



แนะนำ GNU autotools

เทพพิทักษ์ การุญบุญญานันท์

thep@linux.thai.net

Thai Linux Working Group

แนะนำ GNU autotools

- ภาพสุดท้าย: การติดตั้งซอฟต์แวร์สไตล์ GNU
- ก่อน GNU autotools: make
- GNU autoconf
- GNU automake
- ไลบรารีบนยูนิกซ์
- GNU libtool

ภาพสุดท้าย: การติดตั้งซอฟต์แวร์สไตล์ GNU

- แยก tarball

```
$ gzip -cd foo-0.1.1.tar.gz | tar -xf -
```

- ตรวจสอบระบบและ config source

```
$ ./configure
```

- คอมไพล์โปรแกรม

```
$ make
```

- ตรวจสอบ

```
$ make check
```

ภาพสุดท้าย: การติดตั้งซอฟต์แวร์สไตล์ GNU

- ติดตั้ง
`$ make install`
- เคลียร์พื้นที่คอมไพล์
`$ make clean`
- ถอดถอนโปรแกรม
`$ make uninstall`

ภาพสุดท้าย: การติดตั้งซอฟต์แวร์สไตล์ GNU

- เคลียร์พื้นที่ให้เหมือนต้นฉบับ
`$ make distclean`
- เคลียร์พื้นที่สำหรับผู้ดูแลโปรแกรม
`$ make maintainer-clean`
- สร้าง source tarball
`$ make dist`
- สร้าง source tarball และตรวจสอบ
`$ make distcheck`

ภาพสุดท้าย: การติดตั้งซอฟต์แวร์สไตล์ GNU

- การคอมไพล์ด้วย VPATH โดยแยกคอมไพล์นอก source tree

```
$ gzip -cd foo-0.1.1.tar.gz | tar -xf -  
$ cd foo-0.1.1  
$ mkdir build  
$ cd build  
$ ../configure  
$ make
```

ภาพสุดท้าย: การติดตั้งซอฟต์แวร์สไตล์ GNU

- การเปลี่ยนรากสำหรับติดตั้งไฟล์ต่างๆ

```
$ ../configure --prefix=/usr
```

- การสร้าง binary tarball ด้วย DESTDIR

```
$ mkdir /tmp/dist
```

```
$ make install DESTDIR=/tmp/dist
```

```
$ tar -C /tmp/dist -cf - /tmp/dist |  
gzip -9 > foo-0.1.1.tar.gz
```

ก่อน GNU autotools: make

- **make:** โปรแกรมบนยูนิกซ์เพื่อช่วยทำให้การคอมไพล์โปรแกรมขนาดใหญ่ เป็นไปอย่างอัตโนมัติ
 - ▶ auto incremental build
 - ▶ เก็บ option ต่างๆ ของการคอมไพล์
 - ▶ ทำคำสั่งอื่นๆ ที่ไม่เกี่ยวกับการคอมไพล์ เช่น install, clean ฯลฯ

ก่อน GNU autotools: make

- Makefile: ไฟล์สำหรับเก็บกฎการคอมไพล์
 - ▶ รูปแบบคือ:
target: dependencies
TAB *command*
TAB ...
(บรรทัดเปล่า)
 - ▶ *target* คือไฟล์ที่จะต้อง build ใหม่ด้วยชุด *command* ทั้งหมด เมื่อ *dependencies* ไฟล์ใดไฟล์หนึ่งใหม่กว่า *target*

ก่อน GNU autotools: make

ตัวอย่าง Makefile

```
greet: hello.o bye.o
```

```
TAB cc -o greet hello.o bye.o
```

```
hello.o: hello.c bye.h
```

```
TAB cc -c hello.c
```

```
bye.o: bye.c bye.h
```

```
TAB cc -c bye.c
```

```
install:
```

```
TAB mkdir -p /usr/local/bin
```

```
TAB cp -f greet /usr/local/bin
```

ก่อน GNU autotools: make

เทคนิคอื่นๆ ใน Makefile

- ตัวแปร
 - ▶ กำหนด: `var=val`
 - ▶ ใช้ค่า: `$(var)`

ก่อน GNU autotools: make

- กฎ abstract

- ▶ เขียนกฎทั่วไปสำหรับสร้าง *file.s2* จาก *file.s1* :-
.s1.s2:

TAB *command*

TAB ...

- ▶ ตัวแปรพิเศษสำหรับใช้ใน *command*

- $\$@$ หมายถึง target
- $\$<$ หมายถึง dependency ที่เปลี่ยนแปลง และทำให้กฎทำงาน
- $\$^$ หมายถึง dependency ทุกไฟล์ของกฎ

ก่อน GNU autotools: make

- กฎ abstract

- ▶ suffix ของไฟล์ที่ใช้ในกฎ abstract จะต้อง list ไว้:

- `.SUFFIXES: s1 s2 ... sn`

ก่อน GNU autotools: make

ตัวอย่าง Makefile ที่แก้แล้ว

```
CC = gcc
CFLAGS = -Wall -g
PREFIX = /usr/local
.SUFFIXES: .c .o
.c.o:
[TAB] $(CC) $(CFLAGS) -c $<
.o:
[TAB] $(CC) $^ -o $@
greet: hello.o bye.o
hello.o: hello.c bye.h
bye.o: bye.c bye.h
install:
[TAB] mkdir -p $(PREFIX)/bin
[TAB] cp -f greet $(PREFIX)/bin
```

ก่อน GNU autotools: make

ข้อบกพร่องที่ยังเหลือของการใช้ Makefile

- ต้องแกะรอย dependency ของ target จากซอร์สโค้ดเอง (`gcc -MM` สามารถทำได้ แต่ `cc` ของยูนิกซ์ปกติไม่มี)
- การปรับพารามิเตอร์ต่างๆ ใน Makefile เอง เป็นสิ่งที่ผู้ใช้ยอมรับไม่ได้
- จะต้องเขียนกฎสำหรับ target มาตรฐาน เช่น `clean`, `install`, `uninstall` ให้ครบด้วยตัวเอง ซึ่งน่าเบื่อหน่าย
- ในโครงการที่มีไจเรกทอรีย่อยมาก จะต้องจัดการ Makefile ให้เรียกกันแบบ recursive โดยส่งพารามิเตอร์ไปให้หมด

GNU autoconf

- ยูนิกซ์มีการแตกแขนงหลากหลาย (BSD, AT&T System V, SunOS, Ultrix, HP-UX, AIX, SCO Unix, GNU/Linux, etc.) ทำให้เงื่อนไขการคอมไพล์โปรแกรมต่างกัน
- มี solution 4 แบบสำหรับการ config อัตโนมัติ ณ ปี 2535
 - ▶ metaconfig โดย Larry Wall et. al.
 - ▶ configure ของ Cygnus และ GCC
 - ▶ GNU autoconf โดย David MacKenzie
 - ▶ Imake ของ X Window

GNU autoconf

- ทั้งหมดจะแบ่งการทำงานเป็นสองขั้น คือ `config` `Makefile` แล้วจึง `make`
- `metaconfig` และ `autoconf` จะใช้โดยผู้เขียนโปรแกรม โดยจะสร้างเป็น `shell script` สำหรับตรวจสอบระบบ
- `configure` ของ `Cygnus` และ `GCC` เป็น `shell script` ที่ทำงานโดยอาศัยข้อมูลจาก `config file` ย่อยๆ สำหรับ `platform` ต่างๆ
- `imake` เป็นโปรแกรมภาษาซีที่อาศัยข้อมูล `config` ที่กำหนดไว้สำหรับแต่ละ `platform`

GNU autoconf

- script ที่สร้างจาก **metaconfig** จะ interactive โดยถามผู้ใช้เป็นเรื่องๆ ส่วนที่เหลือจะอัตโนมัติทั้งหมด
- ในปี 2537 David MacKenzie ได้รวม feature ของ **configure** ของ Cygnus และ GCC เข้าในระบบของ **autoconf** และ Cygnus และ GCC ก็ได้ย้ายมาใช้ **autoconf** แทน
- **metaconfig** ยังคงใช้อยู่ทุกวันนี้โดย Perl
- **imake** ไม่ค่อยมีการใช้นอกเหนือโครงการของ X Window และโครงการใหม่ๆ ก็ไม่ค่อยใช้ **imake** กันแล้ว

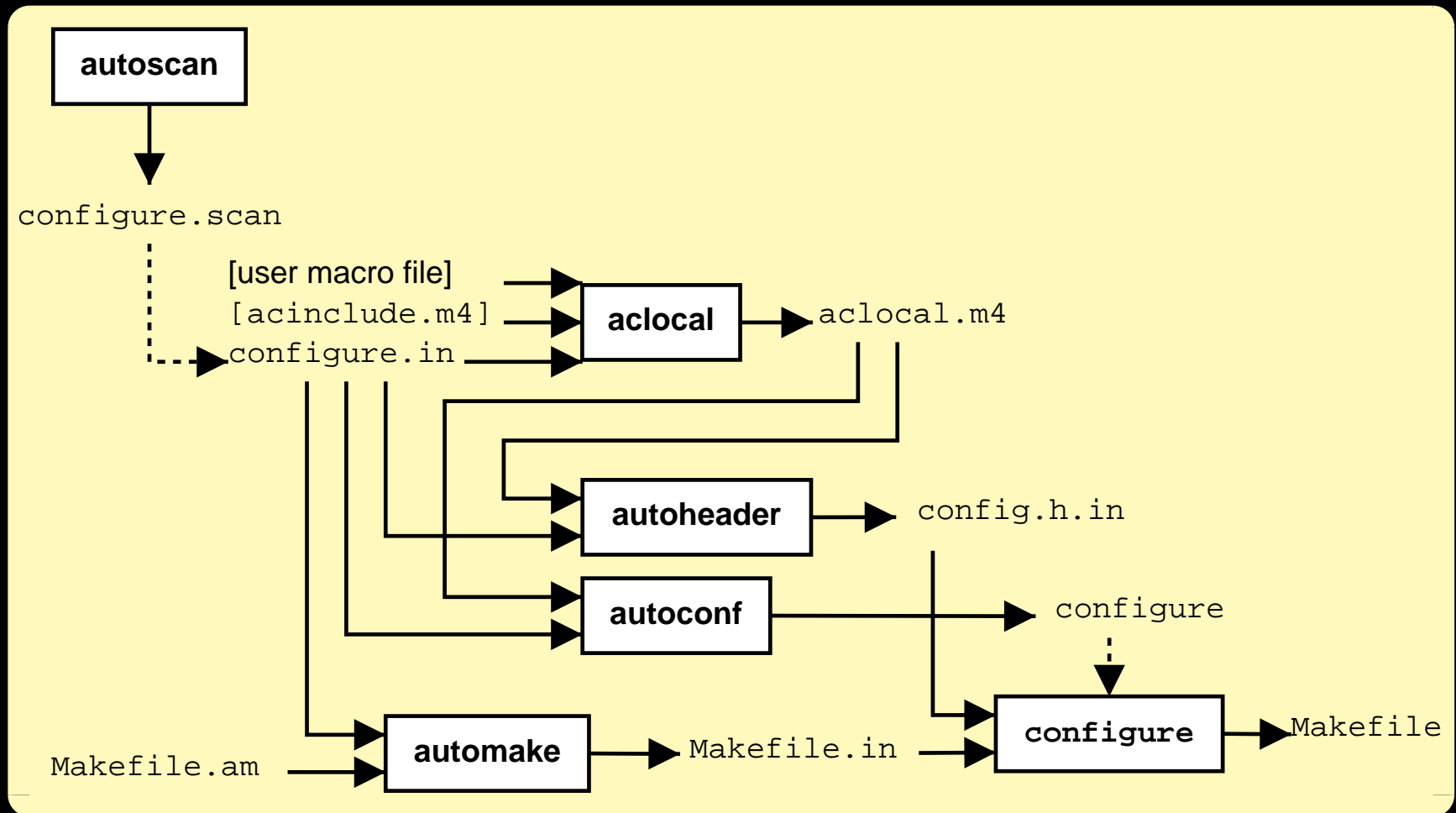
GNU automake

- autoconf ใช้ตรวจสอบระบบได้ดี แต่ยังขาดความสามารถที่ imake มี: การตรวจ dependency อัตโนมัติ
- ด้วย imake ผู้เขียนโปรแกรมเพียงระบุซอร์สสำหรับแต่ละ target ใน `Imakefile` และ imake จะสร้าง `Makefile` ที่มี dependency ให้
- David MacKenzie เริ่มเขียน automake รุ่นแรกเมื่อปี 2537 และ Tom Tromey ได้เขียนขึ้นใหม่อีกในปี 2538

GNU automake

- ด้วย `automake` ผู้เขียนโปรแกรมเพียงระบุซอร์สสำหรับแต่ละ target ใน `Makefile.am` และ `automake` จะสร้าง `Makefile.in` ที่มี dependency ให้
- target มาตรฐานของ GNU เช่น `check`, `install`, `uninstall`, `clean`, `dist`, `distclean` ก็จะถูกสร้างมาให้ด้วย
- สนับสนุนการสร้าง test suite

ระบบคอมไพล์ของ GNU



ระบบคอมไพล์ของ GNU

- สำหรับผู้พัฒนา
 - ▶ **autoconf** สร้าง `configure` shell script จากแมคโครใน `configure.in`^a โดยแทนที่แมคโคร `m4` ด้วยค่าจากแมคโครในคลัง และจาก `aclocal.m4`
 - ▶ **automake** สร้าง `Makefile.in` จาก `Makefile.am` ทั้งหมดที่ลิสต์ไว้ใน `configure.in`

^aใน autoconf 2.50 จะเริ่มใช้ `configure.ac` แทน `configure.in` แต่ก็ยังอ่านชื่อเดิมได้ด้วย

ระบบคอมไพล์ของ GNU

- สำหรับผู้ใช้
 - ▶ `configure` ตรวจสอบระบบและสร้าง `Makefile` ทั้งหมดจาก `Makefile.in` และ `config.h` จาก `config.h.in` ด้วยการแทนค่าตัวแปร
 - ▶ `make` คอมไพล์โปรแกรมตามกฎใน `Makefile` ที่ได้

ตัวอย่างโครงการอย่างง่าย

- `configure.in`:
`AC_INIT(hello.c)`
`AM_INIT_AUTOMAKE(hello,1.0)`
`AC_PROG_CC`
`AC_PROG_INSTALL`
`AC_OUTPUT(Makefile)`
- `Makefile.am`:
`bin_PROGRAMS = greet`
`greet_SOURCES = greet.c hello.c bye.c hello.h bye.h`

ตัวอย่างโครงการอย่างง่าย

■ จัดการ..

```
$ aclocal
```

```
$ autoconf
```

```
$ touch NEWS README AUTHORS ChangeLog
```

```
$ automake --add-missing
```

```
$ mkdir build
```

```
$ cd build
```

```
$ ../configure
```

```
$ make
```

หลักการของ autoconf

- อ่าน `configure.in` แทนที่แมคโคร `m4` ด้วย `definition` จากคลังของ `autoconf` เอง และจาก `aclocal.m4` ในรากปัจจุบัน และสร้างเป็นสคริปต์ `configure`
- ไฟล์ `aclocal.m4` สร้างได้ด้วยคำสั่ง `aclocal` ซึ่งจะค้นหา `definition` ของแมคโคร `m4` ทั้งหมดในระบบที่ใช้ โดย `configure.in` รวมกับแมคโครใน `acinclude.m4` ในรากปัจจุบัน แล้วเขียนรวมกันใน `aclocal.m4` ดังกล่าว

หลักการของ autoconf

- สำคัญว่าจะต้องเรียก `aclocal` ก่อน `autoconf` หรือ `automake` เพราะแมคโครบางตัวไม่ได้มากับ `autoconf` (เช่น `AM_INIT_AUTOMAKE` ในตัวอย่างดังกล่าว ซึ่งเป็นของ `automake`) `autoconf` จะไม่สามารถกระจายแมคโครดังกล่าวได้

หลักการของ autoconf

- สคริปต์ `configure` ที่ได้ เมื่อรันแล้วจะ
 - ▶ ตรวจสอบ `config` ของระบบหรือตรวจสอบ `option` ของผู้ใช้ตามที่กำหนดในแมคโคร
 - ▶ อ่านไฟล์ `out.in` สำหรับ `out` ทุกค่าที่อยู่ในแมคโคร `AC_OUTPUT(out)`, แทนที่ตัวแปร `@var@` ในไฟล์ดังกล่าวทุกตัวที่ปรากฏในคำสั่ง `AC_SUBST(var)` ด้วยค่าที่เซ็ตได้ระหว่าง `configure` ก่อนจะเขียนเป็นไฟล์ `out` ที่ต้องการ

หลักการของ automake

- รายการของ `Makefile.am` ทุกไฟล์ที่จะประมวลผล จะอ่านมาจากแมคโคร `AC_OUTPUT()` ใน `configure.in`
- แต่ละไดเรกทอรีย่อย จะต้องมี `Makefile.am` เฉพาะ ไม่สามารถอ้างถึงไฟล์ที่อยู่ในไดเรกทอรีย่อยได้
- `automake` จะจัดการตัวแปรของ `automake` ทุกตัวใน `Makefile.am` และส่งผ่านส่วนที่เหลือไปยัง `Makefile.in`

หลักการของ automake

- **Primary:** หน่วยของ object ที่ automake จัดการได้
- primary ที่สำคัญใน automake ได้แก่
 - ▶ DATA: ส่วนที่ install โดยไม่มีการประมวลผล
 - ▶ HEADERS: คล้าย DATA แต่ใช้กับ header file
 - ▶ SCRIPTS: คล้าย DATA แต่ติดตั้งแบบ executable
 - ▶ MANS: ติดตั้งเป็น manual page แยก section
 - ▶ TEXINFOS: ติดตั้งเป็น info page

หลักการของ automake

- primary ที่สำคัญใน automake (ต่อ)
 - ▶ PROGRAMS: ส่วนที่ต้องคอมไพล์ก่อนติดตั้งเป็น executable
 - ▶ LIBRARIES: ส่วนที่ต้องคอมไพล์ก่อนติดตั้งเป็น library
 - ▶ LTLIBRARIES: library ที่จัดการด้วย libtool

หลักการของ automake

- การเขียนกฎการสร้าง/ติดตั้ง object ต่างๆ จะอาศัยตัวแปร primary ที่มี prefix ขยาย:

```
bin_PROGRAMS = greet
```

→ กำหนด target เป็นโปรแกรมที่ต้องคอมไพล์ก่อนติดตั้งที่ bindir

```
greet_SOURCES = hello.c bye.c
```

→ source ของ greet ได้แก่สองไฟล์นี้

```
pixmapsdir = @datadir@/pixmaps
```

```
pixmaps_DATA = hello.png bye.png
```

→ ให้ติดตั้งไฟล์ทั้งสองใ้ราก pixmapsdir ที่กำหนด

หลักการของ automake

■ ตัวแปรเพิ่มเติม

- ▶ `_DEPENDENCIES`: เพิ่ม dependency ให้กับ target (โดยดีฟอลต์จะคำนวณอัตโนมัติ)
- ▶ `_LDADD`: เพิ่ม shared object ที่จะลิงก์กับโปรแกรมหรือไลบรารี
- ▶ `_LDFLAGS`: เพิ่มแฟล็กสำหรับการลิงก์
- ▶ `_LIBADD`: คล้าย `_LDADD` แต่ใช้กับ static library เท่านั้น

หลักการของ automake

■ ตัวแปรอื่นๆ

- ▶ SUBDIRS: ไดรกทอรีย่อยที่ต้องเข้าไป build
- ▶ EXTRA_DIST: ไฟล์อื่นๆ ที่ไม่ได้ลิสต์ไว้ใน `_SOURCES` แต่ต้องการให้รวมเข้าไปใน source tarball ด้วยในขณะ “make dist”
- ▶ DISTCLEANFILES: เพิ่มไฟล์ที่ต้องการให้ลบเมื่อสั่ง “make distclean”
- ▶ MAINTAINERCLEANFILES: เพิ่มไฟล์ที่ต้องการให้ลบเมื่อสั่ง “make maintainer-clean”

ไลบรารีบนยูนิกซ์

- static library บนยูนิกซ์เป็น indexed archive ของ object file:

```
$ ar cru libgreet.a hello.o bye.o  
$ ranlib libgreet.a
```
- เมื่อลิงก์โปรแกรมกับ static library ตัวไลบรารีจะถูกรวมเข้าไปใน executable file ของโปรแกรม
- shared library จะถูกโหลดขณะ run-time และ resolve symbol ต่างๆ แบบ dynamic
→ ประหยัดเนื้อที่ฮาร์ดดิสก์ และสะดวกต่อการแก้ bug ในไลบรารีโดยไม่ต้องคอมไพล์โปรแกรมใหม่ทั้งหมด

GNU libtool

- shared library มีขึ้นครั้งแรกใน AT&T Unix System V Release 3 และยูนิกซ์เจ้าต่างๆ ก็ implement ตามกันอย่างรวดเร็ว ...ด้วยวิธีที่ต่างกัน (อีกแล้ว)
- ซอฟต์แวร์ทั้งหลายต่างก็หาทางแก้ปัญหานี้ด้วยวิธีของตนเอง
- ในปี 2539 Gordon Matzigkeit เริ่มเขียน libtool เป็น shell script สำหรับสร้างและลิงก์ shared library บนระบบที่แตกต่างกันนี้

การใช้ GNU libtool ร่วมกับ automake

- automake มี primary `LTLIBRARIES` สำหรับสร้าง/ติดตั้งไลบรารีด้วย libtool:
`lib_LTLIBRARIES = libgreet.la`
`libgreet_la_SOURCES = hello.c bye.c`
`bye.h`
- สังเกตว่าชื่อตัวแปรของ automake ต้อง normalize โดยเปลี่ยนอักขระที่ไม่ใช่ตัวอักษรเป็น underscore ให้หมด

ตัวอย่างการสร้าง shared library

- `configure.in`:
AC_INIT(hello.c)
AM_INIT_AUTOMAKE(libgreet,0.1.1)
AC_PROG_CC
AC_PROG_INSTALL
AC_PROG_LN_S
AC_PROG_LIBTOOL
AC_OUTPUT(Makefile)
- `Makefile.am`:
lib_LTLIBRARIES = libgreet.la
libgreet_la_SOURCES = hello.c bye.c
include_HEADERS = greet.h hello.h bye.h

ตัวอย่างการสร้าง shared library

● จัดการ..

```
$ libtoolize
```

```
$ aclocal
```

```
$ autoconf
```

```
$ touch NEWS README AUTHORS ChangeLog
```

```
$ automake --add-missing
```

```
$ mkdir build
```

```
$ cd build
```

```
$ ../configure
```

```
$ make
```



ขอบคุณครับ

Happy Hacking!!