

mk-configure – легковесная альтернатива GNU autotools

А. В. Чеусов,
Ведущий программист ИП Invention Machine corp.,
vle@gmx.net

7 июля 2009 г.

Созданный более 15 лет назад комплект программного обеспечения autotools в настоящее время является, по всей видимости, наиболее широко используемым пакетом программ, предназначенным для разработки приложений, переносимых на широкий спектр POSIX-совместимых операционных систем. Круг проблем, связанных с разработкой переносимого ПО а также причины их возникновения хорошо известны. Это и многочисленные расширения API, отсутствующие в спецификациях POSIX и присутствующие далеко не во всех вариантах UNIX-like систем, это и “особенности” реализации, отсталость широко используемых систем от новейших спецификаций и многое другое. В настоящее время autotools распространен настолько широко, что трудно, наверное, найти программиста, системного администратора или даже “продвинутого” пользователя, не знакомого со “стандартным” способом сборки программного обеспечения: “configure; make; make install”. Наверное, я не ошибусь, если скажу, что в настоящий момент autotools является стандартом de-facto.

Итак, autotools – стандартный инструмент, знакомый сегодня практически каждому программисту и пользователю UNIX-like систем, доказавший свою работоспособность более чем 15 годами эксплуатации. Но является ли он незаменимым? Отвечает ли он современным требованиям к разработке программного обеспечения, которые в значительной степени изменились со времени его создания? Удобен ли он в работе? Я считаю, что нет.

Прежде всего, необходимо вспомнить основную задачу, которую ставили перед собой разработчики autotools. Основной задачей является предоставление программисту инструмента, с помощью которого он смог бы разрабатывать ПО, которое в свою очередь конечный пользователь смог бы скомпилировать на своей системе без предварительной установки второстепенного программного обеспечения, без редактирования файлов, отвечающих за сборку, файлов настроек и прочих действий, отнимающих огромное количество времени и требующих определенной квалификации пользователя. То есть, в конечном итоге задача заключалась в предоставлении конечному пользователю ПО, которое он смог бы максимально быстро и просто скомпилировать и использовать на своей системе, какой бы она ни была. Итак, есть задача. Решена ли она? Прежде чем ответить на этот вопрос, немного отвлекусь. Около трех лет я являюсь активным пользователем source-based пакетной системы pkgsrc, являющейся дочерним проектом операционной системы NetBSD. В задачи pkgsrc входит сборка программного обеспечения из исходных текстов, создание так называемых бинарных пакетов ПО, и, естественно, управление этими пакетами в рабочей системе. Буквально в первые же годы существования pkgsrc была “перенесена” на другие операционные системы, среди которых в настоящий момент Solaris, FreeBSD, OpenBSD, Linux, HP-UX, AIX и многие другие, всего их сейчас 15. Кроме того, pkgsrc позволяет собирать проекты используя различные компиляторы, предоставляя пакетировщику все необходимые средства для управления этим процессом. Например, на платформе Linux можно воспользоваться не только C/C++ компилятором из комплекта GCC, но и, например, Intel C/C++ или Sun Studio. На платформе Interix возможно использование компилятора компании Microsoft. Естественно степень поддержки всех поддерживаемых платформ и компиляторов различна. Очевидно, наилучшей поддержкой обладают системы NetBSD и DragonFlyBSD, для которых pkgsrc является “родной” пакетной системой. Другие системы поддерживаются несколько хуже. Есть и откровенно проблемные платформы, такие, например, как QNX, Interix и даже намного более известная и распространенная система Solaris. К “другим” системам относится также и Linux, с которым я постоянно работаю уже много лет, и на которой последние три года широко использую pkgsrc в качестве дополнительного репозитория пакетов. В процессе

эксплуатации `pkgsrc` за все это время мне, естественно, много раз приходилось исправлять ошибки в нужных мне пакетах, работающих на NetBSD, но не работающих по тем или иным причинам на Linux. Подавляющее большинство этих программ использовали для сборки как минимум одну из следующих утилит: `autoconf`, `automake` и `libtool`, то есть, в той или иной степени использовали `autotools`. Каков же опыт этой работы? Какие уроки получены? Оказавшись в роли “пакетировщика” многоплатформной пакетной системы, задачей которой (в идеале) является обеспечение работы ПО на почти всех “живых” системах многие вещи выглядят совсем иначе, на многие проблемы смотришь совсем по-другому. В результате неописуемых приключений в веселой стране кроссплатформного программирования, я задался вопросом, а какое вообще количество заплаток прикладывается к программам для обеспечения их работы на хотя бы двух системах, NetBSD и DragonFlyBSD, ни одна из которых не является тем, что называется `mainstream`. У меня нет точной статистики, но, думаю, я не ошибусь, если скажу, что сейчас подавляющее большинство программ разрабатывается сейчас на Linux и FreeBSD. Но никак не NetBSD и DragonFlyBSD. В конце концов я задался вопросом, какое количество программ требуют заплаток и каков процент программ собирается безо всяких проблем “из коробки”, то есть без изменений авторского варианта. Примерно полтора года назад, то есть осенью 2007-ого я получил такие цифры. 54% программ, находящихся в основном репозитории `pkgsrc` и использующих `autoconf`, требуют заплаток на `Makefile`-ы и/или на файлы `configure.ac`. 54%! То есть, более половины. Конечно, заплатки эти очень разные по назначению. Я не анализировал их все, но могу сказать точно, весьма значительная их часть связана с неправильным использованием `autoconf`, `automake`, `libtool` или неправильным написанием `Makefile`-ов. Из чего я сделал следующий вывод. Задача по обеспечению инструментария, упрощающего разработку переносимого программного обеспечения комплектом программ `autotools` не решена.

Возникает законный вопрос. Каковы причины этого “провала”? По-другому я это назвать никак не могу. Далее – без доказательств. `autotools` слишком большой. Исходные коды и `autoconf` и `automake` составляют десятки тысяч строк кода. Задача, которую решают эти инструменты мне представляются гораздо более простой. Я бы ска-

зал, на порядок более простой. На мой взгляд, здесь налицо вопиющий *overdesign*. Объем исходного кода этих инструментов явно не соответствует сложности решаемой задачи. *autotools* – слишком сложен в использовании. По крайней мере об этом говорят виденные мной многочисленные примеры его неправильного использования. К другим недостаткам *autotools* я бы отнес также подход, положенный в основу *autoconf* и *automake* – кодогенерацию. Даже не обращая внимание на низкое качество генерируемого кода (иногда он просто изобилует ошибками, часто не настолько хорошо переносим на другие платформы, насколько обещают авторы), нельзя не отметить его огромный объем. Малейший проект из 500 строк исходного кода на C превращается в сотни килобайт сгенерированных скриптов. Конечно, сейчас 21-й век, сети в наше время быстрые, трафик и ОЗУ дешевые, а накопители просто огромны. Но это ведь просто неэстетично, в конце концов! В случае возникновения каких-либо нестандартных проблем, необходимо анализировать не небольшой по объему исходный код *configure.ac* или *Makefile.am*, а самые что ни на есть нечитабельные *blob*-ы – *configure* и *Makefile*. А ведь в *pkgsrc* и других пакетных системах множество примеров накладки именно на эти файлы, а не на исходные. Кроме того, нельзя не отметить низкую скорость выполнения *configure* скриптов, не слишком большая проблема для современной *desktop*-аппаратуры и современных операционных систем с быстрым системным вызовом *fork(2)*, но тем не менее. Существуют приложения, где хотелось бы значительно снизить время ожидания. Казалось бы, существует давно испытанное средство, кросс-сборка, но не тут то было. Львиная доля проектов, основанных на *autotools*, нельзя собрать таким образом. Увы. Печальный факт. Далее, отсутствует возможность кеширования результатов проверок между различными проектами. Скажем, при сборке нескольких тысяч проектов на одинаковом окружении 99% проверок повторяются от проекта к проекту. Зачем тратить время впустую? Идентичность окружения ведь легко проверить. И последним, очень важным недостатком *autotools* является то, что он крайне неудобен для разработки. Упростить разработку ПО, судя по всему, не было целью авторов проекта, но тем не менее. Кто-то же должен разрабатывать программы, которые потом пользователи “легко и просто” установят и будут использовать. За время существования *autotools* и его победного шествия в мире открытого

программного обеспечения все уже свыклись со схемой “sh configure; make; make install”. Но я позволю себе напомнить, как мы разрабатываем наши программы. Схема в общих чертах такая:

```
while наша программа недостаточно хороша; do
    редактировать исходный код и документацию
    make all && make test
end
выпустить новую версию программы
```

Обратите внимание, в этой схеме нет фазы “конфигурации” или “преобразуем Makefile.am во что-то, а потом это что-то еще во что-то и только потом...” Все это лишние противоестественные сущности, которые, по моему глубокому убеждению, не имеют к собственно разработке никакого отношения. На странице <http://en.wikipedia.org/wiki/File:> приведена схема работы autotools. Я не вижу ни одной причины, почему перечисленные в этой схеме утилиты и команды: autoscan, aclocal, autoheader, automake, configure а также файлы: Makefile.am, configure.ac, и неперечисленные config.status, config.sub, config.guess, install.sh и многие другие должны занимать такое большое место в процессе разработки. Попробуйте создать блок-схему или алгоритм на псевдоязыке, описывающий полный цикл разработки ПО, основанного на autotools. Не слишком ли сложным стал этот процесс? Не слишком ли мы удалились от схемы, приведенной выше? И ради чего?

С критикой покончено. Пора переходить к конструктиву. Учитывая выше изложенное, я пришел к выводу о необходимости поиска альтернативы autotools, построенного на других принципах и другом подходе. Не могу сказать, что в этом была острая практическая необходимость, скорее это был чисто академический интерес. Альтернативы autotools, конечно же есть, CMake, pmk (aka pre make kit) и другие (очень многие из которых к сожалению давно не развиваются и не поддерживаются), но по разным причинам ни одна из них меня не устроила в полной мере. В конце концов я решил создать свою собственную систему, тем более что к этому времени у меня уже сложилось более менее полное представление о том, какой хотелось бы ее видеть. Так несколько месяцев назад на свет появился проект mk-configure (<http://sourceforge.net/projects/mk-configure>), легковесная, простая в использовании замена automake, autoconf и,

в определенной степени, libtool. Языки реализации: bmake (NetBSD make, портированный под различные платформы) и POSIX shell. Принципы, положенные в основу. Все, что необходимо для сборки программы, включая определение особенностей окружения, записывается в Makefile. Никаких Makefile.am, Makefile.in, configure.ac и тому подобных. В них нет никакой необходимости. Для полной сборки проекта необходимо запустить только одну команду – “bmake all”. Эта команда проведет и анализ особенностей платформы и собственно сборку проекта. Такой подход значительно упрощает и ускоряет разработку. Сравните запуск одной команды с тем, что приходится делать при использовании autotools. Что касается содержимого Makefile, то в подавляющем большинстве случаев нет необходимости создавать правила вручную. Все, что необходимо сделать, это указать, что является главной целью проекта (библиотека, программа, набор скриптов и так далее), и перечислить исходные файлы. Все остальное команда “bmake” сделает сама. Также, нет необходимости самостоятельно реализовывать цели “all”, “install”, “clean” и тому подобные, они уже реализованы в библиотечных файлах, созданных специально для “bmake”, все, что необходимо сделать – это включить их в основной Makefile с помощью директивы “include”. mk-configure полностью документирован (на английском языке). Я приведу несколько примеров, демонстрирующих его базовые возможности. Пример Makefile-а проекта hello_world, включающего один единственный исходный файл hello_world.c.

```
PROG=    hello_world

MKMAN=  no

.include <mkc.prog.mk>
```

Более сложный пример проекта, состоящего из двух скриптов, одного компилируемого исполняемого файла и man-страницы для него.

```
# Makefile for more complex project consisting of one compile
# and two scripts
PROG=    hello_world
```

```
SRCS=          main.c msg.c

SCRIPTS=       hello_world2 hello_world3

.include <mkc.prog.mk>
```

Пример Makefile-а для проекта, который использует нестандартную функцию `strncpy`.

```
MKC_SOURCE_FUNCLIBS=   strncpy # add strncpy.c to SRCS if ne
MKC_CHECK_FUNCS3+=     strncpy:string.h

PROG=                 hello4
SRCS=                 hello.c
MKMAN=                no

.include <mkc.configure.mk>
.include <mkc.prog.mk>
```

Пример Makefile-а для разделяемой библиотеки.

```
LIB=      mylib
SRCS=     source1.c source2.c source3.c source4.c

SHLIB_MAJOR=  1
SHLIB_MINOR=  0

INCS=      mylib.h

.include <mkc.lib.mk>
```

В отличие от авторов `autotools` и других конкурирующих систем, разрабатывая `mk-configure`, я не ставил перед собой цели предоставить пользователю возможность сборки без привлечения сторонних утилит. Сторонние утилиты нужны. Это, во-первых, `bmake`, во-вторых `mk-files` для `bmake`-а (вариант из `pkgsrc`), на которых основана определенная функциональность `mk-configure`, и, в-третьих, сам `mk-configure`. С одной стороны, это недостаток. В `ВМАКЕ` отсутствует

(на момент написания этой статьи) в репозиториях подавляющего большинства Linux дистрибутивов, на сколько мне известно, единственным исключением является Fedora Linux. В Debian GNU/Linux и некоторых других дистрибутивах имеется пакет `rmake`, но он соответствует слишком старой версии NetBSD `make`-а. Отсутствует `bmake` и во FreeBSD/OpenBSD ports. То же относится к `mk-files` и, само собой, к `mk-configure`. Таким образом в настоящий момент для проектов, основанных на `mk-configure`, пользователю требуется провести некоторую предварительную работы. Но кто знает, возможно в будущем, ситуация изменится, и все перечисленные утилиты и программы появятся хотя бы в ведущих дистрибутивах. С другой стороны, я не вижу проблем в том, что процесс сборки требует сторонних утилит. Думаю, никому не приходит в голову переписывать GNU Makefile-ы на POSIX `make`. В мире открытого исходного кода “нестандартные” утилиты используются для сборки повсеместно, например, `bison`, `flex`, `pkg-config`, `imake` и многие-многие другие. Почему был выбран мало кому известный за пределами своей родной системы NetBSD `make`? Почему не гораздо более известный и распространенный GNU `make`? Во-первых, `bmake` обладает некоторыми возможностями, которых в GNU `make` просто нет. Прежде всего это касается директив `.for/.endfor`. Именно эта конструкция очень широко использовалась при реализации `mk-configure`. Во-вторых, для GNU `make` отсутствует (или я не знаю о его существовании) аналог `mk-files` для NetBSD `make`, на основе которой реализована значительная часть `mk-configure`. В-третьих, мне представляется, что `bmake` проще в использовании по сравнению с GNU `make`, при этом обладает по меньшей мере такими же возможностями. Ну и в-четвертых, мне больше нравится лицензия BSD, чем GNU GPL. Кстати, `mk-configure` выпущен под лицензией 2-clause BSD.

В данный момент проект находится в стадии разработки. Тем не менее, в его нынешнем виде он реализует все необходимые для полноценной разработки функции: декларативный способ описания проекта (минимум написанных вручную правил), определение размера произвольных типов, проверка декларации макроопределений, типов, переменных, функций, полей структур, проверка наличия реализации функции в заданных библиотеках, преобразование хорошо знакомых пользователям `autotools` @@-конструкций, например, `@prefix@` или `@bindir@`, результаты всех приведенных выше

проверок кешируются и могут использоваться при сборке других проектов, обеспечена поддержка кросс-сборки. Один из своих проектов я уже перевел с autotools на mk-configure – это lldb, легкий, но мощный отладчик обращений к памяти (malloc debugger, <http://sourceforge.net/projects/lldb>). Проект mk-configure полностью документирован (на английском языке), в архиве проекта (.tar.gz) имеется большое количество примеров. Напомню, страницу проекта.

<http://sourceforge.net/projects/mk-configure>.

Я думаю, мне удалось если не создать полностью завершённый проект, решающий все перечисленные выше проблемы, то, по крайней мере, решить большинство из них. Причем, сделано это минимальными усилиями. Объем исходных текстов – менее 2000 строк исходного кода, включая man-страницы и регрессионные тесты. Это на порядок меньше объема исходного кода autosconf и automake. Выбранное направление развития мне кажется правильным и перспективным. Кроме того, mk-configure вполне способен заставить разработчика взглянуть на привычные и “стандартные” вещи по-другому.