

# Cross Compiling For Perl Hackers

Jens Rehsack

Niederrhein Perl Mongers

2016

## Part I

# Introduction

- 1 Introduction
  - Motivation

# Motivation

## Clarify some use-cases

- Cross-Compiling
- Cross-Building
- Canadian Cross
- Foreign builds

# Motivation

## Clarify some use-cases

- Cross-Compiling
- Cross-Building
- Canadian Cross
- Foreign builds

## Sensibilize beyond developer environments

- How can I enable other people using my code?
- What else beside specs, tests and documentation can be provided?
- Why should I care?

## Part II

### Basics

- 2 Cross Compiler
  - API vs. ABI
- 3 Cross Compiling
- 4 Cross SDK
- 5 Pitfalls

# Cross Compiler

- Compiles source into binary objects for another platform than the current host

# Cross Compiler

- Compiles source into binary objects for another platform than the current host
- Platform? What is such a platform?

# Cross Compiler

- Compiles source into binary objects for another platform than the current host
- Platform? What is such a platform?
- A platform is defined by
  - ▶ Architecture
  - ▶ Vendor
  - ▶ Operating System / ABI



# Cross Compiler

- Compiles source into binary objects for another platform than the current host
- Platform? What is such a platform?
- A platform is defined by
  - ▶ Architecture
  - ▶ Vendor
  - ▶ Operating System / ABI
    - ★ i486-pc-linux-gnu
    - ★ x86\_64-apple-darwin64

# Cross Compiler

- Compiles source into binary objects for another platform than the current host
- Platform? What is such a platform?
- A platform is defined by
  - ▶ Architecture
  - ▶ Vendor
  - ▶ Operating System / ABI
    - ★ i486-pc-linux-gnu
    - ★ x86\_64-apple-darwin64
    - ★ arm926ejse-poky-linux-gnueabi
    - ★ cortexa9hf-vfp-neon-mx6qdl-poky-linux-gnueabi

# Cross Compiler

- Compiles source into binary objects for another platform than the current host
- Platform? What is such a platform?
- A platform is defined by
  - ▶ Architecture
  - ▶ Vendor
  - ▶ Operating System / ABI
    - ★ i486-pc-linux-gnu
    - ★ x86\_64-apple-darwin64
    - ★ arm926ejse-poky-linux-gnueabi
    - ★ cortexa9hf-vfp-neon-mx6qdl-poky-linux-gnueabi
    - ★ sparcv9-sun-solaris

# API vs. ABI

## size\_t-size.c

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

int main(int argc, char *argv[]) {
    printf("%zd\n", sizeof(size_t));
    return 0;
}
```

# API vs. ABI

## size\_t-size.c

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

int main(int argc, char *argv[]) {
    printf("%zd\n", sizeof(size_t));
    return 0;
}
```

## 32-bit mode size\_t-size

```
$ cc -O -m32 -o size_t-size size_t-size.c
$ ./size_t-size
4
```

# API vs. ABI

## size\_t-size.c

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

int main(int argc, char *argv[]) {
    printf("%zd\n", sizeof(size_t));
    return 0;
}
```

## 32-bit mode size\_t-size

```
$ cc -O -m32 -o size_t-size size_t-size.c
$ ./size_t-size
4
```

## 64-bit mode size\_t-size

```
$ cc -O -m64 -o size_t-size size_t-size.c
$ ./size_t-size
8
```

# API

- abbreviation for "Application Programming Interface"

# API

- abbreviation for "Application Programming Interface"
- defines compatibility on source level



# API

- abbreviation for "Application Programming Interface"
- defines compatibility on source level

## snprintf declaration

```
#include <stdio.h>
int snprintf(char * restrict str,
             size_t size,
             const char * restrict format,
             ...);
```

# API

- abbreviation for "Application Programming Interface"
- defines compatibility on source level

## snprintf declaration

```
#include <stdio.h>
int snprintf(char * restrict str,
             size_t size,
             const char * restrict format,
             ...);
```

- every STD C conforming C program can call snprintf

## snprintf invocation

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    char buf[_PATH_MAX];
    snprintf(buf, sizeof buf, "%s", argv[0]);
    return 0;
}
```

# ABI

- abbreviation for "Application Binary Interface"

# ABI

- abbreviation for "Application Binary Interface"
- defines compatibility on compiled code level

# ABI

- abbreviation for "Application Binary Interface"
- defines compatibility on compiled code level

## snprintf declaration

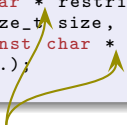
```
#include <stdio.h>
int snprintf(char * restrict str,
             size_t size,
             const char * restrict format,
             ...);
```

# ABI

- abbreviation for "Application Binary Interface"
- defines compatibility on compiled code level

## snprintf declaration

```
#include <stdio.h>
int snprintf(char * restrict str,
             size_t size,
             const char * restrict format,
             ...);
```



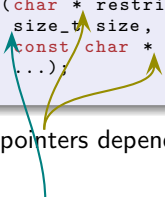
- sizes of pointers depend on memory model (segmented, flat, address width, ...)

# ABI

- abbreviation for "Application Binary Interface"
- defines compatibility on compiled code level

## snprintf declaration

```
#include <stdio.h>
int snprintf(char * restrict str,
             size_t size,
             const char * restrict format,
             ...);
```



- sizes of pointers depend on memory model (segmented, flat, address width, ...)
- size of buffer size depends just on a subset of the memory model: the address width

# ABI influencers

- CPU register sizes



# ABI influencers

- CPU register sizes
- alignment

# ABI influencers

- CPU register sizes
- alignment
- packing of enums/structs

# ABI influencers

- CPU register sizes
- alignment
- packing of enums/structs
- memory model (flat vs. segmented, address width, ...)

# ABI influencers

- CPU register sizes
- alignment
- packing of enums/structs
- memory model (flat vs. segmented, address width, ...)
- calling convention (stack vs. register based, order of arguments, how many registers, ...)

# ABI influencers

- CPU register sizes
- alignment
- packing of enums/structs
- memory model (flat vs. segmented, address width, ...)
- calling convention (stack vs. register based, order of arguments, how many registers, ...)
- byte order

# Cross Compiling "Hello world"

What does such a compiler do?

# Cross Compiling "Hello world"

What does such a compiler do?

Compiles source.

# Cross Compiling "Hello world"

What does such a compiler do?

Compiles source.

hello.c

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

int main(int argc, char *argv[]) {
    printf("Hello world\n");
    return 0;
}
```



# Cross Compiling II

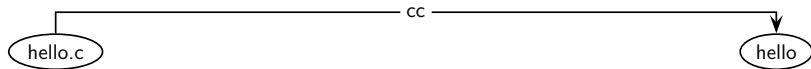
## compiler invocation

```
$ ${CC} -o hello hello.c
```

# Cross Compiling II

## compiler invocation

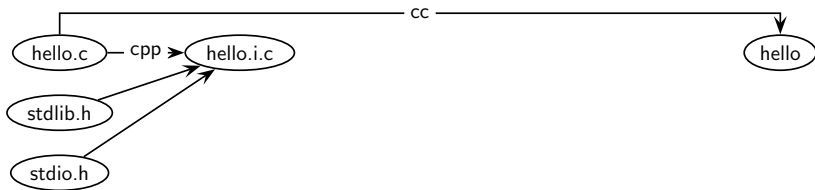
```
$ ${CC} -o hello hello.c
```



# Cross Compiling II

## compiler invocation

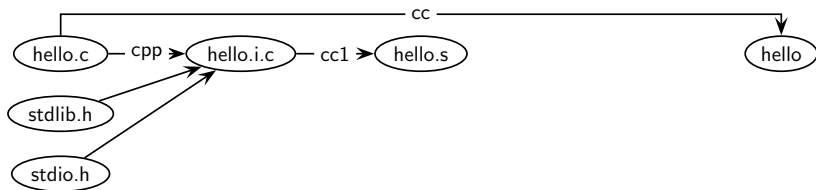
```
$ ${CC} -o hello hello.c
```



# Cross Compiling II

## compiler invocation

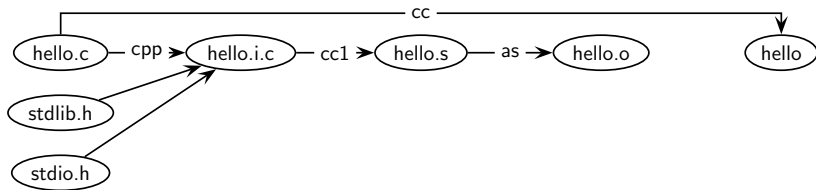
```
$ ${CC} -o hello hello.c
```



# Cross Compiling II

## compiler invocation

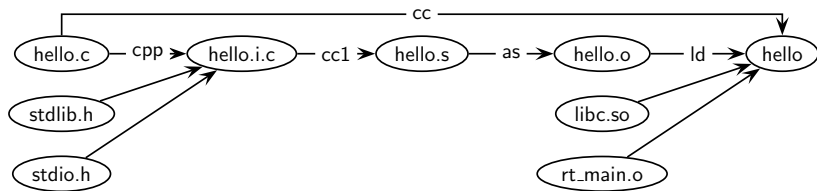
```
$ ${CC} -o hello hello.c
```



# Cross Compiling II

## compiler invocation

```
$ ${CC} -o hello hello.c
```



# Cross Development Kit

What is this ... you're talking about?

# Cross Development Kit

What is this ... you're talking about?

cpp C PreProcessor



# Cross Development Kit

What is this ... you're talking about?

cpp C PreProcessor

cc1 C Compiler

# Cross Development Kit

What is this ... you're talking about?

`cpp` C PreProcessor

`cc1` C Compiler

`as` Assembler

# Cross Development Kit

What is this ... you're talking about?

cpp C PreProcessor

cc1 C Compiler

as Assembler

ld Linker

# Cross Development Kit Location

## Which `stdlib.h`,... is used

```
# locate stdlib.h
...
/foo-bsp/w/tmp/sysroots/arm926ejse-poky-linux-gnueabi/usr/include/stdlib.h
/foo-bsp/w/tmp/sysroots/cortexa9hf-vfp-neon-poky-li.../usr/include/stdlib.h
...
/opt/SolarisStudio12.3-linux-x86-bin/solstudio12.3/prod/include/cc/stdlib.h
...
/usr/include/stdio.h
```

# Cross Development Kit Location

## Which `stdlib.h`,... is used

```
# locate stdlib.h
...
/foo-bsp/w/tmp/sysroots/arm926ejse-poky-linux-gnueabi/usr/include/stdlib.h
/foo-bsp/w/tmp/sysroots/cortexa9hf-vfp-neon-poky-li.../usr/include/stdlib.h
...
/opt/SolarisStudio12.3-linux-x86-bin/solstudio12.3/prod/include/cc/stdlib.h
...
/usr/include/stdio.h
```

Similar picture for

# Cross Development Kit Location

## Which `stdlib.h`,... is used

```
# locate stdlib.h
...
/foo-bsp/w/tmp/sysroots/arm926ejse-poky-linux-gnueabi/usr/include/stdlib.h
/foo-bsp/w/tmp/sysroots/cortexa9hf-vfp-neon-poky-li.../usr/include/stdlib.h
...
/opt/SolarisStudio12.3-linux-x86-bin/solstudio12.3/prod/include/cc/stdlib.h
...
/usr/include/stdio.h
```

Similar picture for `stdio.h`,

# Cross Development Kit Location

## Which `stdlib.h`,... is used

```
# locate stdlib.h
...
/foo-bsp/w/tmp/sysroots/arm926ejse-poky-linux-gnueabi/usr/include/stdlib.h
/foo-bsp/w/tmp/sysroots/cortexa9hf-vfp-neon-poky-li.../usr/include/stdlib.h
...
/opt/SolarisStudio12.3-linux-x86-bin/solstudio12.3/prod/include/cc/stdlib.h
...
/usr/include/stdio.h
```

Similar picture for `stdio.h`, `stdint.h`,

# Cross Development Kit Location

## Which `stdlib.h`,... is used

```
# locate stdlib.h
...
/foo-bsp/w/tmp/sysroots/arm926ejse-poky-linux-gnueabi/usr/include/stdlib.h
/foo-bsp/w/tmp/sysroots/cortexa9hf-vfp-neon-poky-li.../usr/include/stdlib.h
...
/opt/SolarisStudio12.3-linux-x86-bin/solstudio12.3/prod/include/cc/stdlib.h
...
/usr/include/stdio.h
```

Similar picture for `stdio.h`, `stdint.h`, `libc.so`,



# Cross Development Kit Location

## Which `stdlib.h`,... is used

```
# locate stdlib.h
...
/foo-bsp/w/tmp/sysroots/arm926ejse-poky-linux-gnueabi/usr/include/stdlib.h
/foo-bsp/w/tmp/sysroots/cortexa9hf-vfp-neon-poky-li.../usr/include/stdlib.h
...
/opt/SolarisStudio12.3-linux-x86-bin/solstudio12.3/prod/include/cc/stdlib.h
...
/usr/include/stdio.h
```

Similar picture for `stdio.h`, `stdint.h`, `libc.so`, `rt_main.o`, ...

# Convinced

Where can I download it?

# Convinced

Where can I download it?

Which one?

# Build Yourself a Cross-SDK

Use the source, Luke

There're several ways:

# Build Yourself a Cross-SDK

## Use the source, Luke

There're several ways:

- the hard way: do it yourself as described at [Building Cross Toolchains with gcc](#) or [Build a GCC-based cross compiler for Linux](#)

# Build Yourself a Cross-SDK

## Use the source, Luke

There're several ways:

- the hard way: do it yourself as described at [Building Cross Toolchains with gcc](#) or [Build a GCC-based cross compiler for Linux](#)
- Toolchain build helper like [crosstool-NG](#) or [Scratchbox](#)

# Build Yourself a Cross-SDK

## Use the source, Luke

There're several ways:

- the hard way: do it yourself as described at [Building Cross Toolchains with gcc](#) or [Build a GCC-based cross compiler for Linux](#)
- Toolchain build helper like [crosstool-NG](#) or [Scratchbox](#)
- Full flavoured - [Yocto](#) or [T2 SDE](#)

# Vendor Cross-SDK

## Typical cases

- Bare Metal SDK
- Accelerator Libraries (typically not Open-Source)
- Early Adopter
- Enterprise Support



# And now

Which way I ought to go from here?

That depends . . . on where you want to get to.

## And now

Which way I ought to go from here?

That depends . . . on where you want to get to.

Topic was . . .

Cross compiling for Perl Hackers

## And now

Which way I ought to go from here?

That depends ... on where you want to get to.

Topic was ...

Cross compiling for Perl Hackers

we didn't define an audience, reasonable possibilities are

## And now

Which way I ought to go from here?

That depends ... on where you want to get to.

Topic was ...

Cross compiling for Perl Hackers

we didn't define an audience, reasonable possibilities are

- Perl Porters

## And now

Which way I ought to go from here?

That depends ... on where you want to get to.

Topic was ...

Cross compiling for Perl Hackers

we didn't define an audience, reasonable possibilities are

- Perl Porters
- Perl Module Maintainers

## And now

Which way I ought to go from here?

That depends ... on where you want to get to.

Topic was ...

Cross compiling for Perl Hackers

we didn't define an audience, reasonable possibilities are

- Perl Porters
- Perl Module Maintainers

Perl Porters probably have to care for more than Perl Module Maintainers ...

# Build here, run there

## Host vs. Target

# Build here, run there

## Host vs. Target

- Which 'cc' to use to compile bootstrap tools (as miniperl)?



# Build here, run there

## Host vs. Target

- Which 'cc' to use to compile bootstrap tools (as miniperl)?  
mind `HOSTCC` vs. `CC`

# Build here, run there

## Host vs. Target

- Which 'cc' to use to compile bootstrap tools (as miniperl)?  
mind `HOSTCC` vs. `CC`
- ...and which `stdlib.h/libc.so`?

# Build here, run there

## Host vs. Target

- Which 'cc' to use to compile bootstrap tools (as miniperl)?  
mind `HOSTCC` vs. `CC`
- ...and which `stdlib.h/libc.so`?  
modern toolchains know `--sysroot` argument - prior lot's on replacements  
in `-I...` and `-L...` were required

# Build here, run there

## Host vs. Target

- Which 'cc' to use to compile bootstrap tools (as miniperl)?  
mind `HOSTCC` vs. `CC`
- ... and which `stdlib.h/libc.so`?  
modern toolchains know `--sysroot` argument - prior lot's on replacements  
in `-I...` and `-L...` were required
- pick right `CFLAGS`, `BUILD_CFLAGS`, `HOST_CFLAGS` or `TARGET_CFLAGS`  
for the right job, likewise for `LDFLAGS`, `CCLDFLAGS`, `LDDLFLAGS`,  
`CXXFLAGS` and whatever additional tool is used

# Build here, run there

## Host vs. Target

- Which 'cc' to use to compile bootstrap tools (as miniperl)?  
mind `HOSTCC` vs. `CC`
- ... and which `stdlib.h/libc.so`?  
modern toolchains know `--sysroot` argument - prior lot's on replacements  
in `-I...` and `-L...` were required
- pick right `CFLAGS`, `BUILD_CFLAGS`, `HOST_CFLAGS` or `TARGET_CFLAGS`  
for the right job, likewise for `LD_FLAGS`, `CCLDFLAGS`, `LDDLFLAGS`,  
`CXXFLAGS` and whatever additional tool is used
- do not mix build and target configuration

# Build here, run there

## Host vs. Target

- Which 'cc' to use to compile bootstrap tools (as miniperl)?  
mind `HOSTCC` vs. `CC`
- ...and which `stdlib.h/libc.so`?  
modern toolchains know `--sysroot` argument - prior lot's on replacements  
in `-I...` and `-L...` were required
- pick right `CFLAGS`, `BUILD_CFLAGS`, `HOST_CFLAGS` or `TARGET_CFLAGS`  
for the right job, likewise for `LDFLAGS`, `CCLDFLAGS`, `LDDLFLAGS`,  
`CXXFLAGS` and whatever additional tool is used
- do not mix build and target configuration
- do not run target artifacts locally

## Build here, run there II

### Build vs. Run

- during build, several development kits are involved

# Build here, run there II

## Build vs. Run

- during build, several development kits are involved (at least host and target, sometimes host, build and multiple targets)



# Build here, run there II

## Build vs. Run

- during build, several development kits are involved (at least host and target, sometimes host, build and multiple targets)
- **PATHs** vary, eg.  
-L/foo-bsp/w/tmp/sysroots/arm926ejse-poky-linux-gnueabi/usr/lib  
vs. -Wl,-R/usr/lib

## Build here, run there III

mind those differences when invoking wrapper-scripts

### Build vs. Run

```
rakudo-star % make install
# This is a post-compile task, unfortunately placed into install stage
./perl6-j tools/build/install-core-dist.pl /foo-bsp/w/tmp/work/...
  cortexa9hf-vfp-neon-poky-linux-gnueabi/rakudo-star/2016.01-r0/...
  image/usr/share/nqp
Error: Could not find or load main class perl6
```

## Build here, run there III

mind those differences when invoking wrapper-scripts

### Build vs. Run

```
rakudo-star % make install
# This is a post-compile task, unfortunately placed into install stage
./perl6-j tools/build/install-core-dist.pl /foo-bsp/w/tmp/work/...
    cortexa9hf-vfp-neon-poky-linux-gnueabi/rakudo-star/2016.01-r0/...
    image/usr/share/nqp
Error: Could not find or load main class perl6
```

### perl6-j

```
#!/bin/sh
: ${NQP_DIR:=/usr/share/nqp}
: ${NQP_JARS:="${NQP_DIR}/runtime/asm-4.1.jar:${NQP_DIR}/runtime/asm-tree-4.1.jar:\
    ${NQP_DIR}/runtime/jline-1.0.jar:${NQP_DIR}/runtime/jna.jar:\
    ${NQP_DIR}/runtime/nqp-runtime.jar:${NQP_DIR}/lib/nqp.jar"}
: ${PERL6_DIR:=/usr/share/perl6}
: ${PERL6_JARS:="${NQP_JARS}:${PERL6_DIR}/runtime/rakudo-runtime.jar:${PERL6_DIR}/runtime/perl6.jar"}
exec java -noverify -Xms100m -Xbootclasspath/a:${NQP_JARS}:${PERL6_DIR}/runtime/rakudo-runtime.jar:\
    ${PERL6_DIR}/runtime/perl6.jar -cp $CLASSPATH:${PERL6_DIR}/runtime:${PERL6_DIR}/lib:\
    ${NQP_DIR}/lib -Dperl6.prefix=/usr -Djna.library.path=/usr/share/perl6/site/lib \
    -Dperl6.execname="$0" perl6 "$@"
```

## Build here, run there IV

### Build vs. Run

- guess why running that script from  
`/foo-bsp/w/tmp/work/cortexa9hf-vfp-neon-poky-linux-gnueabi/...  
rakudo-star/2016.01-r0/rakudo-star-2016.01/` fails ...

# Build here, run there IV

## Build vs. Run

- guess why running that script from  
`/foo-bsp/w/tmp/work/cortexa9hf-vfp-neon-poky-linux-gnueabi/...  
rakudo-star/2016.01-r0/rakudo-star-2016.01/` fails ...
- remember `sdkroot` (build libraries, can be executed in build environment) and `sysroot` (target runtime chroot, used for linking etc.)

# Build here, run there IV

## Build vs. Run

- guess why running that script from  
`/foo-bsp/w/tmp/work/cortexa9hf-vfp-neon-poky-linux-gnueabi/...  
rakudo-star/2016.01-r0/rakudo-star-2016.01/` fails ...
- remember `sdkroot` (build libraries, can be executed in build environment) and `sysroot` (target runtime chroot, used for linking etc.)
- all path's in `sysroot` are as if the files were already on target

# Configure Stage

## Prerequisites ...

- nowadays Perl Toolchain doesn't support cross compile dependency checks

# Configure Stage

## Prerequisites ...

- nowadays Perl Toolchain doesn't support cross compile dependency checks
- neither external resources (mind wrapper modules as `RRDTool::OO`), so configure stage has to prove on it's own (compile and link test in `Makefile.PL`)



# Configure Stage

## Prerequisites ...

- nowadays Perl Toolchain doesn't support cross compile dependency checks
- neither external resources (mind wrapper modules as `RRDTool::OO`), so configure stage has to prove on it's own (compile and link test in `Makefile.PL`)
- `x_prereqs` was an idea but never completed

# Configure Stage

## Prerequisites ...

- nowadays Perl Toolchain doesn't support cross compile dependency checks
- neither external resources (mind wrapper modules as `RRDTool::OO`), so configure stage has to prove on it's own (compile and link test in `Makefile.PL`)
- `x_prereqs` was an idea but never completed
- workaround in Yocto for module prerequisites: `DEPENDS` (configure stage) contain host packages, `RDEPENDS` (install stage) contain target packages

# Configure Stage

## Prerequisites ...

- nowadays Perl Toolchain doesn't support cross compile dependency checks
- neither external resources (mind wrapper modules as `RRDTool::OO`), so configure stage has to prove on it's own (compile and link test in `Makefile.PL`)
- `x_prereqs` was an idea but never completed
- workaround in Yocto for module prerequisites: `DEPENDS` (configure stage) contain host packages, `RDEPENDS` (install stage) contain target packages
  - ↳ it's slightly more complicated for external libraries when `Makefile.PL` doesn't know about cross compiling

## Part III

### Finish

6 Conclusion

7 Resources

8 Thank you

# Conclusion

- stay as close as possible to existing standards - reinventing the wheel will almost always fail

# Conclusion

- stay as close as possible to existing standards - reinventing the wheel will almost always fail
- use `ExtUtils::MakeMaker` for building

# Conclusion

- stay as close as possible to existing standards - reinventing the wheel will almost always fail
- use `ExtUtils::MakeMaker` for building
- use `Config::AutoConf` when it is really necessary to have configure time checks (as which API is supported by wrapped library)
- prefer `pkgconf` (or `pkg-config`) over `compile` and `link` testing

# Conclusion

- stay as close as possible to existing standards - reinventing the wheel will almost always fail
- use `ExtUtils::MakeMaker` for building
- use `Config::AutoConf` when it is really necessary to have configure time checks (as which API is supported by wrapped library)
- prefer `pkgconf` (or `pkg-config`) over `compile` and `link` testing
- always allow every check being overwritten by environment variables



# Resources

## Cross Compile Perl

[P5P] Remodeling the cross-compilation model

<http://grokbase.com/t/perl/perl5-porters/141gz52519/remodeling-the-cross-compilation-model>

# Resources

## Cross Compile Perl

[P5P] Remodeling the cross-compilation model

<http://grokbase.com/t/perl/perl5-porters/141gz52519/remodeling-the-cross-compilation-model>

## Cross Compile Guides

Building Cross Toolchains with gcc

[https://gcc.gnu.org/wiki/Building\\_Cross\\_Toolchains\\_with\\_gcc](https://gcc.gnu.org/wiki/Building_Cross_Toolchains_with_gcc)

Build a GCC-based cross compiler for Linux

<https://www6.software.ibm.com/developerworks/education/library/wa/crosscompilinggcc.html>

# Resources

## Cross Compile Helper

crosstool-NG <http://crosstool-ng.org/>

Scratchbox <http://www.scratchbox.org/>

# Resources

## Cross Compile Helper

crosstool-NG <http://crosstool-ng.org/>

Scratchbox <http://www.scratchbox.org/>

## Cross Compile Distribution Builder

Yocto <http://yoctoproject.org/>

T2 SDE <http://t2-project.org/>

# Thank You For Listening

Questions?

Jens Rehsack <[rehsack@cpan.org](mailto:rehsack@cpan.org)>

Cologne