COLLABORATORS			
	TITLE :		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		2019-05-18	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Содержание

crosstool-NG

Yann E MORIN <yann.morin.1998@anciens.enib.fr> v1.0, 6 июня 2010: Исходный документ.

1 Введение

crosstool-NG предназначена для создания toolchains. Toolchains являются важным компонентом при разработке программного обеспечения. Она будет компилировать, собирать и связывать код, который в настоящее время разрабатывается. Некоторые части toolchain будут в конечном итоге превращаться в результирующий двоичный файл: статические библиотеки являются лишь примером.

Таким образом, набор инструментов является очень чувствительной частью программного обеспечения, так как любая ошибка в одном из компонентов, или плохо настроенный компонент, может привести к проблемам исполнения, начиная от низкой производительности, до неожиданного окончания приложения, неправильного поведения программного обеспечения (которые очень часто трудно обнаружить), повреждению оборудования, или даже человеческих рисков, (что наиболее прискорбно).

Toolchains сделаны из различных кусок программного обеспечения, каждый из которых довольно сложен и требует специально подобранных настроек для построения и работы без проблем на стыках. Обычно это не так легко, даже в случае не такой уж тривиальной нативной Toolchain. Разработка достигает более высокой степени сложности, когда речь заходит о кросс компиляции, где она может стать настоящим кошмаром…

Некоторые кросс Toolchains существуют в Интернете и могут использоваться для общего развития, но они имеют ряд ограничений:

- Они могут быть общего назначения, в том смысле, что они настроены для наиболее используемых вариантов: Без оптимизации для вашей конкретной цели,
- Они могут быть настроены для конкретной целевой платформы и таким образом, их будет не просто использовать для Вашей, так как они не оптимизированны или даже не поддерживают Вашу целевую платформу,
- Они часто используют устаревшие компоненты (компилятор, библиотеки C, и т.д…) не поддерживают специальные функции вашего новейшего процессора;

С другой стороны эти toolchain предоставляют ряд преимуществ:

- они готовы к использованию и их довольно легко установить и настроить,
- они доказали свою работоспособность, если используются большим сообществом.

Но после того, как вы захотите выжать все соки из вашего конкретного оборудования, вы захотите построить свой собственный toolchain. Здесь в игру вступает crosstool-NG.

Есть также ряд инструментов, которые строят Toolchains для конкретных потребностей, которые не являются действительно масштабируемыми. Примерами являются:

- buildroot (buildroot.uclibc.org), чья главная цель заключается в создании корневых файловых систем, отсюда и название. Но как только вы получите Ваш toolchain с помощью buildroot, часть его должна быть установлена в корень, так что если вы хотите построить совершенно новую корневую ФС, Вы либо должны сохранить существующий шаблон и восстановить его позже или перезапустить всё с нуля. Это не удобно,
- ptxdist (www.pengutronix.de/software/ptxdist), целью которого является нечто очень похожее на buildroot,

- другие проекты (openembedded.org например), которые опять-таки используются для построения корневых файловых систем.

crosstool-NG действительно ориентирован на построение Toolchains и только Toolchains. Именно для того, чтобы использовать её так, как вы хотите.

1.1 История

crosstool был впервые «задуман» Dan Kegel, который предложил его сообществу как набор скриптов, репозиторий патчей и несколько предварительно сконфигурированных файлов установки, общего назначения, которые должны использоваться для настройки crosstool. Это всё доступно на http://www.kegel.com/crosstool, и хранилище subversion размещается на google в http://code.google.com/p/crosstool/.

Однажды мне удалось добавить поддержку на основе uClibc Toolchains, но это не стало магистральным направлением, главным образом потому, что у меня не было времени на портирование патчей на новые версии, из-за больших усилий, которые эти патчи занимают.

Так что я решил прояснить состояние crosstool, в котором они были, изменив порядок вещей, добавив соответствующую поддержку, того, что мне было нужно, то есть поддержку uClibc и меню-ориентированную конфигурацию, назвал новую реализацию crosstool-NG, (объявляя crosstool следующего поколения, как это делают многие другие проекты и как в мульт-сериале «Звездный путь: Следующее поколение» ;-)) и сделали их доступными для сообщества, в случае, если они представляют интерес для всех.

1.2 Ссылки на crosstool-NG

Длинное имя проекта: crosstool-NG:

- без ведущих прописных (за исключением первого слова в предложении)
- crosstool и NG, разделенные дефисом (тире)
- NG в верхнем регистре

Crosstool-NG может также быть передано его короткое имя CT-NG:

- NG в верхнем регистре
- Crosstool и NG, разделенных дефисом (тире)

Длинное имя предпочтительнее краткого имени, за исключением subj в почте, где короткое имя подходит лучше.

Когда речь идет об определенной версии crosstool-NG, добавить информацию о версии либо номере, можно таким образом:

- crosstool-NG X.Y.Z
  - Длинное имя, пробел и строка версии
- crosstool-ng-X.Y.Z
  - Длинное имя в нижнем, дефис и строка версии
  - Такой формат используется для присвоения имени выпуска архивы
- crosstool-ng-X.Y.Z+hg_id

- Длинное имя в нижнем регистре, дефис, строка версии и идентификатор Hg (возвращается вызовом: ct-ng version)
- Этот формат используется для различения между релизами и snapshots.

Интерфейсом для crosstool-NG является команда ct-ng:

- все в нижнем регистре
- ct и ng, разделенных дефисом (тире)

2 Установка crosstool-NG

Существует два способа, как Вы можете использовать crosstool-NG:

- построить и установить его, а затем избавиться от исходных текстов, как Вы могли бы сделать для большинства программ,
- или только построить его и запустить из исходного каталога.

Первый способ следует использовать, если вы получили crosstool-NG из упакованного архива, смотрите «Метод установки», ниже, в то время как последний является наиболее полезным для разработчиков, которые используют клон репозитория и хотят отправить патчи, смотрите «Путь хакера», ниже.

2.1 Метод установки

Если Вы склоняетесь к установке, то просто следуйте классическим, но легким путём ./configure:

```
./configure  --prefix=/some/place
make
make install
export PATH=”${PATH}:/some/place/bin”
```

Затем, вы можете избавиться от исходного текста crosstool-NG. Далее создайте каталог, который будет служить рабочим местом, перейдите туда и запускайте:

```
ct-ng help
```

Смотрите ниже для полного использования.

2.2 Путь хакера

Если вы пойдете путем Хакера, то использование немного отличается, хотя тоже очень просто:

```
./configure --local
make
```

Теперь **НЕ** удаляйте исходы crosstool-NG. Они понадобятся для запуска crosstool-NG! Оставаясь в каталоге исходов запустите:

```
./ct-ng help
```

Смотрите ниже для полного использования.

Теперь если вы использовали клон репозитория, вы можете прислать мне ваши изменения. В разделе под названием ”Вклад в crosstool-NG”, ниже, написано о том, как представить изменения.

2.3 Подготовка для упаковки

Если Вы планируете упаковку crosstool-NG, Вы наверняка не хотите установить архивы в корень Вашей файловой системы. Процедура установки crosstool-NG отличается DESTDIR переменной:

```
./configure --prefix=/usr
make
make DESTDIR=/packaging/place install
```

2.4 Коды завершения shell для crosstool-NG

crosstool-NG поставляется с фрагментом скрипта shell, который определяет bash-совместимые коды завершения. Этот фрагмент shell в настоящее время не устанавливается автоматически, но это запланировано.

Чтобы установить фрагмент shell скрипта, у вас есть два варианта:

- Установив на всю систему, полностью, скопировав ct-ng.comp в /etc/bash_completion.d/
- Установка для одного пользователя, путем копирования ct-ng.comp в ${HOME}/ и включив этот файл в Ваш ${HOME}/.bashrc

2.5 Вклад в код

Некоторые люди вложили код, который невозможно объединить по различным причинам. Этот код доступен в виде lzma-сжатых patches, в подкаталоге contrib/. Эти патчи должны быть применены к исходным текстам crosstool-NG, перед установкой, используя нечто вроде следующего:

```
lzcat contrib/foobar.patch.lzma |patch-p1
```

Нет никакой гарантии, что конкретный вклад применяется к текущей версии crosstool-ng, или что он будет работать на всех. Используйте вклады на свой собственный риск.

3 Настройка crosstool-NG

crosstool-NG настраивается с помощью конфигуратора, представляющего структуру меню для установки значений параметров. Эти параметры позволяют указать, путь построения Вашего toolchain как Вы хотите , где Вы хотите его установить, какие архитектуры и конкретные процессоры, он будет поддерживать, версии компонентов, которые вы хотите использовать, и т.д.... Значения для этих параметров, затем сохраняются в файле конфигурации.

Конфигуратор работает аналогично настройщику конфигурации ядра Linux. Предполагается, что Вы знаете, как справиться с этим.

Чтобы войти в меню, введите следующую команду:

```
ct-ng menuconfig
```

Почти каждый элемент config имеет справку. Прочитайте их внимательно.

Строковые и числовые опции могут ссылаться на переменные среды. В таком случае необходимо использовать синтаксис shell: ${VAR}. Вы должны не использовать ни одно, ни двойных кавычек в опциях.

Существует три переменные среды, которые вычисляются в crosstool-NG, и Вы можете их использовать:

CT_TARGET Представляет собой кортеж цели, для которой создаётся. Можно использовать его например в каталоге installation/prefix, таким образом: /opt/x-tools/${CT_TARGET}

CT_TOP_DIR Верхняя директория, в которой работатет crosstool-NG. Вам не нужно это в большинстве случаев. Существует один случай, когда вам может понадобиться это: если у Вас используются локальные patches, и хранить их нужно в рабочей папке, Вы можете обратиться к ним с помощью CT_TOP_DIR, таким образом: ${CT_TOP_DIR}/patches.myproject

CT_VERSION Версия crosstool-NG, которую вы используете. Не так уж нужно Вам, но используется если Вам это нужно.

3.1 Интересные опции конфигурации

CT_LOCAL_TARBALLS_DIR Если у вас уже есть некоторые архивы исходов, в некотром каталоге, введите его имя здесь. Это позволит ускорить фазу извлечения, так как в противном случае crosstool-NG будет загружать эти архивы.

CT_PREFIX_DIR Это куда toolchain будет устанавливаться в (и где он будет выполняться позже). Обычно используется для добавления кортежа цели в путь к каталогу, например (см. выше): /opt/x-tools/${CT_TARGET}

CT_TARGET_VENDOR Идентификатор для вашей toolchain, будет передаваться в поле ”поставщик” как часть кортежа цели. Он **НЕ** содержат пробелы или дефисы. Как правило сохраняйте его в строке из одного слова, или используйте знаки подчеркивания для разделения слов, если Вам нужно. Избегайте точки, запятые и специальные символы.

CT_TARGET_ALIAS Псевдоним для toolchian. Он будет использоваться в качестве префикса к средствам toolchain. Например, вы будете иметь ${CT_TARGET_ALIAS}-gcc. Кроме того если Вы думаете, что вы не видите достаточно версий, Вы можете попробовать разрешить одно из:

CT_OBSOLETE Показывает устаревшие версии или инструменты. В большинстве случаев, Вы не захотите строить ваш toolchain на слишком старых версиях (gcc, например). Но иногда, может случиться так, что сподручнее использовать старую версию для тестов регрессии. Эти старые версии скрыты в CT_OBSOLETE. Эти версии (особенности) помечены так потому, что поддержание их является слишком дорогостоящим, отнимая время и деньги у тех, кто поддерживает crosstool-NG.

CT_EXPERIMENTAL Показывает экспериментальные версии или инструменты. Опять же, может и не нужно ориентировать свой набор инструментов на самые последние версии (например, gcc). Но, если нужная Вам функция присутствует только в последней версии, или новейшем инструменте, Вы можете найти их в CT_EXPERIMENTAL. Эти версии (или функции) не прошли (пока) тщательного тестирования в crosstool-NG, и/или не являются достаточно зрелыми для того, чтобы слепо им доверять.

3.2 Пересборка существующих toolchain

Если у вас есть существующая toolchain, вы можете повторно использовать параметры, используемые для её построения, чтобы создать новый toolchain. Это потребует немного усилий с вашей стороны, но зато довольно легко. Параметры для построения toolchain сохраняются в самом toolchain, и Вы можете получить эту конфигурацию, выполнив:

```
${CT_TARGET}-ct-ng  .config
```

Альтернативный метод заключается в том, чтобы извлечь конфигурацию из файла build.log. Это будет необходимо, если ваш toolchain был построен с crosstool-NG до 1.4.0, но может быть использован с файломи build.log из любой версии:

```
ct-ng extractconfig <build.log >  .config
```

Или, если ваш build.log файл сжат (наиболее вероятно!):

```
bzcat build.log.bz2 |ct-ng extractconfig > .config
```

Вышеприведённая команда будет выдавать конфигурацию в stdout, так что просто перенаправив вывод в файл .config, восстановим набор инструментов с помощью этой конфигурации:

```
${CT_TARGET}-ct-ng.config >.config
ct-ng oldconfig
```

Затем можно просмотреть и изменить конфигурацию, выполнив:

```
ct-ng menuconfig
```

3.3 Использование в качестве backend для построения системы

Crosstool-NG может использоваться в качестве backend для автоматизированной сборки системы. В этом случае, некоторые компоненты, которые, как ожидается, будут выполняться на целевом компьютере (например, нативные gdb, ltrace, DUMA…) не доступны в menuconfig, и они не будут собраны вообще, либо будут построены под ответственность системы построения, в качестве своих собственных версий этих средств.

Если вы хотите использовать crosstool-NG как бэкэнд для создания ваших toolchains для вашей системы построения, необходимо установить и экспортировать эту переменную среды:

```
CT_IS_A_BACKEND=y
```

(Регистр не чувствителен, вы можете сказать, Y).

4 Построение toolchain

Чтобы построить toolchain, просто наберите:

```
ct-ng build
```

Это позволит использовать ранее опредлённую конфигурацию для извлечения, наложения патчей на компоненты, сборки, установки и, в конечном итоге, тестирвания только что построенной Вами toolchain.

Затем Вы можете добавить toolchain/bin директории в Вашу переменную PATH и свободно использовать её.

В любом случае вы можете получить некоторую лаконичную помощь. Просто наберите:

```
ct-ng help
```

или:

```
man 1 ct-ng
```

4.1 Остановка и перезапуск построения

Если вы хотите остановить построение после некоторого шага, при отладке, можно передать переменную STOP, чтобы это сделать:

```
ct-ng build STOP=some_step
```

И наоборот Если вы хотите перезапустить построение с конкретного шага при отладке, можно передать переменную RESTART, что бы это сделать:

```
ct-ng build RESTART=some_step
```

Кроме того, вы можете вызвать make с именем шага, что бы просто выполнить этот шаг:

```
ct-ng libc_headers
```

эквивалентна:

```
ct-ng build RESTART=libc_headers STOP=libc_headers
```

Сочетания клавиш + step_name и step_name + позволяют соответственно остановить или перезапустить с этого шага. Таким образом:

```
ct-ng +libc_headers              and:    ct-ng libc_headers+
```

эквивалентно:

```
ct-ng build STOP=libc_headers    and:    ct-ng build RESTART=libc_headers
```

Чтобы получить список приемлемых шагов, вызовете:

ct-ng list-steps

Обратите внимание, что для того, чтобы перезапустить построение, вам придется сказать *Y* для параметра config CT_DEBUG_CT_SAVE_STEPS, и что предыдущее построение действует до текущего момента.

4.2 Строительство всех цепей сразу

Вы можете построить все примеры; просто вызовите:

```
ct-ng build-all
```

4.3 Переопределение количества / / jobs

Если вы хотите переопределить количество заданий для запуска в / / (-j опция make), можно повторно ввести menuconfig, или просто добавить его в командной строке, таким образом:

```
ct-ng build.4
```

которая говорит crosstool-NG переопределить количество / / jobs до 4. Вы можете увидеть действия, которые поддерживаются при задании разного количества в / / jobs, в меню Справка. Т.е. с окончанием [. #] после них (например. build [. #] или build-all [. #], и так далее...).

4.4 Замечания по поводу / / jobs

Crosstool-NG сценарий «ct-ng» является сценарием Makefile. Он **НЕ** выполняется параллельно (есть немного, для увеличения). Говоря о // jobs, мы имеете в виду количество // jobs при выполнении **компоненты**. То есть, мы говорим о количестве / / jobs используемом для построения gcc, glibc и так далее...

4.5 Инструменты оболочки

Начиная с gcc-4.3 появляются две новые зависимости: GMP и MPFR. С gcc-4.4 приходят три новых: PPL, CLooG/ppl и MPC. С gcc-4.5 появляется новая зависимость от libelf. Это библиотеки, которые включают расширенные функции в gcc. Кроме того некоторые из этих библиотек могут использоваться binutils и gdb. К сожалению не все системы, на которых выполняется crosstool-NG имеют все эти библиотеки. И для тех, кто работает с версиями этих библиотек, которые могут быть старше, чем версия gcc (и binutils и gdb) . На сегодняшний день, Debian стабильный (aka Lenny) отстает на некоторых библиотеках, и отсутствуют другие.

Вот почему crosstool-NG строит свой собственный набор библиотек в рамках toolchain.

Комплектные библиотеки могут быть построены как статические библиотеки, или как разделяемые библиотеки. По умолчанию строятся статические библиотеки и это - безопасным способ. Если вы решите использовать статические комплектные библиотеки, то Вы можете далее не читать этот раздел.

Но если вы предпочитаете разделяемые библиотеки, то читайте дальше...

Построение разделяемых комплектных библиотек не создает никаких проблем во время построения, так как crosstool-NG правильно указывает gcc (и binutils и gdb) на место, где установлены библиотеки нашей собственной версии. Но это создает проблемы при запуске gcc (и других): место, где библиотеки наиболее вероятно расположены, не известно динамическому компоновщику на хосте. Что еще хуже, если на хост-системе есть свои собственные версии, тогда ld.so будет загружать не те библиотеки!

Поэтому мы должны заставить динамический компоновщик, загружать правильную версию. Мы делаем это с помощью переменной, LD_LIBRARY_PATH которая сообщает динамическому компоновщику, где искать разделяемые библиотеки до поиска в стандартных местах. Но мы не можем навязывать это бремя на всю систему (потому что это будет кошмаром для настройки, и потому, что две toolchains на одной и той же системе могут использовать различные версии библиотек); Поэтому мы должны делать это отдельно для каждой toolchain.

Поэтому мы переименовали все двоичные файлы toolchain (путем добавления точки . как их первый символ) и добавили небольшую программу, так называемый «tools wrapper», которая правильно устанавливает LD_LIBRARY_PATH перед запуском конкретного инструмента.

Во-первых оболочка была написана как POSIX-совместимые shell сценарий. Этот shell сценарий очень прост, если не тривиален и прекрасно работает. Единственным недостатком является то, что он не работает на хост-системах, которые не имеют shell, например окружающая среда MingW32. Для решения проблемы, оболочки был переписать в C и скомпилированы во время построения. Эта обертка на C гораздо более сложная, чем shell сценарий и хотя она выглядит работающей, она лишь слегка тестировалась. Некоторые из ожидаемых недостатков такой оболочки на C являются;

- имена файлов в мультибайтной кодировке не могут быть обработаны правильно
- Это действительно много, для того, что он делает

Таким образом оболочки по умолчанию, установленные с вашей toolchain является shell сценарий. Если вы знаете, что на Вашей системе нет shell, то Вы должны использовать оболочку C (и сообщить, работает ли он у Вас, или не работает).

Последнее замечание по этому вопросу: не создавать разделяемые библиотеки. Строя их статически, Вы будете в безопасности.

5 Использование toolchain

Использование toolchain также просто, как добавление каталога toolchain/bin в Ваш путь, таким образом:

```
export PATH=”${PATH}:/your/toolchain/path/bin”
```

а затем с помощью кортежа цели указать системе построения использование Вашей toolchai:

```
./configure --target=your-target-tuple
```

или:

```
make CC=your-target-tuple-gcc
```

или:

```
make CROSS_COMPILE=your-target-tuple-
```

И так далее...

Настоятельно рекомендуется не использовать каталог toolchain sys-root в качестве каталога установки для пакетов программ. Если вы это сделаете, вы не сможете использовать вашу toolchain для другого проекта. Настоятельно рекомендуется, сделать Вашу toolchain только для чтения с помощью chmod, после успешного построения, так что Вы не сможете загрязнить Вашу toolchain Вашими программами/пакетами.

Таким образом когда Вы строите пакет программ, установите его в отдельный каталог, например ./ your/root. Этот каталог является образом того, что будет в корневой файловой системе Вашей целевой платформы, и будет содержать все, что необходимо для установки Вашей программы/пакета.

5.1 Сценарий «заполнить»

Когда Ваш корневой каталог будет готов, в нём все еще отстутствуют некоторые важные детали: библиотеки toolchain. Чтобы поместить в Ваш корневой каталог эти библиотеки, просто запустите:

```
your-target-tuple-populate -s /your/root -d /your/root-populated
```

Он будет копировать /your/root в /your/root-populated и поместит туда только необходимые библиотеки. Таким образом вы не загрязните /your/root всяким хламом, который больше не требуется, и Вам не нужно удалять его вручную. /your/root всегда содержит только те вещи, которые Вы установите в нем.

Затем можно использовать /your/root-populated для создания Вашего образа файловой системы, архивировать его, или смонтировать его по NFS на Вашей целевой машине, или все, что Вам понадобится.

Сценарий заполнения принимает следующие параметры:

- -s src_dir использовать «src_dir» как не-заполняемый корневой каталог.
- -d dst_dir Заполняемый корневой каталог поместить в «dst_dir».
- -l lib1 [...] Всегда добавлять указанные библиотеки.
- -L файл Всегда добавлять библиотеки, перечисленные в «файл».

- -f Удалить «dst_dir» если существовали ранее; Продолжать, даже если любая библиотека, указанный с -l или -L отсутствует.
- -v Быть многословным и показывать, что происходит (вы можете увидеть точно какая libs откуда).
- -h Распечатка помощи.

Смотрите «your-target-tuple-populate - h» для получения дополнительной информации о параметрах.

Здесь описывается, как работает заполнение:

1. Выполняется ряд проверок исправности:
   - src_dir и dst_dir, указаны
   - src_dir существует
   - dst_dir не существует, если не установлен forced
   - src_dir != dst_dir
2. Копирует src_dir в dst_dir
3. Добавляет принудительно устанавливаемые библиотеки в dst_dir
   - Строит список с помощью опций -l и -L
   - Получает принудительно устанавливаемые библиотеки от sysroot (эвристики см. ниже)
     - Прерывает процес на первой же отсутствующей библиотеке, если -f не указан
4. Добавляет все недостающие библиотеки в dst_dir
   - Сканирует dst_dir для каждого ELF-файла, которые являются «исполняемыми» или «разделяемыми» объектами
   - Заполняет поля списка «нужных Shared библиотек»
     - Проверяет, если ли библиотека уже в dst_dir/lib или dst_dir/usr/lib
     - Если нет, то берёт библиотеку от sysroot
       * Если она расположена в sysroot/lib, копирует её в dst_dir/lib
       * Если она в sysroot/usr/lib, копирует её в dst_dir/usr/lib
       * В обоих случаях использует SONAME этой библиотеки для создания файла в dst_dir
       * Если библиотека не была найдена в sysroot, это приводит к сообщению об ошибке.

6 Типы toolchain

Существует четыре вида toolchain, с которыми Вы могли бы столкнуться. Во-первых, Вы должны понять следующее: когда речь заходит о компиляторах, то там до четырех машин:

1. Машина настройка компонентов toolchain: Компьютер конфигурации
2. Машина для построения компонентов toolchain: Компьютер для построения toolchain
3. Компьютер, на котором работатет toolchain: Компьютер хоста
4. Компьютер, для которого toolchain генерирует код: Целевой компьютер

Мы можем, в большинстве случаев, предположить, что копьютеры конфигурации и построения - одно и тоже. Почти всегда, это будет верно. Единственный случай, когда это не так, это если Вы используете распределенную компиляцию (например, distcc). Для простоты, давайте пока забудем об этом.

Поэтому мы остались с тремя машинами:

- Построения
- Хост
- Целевая

Любой набор инструментов будет включать эти три машины. Вы можете быть абсолютно уверены в этом, как в том, что 2 + 2 = 4. Вот как они взаимодействуют:

1) **build == host == target** Это нативный toolchain, предназначенный для точно той же машины, что и та, на которой он построен и работает на этой же машине. Вы должны построить такой toolchain, когда вы хотите использовать обновленный компонент, например новые gcc. crosstool-NG называет его «нативный».
2) **build == host != target** Это классический кросс-toolchain, который должен быть запущен на той же машине, на которой он был скомпилирован и создаёт код для выполнения на второй машине, целевой. crosstool-NG называет его «cross».
3) **build != host == target** Такой toolchain также является нативным toolchain, так как его целевой машиной является та, на которой он работает. Но он был построен на другом компьютере. Вы захотите такой инструмент при переносе на новую архитектуру, или если компьютер построения гораздо быстрее, чем машина хоста. crosstool-NG называет его «кросс нативный».
4) **build != host != target** Этот вариант называется канадский toolchain (*) и он достаточно сложен. Три машины в этом процессе различаются. Вам может потребоваться такой toolchain, если у вас есть машина быстрого построения, но пользователи будут использовать его на другом компьютере и будет производить запуск кода на третьем компьютере. crosstool-NG называет его «канадским».

crosstool-NG может построить все эти виды toolchain (В любом случае, он нацелен на это!)

(*) Термин Канадский Крест произошло потому, что в то время, когда эти вопросы обсуждались, в Канаде было три национальных политических партии. http://EN.wikipedia.org/wiki/Cross_compiler

7 Вклад в crosstool-NG

7.1 Отправка отчета об ошибке

Если вам нужно отправить отчет об ошибке, пожалуйста, отправьте письмо с темой, именующей префикс «[CT_NG]» по следующим адресам:

Example 7.1 Заполнение заголовка письма

```
TO: yann.morin.1998 (at) anciens.enib.fr
CC: crossgcc (at) sourceware.org
```

7.2 Отправка патчей

Если вы хотите улучшить crosstool-NG, есть список в файле TODO.

Патчи должны прийти с соответствующей строкой SoB. SoB строка обычно является чем-то вроде:

Example 7.2 Строка подписи

Signed-off-by: John DOE <john.doe@somewhere.net>

Строки SoB детально описаны в Documentation/SubmittingPatches, в разделе 12, Вашего любимого дерева исходников ядра Linux.

Для более крупных или более частых вкладов должны использоваться mercurial. В разделе «C» есть приятный, полный и пошаговый учебник.

8 Внутреннее устройство

Внутренне crosstool-NG, основан на скрипте. Для простоты использования, интерфейс сделан на основе Makefile.

8.1 Makefile интерфейс

Точкой входа для crosstool-NG является Makefile скрипт «ct-ng». Вызвав этот сценарий с ”действием” даёт точно такой эффект, как если бы в текущем рабочем каталоге имелся Makefile и make был вызван с целью ”действие” как правилом из Makefila. Таким образом:

```
ct-ng menuconfig
```

Эквивалентно наличию Makefile в текущей директории и вызову:

```
make menuconfig
```

Существующий ct-ng избегает копирования Makefile везде и действует как традиционные команды.

ct-ng загружает sub-Makefile из директории $(CT_LIB_DIR), как и во время настройки с помощью ./configure.

ct-ng также ищет конфигурационные файлы, вложенные инструменты, примеры, сценарии и исправления в этом каталоге библиотеки.

Из-за тупого поведения make, я не смог отследить ошибки, неявные правила make отключены: Установка с ключом --local будет запускать эти правила, и mconf было невозможно построить.

8.2 Парсер Kconfig

Язык kconfig является взломанной версией, высосаной из ядра Linux (http://www.kernel.org/) и (сильно) адаптирована к моим потребностям.

Список наиболее заметных изменений (по крайней мере те, которые я помню) выглядит следующим образом:

- префикс CONFIG_ была заменена на CT_
- Ведущий | в промпте пропускается, а последующие начальные пробелы, не удаляются; в противном случае пробелы удаляются молча

- удалено предупреждение о неопределенной переменной окружения

Kconfig парсеры (conf и mconf) не установлены заранее, но построены из исходных файлов. Таким образом вы можете иметь каталог, где установлен crosstool-NG, который экспортируется (через NFS или както иначе) и клиенты с другой архитектурой могут использовать одну и ту же установку crosstool-NG, и прежде всего, тот же набор патчей.

8.3 Архитектурные зависимости

Примечание: эта глава не очень хорошо написана и может быть немного сложна для понимания. Чтобы лучше понять то, что такое архитектура, читателю любезно предлагается взглянуть на подкаталог ”arch/” и на существующие архитектуры, чтобы увидеть, как они там представлены.

Архитектура определяется:

- Понятным человеку именем, записанном строчными буквами, и числами в случае необходимости. Допускается символ подчеркивания; пробел и специальные символы не допускаются. Например.: arm, x86_64
- файлом в каталоге «config/arch/», который назван по имени архитектуры и с суфиксом «.in». Например: config/arch/arm.in
- файлом в «scripts/build/arch/», назван по имени архитектуры и с суфиксом «.sh». Например: scripts/build/arch/arm.sh

8.3.1 Архитектура «.in» файла API:

Параметр конфигурации «ARCH_ %arch%» (где %arch% — заменяется фактическим именем архитектуры). Параметр config не должен иметь **НИ** типа, **НИ** промпта! Кроме того, он может **НЕ** зависеть от любых других опций конфигурации (EXPERIMENTAL управляются как указано выше).

Example 8.1 Задать архитектуру

```
config ARCH_arm
```

- обязательные опции: Определить запись (сжатой) справки для этой архитектуры:

ARCH_arm

```
config ARCH_arm
  help
    The ARM architecture.
```

- Дополнительные опции: Выберите адекватно связанные опции конфигурации. Примечание: на 64-битных архитектурах Вы **ДОЛЖНЫ** выбирать опцию ARCH_64

ARC_arm

```
config ARCH_arm
  select ARCH_SUPPORTS_BOTH_ENDIAN
  select ARCH_DEFAULT_LE
  help
     The ARM architecture.
```

ARCH_x86_64:

```
config ARCH_x86_64
  select ARCH_64
  help
    The x86_64 architecture.
```

- другие целевые варианты по вашему усмотрению. Однако, обратите внимание, что чтобы избежать конфликтов имён, такие опции должны иметь префикс «ARCH_ %arch%», где %arch% снова заменяется на фактическое имя архитектуры. (Примечание: из-за исторических причин и нехватки времени для очистки кода, я возможно, оставили некоторые параметры конфигурации, которые не полностью соответствуют тому, что имя архитектуры должно быть написано только заглавными. Однако префикс является уникальным в архитектуре и не наносят вреда).

8.3.2 Архитектура файла «.sh» API:

8.3.2.1 Функция «CT_DoArchTupleValues»

- параметры: нет
- Переменные окружающей среды:
  - все переменные из файла «config»,
  - две переменные «target_endian_eb» и «target_endian_el», которые являются суффиксами порядка байтов
- Возвращаемое значение: 0 в случае успеха,! 0 при неудаче
- предоставляет:
  - обязательно
    * переменную среды CT_TARGET_ARCH
  - содержит: часть архитектуры кортежа целевого объекта. Например.: «armeb» для big endian ARM и «i386» для i386
- пердоставляет:
  - Необязательно
    * переменную среды CT_TARGET_ARCH
  - содержит: часть sytem кортежа целевого объекта. Например.: «gnu» для glibc на большинстве архитектур «gnueabi» для glibc на ARM EABI
  - по умолчанию:
    * для основанной на glibc toolchain: ”gnu”
    * для основанных на uClibc toolchain: ”uclibc”
- предоставляет:
  - Необязательно
  - переменные среды для настройки кросс gcc (по умолчанию)
    * CT_ARCH_WITH_ARCH: gcc ./configure переключатель для выбора уровня архитектуры («--with-arch=${CT_ARCH_ARCH}»)
    * CT_ARCH_WITH_ABI: gcc ./configure переключатель для выбора уровня ABI («--with-abi = ${CT_ARCH_ABI}»)

    * CT_ARCH_WITH_CPU: gcc ./configure переключатель для выбора набора инструкций процессора (”--with-cpu = ${CT_ARCH_CPU}»)
    * CT_ARCH_WITH_TUNE: gcc ./configure переключатель для выбора планирования («--with-tune = ${CT_ARCH_TUNE}»)
    * CT_ARCH_WITH_FPU: gcc ./configure переключатель для выбора типа FPU («--with-fpu = ${CT_ARCH_FPU}»)
    * CT_ARCH_WITH_FLOAT: gcc ./configure переключатель для выбора плавающей точкой арифметика («--with-float= soft» или /empty/)

- предоставляет:
  - опционально - переменные среды для передачи кросс gcc для создания целевого двоичные файлы (по умолчанию)
    * CT_ARCH_ARCH_CFLAG: gcc переключатель для выбора уровня архитектуры («-march=${CT_AR
    * CT_ARCH_ABI_CFLAG: gcc переключатель для выбора уровня ABI (”-mabi= ${CT_ARCH_ABI}»)
    * CT_ARCH_CPU_CFLAG: gcc переключатель для выбора набора инструкций процессора («-mcpu=${CT_ARCH_CPU}»)
    * CT_ARCH_TUNE_CFLAG: gcc переключатель для выбора планирования (”-mtune = ${CT_ARCH_T
    * CT_ARCH_FPU_CFLAG: gcc переключатель для выбора типа FPU (”-mfpu = ${CT_ARCH_FPU}»)
    * CT_ARCH_FLOAT_CFLAG: gcc переключатель для выбора арифметики с плавающей точкой («- msoft-float» или /empty/)
    * CT_ARCH_ENDIAN_CFLAG: gcc переключатель для выбора big или little порядка байт («-mbig-endian» или «- mlittle-endian»)
  - по умолчанию: Смотрите выше.

- обеспечивает:
  - Необязательно
  - Переменные окружения для настройки ядра и окончательный компилятор, специфичный для этой архитектуры:
    * CT_ARCH_CC_CORE_EXTRA_CONFIG: дополнительные, архитектура конкретное ядро gcc. / configure флаги
    * CT_ARCH_CC_EXTRA_CONFIG: дополнительные, архитектура конкретные gcc. / configure флаги
    * по умолчанию:
    * все пустые

- предоставляет:
  - Необязательно
    * архитектурно зависимые CFLAGS и LDFLAGS:
    * CT_ARCH_TARGET_CLFAGS
    * CT_ARCH_TARGET_LDFLAGS
  - по умолчанию:
    * все пустые

Вы можете взглянуть на «config/arch/arm.in» и «scripts/build/arch/arm.sh», что бы увидеть, как выглядит довольно полный пример описания фактической архитектуры.

8.4 Особенности ядра

Ядро определяется псредством:

- Понятное человеку имя, строчными буквами, с числами в случае необходимости. Допускается символ подчеркивания; пробелы и специальные символы, не допускаются (хотя внутренне они заменяются символами подчеркивания.) Например.: linux, bare-metal
- файл в каталоге «config/arch/», назван по имени архитектуры и с суфиксом «.in».

Файлы ".in"

```
config/kernel/linux.in
config/kernel/bare-metal.in
```

- файл в «scripts/build/arch/», назван по имени архитектуры и с суфиксом «.sh».

Файлы ".sh"

```
scripts/build/kernel/linux.sh
scripts/build/kernel/bare-metal.sh
```

8.4.1 Файл ядра «.in» должен содержать:

- Дополнительные строки, содержащие в точности слово «#EXPERIMENTAL», начиная с первого столбца и без каких-либо последующих пробелов или других знаков. Если эта строка присутствует, то это ядро считается ЭКСПЕРИМЕНТАЛЬНЫМ, и будут заданы правильные зависимости.
- параметр конфигурации «KERNEL_%kernel_name%» (где %kernel_name% будет заменено именем фактического ядра, с использованием всех специальных символов и пробелы будут заменены на символы подчеркивания). Параметр config не должен иметь **НИ** типа, **НИ** промпта! Кроме того, он **не** может зависить от EXPERIMENTAL. Например.: KERNEL_linux, KERNEL_bare_meta
  - обязательно: Определить запись (сжатой) справки для этой архитектуры:

KERNEL_bare_metal:

```
config KERNEL_bare_metal
  help
    Создание компилятора для использования без какого-либо ядра.
```

- Дополнительные: Выбор адекватно связанных опций конфигурации.

KERNEL_bare_metal:

```
config KERNEL_bare_metal
select BARE_METAL
  help
    Создание компилятора для использования без какого-либо ядра.
```

- другие целевые варианты по вашему усмотрению. Однако, обратите внимание, что, для того, чтобы избежать конфликтов имён, такие опции должны начинаться с «KERNEL_%kernel_name%», где %kernel_name% — заменяются именем фактического ядра. (Примечание: из-за исторических причин и нехватки времени для очистки кода, я возможно, оставили некоторые параметры конфигурации, которые не полностью соответствуют тому, что имя ядра должно быть написано только заглавными буквами. Однако префикс является уникальным в архитектуре и не наносят вреда).

8.4.2 API файлов ядра «.sh»:

- Является фрагментом скрипта bash
- Определяет функцию CT_DoKernelTupleValues
  - Просматривает CT_DoArchTupleValues архитектуры, за исключением:
  - Устанавливает переменную среды CT_TARGET_KERNEL, как часть кортежа цели, задающую ядро.
  - Возвращаемое значение: игнорируется
- Определяет функцию CT_DoKernelTupleValues
  - параметры: нет + переменные окружающей среды: - все переменные из файла «.config»,
  - Возвращаемое значение: 0 в случае успеха, ! 0 при неудаче
  - поведение: скачать исходники ядра и сохранить архив в «${CT_TARBALLS_DIR}». С этой целью функции доступны доступны для загрузки архивы:
    * CT_DoGet <tarball_base_name> <URL1 [URL...]>

CT_DoGet

```
CT_DoGet linux-2.6.26.5 ftp://ftp.kernel.org/pub/linux/kernel/v2.6
```

```
Примечание: получение исходников из svn, cvs, git и подобного не поддерживается ↩
    CT_DoGet. Вам придется делать это вручную, как это делается для eglibc в ↩
    «scripts/build/libc/eglibc.sh»
```

- Определяет функцию «do_kernel_extract»:
  - параметры: нет
  - Переменные окружающей среды:
    * все переменные из файла «.config»,
  - Возвращаемое значение: 0 в случае успеха, ! 0 при неудаче
  - поведение: Распаковывает архив ядра в ”${CT_SRC_DIR}” и применяет необходимые патчи. С этой целью функции доступны для загрузки архивы:
    * CT_ExtractAndPatch <tarball_base_name> .CT_ExtractAndPatch:

```
CT_ExtractAndPatch linux-2.6.26.5
```

- Определяет функцию «do_kernel_headers»:
  - параметры: нет
  - Переменные окружающей среды:
    * все переменные из файла «.config»,
  - Возвращаемое значение: 0 в случае успеха, ! 0 при неудаче
  - поведение: установить заголовки ядра (если таковые имеются) в «${CT_SYSROOT_DIR}/usr/include
- определяет любые зависимые от ядра вспомогательные функции. Эти функции, если таковые имеются, должны быть с префиксом «do_kernel_%CT_KERNEL%_», где «%CT_KERNEL%» должен быть заменен именем фактического ядра, чтобы избежать конфликтов имён.

Вы можете посмотреть на «config/kernel/linux.in» и «scripts/build/kernel/linux.sh» на пример того, как выглядит сложное описание ядра.

8.4.3 Добавление новой версии компонента

Когда новый компонент, например ядро Linux, gcc или любой другой освобождается, добавление новой версии в crosstool-NG выполняется довольно легко. Существует сценарий, который будет делать все это для вас:

```
scripts/addToolVersion.sh
```

Запустите его без опций, что бы получить подсказку.

8.5 Создавать сценарии

Будет написана позже...

9 Список разработчиков

Я хотел бы поблагодарить этих прекрасных людей за то, что они сделали возможным создание crosstool-NG:

Dan KEGEL, прервоначальный автор crosstool: http://www.kegel.com/ Дэн был очень полезным и всегда был готов помочь, когда я делал мой первый toolchains. Я задолжаю ему одному. Dan Спасибо! Некоторые crosstool-NG скрипты имеют фрагменты кода, происходящие почти от оригинальной работы Дэна.

И в порядке появления на crossgcc ML:

Аллан Кларк За его исследования по созданию toolchains на MacOS X. Аллан сделали обширные испытания первой альфа crosstool-NG на его MacOS X и представила некоторые странности bash-2.05.

Энрико ВАЙГЕЛЬТ

- некоторые улучшения в процедуре сборки
- cxa_atexit запрещённая в библиотеках C не поддерживающих её (старый uClibc)
- Прочие предложения (Перезапускаемое построение,...)
- чтобы избавиться от некоторых ”башизмов” в ./configure
- Поддержка OpenRISC or32

Роберт П. Ж. Дэй

- Некоторые небольшие усовершенствования конфигуратор, разнообразное предупреждение небольших проблем
- «sanitised» патчи для binutils-2.17
- патчи для glibc-2.5
- Разные патчи, typos и eye candy
- Слишком много, чтобы перечислить всё!

Al Stone

- Первоначальная поддержка ia64
- Некоторые косметика

Szilveszter Ordog

- Исправление плавающей точки в uClibc

- Начальная поддержка для ARM EABI

Марк Джонас

- Инициатор портирования на Super-H

Майкл Эббот

- Сделал это, построив из древних findutils

Вилли Tarreau

- патч для glibc постренный на «древних» shell
- Сообщил о неправильном использовании $CT_CC_NATIVE

Маттиас Kaehlcke

- исправил построение glibc-2.7 (и 2.6.1) с новыми ядрами

Даниэль Dittmann

- Поддержка PowerPC

Иоаннис E. Венетис

- Предварительная поддержка Alpha
- интенсивной мозговой штурм gcc-4.3

Томас Jourdan

- интенсивной мозговой штурм gcc-4.3
- Поддержка eglibc

Многое другое было внесено, либо в виде патчей, предложений, комментариев или тестирования… Спасибо всем вам!

Специальные посвящение людям из buildroot за поддержание набор патчей, которых я счастливо и бесстыдно вампирил их время от времени… :-)

20100530: Статус этого файла

Уже около года, как мы переехали в репозиторий Mercurial. Репозиторий теперь имеет надлежащее авторства для каждого набора изменений, и это используется для создания изменений в каждом выпуске. Этот файл, сорее всего, не будет обновляться и передоверяем его людям, которые выполнят миграцию на Mercurial, или для обсуждения идей, или иным образом помогая без кода.

Если вы думаете, что вы заслуживаете, упоминания в этом файле, сообщите мне! ;-)

10 Известные проблемы

В этих файлах перечислены известные проблемы, возникающие при разработке crosstool-NG, но которые не могут быть решены до выхода очередного релиза.

Файл имеет по одной секции для каждой из известных проблем, каждая секция содержащий четыре подраздела: Симптомы, разъяснения, исправления и обход.

Каждая секция отделена от других линиями, по крайней мере в 4 тире длиной.

Следующая секция поясняет все на макете секции.

Симптомы Одна строка из того, что Вы увидите в выдаче

Пояснения Как можно более углубленное объяснение контекста, почему это происходит, что исследовано, насколько и возможные ориентиры, как попытаться решить эту проблему (например. URL-адреса, фрагменты кода...).

Исправьте Что нужно сделать, чтобы исправить ошибку, если это вообще возможно. Тот факт, что существует исправление, и пока это проблема известна, означает, что время для включения исправления в crosstool-NG был упущено, или планируется в будущем выпуске.

Обход проблемы Что нужно сделать, чтобы исправить ошибку, если это вообще возможно. Обходной путь не является реальным исправлением, так как это может нарушить работу других частей crosstool-NG, но по крайней мере, Вы выполните сборку в вашем конкретном случае.

Теперь, о реальных проблемах...

Симптомы GCC не найден, хотя я **ВЫПОЛНИЛ** установку gcc.

Пояснения Эта проблема возникает по крайней мере на RHEL систем, где gcc является символической ссылкой на ccache. Потому, что crosstool-NG создаёт ссылки на gcc для построения в хост среде, эти символические ссылки фактически указывают к ccache, который затем не знает, как запустить компилятор. Можно исправить проблему, если задать переменную среды CCACHE_CC на компилятор, который фактически используется.

Исправьте Не известно.

Обход проблемы Удалите ccache.

Симптомы Этапы извлечения и/или пропачивания под Cygwin завершаются неудачно.

Пояснения Это не связано с crosstool-NG. Монтирование под Cygwin, по умолчанию, не учитывает регистр. Вы должны использовать так называемые «управляемую» монтировку. См.: http://cygwin.com/faq.html section 4, question 32.

Исправьте Используйте «управляемую» монтировку для каталогов, где Вы собираете **и** устанавливаете Ваш toolchains.

Обход проблемы Нет.

Симптомы uClibc не собирается под Cygwin.

Пояснения С помощью uClibc можно построить кросс ldd. К сожалению, не возможно (в настоящее время) построить этот кросс ldd под Cygwin.

Исправьте Пока нет ничего.

Обход проблемы Отключение построения кросс ldd.

Симптомы При построении 64-битной системы, построение glibc (возможно eglibc тоже) завершается неудачно для 64-разрядные мишени, потому что не может найти libgcc.

Пояснения Эта проблема наблюдалась при статически скомпонованых дополнительных библиотеках. По неизвестной причине, в данном случае libgcc строится ядром gcc не в том же месте , где оно само расположено, когда строятся с разделяемыми дополнительными библиотеками.

Исправьте Пока нет ничего.

Обход проблемы Постройте разделяемые (so) дополнительные библиотеки.

Симптомы При построении окончательного gcc, я получаю сообщение об ошибке, которое оканчивается на: libtool.m4: error: problem compiling FC test program

Пояснения Процедура построения gcc пытается запустить Fortran тест, чтобы увидеть, имеется ли нативный компилятор fortran, установлен он на компьютере построения, и она не может его найти. Нативный компилятор Fortran необходим (кажется необходимым) для построения Фортран frontend кросс компилятора. Даже если вы не хотите строить Фортран frontend, gcc пытается увидеть, если он, но это завершается неудачей. Это не проблема, так как Фортран интерфейс не будет построен. Нечего беспокоиться (если Вы не хотите построить Фортран frontend, конечно).

Исправьте Пока нет ничего. Это ложная ошибка, так что, вероятно, никогда не будет исправление для этой проблемы.

Обход проблемы Ничего не требуется, это ложная ошибка.

Симптомы GCC ругается, потому что «не удается обнаружить модель исключений».

Пояснения На некоторых архитектурах, правильная обработка стека (C++) при использовании setjmp/longjmp требует sjlj, в то время как на других архитектурах не нужно sjlj. На некоторых архитектурах gcc не может определить, нужны ли sjlj или нет.

Исправьте Пока нет ничего.

Обход проблемы Попытайтесь задать использование sjlj либо «Y» или *N* (вместо по умолчанию *M*) в menuconfig, в опции CT_CC_GCC_SJLJ_EXCEPTIONS, помеченной как «Использовать sjlj для исключения».

11 Разные учебники

11.1 Использование crosstool-NG в FreeBSD (и другие * BSD)

Предоставлено: Титус фон Боксберг

Предпосылки и инструкции по использованию ct-ng для строительства кросс toolchain на FreeBSD, как хосте.

1. испытан на FreeBSD 8.0
2. Установлены (как минимум) следующие порты
   - archivers/lzma
   - textproc/gsed
   - devel/gmake
   - devel/patch
   - shells/bash
   - devel/bison
   - lang/gawk
   - devel/automake110
   - ftp/wget Конечно вы должны иметь /usr/local/bin в переменной PATH.
3. Запустите конфигурацию ct-ng со следующими настройками конфигуратора: ---------- ---------- ./configure --with-sed=/usr/local/bin/gsed --with-make=/usr/loca \ --with-patch=/usr/local/bin/gpatch [.. .другие настроечные параметры, как вам нравится…] ----------
4. Действуйте, как описано в обычной документации но используйте gmake вместо make

11.2 Использование crosstool-NG на MacOS X

Предоставлено: Титус фон Боксберг

Предварительные требования и инструкции по использованию ct-ng для построения кросс toolchain на FreeBSD, как хосте.

1. Установлена Mac OS Snow Leopard с 3.2 с инструментами разработчика , или Mac OS Leopard, с инструментами разработчика & новый gcc (> = 4.3) установленные посредством macports
2. Вы должны использовать чувствительную к регистру файловую систему для построения ct-ng и целевых каталогов. Используйте диск или образ диска с fs, чувствительной к регистру, которые Вы должны где-то смонтировать.
3. Установить macports (или аналогичные средства лёгкой установки программного обеспечения сторонних производителей), убедитесь, что macport директория bin находится в PATH. Предположим далее, что это /opt/local/bin.
4. Установлены (как минимум) следующие порты
   - ncurses
   - lzmautils
   - libtool
   - binutils
   - gsed
   - gawk
   - gcc43 (необходим тоолько для Leopard OSX 10.5)

На Leopard убедитесь, что gcc macport вызывается с помощью команд по умолчанию (gcc, g ++,...), например через macport gcc_select 4) Запустите конфигурацию ct-ng со следующими настройками конфигуратора: (Предполагая, что Вы установили средства через macports в /opt/local):

```
./configure --with-sed=/opt/local/bin/gsed           \
--with-libtool=/opt/local/bin/glibtool   \
--with-objcopy=/opt/local/bin/gobjcopy   \
--with-objdump=/opt/local/bin/gobjdump   \
--with-readelf=/opt/local/bin/greadelf   \
[.. .другие настроечные параметры, как вам нравится...]
```

5) Действуйте, как описано в стандартной документации

ПОДСКАЗКИ:

- По-видимому встроенная переменная GNU make .LIBPATTERNS неправильно работает под MacOS: Она не включает lib%.dylib. Это влияет на сборку (по крайней мере) gdb-7.1 Поместите «lib%.a lib%.so lib%.dylib» в .LIBPATTERNS в Вашу среду перед выполнением построения ct-ng. Смотри http://www.gnu.org/software/make/manual/html_node/Libraries_002fSearch.html как объяснение.

11.3 Взломать crosstool-NG, используя Mercurial

Предоставлено: Титус фон Боксберг

11.3.1 ПРЕДВАРИТЕЛЬНЫЕ ТРЕБОВАНИЯ:

11.3.1.1 Настройка Mercurial:

Вам нужно mercurial со следующими расширениями:

- mq : http://mercurial.selenic.com/wiki/MqExtension
- patchbomb : http://mercurial.selenic.com/wiki/PatchbombExtension

Обычно эти два расширения уже являются частью пакета установки. Расширение mq поддерживает отдельную очередь ваших локальных изменений, которые можно исправить в любое время позднее. С расширением patchbomb вы можете отправить email с этими патчами непосредственно из вашего местного репо.

Ваш конфигурационный файл для mercurial, например ~/.hgrc должен содержать по крайней мере следующие разделы (но достаточно взглянуть на «man hgrc»):

```
# ---
[email]
# настроить отправку патчей непосредственно через Mercurial
from = ”Your Name” <your@email.address>
# Как послать email:
method = smtp


[smtp]
# SMTP конфигурации (только для метода = smtp)
host = localhost
tls = true
username =
password =

[extensions]
# Следующие строки включить два расширения:
hgext.mq =   hgext.patchbomb =
```

11.3.1.2 Создайте локальный репозиторий как клон:

```
hg clone http://ymorin.is-a-geek.org/hg/crosstool-ng crosstool-ng
```

11.3.1.3 Настройте расширение mq в вашей локальной копии:

```
cd crosstool-ng
hg qinit
```

11.3.2 СОЗДАНИЕ ИСПРАВЛЕНИЯ:

Запись изменений в очередь исправлений управляется mq:

```
# Во-первых создайте новый патч запись в очереди исправлений:
hg qnew -D -U -e short_patch_name1

# Отредактируйте описания исправления в сообщении commit (Ниже приводится пример)

# Отредактируйте исходный текст ct-ng и проверьте его
```

```
# Если Вы сейчас выполняете «hg статус», ваши изменения рабочей
# копии должны быть показаны.

# Теперь следующая команда берёт Ваши изменения из рабочей копии
# и создаёт  patch-запись
hg qrefresh -D [-e]

# повторно отредактируйте описание исправления [-e] если есть такая надобность>

# Теперь Ваши изменения будут записаны, и «hg статус» должен это чётко показывать
# в рабочей копии
```

Повторите описанные выше действия для всех ваших изменений. Команда «hg qseries» информирует вас о содержании вашей очереди patch.

11.3.3 ВКЛАД С ПОМОЩЬЮ ВАШИХ ПАТЧЕЙ:

Как только Вы удовлетворены Вашей серией patch, Вы можете (Вы должны!) поместить их обратно в главный поток. Это легко сделать с помощью команды «hg ”адрес электронной почты”».

«hg электронная почта» посылает набор новых изменений по указанному списку получателей, каждый патч в отдельном письме, в том порядке, как Вы, ввели их (старейший первый). Флаг командной строки --outgoing выбирает все наборы изменений, которые находятся у Вас, но еще не в главном тпотоке репозитория. Это именно тео, что Вы ввли в вашу локальную очередь patch выше, так что --outgoing является именно тем, что Вы хотите.

Каждое письмо получает такую тему: «[PATCH x of n] <сводка серии>””, где *x' - серийный номер письма в серии, а 'n* - общее количество патчей в серии. Тело письма представляет собой полный patch, плюс гнекоторые метаданные, которые помогают правильно применить патч, сохранить сообщения в лог, присвоить им дату, для отслеживания изменений (перемещение, удаление, режимы…) файла.

«hg <электронная почта>» также сцепляет все исходящие patch сообщения электронной почты, с помощью вступительного сообщения. Вы должны использовать вступительное сообщение (флаг командной строки --intro) для описания сферы охвата и мотивации для всей серии патча. Тема для вступительного сообщения серии сообщений: «[PATCH 0 из n] <резюме серии>” даёт Вам возможность установить <резюме серии>.

Вот пример «hg адрес» полной командной строки: Примечание: замените «(at)» на «@»

```
hg email --outgoing --intro   \
  --to '"Yann E. MORIN" <yann.morin.1998 (at) anciens.enib.fr>'    \
  --cc 'crossgcc (at) sourceware.org'
```

Открывает редактор и позволяет ввести тему # и тело для вводного сообщения.

Используйте «hg адрес» с дополнительным параметром командной строки - n для того, что бы сначала просмотреть письма, не отправляя их.

11.3.4 УПРАВЛЕНИЕ ВАШИМИ ПАТЧАМИ:

Когда патчи уточняются путем их обсуждения в списке рассылки, вы можете завершить обсуждение и повторно отправить их.

Расширение mq плохо переносит укладывагние стека в очередь: Вы можете всегда повторно отредактировать/обновить только патч на вершине стека. Очередь состоит из применённых и неприменённых исправлений (если Вы достигли этого места через описанные выше шаги, то все Ваши патчи применяются), где «стек» содержит применённые исправления, а верхняя часть стека - последнее применённые патчи.

```
hg qseries

0. A short_patch_name1
1. A short_patch_name2
2. A short_patch_name3
3. A short_patch_name4
```

Вы можете теперь редактировать патч «short_patch_name4» (который расположен в верхней части стека):

```
<Редактируем исходы>
# и выполняем снова
hg qrefresh -D [-e]
```

<Опция [-e]=”” позволяет повторно редактировать закомиченное сообщение>

Если Вы, например, хотите изменить патч short_patch_name2, Вам придется изменить стек mq, таким образом, что бы этот патч оказался в верхней части стека. Для этой цели посмотрите «hg help qgoto», «hg hekp qpop» и «hg help qpush».

hg qgoto short_patch_name2 # Очередь патчей теперь должна выглядеть как

```
hg qseries

0. A short_patch_name1
1. A short_patch_name2
2. U short_patch_name3
3. U short_patch_name4
```

Так, чтобы патч # 1 (short_patch_name2) оказался в верхней части стека. <Теперь повторно редактируйте исходники short_patch_name2>

```
# и выполняем снова
hg qrefresh -D [-e]
```

<Опция [-e]=”” позволяет повторно редактировать закомиченное сообщение>

```
# Следующая команда применяет теперь два неприменённых ранее исправления:
hg qpush -a
# Вы также можете использовать ’hg qgoto short_patch_name4’ что бы попасть туда снова.
```

11.3.5 ПОВТОРНАЯ ПЕРЕДАЧА ВНОВЬ ОТРЕДАКТИРОВАННЫХ PATCH:

В соответствии с политикой списка рассылки, пожалуйста, перешлите полный патч серии. -→ Вернитесь к разделу «CONTRIBUTING YOUR PATCHES» и повторно представьте полный набор.

СИНХРОНИЗАЦИИ С ГЛАВНЫМ ПОТОКОМ:

Вы можете синхронизировать Ваш репозитарий с главным потоком в любое время, выполнив

```
# Сначала отменить все ваши патчи:
hg qpop - # Затем выбираем новые изменения из главного потока
# После чего обновляем Вашу рабочую копию hg up
# При необходимости удаляем уже интегрированные в главный поток исправления (см. ниже)
hg qdelete <short_name_of_already_applied_patch>
# и повторно применяем Ваши патчи, если всё ещё есть что-то не интегрированное в главный ↩
    поток (но см. ниже)
hg qpush -a
```

В конце концов ваши патчи включаются в главное хранилище, которое вы изначально склонировали. В этом случае перед выполнением hg qpush -a, сверху Вы должны вручную выполнить удаление патчей «hg qdelete», которые уже интегрированы в главный поток.

КАК ФОРМАТИРОВАТЬ СООБЩЕНИЕ во вемя commit (ака патч desciptions):

Example 11.1 Сообщение в commit, должно выглядеть (без ведущего |):

|component: short, one-line description |

Необязательный символ | позволяет вводить описание из нескольких строк при необходимости

Example 11.2 Вот пример сообщения фиксации (см. a53a5e1d61db редакция):

|comp-libs/cloog: fix building | |For CLooG/PPL 0.15.3, the directory name was simply cloog-ppl. |For any later versions, the directory name does have the version, such as |cloog-ppl-0.15.4.