invece avrete l'**intero repository**, compresa la history
  - In questo modo potrete effettuare molte delle operazioni off-line

History delle modifiche

- `git log` permette di visualizzare lo storico delle modifiche effettuate al software
  - L'opzione `-p` mostra anche il corrispondente diff
- `gitk` permette una visualizzazione grafica della history
- La history non è lineare (come in Subversion), ma è un grafo
  - Più complicato (~learning curve)
  - Molto più potente

Aggiornare il repository locale

- Per applicare le modifiche fatte da terzi al repository (l'equivalente dell'*update* di Subversion) si utilizza

```
git pull
```

- Il comando effettua due operazioni
  - Recupera le modifiche dal repository remoto
  - Effettua un merge di tali modifiche con il repository locale

Ramificazioni

- Le ramificazioni (~*branches*) rappresentano un potente strumento in GIT
  - Sono visibili solamente in locale
  - Permettono di suddividere il lavoro in tematiche/categorie differenti
- GIT **incoraggia** l'uso delle branch
  - `git branch <branchname>` creazione
  - `git checkout <branchname>` per lavorare su quel particolare branch
  - `git branch -l` lista delle branches

Applicare le modifiche

- `git status` permette di visualizzare quelle che sono le modifiche apportate al progetto
- `git add <nome_file>` prepara `<nome_file>` per il commit
- `git commit` applica in locale le modifiche (commit locale, non necessita di connessione alla rete)
- `git push` rende visibili/disponibili le modifiche ai repository in remoto

References 1/3

- **Informazioni sul software GNU/Linux**
  - `www.gnu.org`
  - `www.kernel.org`
  - `www.nic.funet.fi`

- **Programmazione di sistema**

  Mitchell, M.; Oldham, J and Samuel, A. *"Advanced Linux Programming"*. New Riders Publishing, 2001. Disponibile sul sito *`http://www.advancedlinuxprogramming.com/`*

References 2/3

- **Sistemi operativi** *(slide del Deitel&Deitel)*
  http://www.deitel.com/books/os3e/slides.html

- **GNU/Linux per sistemi embedded e introduzione**
  `http://free-electrons.com/`

- **Introduzione a Subversion**
  `http://svnbook.red-bean.com/`

References 3/3

- **Git Manual**
  `http://www.kernel.org/pub/software/scm/git/docs/usermanual.html`
- **Git home-page**
  `http://gitscm.com/`
- **Presentazione Git**
  `http://excess.org/article/2008/07/ogre-git-tutorial/`