从零开始为 RISC-V 构建一个 Linux 系统

第 5 章 制作交叉编译工具链

汪辰

目录

01

编译与链接

GCC 简介

- GCC（GNU Compiler Collection）https://gcc.gnu.org/
- 由 GNU 开发的，遵循 GPL 许可证发行的编译器套件。
- 支持 C、C++、Objective-C、Fortran、Ada 和 Go 语言等多种语言前端，已被移植到多种计算机体系架构上，如 x86、ARM、RISC-V 等。
- GCC 的初衷是为 GNU 操作系统专门编写一款编译器，现已被大多数 “Unix-like” 操作系统（如 Linux、BSD、MacOS 等）采纳为标准的编译器。

GCC 的命令格式

gcc [options] [filenames]

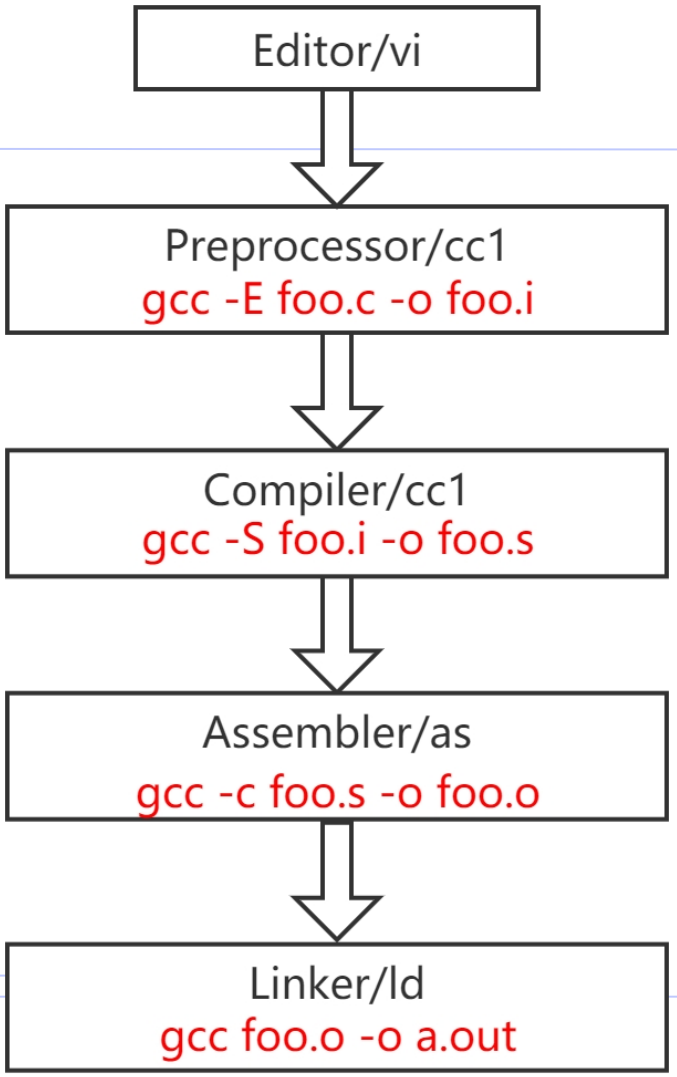
常用选项	含义
-E	只做预处理
-c	只编译不链接，生成目标文件“.o”
-S	生成汇编代码
-o file	把输出生成到由 file 指定文件名的文件中
-g	在输出的文件中加入支持调试的信息
-v	显示输出详细的命令执行过程信息

GCC 的主要执行步骤

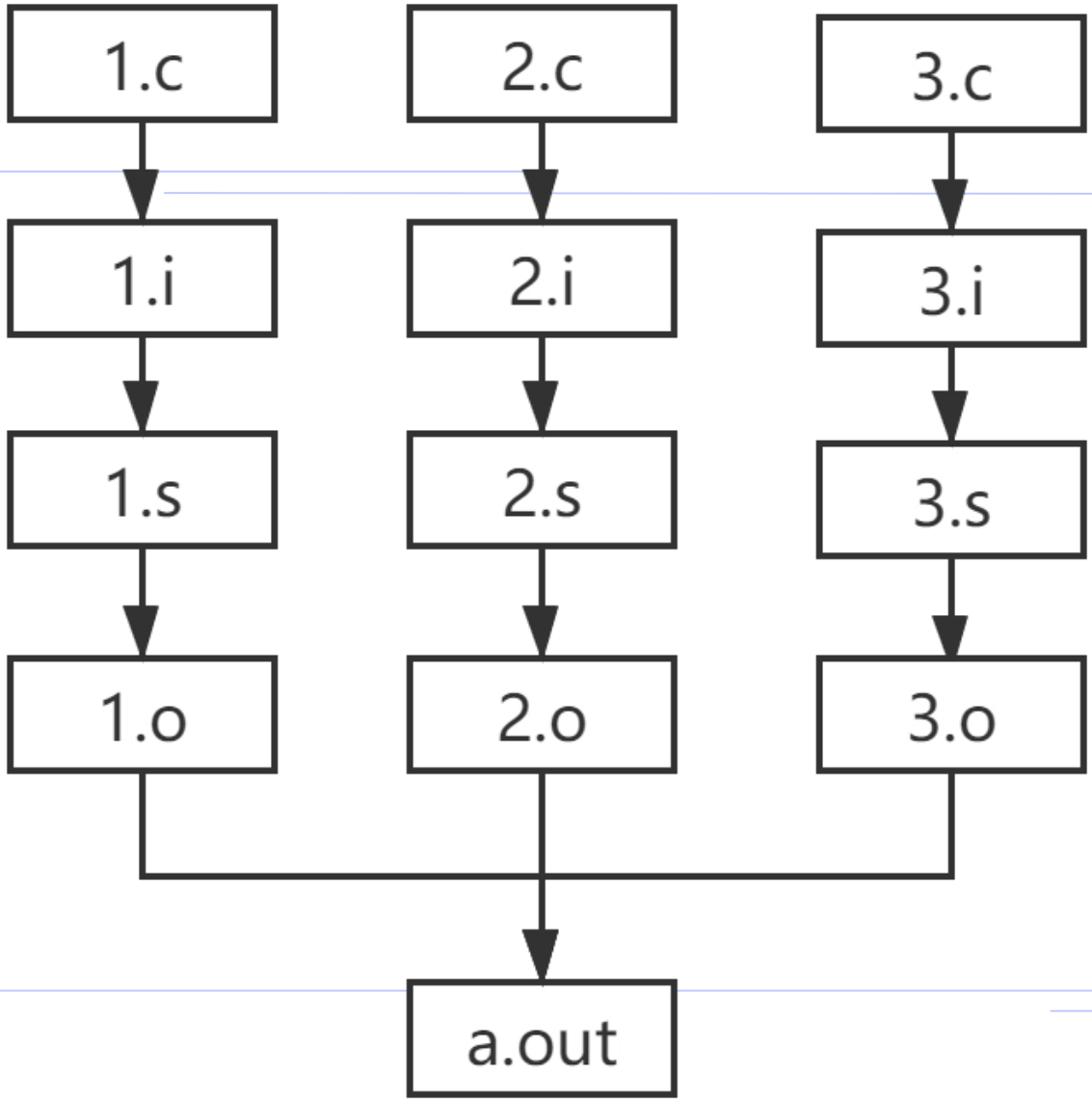
- **编译**（cc1，这里针对 C 语言，不同的语言有自己的编译器）：编译器完成 **预处理** 和 **编译**，**预处理** 指处理源文件中以“#”开头的预处理指令，譬如 #include、#define 等；**编译** 则针对预处理的结果进行一系列的词法分析、语法分析、语义分析，优化后生成汇编指令，存放在 .o 为后缀的目标文件中。

- **汇编**（as）：汇编器将汇编语言代码转换为机器（CPU）可以执行的指令。

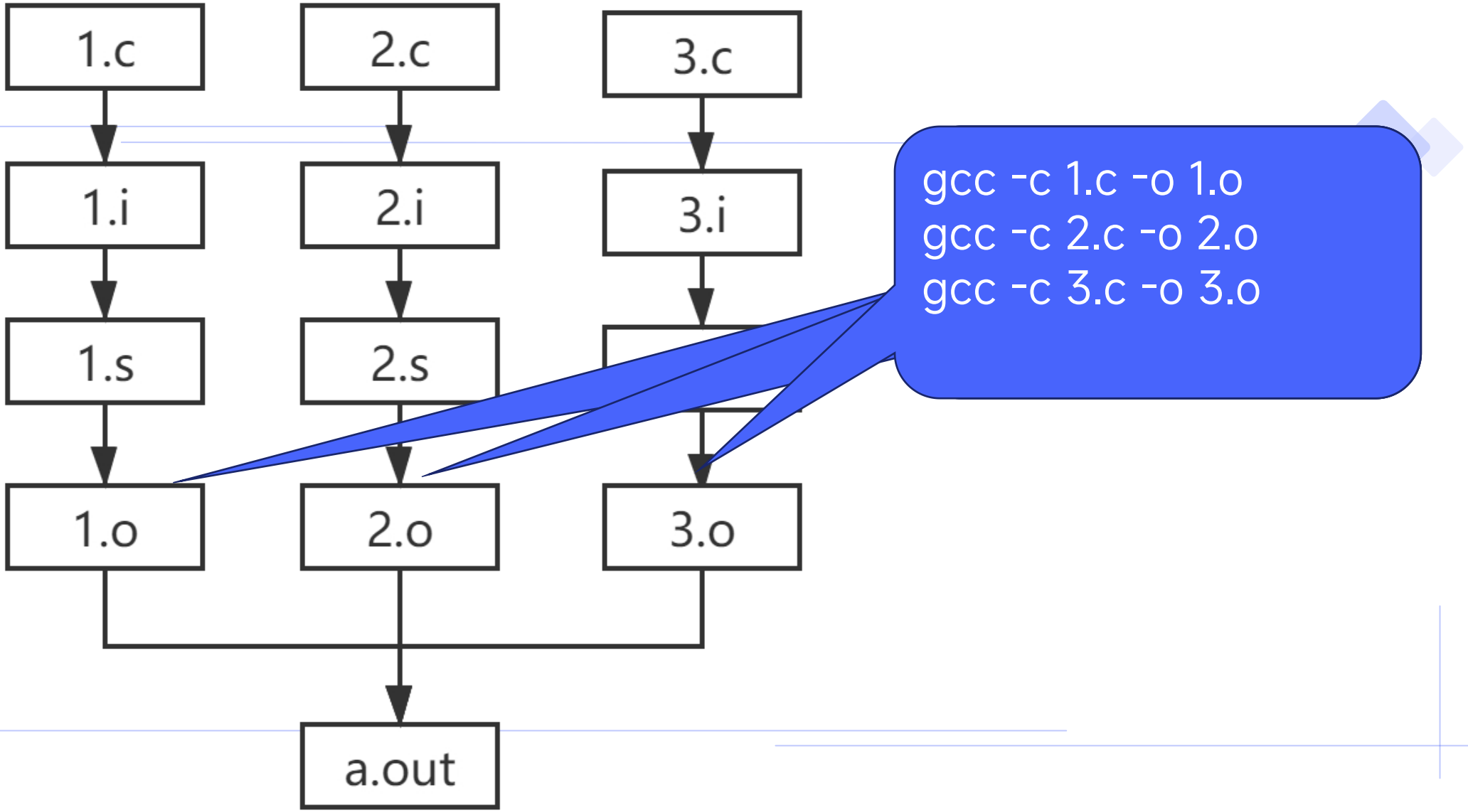
- **链接**（ld）：链接器将汇编器生成的目标文件和一些标准库（譬如 libc）文件组合，形成最终可执行的应用程序。



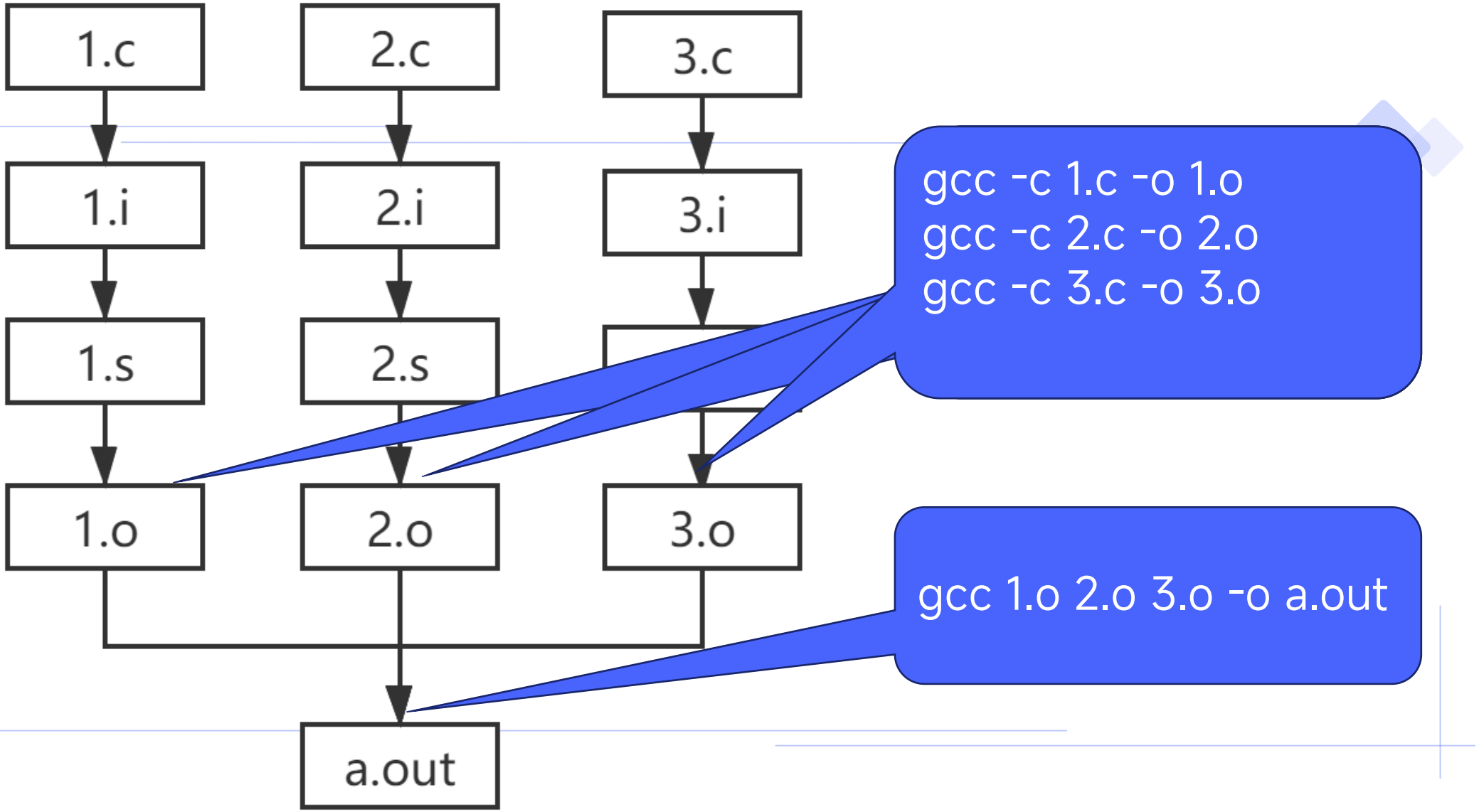
GCC 针对多个源文件的处理



GCC 针对多个源文件的处理

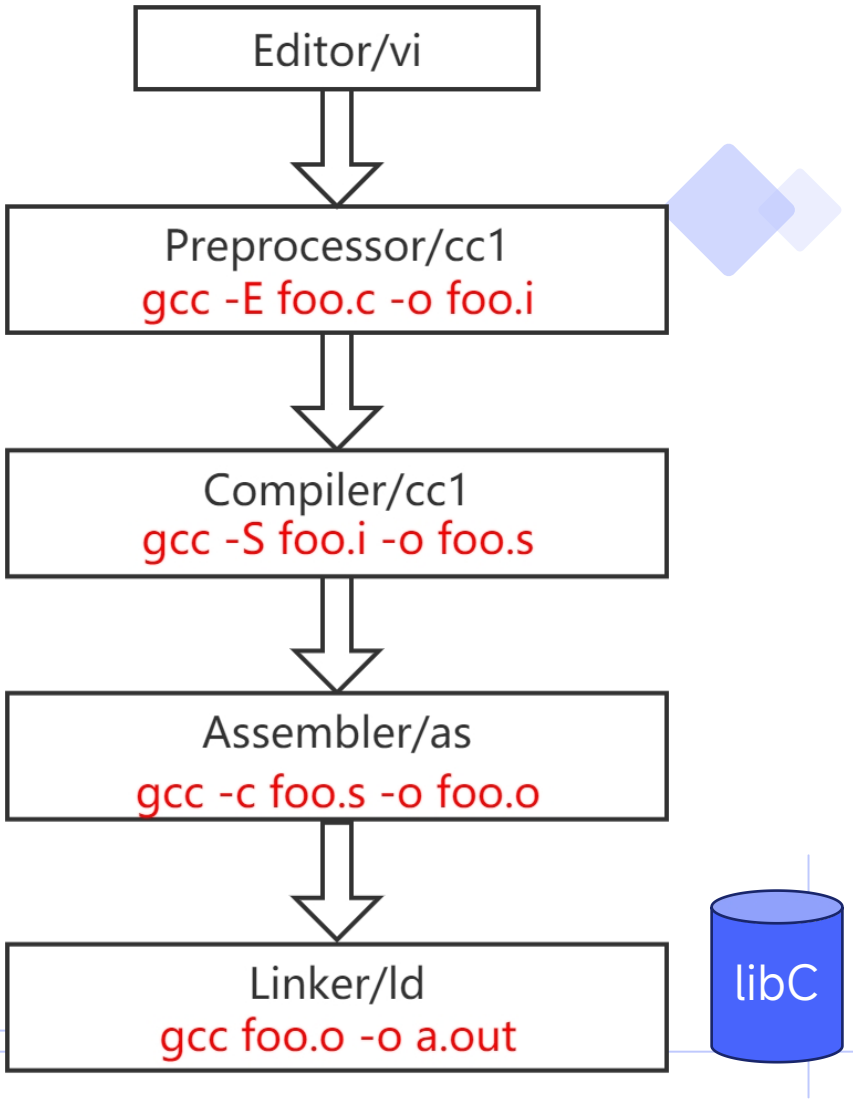


GCC 针对多个源文件的处理



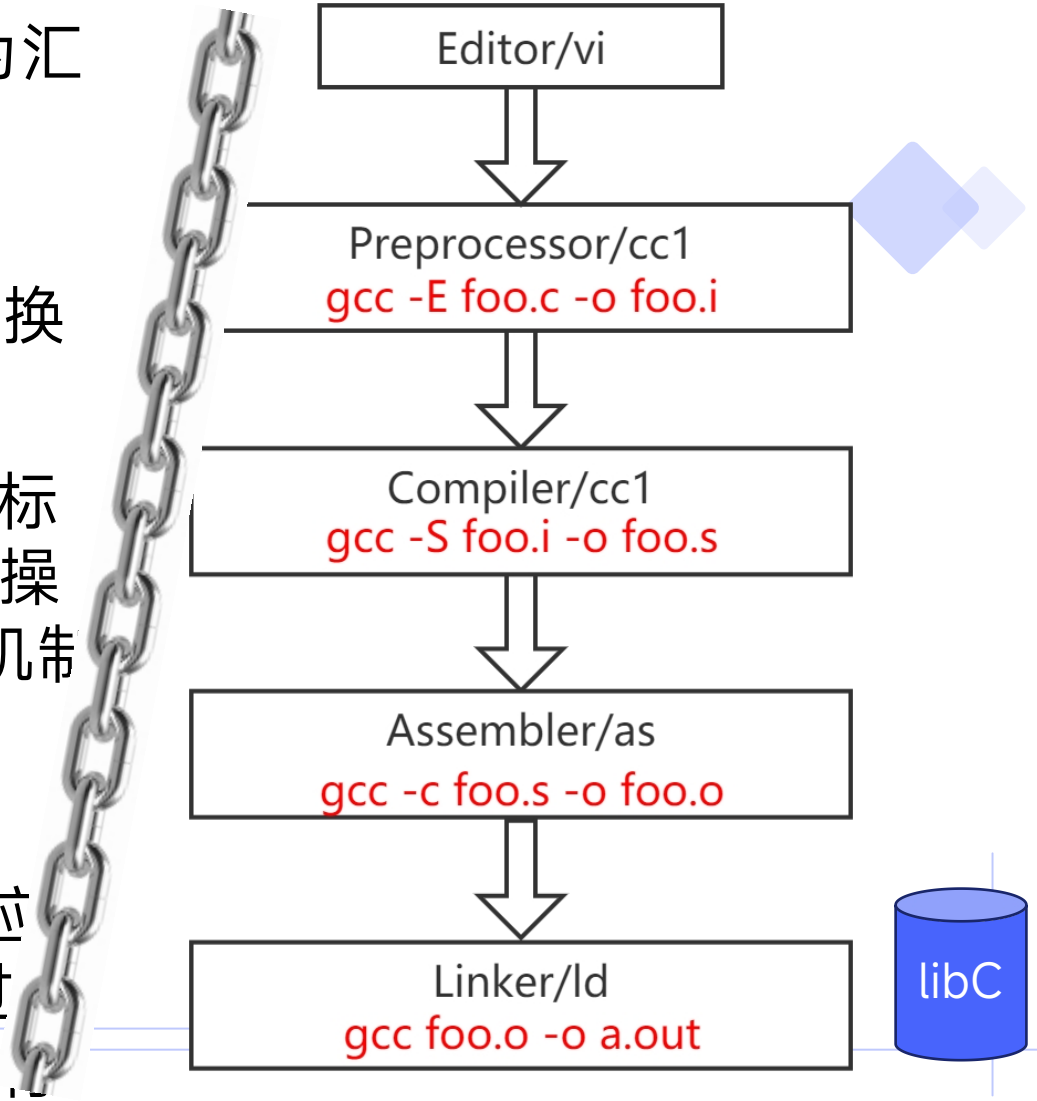
工具链（Tool Chain）的主要组成

- **编译器（cc1）**：将高级语言（C）转换为汇编语言。编译器需要了解目标汇编语言（RISC-V）。
- **汇编器（as）**：汇编器将汇编语言代码转换为 CPU 可执行的字节码（机器语言）。
- **链接器（ld）**：链接器将汇编器生成的目标文件组合成一个可执行应用程序。不同的操作系统和 CPU 组合通常使用不同的封装机制和标准（ELF）。
- **标准 C 库（glibc）**：核心 C 函数（例如 printf）在一个统一的 C 库中提供。如果应用程序中使用了 C 库中的函数，则该库通过和源代码的目标文件链接，生成最终的可执行文件。



工具链（Tool Chain）的主要组成

- **编译器（cc1）**：将高级语言（C）转换为汇编语言。编译器需要了解目标汇编语言（RISC-V）。
- **汇编器（as）**：汇编器将汇编语言代码转换为 CPU 可执行的字节码（机器语言）。
- **链接器（ld）**：链接器将汇编器生成的目标文件组合成一个可执行应用程序。不同的操作系统和 CPU 组合通常使用不同的封装机制和标准（ELF）。
- **标准 C 库（glibc）**：核心 C 函数（例如 printf）在一个统一的 C 库中提供。如果应程序中使用了 C 库中的函数，则该库通过源代码的目标文件链接，生成最终的可执行文件。



Binutils 简介

- Binutils（Binary Utilities）二进制格式（ELF）文件处理工具
- 由 GNU 开发的，遵循 GPL 许可证发行的工具套件 https://www.gnu.org/software/binutils/ 。
  - ✓ ar：归档文件，将多个文件打包成一个大文件。
  - ✓ as：被 gcc 调用，输入汇编文件，输出目标文件供链接器 ld 连接。
  - ✓ ld：链接器。被 gcc 调用，它把目标文件和各种库文件结合在一起，重定位数据，并链接符号引用。
  - ✓ objcopy：执行文件格式转换。
  - ✓ objdump：显示 ELF 文件的信息。
  - ✓ readelf：显示更多 ELF 格式文件的信息（包括 DWARF 调试信息）。
  - ✓ .....

ELF 简介

- ELF（Executable Linkable Format）是一种 Unix-like 系统上的二进制文件格式标准。ELF 标准中定义的采用 ELF 格式的文件分为 4 类：

ELF 文件类型	说明	实例
可重定位文件（Relocatable File）	内容包含了代码和数据，可以被链接成可执行文件或共享目标文件。	Linux 上的 .o 文件
可执行文件（Executable File）	可以直接执行的程序	Linux 上的 a.out
共享目标文件（Shared Object File）	内容包含了代码和数据，可以作为链接器的输入，在链接阶段和其他的 Relocatable File或者 Shared Object File一起链接成新的 Object File；或者在运行阶段，作为动态链接器的输入，和 Executable File 结合，作为进程的一部分来运行	Linux 上的 .so
核心转储文件（Core Dump File）	进程意外终止时，系统可以将该进程的部分内容和终止时的其他状态信息保存到该文件中以供调试分析。	Linux 上的 core 文件

02

交叉编译

什么是交叉编译

- 假设我们在一个**平台** A 上执行编译和链接，生成的程序（二进制文件）用于在**平台** B 上运行。**平台** 指不同的 CPU 架构的机器 **和** 操作系统的组合。
- 如果平台 A 和 平台 B 相同，则该编译和链接过程我们称之为 **本地编译（Native Compile）**；
- 如果平台 A 和 平台 B 不同，则该编译和链接过程我们称之为 **交叉编译（Cross Compile）**。
- 支持“本地编译”的工具链称之为 **本地工具链**，反之支持“交叉编译”的工具链则称之为 **交叉工具链**。

例子	本地编译	交叉编译
平台 A	x86_64 + Linux	x86_64 + Windows
平台 B	x86_64 + Linux	riscv64 + Linux

为什么需要交叉编译

- **目标机资源有限**：嵌入式设备（如路由器、智能手表、IoT设备）的 CPU 性能弱、内存小、没有操作系统或存储空间不足，无法在其上直接运行庞大的编译器。
- **提高开发效率**：在性能强大的平台上编译，速度远快于在性能羸弱的平台上编译。
- **统一构建环境**：在同一个服务器上进行自动化构建，可以为多种不同的平台生成程序，保证环境一致。

交叉编译涉及的三个核心概念

- **构建（Build）系统**：执行构建的平台。
- **主机（Host）系统**：这是运行构建结果的平台。对于本地编译，Build 系统和 Host 系统相同，对于交叉编译，Build 系统和 Host 系统不同。
- **目标（Target）系统**：只对构建结果是 gcc/binutils 这类特殊软件有意义，gcc/binutils 这些软件的特殊之处在于它们将被进一步用于构建新的软件，而目标系统描述了这些新的软件将运行在什么样的平台上。

交叉编译涉及的三个核心概念

- **构建（Build）系统**：执行构建的
- **主机（Host）系统**：这是运行构
，Build 系统和 Host 系统相同，对于交叉编
不同。
- **目标（Target）系统**：只对构建结果是 gcc/binutils 这类特殊软件有意义，gcc/binutils 这些软件的特殊之处
建新的软件，而目标系统描述了这些新的软件将

./configure --build=……

./configure --host=……

./configure --target=……

交叉编译涉及的三个核心概念

- **构建（Build）系统**：执行构建的
- **主机（Host）系统**：这是运行构建的
- **目标（Target）系统**：只对构建结果是 gcc/binutils 这类特殊软件有意义，gcc/binutils 这些软件的特殊之处
而目标系统描述了这些新的软件将

./configure --build=……

./configure --host=……

./configure --target=……

--build/--host/--target 的取值格式为三/四元组(triplet/quadruplet)，遵循以下模式：

arch[-vendor]-os[-abi]

例子：

- x86_64-pc-linux-gnu
- riscv64-unknown-linux-gnu

交叉编译涉及的三个核心概念

- **构建（Build）系统**：执行构建的平台。
- **主机（Host）系统**：这是运行构建结果的平台。对于本地编译，Build 系统和 Host 系统相同，对于交叉编译，Build 系统和 Host 系统不同。
- **目标（Target）系统**：只对构建结果是 gcc/binutils 这类特殊软件有意义，gcc/binutils 这些软件的特殊之处在于它们将被进一步用于构建新的软件，而目标系统描述了这些新的软件将运行在什么样的平台上。

例子 1：本地编译 pkgconf	平台
Build	--build=x86_64-pc-linux-gnu
Host	--host=x86_64-pc-linux-gnu
Target	N/A

交叉编译涉及的三个核心概念

- **构建（Build）系统**：执行构建的平台。
- **主机（Host）系统**：这是运行构建结果的平台。对于本地编译，Build 系统和 Host 系统相同，对于交叉编译，Build 系统和 Host 系统不同。
- **目标（Target）系统**：只对构建结果是 gcc/binutils 这类特殊软件有意义，gcc/binutils 这些软件的特殊之处在于它们将被进一步用于构建新的软件，而目标系统描述了这些新的软件将运行在什么样的平台上。

例子 2：本地编译用于交叉编译的 gcc	平台
Build	--build=x86_64-pc-linux-gnu
Host	--host=x86_64-pc-linux-gnu
Target	--target=riscv64-unknown-linux-gnu

交叉编译涉及的三个核心概念

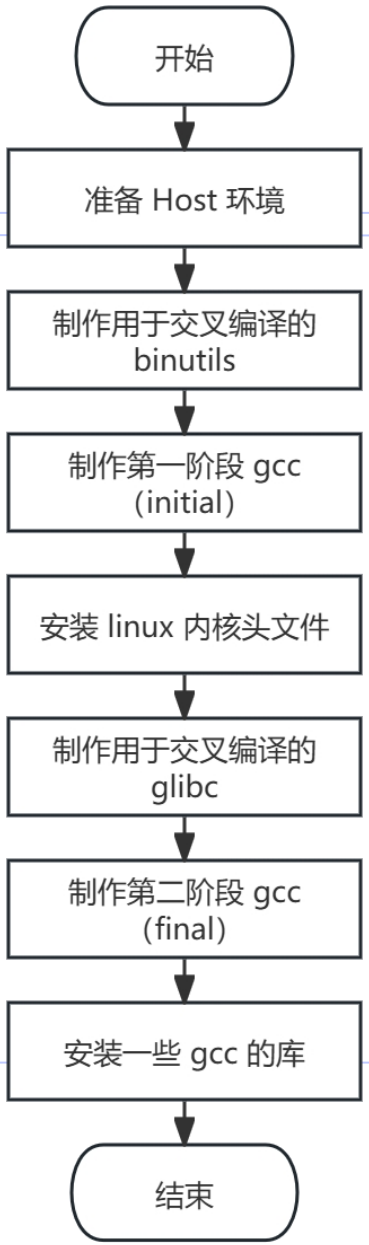
- **构建（Build）系统**：执行构建的平台。
- **主机（Host）系统**：这是运行构建结果的平台。对于本地编译，Build 系统和 Host 系统相同，对于交叉编译，Build 系统和 Host 系统不同。
- **目标（Target）系统**：只对构建结果是 gcc/binutils 这类特殊软件有意义，gcc/binutils 这些软件的特殊之处在于它们将被进一步用于构建新的软件，而目标系统描述了这些新的软件将运行在什么样的平台上。

例子 3：交叉编译 glibc	平台
Build	--build=x86_64-pc-linux-gnu
Host	--host=riscv64-unknown-linux-gnu
Target	N/A

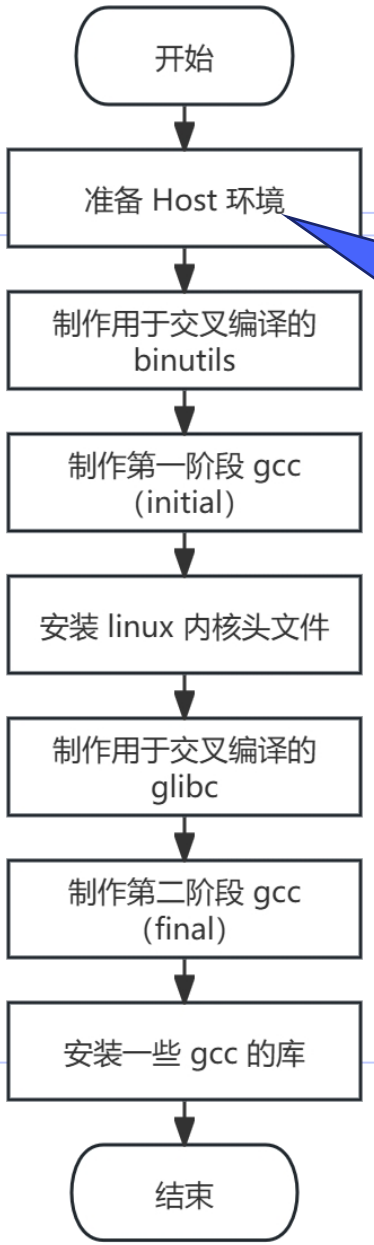
03

制作交叉工具链

制作交叉工具链的步骤

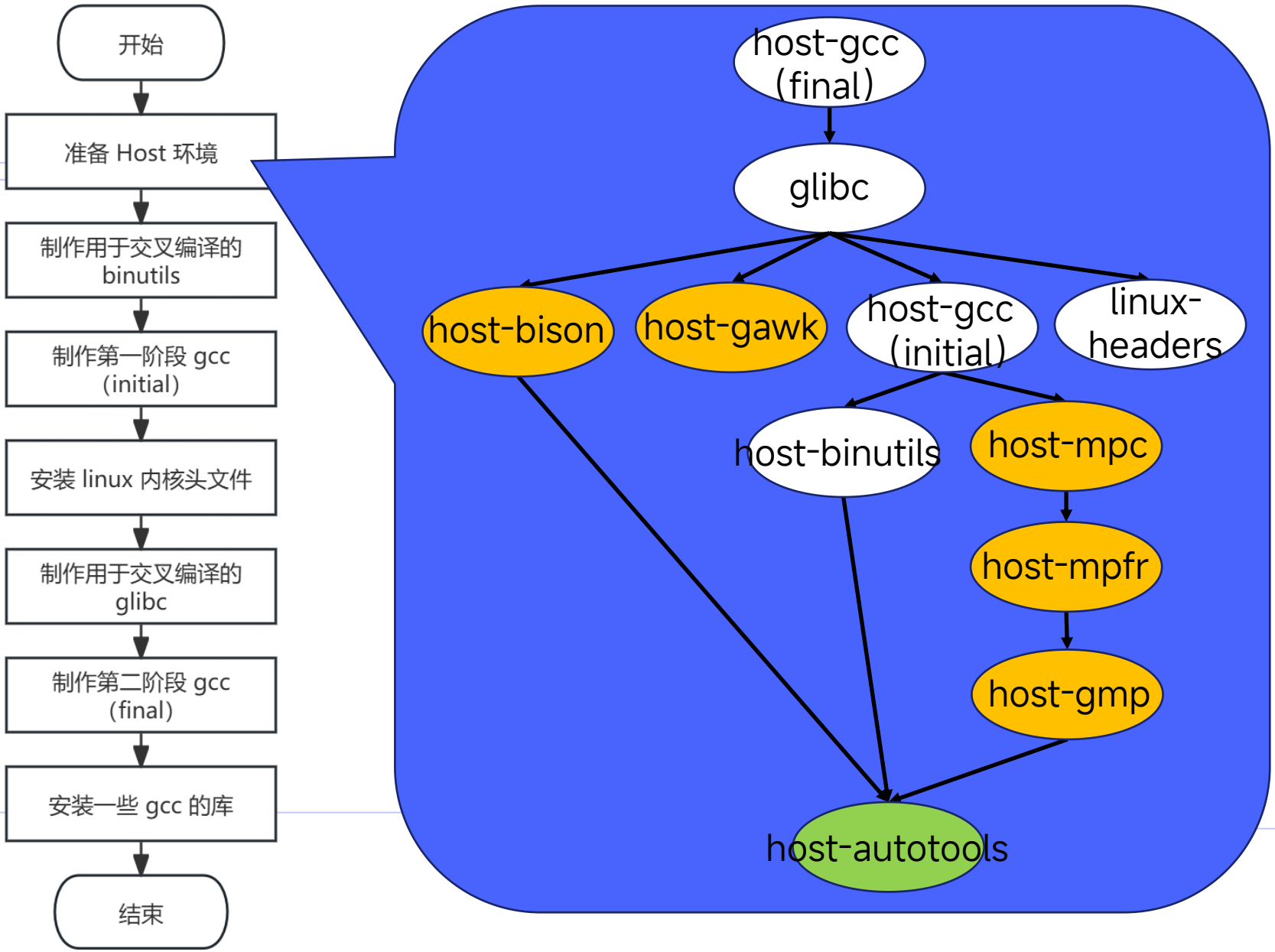

开始
准备 Host 环境
制作用于交叉编译的 binutils
制作第一阶段 gcc (initial)
安装 linux 内核头文件
制作用于交叉编译的 glibc
制作第二阶段 gcc (final)
安装一些 gcc 的库
结束

制作交叉工具链的步骤



制作 gcc 和 glibc 过程中会用到一些运行在本地主机上的工具（譬如 bison/gawk/gmp/mpfr/mpc ），原主机系统可能没有或者版本太低，做好后安装在 ${HOST_DIR} 下。

制作交叉工具链的步骤


开始
准备 Host 环境
制作用于交叉编译的 binutils
制作第一阶段 gcc (initial)
安装 linux 内核头文件
制作用于交叉编译的 glibc
制作第二阶段 gcc (final)
安装一些 gcc 的库
结束
host-gcc (final)
glibc
host-bison
host-gawk
host-gcc (initial)
linux-headers
host-binutils
host-mpc
host-mpfr
host-gmp
host-autotools

Bison - 简介

https://www.gnu.org/software/bison/

- GNU Bison 是一个通用语法分析器生成器（Parser Generator），通过解析输入的 BNF（巴科斯范式）或类似格式的语法规则（*.y），输出可嵌入应用程序的（C、C++ 等形式的）语法分析代码。
- Bison 是用于构建编程语言编译器、解释器或其他文本处理工具的基础核心工具，为 GCC/Glibc 所使用。
- 注意：Bison 是 构建时依赖，不是运行时依赖。

Bison -制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} ac_cv_libtextstyle=no
CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --
disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --
disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --
disable-dependency-tracking  --enable-relocatable"
```

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j1  -C ${PKGBUILD_DIR}"
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j1  install -C ${PKGBUILD_DIR}"
```

Bison -制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} ac_cv_libtextstyle=no
CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=${HOST_DIR}/var\" --enable-shared --
disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --
disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --
disable-dependency-tracking  --enable-relocatab
```

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -
```

- 这是一个 autoconf 的环境变量。
- libtextstyle: 指 GNU libtextstyle 库，提供文本样式化/格式化功能
- 设置为“no”表示不启用/跳过检测该库。减少不必要的依赖，最小化安装。

Bison – 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} ac_cv_libtextstyle=no
CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --
disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --
disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --
disable-dependency-tracking  --enable-relocatable"
```

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j1
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j1
```

选项	含义/用途
--enable-relocatable	• 允许将软件安装到任意目录，整体移动到其他位置也不影响软件的运行。注意和编译选项 –fPIC 不是一个概念。 • 需要软件本身支持该特性。

Gawk - 简介

https://www.gnu.org/software/gawk/

- Gawk 是 AWK 语言的 GNU 实现，AWK 得名于三位创始人（Aho, Weinberger, Kernighan）的姓氏首字母。
- Gawk 自动逐行读取输入文件，检查每行是否匹配程序中定义的各个“模式”，若匹配则执行对应的“动作”
- Gawk 的设计目标是用较少的代码完成复杂的文本处理任务，譬如自动将每行文本按分隔符；从文本中过滤、提取特定信息，并重新格式化输出等。
- Glibc 的构建依赖于 Gawk。

Gawk – 制作步驟

配置

```
eval "${HOST_CONFIGURE_OPTS} CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking  --without-readline --without-mpfr"
```

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS}  -C ${PKGBUILD_DIR}"
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install -C ${PKGBUILD_DIR}"
```

Gawk – 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking  --without-readline --without-mpfr"
```

构建

```
eval "${HOST_MAKE_ENV}
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV
```

选项	含义/用途
--without-readline	• 明确构建时不要使用/依赖 readline 库 • Readline 是一个提供命令行编辑和历史记录功能的库。它允许用户在命令行中使用方向键移动光标、编辑命令行、搜索历史命令等，这对于制作和使用 gawk 不是必需的，可以忽略。
--without-mpfr	• 明确构建时不要使用/依赖 mpfr 库 • MPFR（Multiple Precision Floating-Point Reliable）是一个用于高精度浮点计算的库。，同样对于 gawk 来说不是必需的，可以忽略。

Gmp - 简介

https://gmplib.org/

- GNU MP 库，简称 GMP（全称 GNU Multiple Precision Arithmetic Library）是一个用于执行高精度数学运算的免费开源 C 语言库。
- 它的核心特点是超越硬件限制，让你能处理任意大（只受限于计算机内存）的整数、有理数和浮点数运算。
- GMP 是许多著名开源项目的基石，如 GCC，MPFR 等。

Gmp - 制作步骤

配置

```
eval "AUTOPOINT=/bin/true ${AUTORECONF_OPTS}  ${HOST_DIR}/bin/autoreconf -f -i"

patch_libtool ${PKGBUILD_DIR}

eval "${HOST_CONFIGURE_OPTS} CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking"
```

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS}  -C ${PKGBUILD_DIR}"
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install -C ${PKGBUILD_DIR}"
```

Gmp – 制作步骤

配置

```
eval "AUTOPOINT=/bin/true ${AUTORECONF_OPTS}  ${HOST_DIR}/bin/autoreconf -f -i"

patch_libtool ${PKGBUILD_DIR}

eval "${HOST_CONFIGURE_OPTS} C                e --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --
disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-
debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking"
```

应用补丁后需要重建 configure。

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS}  -C ${PKGBUILD_DIR}"
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install -C ${PKGBUILD_DIR}"
```

Gmp - 制作步骤

配置

```
eval "AUTOPOINT=/bin/true ${AUTORECONF_OPTS}  ${HOST_DIR}/bin/autoreconf -f -i"

patch_libtool ${PKGBUILD_DIR}

eval "${HOST_CONFIGURE_C --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/e enable-shared --disable-static --
disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-
debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking"
```

用 /bin/true 替代真正的 autopoint 命令，跳过多语言支持的处理

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS}  -C ${PKGBUILD_DIR}"
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install -C ${PKGBUILD_DIR}"
```

Gmp - 制作步骤

配置

```
eval "AUTOPOINT=/bin/true ${AUTORECONF_OPTS}  ${HOST_DIR}/bin/autoreconf -f -i"

patch_libtool ${PKGBUILD_DIR}

eval "${HOST_CONFIGURE_OPTS
                                SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/
                                dir=\"${HOST_DIR}/var\" --enable-shared --disable-static --
disable-gtk-d
                                tation --disable-
debug --wit
```

构建

```
eval "${HOS
```

安装 （insta

```
eval "${HOS
                                "
```

```
GIT_DIR=. PATH="${HOST_DIR}/bin:${HOST_DIR}/sbin:${PATH}"
PKG_CONFIG="${HOST_DIR}/bin/pkg-config" PKG_CONFIG_SYSROOT_DIR="/"
PKG_CONFIG_ALLOW_SYSTEM_CFLAGS=1 PKG_CONFIG_ALLOW_SYSTEM_LIBS=1
PKG_CONFIG_LIBDIR="${HOST_DIR}/lib/pkgconfig:${HOST_DIR}/share/pkgconfig"
AR="/usr/bin/ar" AS="/usr/bin/as" LD="/usr/bin/ld" NM="/usr/bin/nm" CC="/usr/bin/gcc"
GCC="/usr/bin/gcc" CXX="/usr/bin/g++" CPP="/usr/bin/cpp" OBJCOPY="/usr/bin/objcopy"
RANLIB="/usr/bin/ranlib" CPPFLAGS="-I${HOST_DIR}/include" CFLAGS="-O2 -
I${HOST_DIR}/include" CXXFLAGS="-O2 -I${HOST_DIR}/include" LDFLAGS="-L${HOST_DIR}/lib
-Wl,-rpath,${HOST_DIR}/lib" INTLTOOL_PERL=/usr/bin/perl
ACLOCAL="${HOST_DIR}/bin/aclocal" AUTOCONF="${HOST_DIR}/bin/autoconf -I
"${STAGING_DIR}/usr/share/aclocal" -I "${HOST_DIR}/share/aclocal""
AUTOHEADER="${HOST_DIR}/bin/autoheader -I "${STAGING_DIR}/usr/share/aclocal" -I
"${HOST_DIR}/share/aclocal"" AUTOMAKE="${HOST_DIR}/bin/automake" GTKDOCIZE=/bin/true
```

Gmp - 制作步骤

配置

```
eval "AUTOPOINT=/bin/true ${AUTORECONF_OPTS}  ${HOST_DIR}/bin/autoreconf -f -i"

patch_libtool ${PKGBUILD_DIR}

eval "${HOST_CONFIGURE_OPTS}                SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/                   dir=\"${HOST_DIR}/var\" --enable-shared --disable-static --
disable-gtk-d                                                                    tation --disable-
debug --wi
```

构建

```
eval "${HOS
```

安装 （insta

```
eval "${HOS                                                                      "
```

```
GIT_DIR=. PATH="${HOST_DIR}/bin:${HOST_DIR}/sbin:${PATH}"
PKG_CONFIG="${HOST_DIR}/bin/pkg-config" PKG_CONFIG_SYSROOT_DIR="/"
PKG_CONFIG_ALLOW_SYSTEM_CFLAGS=1 PKG_CONFIG_ALLOW_SYSTEM_LIBS=1
PKG_CONFIG_LIBDIR="${HOST_DIR}/lib/pkgconfig:${HOST_DIR}/share/pkgconfig"
AR="/usr/bin/ar" AS="/usr/bin/as" LD="/usr/bin/ld" NM="/usr/bin/nm" CC="/usr/bin/gcc"
GCC="/usr/bin/gcc" CXX="/usr/bin/g++" CPP="/usr/bin/cpp" OBJCOPY="/usr/bin/objcopy"
RANLIB="/usr/bin/ranlib" CPPFLAGS="-I${HOST_DIR}/include" CFLAGS="-O2 -
I${HOST_DIR}/include" CXXFLAGS="-O2 -I${HOST_DIR}/include" LDFLAGS="-L${HOST_DIR}/lib
-Wl,-rpath,${HOST_DIR}/lib" INTLTOOL_PERL=/usr/bin/perl
ACLOCAL="${HOST_DIR}/bin/aclocal" AUTOCONF="${HOST_DIR}/bin/autoconf -I
"${STAGING_DIR}/usr/share/aclocal" -I "${HOST_DIR}/share/aclocal""
AUTOHEADER="${HOST_DIR}/bin/autoheader -I "${STAGING_DIR}/usr/share/aclocal" -I
"${HOST_DIR}/share/aclocal"" AUTOMAKE="${HOST_DIR}/bin/automake" GTKDOCIZE=/bin/true
```

Gmp - 制作步骤

配置

```
eval "AUTOPOINT=/bin/true ${AUTORECONF_OPTS}  ${HOST_DIR}/bin/autoreconf -f -i"

patch_libtool ${PKGBUILD_DIR}

eval "${HOST_CONFIGURE_OPTS}
sysconfdir=\"${HOST_DIR}/
disable-gtk-d
debug --wi
```

构建

```
eval "${HOS
```

安装 （insta

```
eval "${HOS
```

GIT_DIR=. PATH="${HOST_DIR}/bin:${HOST_DIR}/sbin:${PATH}"
PKG_CONFIG="${HOST_DIR}/bin/pkg-config" PKG_CONFIG_SYSROOT_DIR="/"
PKG_CONFIG_ALLOW_SYSTEM_CFLAGS=1 PKG_CONFIG_ALLOW_SYSTEM_LIBS=1
PKG_CONFIG_LIBDIR="${HOST_DIR}/lib/pkgconfig:${HOST_DIR}/share/pkgconfig"
AR="/usr/bin/ar" AS="/usr/bin/as" LD="/usr/bin/ld" NM="/usr/bin/nm" CC="/usr/bin/gcc"
GCC="/usr/bin/gcc" CXX="/usr/bin/g++" CPP="/usr/bin/cpp" OBJCOPY="/usr/bin/objcopy"
RANLIB="/usr/bin/ranlib" CPPFLAGS="-I${HOST_DIR}/include" CFLAGS="-O2 -
I${HOST_DIR}/include" CXXFLAGS="-O2 -I${HOST_DIR}/include" LDFLAGS="-L${HOST_DIR}/lib
-Wl,-rpath,${HOST_DIR}/lib" INTLTOOL_PERL=/usr/bin/perl
ACLOCAL="${HOST_DIR}/bin/aclocal" AUTOCONF="${HOST_DIR}/bin/autoconf -I
"${STAGING_DIR}/usr/share/aclocal" -I "${HOST_DIR}/share/aclocal""
AUTOHEADER="${HOST_DIR}/bin/autoheader -I "${STAGING_DIR}/usr/share/aclocal" -I
"${HOST_DIR}/share/aclocal"" AUTOMAKE="${HOST_DIR}/bin/automake" GTKDOCIZE=/bin/true

$ {HOST_CONFIGURE_OPTS}

Gmp - 制作步骤

配置

```
eval "AUTOPOINT=/bin/true ${AUTORECONF_OPTS}  ${HOST_DIR}/bin/autoreconf -f -i"

patch_libtool ${PKGBUILD_DIR}

eval "${HOST_CONFIGURE_OPTS}                    SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/                       dir=\"${HOST_DIR}/var\" --enable-shared --disable-static --
disable-gtk-d                                                                          tation --disable-
debug --wit
```

构建

```
eval "${HOS
```

安装 （insta

```
eval "${HOS                                                                                    "
```

```
GIT_DIR=. PATH="${HOST_DIR}/bin:${HOST_DIR}/sbin:${PATH}"
PKG_CONFIG="${HOST_DIR}/bin/pkg-config" PKG_CONFIG_SYSROOT_DIR="/"
PKG_CONFIG_ALLOW_SYSTEM_CFLAGS=1 PKG_CONFIG_ALLOW_SYSTEM_LIBS=1
PKG_CONFIG_LIBDIR="${HOST_DIR}/lib/pkgconfig:${HOST_DIR}/share/pkgconfig"
AR="/usr/bin/ar" AS="/usr/bin/as" LD="/usr/bin/ld" NM="/usr/bin/nm" CC="/usr/bin/gcc"
GCC="/usr/bin/gcc" CXX="/usr/bin/g++" CPP="/usr/bin/cpp" OBJCOPY="/usr/bin/objcopy"
RANLIB="/usr/bin/ranlib" CPPFLAGS="-I${HOST_DIR}/include" CFLAGS="-O2 -
I${HOST_DIR}/include" CXXFLAGS="-O2 -I${HOST_DIR}/include" LDFLAGS="-L${HOST_DIR}/lib
-Wl,-rpath,${HOST_DIR}/lib" INTLTOOL_PERL=/usr/bin/perl
ACLOCAL="${HOST_DIR}/bin/aclocal" AUTOCONF="${HOST_DIR}/bin/autoconf -I
"${STAGING_DIR}/usr/share/aclocal" -I "${HOST_DIR}/share/aclocal""
AUTOHEADER="${HOST_DIR}/bin/autoheader -I "${STAGING_DIR}/usr/share/aclocal" -I
"${HOST_DIR}/share/aclocal"" AUTOMAKE="${HOST_DIR}/bin/automake" GTKDOCIZE=/bin/true
```

Gmp - 制作步骤

配置

```
eval "AUTOPOINT=/bin/true ${AUTORECONF_OPTS}  ${HOST_DIR}/bin/autoreconf -f -i"

patch_libtool ${PKGBUILD_DIR}

eval "${HOST_CONFIGURE_OPTS}                    SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/                       dir=\"${HOST_DIR}/var\" --enable-shared --disable-static --
disable-gtk-d                                                                          tation --disable-
debug --wit
```

构建

```
eval "${HOS
```

安装 （insta

```
eval "${HOS                                                                                    "
```

```
GIT_DIR=. PATH="${HOST_DIR}/bin:${HOST_DIR}/sbin:${PATH}"
PKG_CONFIG="${HOST_DIR}/bin/pkg-config" PKG_CONFIG_SYSROOT_DIR="/"
PKG_CONFIG_ALLOW_SYSTEM_CFLAGS=1 PKG_CONFIG_ALLOW_SYSTEM_LIBS=1
PKG_CONFIG_LIBDIR="${HOST_DIR}/lib/pkgconfig:${HOST_DIR}/share/pkgconfig"
AR="/usr/bin/ar" AS="/usr/bin/as" LD="/usr/bin/ld" NM="/usr/bin/nm" CC="/usr/bin/gcc"
GCC="/usr/bin/gcc" CXX="/usr/bin/g++" CPP="/usr/bin/cpp" OBJCOPY="/usr/bin/objcopy"
RANLIB="/usr/bin/ranlib" CPPFLAGS="-I${HOST_DIR}/include" CFLAGS="-O2 -
I${HOST_DIR}/include" CXXFLAGS="-O2 -I${HOST_DIR}/include" LDFLAGS="-L${HOST_DIR}/lib
-Wl,-rpath,${HOST_DIR}/lib" INTLTOOL_PERL=/usr/bin/perl
ACLOCAL="${HOST_DIR}/bin/aclocal" AUTOCONF="${HOST_DIR}/bin/autoconf -I
"${STAGING_DIR}/usr/share/aclocal" -I "${HOST_DIR}/share/aclocal""
AUTOHEADER="${HOST_DIR}/bin/autoheader -I "${STAGING_DIR}/usr/share/aclocal" -I
"${HOST_DIR}/share/aclocal"" AUTOMAKE="${HOST_DIR}/bin/automake" GTKDOCIZE=/bin/true
```

MPFR - 简介

https://www.mpfr.org/

- GNU MPFR（Multiple Precision Floating-Point Reliable）是一个提供高精度可靠浮点数运算的 C 语言库。
- 相较于 GMP 保证了整数运算的正确性，MPFR 的核心使命就是为浮点数运算提供“数学上可靠”的结果。遵循 IEEE 754 标准，支持精度任意可调（仅受内存限制），并提供多种舍入模式并保证结果正确的舍入。
- MPFR 基于 GMP 库构建。
- MPFR 被广泛应用数学常数计算、数值分析研究等。
- MPFR 被 GCC 所使用，用于编译时对浮点常量表达式求值。

MPFR – 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking"
```

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS}  -C ${PKGBUILD_DIR}"
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install -C ${PKGBUILD_DIR}"
```

MPC – 简介

https://www.multiprecision.org/mpc/

- GNU MPC（ Multi-Precision Complex ）库是 GNU 项目下用于高精度复数运算的 C 语言库。
- MPC 将处理浮点数的 IEEE-754 标准扩展到了复数域，提供任意高精度的复数运算，并保证结果的正确舍入。
- 构建在 GNU MPFR（高精度浮点库）之上。
- MPC 被用作 GCC 编译器工具链的一部分。

MPC – 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking"
```

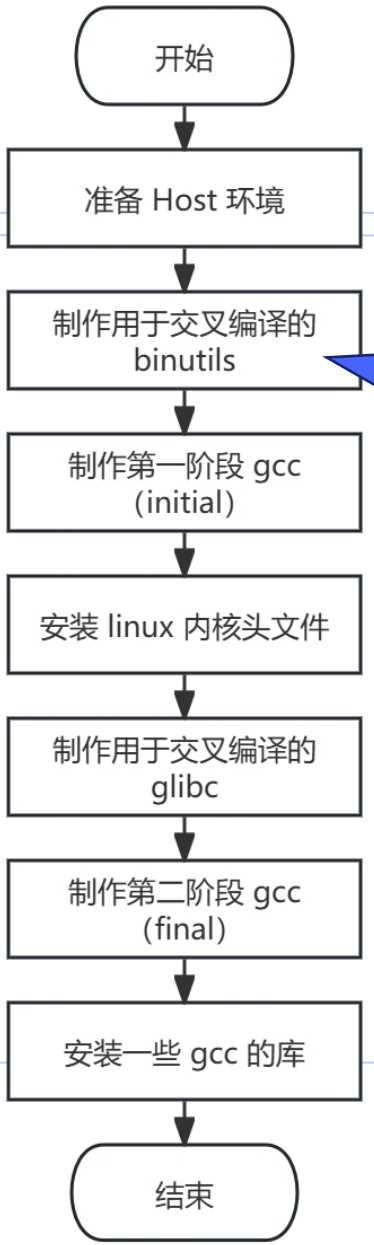
构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS}  -C ${PKGBUILD_DIR}"
```

安装 （install-host）

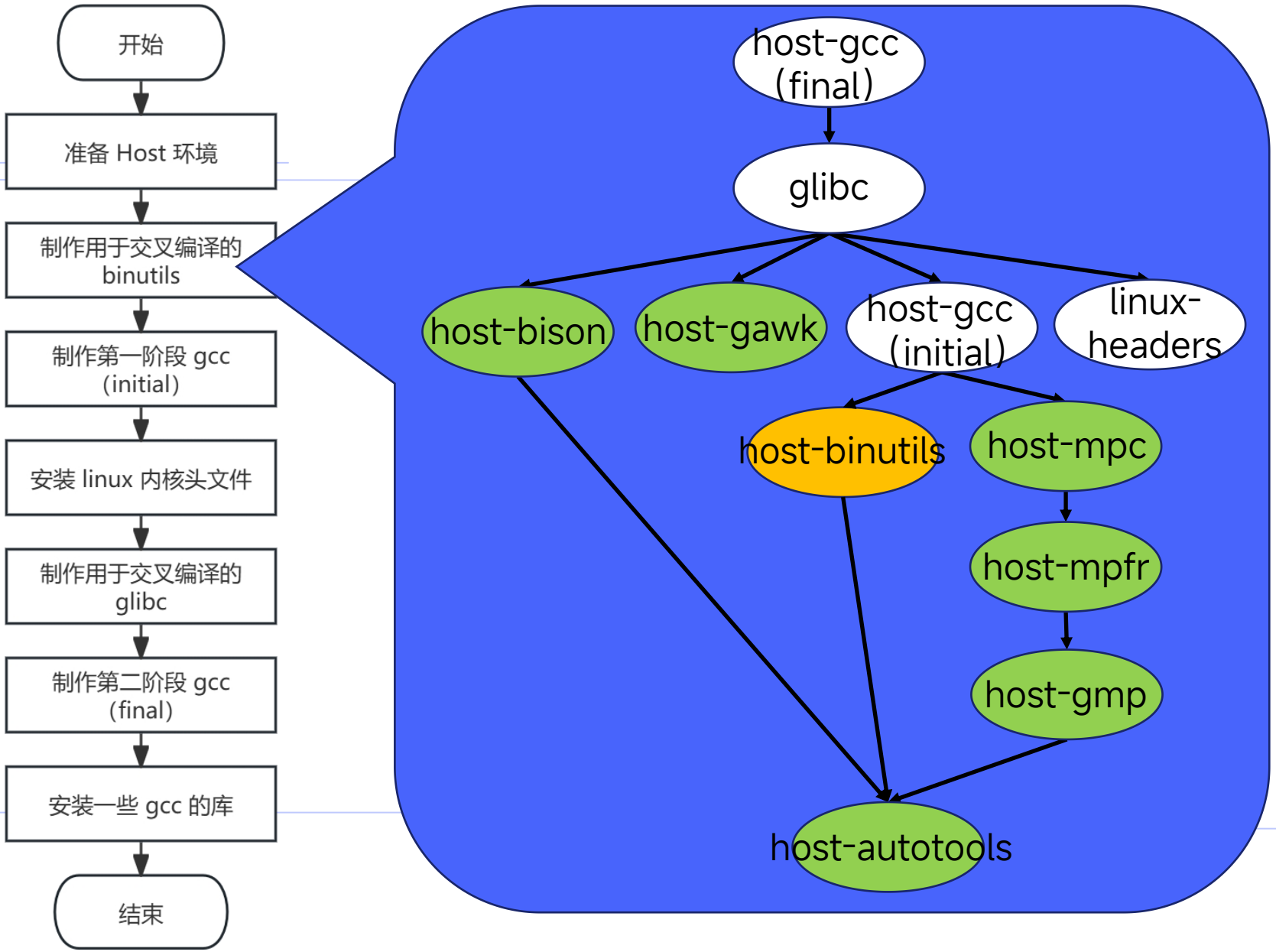
```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install -C ${PKGBUILD_DIR}"
```

制作交叉工具链的步骤



- 制作用于 **交叉编译** 的 binutils，该 binutils 将用于制作需要安装在 Target 系统（riscv64 linux）上并在 Target 系统上运行的软件模块，譬如这里的 glibc 以及后面的所有需要安装到 ${TARGET_DIR} 上的软件。
- 制作 binutils 使用 **Build 系统** 的 **本地工具链**。

制作交叉工具链的步骤


开始
准备 Host 环境
制作用于交叉编译的 binutils
制作第一阶段 gcc (initial)
安装 linux 内核头文件
制作用于交叉编译的 glibc
制作第二阶段 gcc (final)
安装一些 gcc 的库
结束
host-gcc (final)
glibc
host-bison
host-gawk
host-gcc (initial)
linux-headers
host-binutils
host-mpc
host-mpfr
host-gmp
host-autotools

Binutils – 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} MAKEINFO=true CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking  --disable-multilib --disable-werror --target=${GNU_TARGET_NAME} --disable-shared --enable-static --with-sysroot=${STAGING_DIR} --enable-poison-system-directories --without-debuginfod --enable-plugins --enable-lto --disable-sim --disable-gdb  --without-zstd --with-system-readline --disable-gprofng"
```

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} MAKEINFO=true -C ${PKGBUILD_DIR}"
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} MAKEINFO=true install -C ${PKGBUILD_DIR}"
```

Binutils - 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} MAKEINFO=true CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking  --disable-multilib --disable-werror --target=${GNU_TARGET_NAME} --disable-shared --enable-static --with-sysroot=${STAGING_DIR} --enable-poison-system-directories --without-debuginfod --enable-plugins --enable-lto --disable-gdb  --without-zstd --with-system-readline --disable-gprofng"
```

选项	含义/用途
--disable-multilib	禁止多架构或多 ABI 支持，我们只支持 riscv64
--disable-werror	不将编译警告（warning）视为错误（error）

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} MAKEINFO=true -C ${PKGBUILD_DIR}"
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} MAKEINFO=true install -C ${PKGBUILD_DIR}"
```

Binutils – 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} MAKEINFO=true CONFIG_SITE=/dev/null ./configure --
prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-
shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-
documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-
tracking  --disable-multilib --disable-werror --target=${GNU_TARGET_NAME} --disable-shared --enable-
static --with-sysroot=${STAGING_DIR} --enable-pacon-system-directories --without-debuginfod --
enable-plugins --enable-lto --disable-sim --di                --without-zstd --with-system-readline --
disable-gprofng"
```

选项	含义/用途
--target	• 制作交叉工具链的关键选项之一，用于指定 Target 系统架构，定义了这里制作的交叉工具链后期将用于构建在什么平台上运行的软件 • GNU_TARGET_NAME=riscv64-unknown-linux-gnu • 这里没有指定 --host 和 --build，所以默认它们的值就是和本地构建系统相同（x86_64 的 linux）

构建

```
eval "${HOST_MAKE_ENV} /usr/bi                                                    UILD_DIR}"
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/                                                  ${PKGBUILD_DIR}"
```

Binutils - 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} MAKEINFO=true CONFIG_SITE=/dev/null ./configure --
prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-
shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-
documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-
tracking  --disable-multilib --disable-werror --target=${GNU_TARGET_NAME} --disable-shared --enable-
static --with-sysroot=${STAGING_DIR} --enable-poison-system-directories --without-debuginfod --
enable-plugins --enable-lto --disable-sim --disable-gdb  --without-zstd --with-[illegible]-readline --
disable-gprofng"
```

构建

```
eval "${HOST_MAKE_ENV} /usr/b[illegible]D_DIR}"
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/b[illegible]GBUILD_DIR}"
```

- 在 GNU Autotools 的 configure 脚本中，后出现的选项会覆盖前面的选项，这也是 Unix/Linux 命令行参数的通用规则。所以这里的效果就是对于 binutils 来说将只生成静态库而不会生成动态库。
- 确保生成的交叉工具 binutils 是自包含的，静态链接到它们依赖的库（例如 libiberty、libbfd 和 libopcodes）。这些工具将不依赖于主机系统上存在的共享库，实现了与主机系统隔离。
- 同时简化制作工具链后续阶段的依赖关系。

Binutils - 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} MAKEINFO=true CONFIG_SITE=/dev/null ./configure --
prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-
shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-
documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-
tracking  --disable-multilib --disable-werror --target=${GNU_TARGET_NAME} --disable-shared --enable-
static --with-sysroot=${STAGING_DIR} --enable-poison-system-directories --without-debuginfod --
enable-plugins --enable-lto --disab      --disable-gdb  --without-zstd --with-system-readline --
disable-gprofng"
```

构建

```
eval "${HOST_MAKE_EN
```

安装 （install-host）

```
eval "${HOST_MAKE_EN                                                    _DIR}"
```

选项	含义/用途
--with-sysroot	• 制作交叉工具链的关键的选项之一。配置时指定了 --with-sysroot 后，在制作交叉工具链（如这里的 binutils）时，其值会被记录到交叉工具链程序内部；确保我们在运行交叉工具链时会去 --with-sysroot 指定的目录下寻找需要的头文件或者依赖库，而不是在主机系统的 /usr/include、/lib 中查找。 • ${STAGING_DIR} 目录下安装了用于编译和链接 Target 系统上二进制文件所需的依赖文件（头文件、库文件等）； • 同时，当我们运行构建出的交叉工具链进行交叉编译时，可以避免重复指定额外的 –I 或者 –L 通知交叉工具链去 ${STAGING_DIR} 目录下搜索需要的依赖文件。
--enable-poison-system-directories	明确告诉 binutils，在交叉编译过程中不要使用 Host 系统的标准目录（如 /usr/include、/lib 等），只使用 –with-sysroot 指定的 ${STAGING_DIR} 目录下的文件。是 GCC 配置中一个安全防护选项，用于防止编译器意外使用宿主系统的头文件和库文件。

Binutils – 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} MAKEINFO=true CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking  --disable-multilib --disable-werror --target=${GNU_TARGET_NAME} --disable-shared --enable-static --with-sysroot=${STAGING_DIR} --enable-poison-system-directories --without-debuginfod --enable-plugins --enable-lto --disable-sim --disable-gdb  --without-zstd --with-system-readline --disable-gprofng"
```

构建

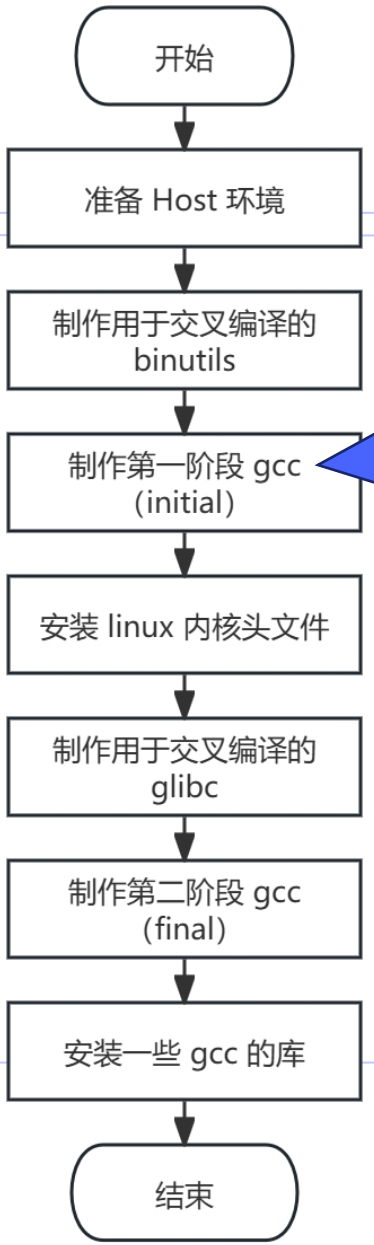
```
eval "${HOST_MAKE_ENV} /usr/bir
```

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bir                                                                                    DIR}"
```

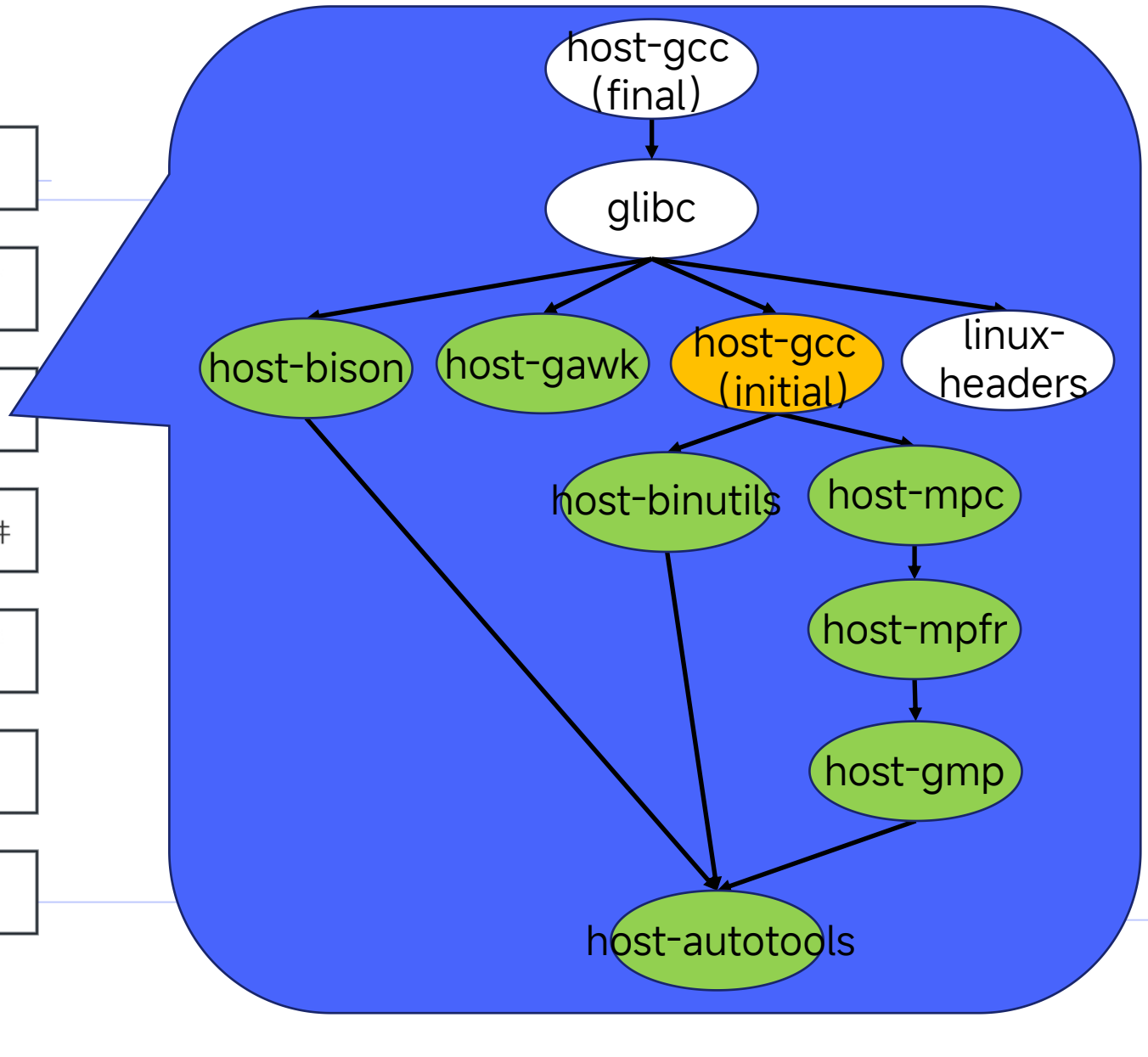
选项	含义/用途
--without-debuginfod	禁用网络获取调试信息的功能，减少构建依赖，提高构建效率
--enable-lto	启用链接时优化（Link Time Optimization）
--disable-sim	禁用模拟器（SIMulator）支持；不支持调试，提高构建效率
--disable-gdb	避免构建 gdb，兼容旧的 binutils 软件包（可能包含 GDB）
--without-zstd	禁用 Zstandard 压缩支持
--with-system-readline	使用 Host 系统已安装的 readline 而不要编译下载的 binutils 软件包内置的 readline，binutils 本身并不使用 readline（gdb 会使用）
--disable-gprofng	禁用新一代 GNU 性能分析器（GNU profiler next generation）

制作交叉工具链的步骤



- 制作一个“initial”版本的、非完全功能的、支持 **交叉编译** 的 GCC，只要能够为 **Target 系统** 构建出一个“C 库”即可。。
- 制作 initial GCC 使用 **Build 系统** 的 **本地工具链**。
- 制作 initial GCC 依赖于 **“准备 Host 环境”** 阶段在 **${HOST_DIR}** 下安装的一些工具以及上一步制作的 binutils。

制作交叉工具链的步骤


开始
准备 Host 环境
制作用于交叉编译的 binutils
制作第一阶段 gcc (initial)
安装 linux 内核头文件
制作用于交叉编译的 glibc
制作第二阶段 gcc (final)
安装一些 gcc 的库
结束
host-gcc (final)
glibc
host-bison
host-gawk
host-gcc (initial)
linux-headers
host-binutils
host-mpc
host-mpfr
host-gmp
host-autotools

GCC(initial) – 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} MAKEINFO=missing CFLAGS_FOR_TARGET=\"-
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -
D_FORTIFY_SOURCE=1 \" CXXFLAGS_FOR_TARGET=\"-D_LARGEFILE_SOURCE -
D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1 \"
LDFLAGS_FOR_TARGET=\" \" AR_FOR_TARGET=gcc-ar NM_FOR_TARGET=gcc-nm
RANLIB_FOR_TARGET=gcc-ranlib CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --
disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-
debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking  --
target=${GNU_TARGET_NAME} --with-sysroot=${STAGING_DIR} --enable-__cxa_atexit --with-gnu-ld --
disable-libssp --disable-multilib --disable-decimal-float --enable-plugins --enable-lto --with-
gmp=${HOST_DIR} --with-mpc=${HOST_DIR} --with-mpfr=${HOST_DIR} --without-zstd --disable-
libquadmath --disable-libquadmath-support --enable-tls --enable-threads --without-isl --without-cloog -
-with-arch=\"rv64imafd_zicsr_zifencei\" --with-abi=\"lp64d\" --enable-languages=c --disable-shared --
without-headers --disable-threads --with-newlib --disable-largefile"
```

GCC(initial) - 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} MAKEINFO=missing CFLAGS_FOR_TARGET=\"-
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -
D_FORTIFY_SOURCE=1 \" CXXFLAGS_FOR_TARGET=\"-D_LARGEFILE_SOURCE -
D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1 \"
LDFLAGS_FOR_TARGET=\" \" AR_FOR_TARGET=gcc-ar NM_FOR_TARGET=gcc-nm
RANLIB_FOR_TARGET=gcc-ranlib CON                    ./configure --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/                                                    --
disable-gtk-doc --disable-                                                   e-
debug --with-xmlto=no --v
target=${GNU_TARGET_N
disable-libssp --disable-m
gmp=${HOST_DIR} --with-
libquadmath --disable-libc                                                   g -
-with-arch=\"rv64imafd_zi
without-headers --disable-
```

选项	含义/用途
MAKEINFO=missing	跳过执行 makeinfo
CFLAGS_FOR_TARGET="....."	构建工具链时才会指定，其值会记录在这里新构建出来的交叉编译器 gcc 中，当我们使用它去构建 Target 的 C 程序（如 libgcc）时作为默认编译选项影响 cc1 的行为。
CXXFLAGS_FOR_TARGET="....."	构建工具链时才会指定，其值会记录在这里新构建出来的交叉编译器 gcc 中，当我们使用它去构建 Target 的 C++ 程序（如 libstdc++）时作为默认编译选项影响 cc1plus 的行为。
LDFLAGS_FOR_TARGET=""	构建工具链时才会指定，其值会记录在这里新构建出来的交叉编译器 gcc 中，当我们使用它去链接 Target 的 程序时作为默认链接选项影响 ld 的行为。
AR_FOR_TARGET=gcc-ar	设置处理交叉构建时所使用的运行时库的归档工具 ar
NM_FOR_TARGET=gcc-nm	设置处理交叉构建时所使用的运行时库的符号列表工具 nm
RANLIB_FOR_TARGET=gcc-ranlib	设置处理交叉构建时所使用的运行时库的创建归档索引工具 ranlib

GCC(initial) - 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} MAKEINFO=missing CFLAGS_FOR_TARGET=\"-
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -
D_FORTIFY_SOURCE=1 \" CXXFLAGS_FOR_TARGET=\"-D_LARGEFILE_SOURCE -
D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1 \"
LDFLAGS_FOR_TARGET=\" \" AR_FOR_TARGET=gcc-ar NM_FOR_TARGET=gcc-nm
RANLIB_FOR_TARGET=gcc-ranlib CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --
disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-
debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking  --
target=${GNU_TARGET_NAME} --with-sysroot=${STAGING_DIR} --enable-__cxa_atexit --with-gnu-ld --
disable-libssp --disable-multilib --disable-decimal-float --enable-pl
gmp=${HOST_DIR} --with-mpc=${HOST_DIR} --with-mpfr=${H
libquadmath --disable-libquadmath-support --enable-tls --enab
-with-arch=\"rv64imafd_zicsr_zifencei\" --with-abi=\"lp64d\" --e
without-headers --disable-threads --with-newlib --disable-large
```

- 和构建 binutils 类似
- 尤其关注一下 --build/--host/--target

GCC(initial) - 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} MAKEINFO=missing CFLAGS_FOR_TARGET=\"-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1 \" CXXFLAGS_FOR_TARGET=\"-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1 \" LDFLAGS_FOR_TARGET=\" \" AR_FOR_TARGET=gcc-ar NM_FOR_TARGET=gcc-nm RANLIB_FOR_TARGET=gcc-ranlib CONFIG_SITE=/dev/null ./configure --prefix=\"${HOST_DIR}\" --sysconfdir=\"${HOST_DIR}/etc\" --localstatedir=\"${HOST_DIR}/var\" --enable-shared --disable-static --disable-gtk-doc --disable-gtk-doc-html --disable-doc --disable-docs --disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-nls --disable-dependency-tracking  --target=${GNU_TARGET_NAME} --with-sysroot=${STAGING_DIR} --enable-__cxa_atexit --with-gnu-ld --disable-libssp --disable-multilib --disable-decimal-float --enable-plugins --enable-lto --with-gmp=${HOST_DIR} --with-mpc=${HOST_DIR} --with-mpfr=${HOST_DIR} --without-zstd --disable-libquadmath --disable-libquadmath-support --enable-tls --enable-threads --without-isl --without-cloog --with-arch=\"rv64imafd_zicsr_zifencei\" --with-abi=\"lp64d\" --enable-languages=c --disable-shared --without-headers --disable-threads --with-newlib --disable-largefile"
```

GCC(initi

配置

```
eval "${HOST_CON
D_LARGEFILE_SOU
D_FORTIFY_SOUR
D_LARGEFILE64_S
LDFLAGS_FOR_TA
RANLIB_FOR_TAR
sysconfdir=\"${HO                                                    c --
disable-gtk-doc --                                                   able-
debug --with-xmlt
target=${GNU_TAR                                                     d --
disable-libssp --dis
gmp=${HOST_DIR} --                                                   e-
libquadmath --disable-libquad          support --enable-tls --enable-threads --without-isl --without-cloog -
-with-arch=\"rv64imafd_zicsr_zifencei\" --with-abi=\"lp64d\" --enable-languages=c --disable-shared --
without-headers --disable-threads --with-newlib --disable-largefile"
```

选项	含义/用途
--with-arch="rv64imafd_zicsr_zifencei"	指定目标架构的指令集扩展，当我们使用这里构建的 gcc 去编译 Target（riscv64）机器上运行的二进制程序时，如果没有显式指定 –march，就会用这个默认值。它要求 Target 机器支持： • rv64: 64位 RISC-V 基础指令集 • i: 基础整数指令集 • m: 整数乘除法指令扩展 • a: 原子操作指令扩展（AMO） • f: 单精度浮点指令扩展 • d: 双精度浮点指令扩展 • zicsr: 控制和状态寄存器（CSR）操作指令 • zifencei: 指令存储器屏障（fence.i）指令
--with-abi="lp64d"	指定目标架构的 ABI（Application Binary Interface） • l: long 类型是 64 位 • p: pointer 类型是 64 位 • 64: 表示整个 ABI 针对 64 位架构 • d: 所有浮点类型参数使用双精度浮点寄存器传递 当我们使用这里构建的 gcc 去编译 Target（riscv64）机器上运行的二进制程序时，如果没有显式指定 –mabi，就会用这个默认值。

GCC(initial) - 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} MA
D_LARGEFILE_SOURCE -D_LARGEF
D_FORTIFY_SOURCE=1 \" CXXFLA
D_LARGEFILE64_SOURCE -D_FILE_
LDFLAGS_FOR_TARGET=\" \" AR_F
RANLIB_FOR_TARGET=gcc-ranlib
sysconfdir=\"${HOST_DIR}/etc\" --l
disable-gtk-doc --disable-gtk-doc-l
debug --with-xmlto=no --with-fop=
target=${GNU_TARGET_NAME} --w
disable-libssp --disable-multilib --disa
gmp=${HOST_DIR} --with-mpc=${HOST_DIR} --with-mpfr=${HOST_DIR}
libquadmath --disable-libquadmath-support --enable-tls --enable-threads
-with-arch=\"rv64imafd_zicsr_zifencei\" --with-abi=\"lp64d\" --enable-languages=c --disable-shared --
without-headers --disable-threads --with-newlib --disable-largefile"
```

选项	含义/用途
--enable-languages=c	即将构建的 GCC 只使能支持 C 语言
--disable-shared	只生成静态库，譬如 libgcc.a
--without-headers	即将构建的 GCC 不使用标准 C 头文件
--disable-threads	禁用多线程支持
--with-newlib	告诉 configure 无需检查标准 C 头文件
--disable-largefile	禁用大文件支持（>2GB）

GCC(initial) – 制作步骤

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} gcc_cv_prog_makeinfo_modern=no gcc_cv_libc_provides_ssp=yes all-gcc all-target-libgcc -C ${PKGBUILD_DIR}/build"

/usr/bin/gcc -O2 -I${HOST_DIR}/include -DBR_CROSS_PATH_SUFFIX='".br_real"' -DBR_SYSROOT="\"${STAGING_SUBDIR}\"" -DBR_ADDITIONAL_CFLAGS='"-fstack-protector-strong",' -DBR2_PIC_PIE -DBR2_RELRO_FULL -s -Wl,--hash-style=both ${PROJECT_DIR}/toolchain/toolchain-wrapper.c -o ${PKGBUILD_DIR}/toolchain-wrapper
```

GCC(initial) - 制作步骤

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} gcc_cv_prog_makeinfo_modern=no
gcc_cv_libc_provides_ssp=yes all-gcc all-target-libgcc -C ${PKGBUILD_DIR}/build"

/usr/bin/gcc -O2 -I${H                    -DBR_CROSS_PATH_SUFFIX="" br_real"" -
DBR_SYSR                                                                    ector-strong"," -
DBR2_PIC_                                                                   in/toolchain-
wrapper.c
```

选项	含义/用途
gcc_cv_prog_makeinfo_modern=no	告诉配置系统本地并不支持现代版本的 makeinfo（Texinfo 文档生成工具）所以跳过文档生成，从而加速编译速度。
gcc_cv_libc_provides_ssp=yes	明确指定我们后面要制作的 C 库支持栈溢出保护（Stack Smashing Protection）

GCC(initial) - 制作步骤

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} gcc_cv_prog_makeinfo_modern=no
gcc_cv_libc_provides_ssp=yes all-gcc all-target-libgcc -C ${PKGBUILD_DIR}/build"

/usr/bin/gcc -O2 -I${HOST_DIR}/include -D    CROSS_PATH_SUFFIX='".br_real"' -
DBR_SYSROOT="\"${STAGING_SUBDIR}\"" -    ADDITIONAL_CFLAGS='"-fstack-protector-strong",' -
DBR2_PIC_PIE -DBR2_RELRO_FULL -s -Wl,-    le=both ${PROJECT_DIR}/toolchain/toolchain-
wrapper.c -o ${PKGBUILD_DIR}/toolchain-w
```

- all-gcc: 将制作出在 Host 上运行的交叉编译器
- all-target-libgcc：利用 all-gcc 生成的交叉编译器制作出针对 Target 的 libgcc.a

GCC(initial) – 制作步骤

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} gcc_cv_prog_makeinfo_modern=no
gcc_cv_libc_provides_ssp=yes all-gcc all-target-libgcc -C ${PKGBUILD_DIR}/build"

/usr/bin/gcc -O2 -I${HOST_DIR}/include -DBR_CROSS_PATH_SUFFIX='".br_real"' -
DBR_SYSROOT="\"${STAGING_SUBDIR}\"" -DBR_ADDITIONAL_CFLAGS='"-fstack-protector-strong",' -
DBR2_PIC_PIE -DBR2_RELRO_FULL -s -Wl,--hash-style=both ${PROJECT_DIR}/toolchain/toolchain-
wrapper.c -o ${PKGBUILD_DIR}/toolchain-wrapper
```

- toolchain-wrapper 是一个工具链包装:
  - ✓ 原始调用： riscv64-unknown-linux-gnu-gcc hello.c -o hello
  - ✓ 实际执行： toolchain-wrapper gcc hello.c -o hello
- toolchain-wrapper 用于**透明地处理交叉编译环境**的各种复杂问题:
  - ✓ 环境隔离，防止宿主机的环境变量影响交叉编译过程
  - ✓ 确保编译器在交叉编译时使用目标系统的头文件（STAGING_DIR），而不是 Host 系统
  - ✓ 检查并过滤不安全的编译选项和参数，自动添加必要的架构相关或者是特性相关的选项等。
- 参考了 Buildroot （2025.08.1）的实现。

GCC(initial) – 制作步骤

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install-gcc install-target-libgcc -C ${PKGBUILD_DIR}/build"

/usr/bin/install -D -m 0755 ${PKGBUILD_DIR}/toolchain-wrapper ${HOST_DIR}/bin/toolchain-wrapper

cd ${HOST_DIR}/bin;

for i in ${GNU_TARGET_NAME}-*; do
        case "$i" in
        *.br_real)
                ;;
        *-ar|*-ranlib|*-nm)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        *cc|*cc-*|*++|*++-*|*cpp|*-gfortran|*-gdc)
                rm -f $i.br_real;
                mv $i $i.br_real;
                ln -sf toolchain-wrapper $i;
                ln -sf toolchain-wrapper ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ln -snf $i.br_real ${ARCH}-linux${i##${GNU_TARGET_NAME}}.br_real;
                ;;
        *)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        esac;
done
```

GCC(initial) – 制作步骤

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install-gcc install-target-libgcc -C ${PKGBUILD_DIR}/build"

/usr/bin/install -D -m 0755 ${PKGBUILD_DIR}/toolchain-wrapper ${HOST_D    /bin/toolchain-wrapper

cd ${HOST_DIR}/bin;

for i in ${GNU_TARGET_NAME}-*; do
        case "$i" in
        *.br_real)
                ;;
        *-ar|*-ranlib|*-nm)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME
                ;;
        *cc|*cc-*|*++|*++-*|*cpp|*-gfortran|*-gdc)
                rm -f $i.br_real;
                mv $i $i.br_real;
                ln -sf toolchain-wrapper $i;
                ln -sf toolchain-wrapper ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ln -snf $i.br_real ${ARCH}-linux${i##${GNU_TARGET_NAME}}.br_real;
                ;;
        *)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        esac;
done
```

- installl-gcc: 将 ${PKGBUILD_DIR} 中的编译器安装到 ${HOST_DIR}
- all-target-libgcc：将 ${PKGBUILD_DIR} 中的 libgcc 等库安装到 ${HOST_DIR}

GCC(initial) - 制作步骤

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install-gcc install-target-libgcc -C ${PKGBUILD_DIR}/build"

/usr/bin/install -D -m 0755 ${PKGBUILD_DIR}/toolchain-wrapper ${HOST_DIR}/bin/toolchain-wrapper

cd ${HOST_DIR}/bin;

for i in ${GNU_TARGET_NAME}-*; do
        case "$i" in
        *.br_real)
                ;;
        *-ar|*-ranlib|*-nm)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        *cc|*cc-*|*++|*++-*|*cpp|*-gfortran|*-gdc)
                rm -f $i.br_real;
                mv $i $i.br_real;
                ln -sf toolchain-wrapper $i;
                ln -sf toolchain-wrapper ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ln -snf $i.br_real ${ARCH}-linux${i##${GNU_TARGET_NAME}}.br_real;
                ;;
        *)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        esac;
done
```

将制作出的 toolchain-wrapper 安装到 ${HOST_DIR}

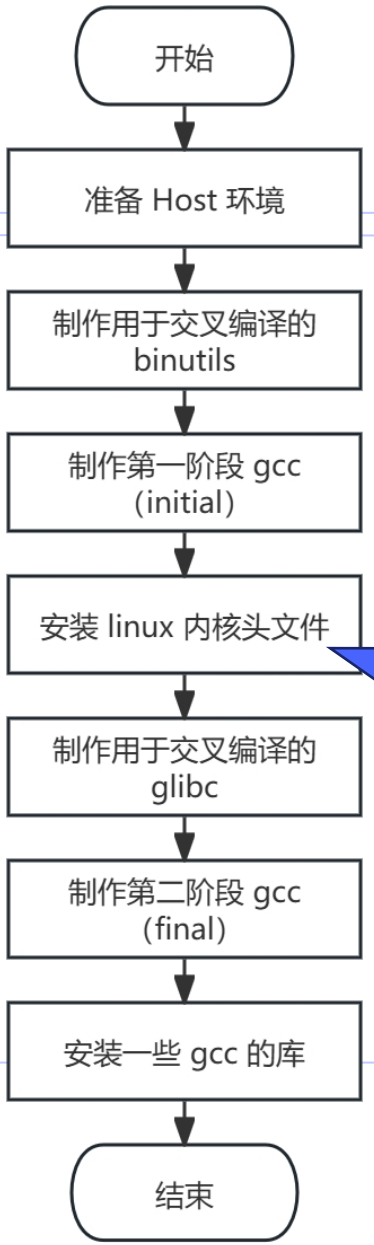
GCC(initial) - 制作步骤

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install-gcc install-target-libgcc -C ${PKGBUILD_DIR}/build"

/usr/bin/install -D -m 0755 ${PKGBUILD_DIR}/toolchain-wrapper ${HOST_DIR}/bin/toolchain-wrapper

cd ${HOST_DIR}/bin;

for i in ${GNU_TARGET_NAME}-*; d
        case "$i" in
        *.br_real)
                ;;
        *-ar|*-ranlib|*-nm)
                ln -snf $i ${ARCH}-linux$
                ;;
        *cc|*cc-*|*++|*++-*|*cpp|*-gfo
                rm -f $i.br_real
                mv $i $i.br_real;
                ln -sf toolchain-wrapp
                ln -sf toolchain-wrapper
                ln -snf $i.br_real ${ARCH}
                ;;
        *)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        esac;
done
```

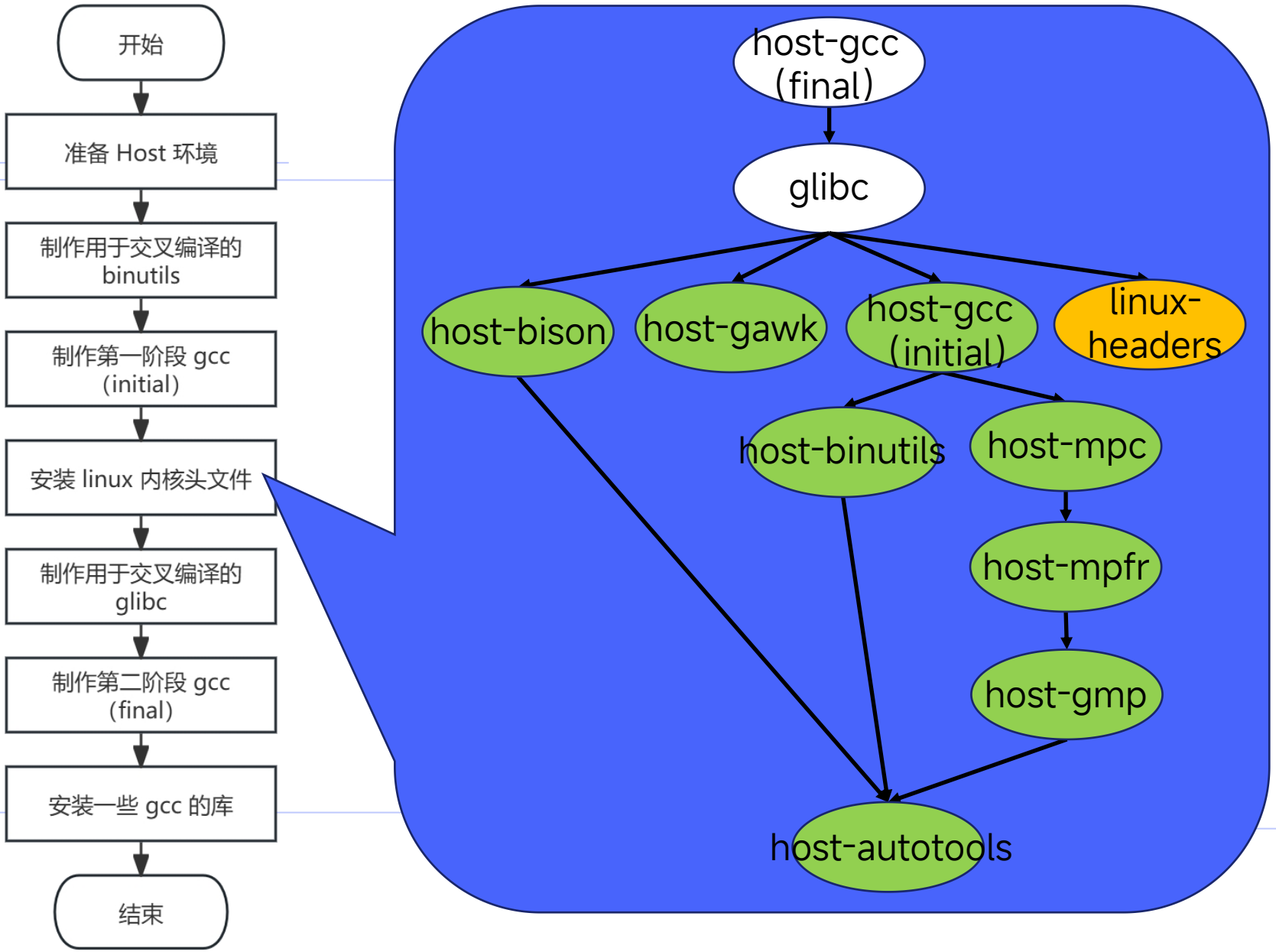
处理类型	例子
将部分核心工具（例如 cc，gcc 等）重命名，添加 .br_real 后缀，原名字创建为指向 toolchain-wrapper 的符号链接	riscv64-unknown-linux-gnu-cc.br_real riscv64-unknown-linux-gnu-cc -> toolchain-wrapper
为所有工具创建不带完整 triplet 的短链接，提升易用性	riscv64-linux-cc -> toolchain-wrapper riscv64-linux-addr2line -> riscv64-unknown-linux-gnu-addr2line riscv64-linux-ar -> riscv64-unknown-linux-gnu-ar

制作交叉工具链的步骤



因为 glibc 是用户空间（应用程序）与 Linux 内核空间进行通信的桥梁。它需要知道内核提供的“接口”（系统调用、数据结构、常量）长什么样，所以我们需要将内核的头文件安装到 ${STAGING_DIR} 确保编译 C 库时依赖了正确的内核头文件。

制作交叉工具链的步骤


开始
准备 Host 环境
制作用于交叉编译的 binutils
制作第一阶段 gcc (initial)
安装 linux 内核头文件
制作用于交叉编译的 glibc
制作第二阶段 gcc (final)
安装一些 gcc 的库
结束
host-gcc (final)
glibc
host-bison
host-gawk
host-gcc (initial)
linux-headers
host-binutils
host-mpc
host-mpfr
host-gmp
host-autotools

Linux-headers – 制作步骤

安装 （install-staging）

```
eval "${TARGET_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} ARCH=riscv
HOSTCC=\"/usr/bin/gcc\" HOSTCFLAGS=\"\"
HOSTCXX=\"/usr/bin/g++\“ INSTALL_HDR_PATH=${STAGING_DIR}/usr headers_install"

HOSTCC="/usr/bin/gcc" check_kernel_headers_version ${BUILD_DIR} ${STAGING_DIR} 6.12 strict
```

Linux-headers – 制作步骤

安装 （install-staging）

```
eval "${TARGET_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} ARCH=riscv
HOSTCC=\"/usr/bin/ \" HOSTCFLAGS=\"\"
HOSTCXX=\"/usr/bin/ NSTALL_HDR_PATH=${STAGING_DIR}/usr headers_install"

HOSTCC="/usr/bin/gc  headers_version ${BUILD_DIR} ${STAGING_DIR} 6.12 strict
```

GIT_DIR=. PATH="${HOST_DIR}/bin:${HOST_DIR}/sbin:${PATH}"

Linux-headers – 制作步骤

安装 （install-staging）

```
eval "${TARGET_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} ARCH=riscv
HOSTCC=\"/usr/bin/gcc\" HOSTCFLAGS=\"\"
HOSTCXX=\"/usr/bin/g++\“ INSTALL_HDR_PATH=${STAGING_DIR}/usr headers_install"

HOSTCC="/usr/bin/gcc" check_kernel_headers_version ${BUILD_DIR} ${...GING_DIR} 6.12 strict
```

make headers_install 是 Linux 内核构建系统中的一个重要命令，该命令会对内核源码中的头文件进行额外处理（移除只在内核内部使用的定义，处理条件编译，只保留用户空间相关部分），然后安装到指定目录（INSTALL_HDR_PATH）供用户空间程序（如 C 库）使用，确保用户空间程序使用正确的系统调用接口。

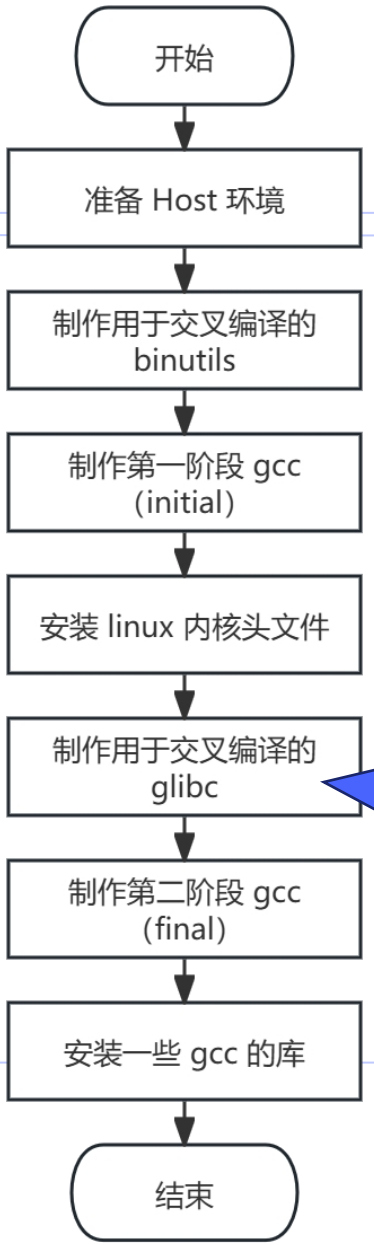
Linux-headers – 制作步骤

安装 （install-staging）

```
eval "${TARGET_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} ARCH=riscv
HOSTCC=\"/usr/bin/gcc\" HOSTCFLAGS=\"\"
HOSTCXX=\"/usr/bin/g++\“ INSTALL_HDR_PATH=${STAGING_DIR}/usr headers_install"

HOSTCC="/usr/bin/gcc" check_kernel_headers_version ${BUILD_DIR} ${STAGING_DIR} 6.12 strict
```

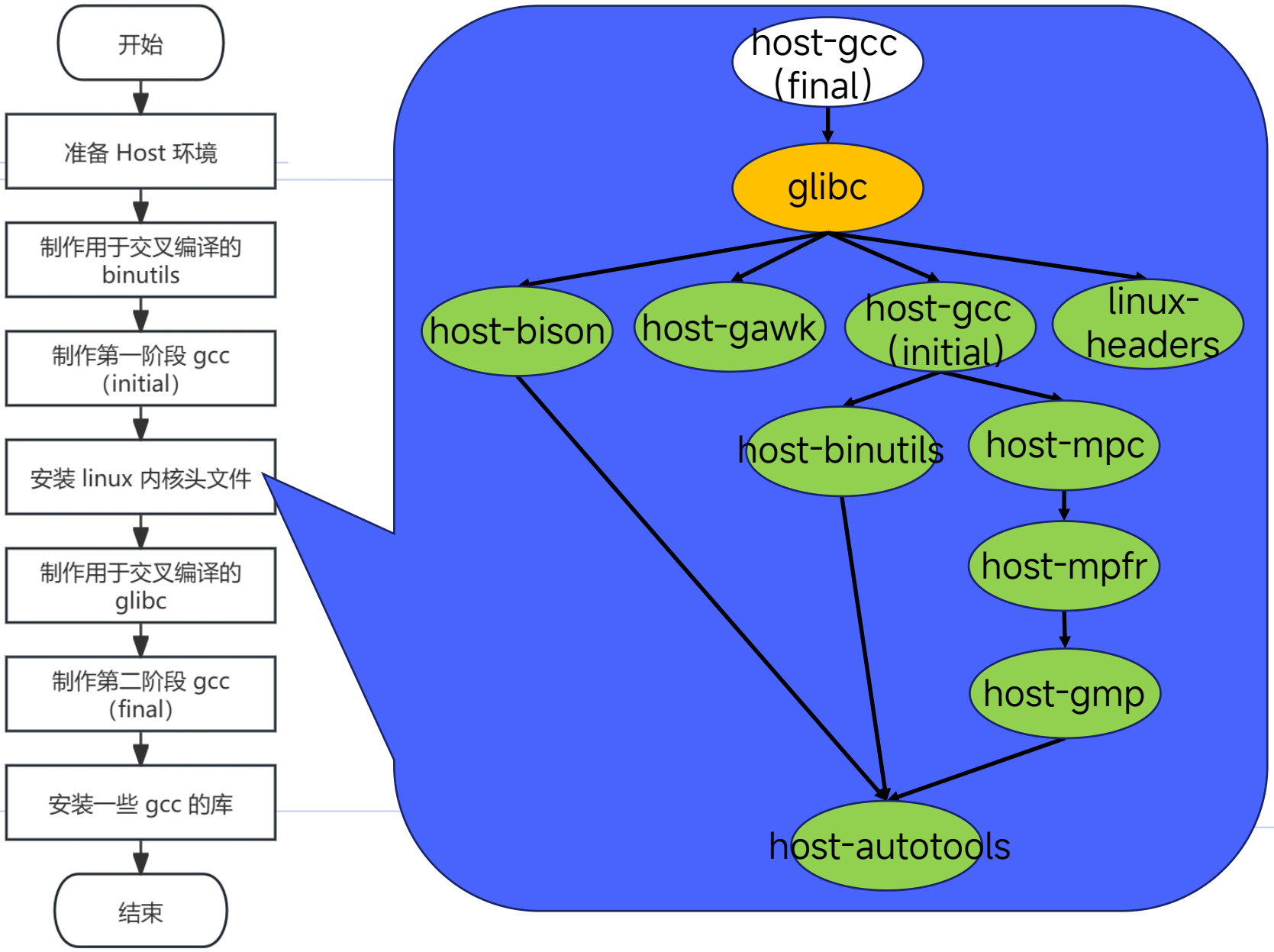
检查并确保我们使用的内核头文件对应的内核版本不可以比实际 Target 系统上运行的内核的版本新。

制作交叉工具链的步骤



- 使用 交叉编译版本的 initial GCC 和 binutils 构建可以在 Target 系统上运行的 glibc。
- 将该 glibc 安装到 **${STAGING_DIR}** 供以后在 Host 系统上交叉编译 Target 系统上的软件时使用。
- 将该 glibc 中的共享库（lib*.so) 安装到 **${TARGET_DIR}** 上供 Target 系统上的软件运行期时使用。

制作交叉工具链的步骤


开始
准备 Host 环境
制作用于交叉编译的 binutils
制作第一阶段 gcc (initial)
安装 linux 内核头文件
制作用于交叉编译的 glibc
制作第二阶段 gcc (final)
安装一些 gcc 的库
结束
host-gcc (final)
glibc
host-bison
host-gawk
host-gcc (initial)
linux-headers
host-binutils
host-mpc
host-mpfr
host-gmp
host-autotools

Glibc – 制作步骤

配置

```
eval "${TARGET_CONFIGURE_OPTS} CXX=no CFLAGS=\"-O2 -fno-lto\" CPPFLAGS=\"\" CXXFLAGS=\"-O2 -fno-lto\" ac_cv_path_BASH_SHELL=/bin/sh libc_cv_forced_unwind=yes libc_cv_ssp=no libc_cv_slibdir=/lib64 libc_cv_rtlddir=/lib ac_cv_prog_MAKE=\"/usr/bin/make -j9\" /bin/bash ${PKGBUILD_DIR}/configure --target=${GNU_TARGET_NAME} --host=${GNU_TARGET_NAME} --build=x86_64-pc-linux-gnu --prefix=/usr --enable-shared --disable-profile --disable-werror --without-gd --with-headers=${STAGING_DIR}/usr/include  --enable-kernel=6.12")

mkdir -p ${STAGING_DIR}/usr/include/gnu

touch ${STAGING_DIR}/usr/include/gnu/stubs.h
```

Glibc - 制作步骤

配置

```
eval "${TARGET_CONFIGURE_OPTS} CXX=no CFLAGS=\"-O2 -fno-lto\" CPPFLAGS=\"\" CXXFLAGS=\"-
O2 -fno-lto\" ac_cv_path_BA
libc_cv_slibdir=/lib64 libc_cv_r
${PKGBUILD_DIR}/configure --ta
build=x86_64-pc-lin
--with-headers=${S

mkdir -p ${STAGIN

touch ${STAGING_
```

```
SHELL=/bin/sh libc_cv_forced_unwind=yes libc_cv_ssp=no
cv_prog_MAKE=\"/usr/bin/make -j9\" /bin/bash
GET_NAME} --host=${GNU_TARGET_NAME} --
d
```

```
GIT_DIR=. PATH="${HOST_DIR}/bin:${HOST_DIR}/sbin:${PATH}" AR="${CROSS_COMPILE}gcc-ar"
AS="${CROSS_COMPILE}as" LD="${CROSS_COMPILE}ld" NM="${CROSS_COMPILE}gcc-nm"
CC="${CROSS_COMPILE}gcc" GCC="${CROSS_COMPILE}gcc" CPP="${CROSS_COMPILE}cpp"
CXX="${CROSS_COMPILE}g++" FC="${CROSS_COMPILE}gfortran" F77="${CROSS_COMPILE}gfortran"
RANLIB="${CROSS_COMPILE}gcc-ranlib" READELF="${CROSS_COMPILE}readelf"
STRIP="${CROSS_COMPILE}strip" OBJCOPY="${CROSS_COMPILE}objcopy"
OBJDUMP="${CROSS_COMPILE}objdump" AR_FOR_BUILD="/usr/bin/ar" AS_FOR_BUILD="/usr/bin/as"
CC_FOR_BUILD="/usr/bin/gcc" GCC_FOR_BUILD="/usr/bin/gcc" CXX_FOR_BUILD="/usr/bin/g++"
LD_FOR_BUILD="/usr/bin/ld" CPPFLAGS_FOR_BUILD="-I${HOST_DIR}/include" CFLAGS_FOR_BUILD="-
O2 -I${HOST_DIR}/include" CXXFLAGS_FOR_BUILD="-O2 -I${HOST_DIR}/include"
LDFLAGS_FOR_BUILD="-L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib" FCFLAGS_FOR_BUILD=""
DEFAULT_ASSEMBLER="${CROSS_COMPILE}as" DEFAULT_LINKER="${CROSS_COMPILE}ld"
CPPFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64"
CFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -
D_FORTIFY_SOURCE=1" CXXFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1" LDFLAGS="" FCFLAGS=" -O2 -g0"
FFLAGS=" -O2 -g0" PKG_CONFIG="${HOST_DIR}/bin/pkg-config" STAGING_DIR="${STAGING_DIR}"
INTLTOOL_PERL=/usr/bin/perl
```

Glibc - 制作步骤

配置

```
eval "${TARGET_CONFIGURE_OPTS} CXX=no CFLAGS=\"-O2 -fno-lto\" CPPFLAGS=\"\" CXXFLAGS=\"-
O2 -fno-lto\" ac_cv_path_BA
libc_cv_slibdir=/lib64 libc_cv_r
${PKGBUILD_DIR}/configure --ta
build=x86_64-pc-lin
--with-headers=${S

mkdir -p ${STAGIN

touch ${STAGING_
```

```
SHELL=/bin/sh libc_cv_forced_unwind=yes libc_cv_ssp=no
cv_prog_MAKE=\"/usr/bin/make -j9\" /bin/bash
GET_NAME} --host=${GNU_TARGET_NAME} --
d
```

```
GIT_DIR=. PATH="${HOST_DIR}/bin:${HOST_DIR}/sbin:${PATH}" AR="${CROSS_COMPILE}gcc-ar"
AS="${CROSS_COMPILE}as" LD="${CROSS_COMPILE}ld" NM="${CROSS_COMPILE}gcc-nm"
CC="${CROSS_COMPILE}gcc" GCC="${CROSS_COMPILE}gcc" CPP="${CROSS_COMPILE}cpp"
CXX="${CROSS_COMPILE}g++" FC="${CROSS_COMPILE}gfortran" F77="${CROSS_COMPILE}gfortran"
RANLIB="${CROSS_COMPILE}gcc-ranlib" READELF="${CROSS_COMPILE}readelf"
STRIP="${CROSS_COMPILE}strip" OBJCOPY="${CROSS_COMPILE}objcopy"
OBJDUMP="${CROSS_COMPILE}objdump" AR_FOR_BUILD="/usr/bin/ar" AS_FOR_BUILD="/usr/bin/as"
CC_FOR_BUILD="/usr/bin/gcc" GCC_FOR_BUILD="/usr/bin/gcc" CXX_FOR_BUILD="/usr/bin/g++"
LD_FOR_BUILD="/usr/bin/ld" CPPFLAGS_FOR_BUILD="-I${HOST_DIR}/include" CFLAGS_FOR_BUILD="-
O2 -I${HOST_DIR}/include" CXXFLAGS_FOR_BUILD="-O2 -I${HOST_DIR}/include"
LDFLAGS_FOR_BUILD="-L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib" FCFLAGS_FOR_BUILD=""
DEFAULT_ASSEMBLER="${CROSS_COMPILE}as" DEFAULT_LINKER="${CROSS_COMPILE}ld"
CPPFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64"
CFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -
D_FORTIFY_SOURCE=1" CXXFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1" LDFLAGS="" FCFLAGS=" -O2 -g0"
FFLAGS=" -O2 -g0" PKG_CONFIG="${HOST_DIR}/bin/pkg-config" STAGING_DIR="${STAGING_DIR}"
INTLTOOL_PERL=/usr/bin/perl
```

Glibc - 制作步骤

配置

```
eval "${TARGET_CONFIGURE_OPTS} CX
O2 -fno-lto\" ac_cv_path_BA  SHELL=/
libc_cv_slibdir=/lib64 libc_cv_r  c_cv_p  /usr/bin/make -j9\" /bin/bash
${PKGBUILD_DIR}/configure --ta  NAME} --host=${GNU_TARGET_NAME} --
build=x86_64-pc-lin
--with-headers=${S

mkdir -p ${STAGIN

touch ${STAGING_
```

CROSS_COMPILE=${HOST_DIR}/bin/${GNU_TARGET_NAME}-
即 ${HOST_DIR}/bin/riscv64-unknown-linux-gnu-

```
GIT_DIR=. PATH="${HOST_DIR}/bin:${HOST_DIR}/sbin:${PATH}" AR="${CROSS_COMPILE}gcc-ar"
AS="${CROSS_COMPILE}as" LD="${CROSS_COMPILE}ld" NM="${CROSS_COMPILE}gcc-nm"
CC="${CROSS_COMPILE}gcc" GCC="${CROSS_COMPILE}gcc" CPP="${CROSS_COMPILE}cpp"
CXX="${CROSS_COMPILE}g++" FC="${CROSS_COMPILE}gfortran" F77="${CROSS_COMPILE}gfortran"
RANLIB="${CROSS_COMPILE}gcc-ranlib" READELF="${CROSS_COMPILE}readelf"
STRIP="${CROSS_COMPILE}strip" OBJCOPY="${CROSS_COMPILE}objcopy"
OBJDUMP="${CROSS_COMPILE}objdump" AR_FOR_BUILD="/usr/bin/ar" AS_FOR_BUILD="/usr/bin/as"
CC_FOR_BUILD="/usr/bin/gcc" GCC_FOR_BUILD="/usr/bin/gcc" CXX_FOR_BUILD="/usr/bin/g++"
LD_FOR_BUILD="/usr/bin/ld" CPPFLAGS_FOR_BUILD="-I${HOST_DIR}/include" CFLAGS_FOR_BUILD="-
O2 -I${HOST_DIR}/include" CXXFLAGS_FOR_BUILD="-O2 -I${HOST_DIR}/include"
LDFLAGS_FOR_BUILD="-L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib" FCFLAGS_FOR_BUILD=""
DEFAULT_ASSEMBLER="${CROSS_COMPILE}as" DEFAULT_LINKER="${CROSS_COMPILE}ld"
CPPFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64"
CFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -
D_FORTIFY_SOURCE=1" CXXFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1" LDFLAGS="" FCFLAGS=" -O2 -g0"
FFLAGS=" -O2 -g0" PKG_CONFIG="${HOST_DIR}/bin/pkg-config" STAGING_DIR="${STAGING_DIR}"
INTLTOOL_PERL=/usr/bin/perl
```

Glibc - 制作步骤

配置

```
eval "${TARGET_CONFIGURE_OPTS} CXX=no CFLAGS=\"-O2 -fno-lto\" CPPFLAGS=\"\" CXXFLAGS=\"-
O2 -fno-lto\" ac_cv_path_BA
SHELL=/bin/sh libc_cv_forced_unwind=yes libc_cv_ssp=no
libc_cv_slibdir=/lib64 libc_cv_r
cv_prog_MAKE=\"/usr/bin/make -j9\" /bin/bash
${PKGBUILD_DIR}/configure --ta
NAME} --host=${GNU_TARGET_NAME} --
build=x86_64-pc-lin
--with-headers=${S

mkdir -p ${STAGIN

touch ${STAGING_
```

```
GIT_DIR=. PATH="${HOST_DIR}/bin:${HOST_DIR}/sbin:${PATH}" AR="${CROSS_COMPILE}gcc-ar"
AS="${CROSS_COMPILE}as" LD="${CROSS_COMPILE}ld" NM="${CROSS_COMPILE}gcc-nm"
CC="${CROSS_COMPILE}gcc" GCC="${CROSS_COMPILE}gcc" CPP="${CROSS_COMPILE}cpp"
CXX="${CROSS_COMPILE}g++" FC="${CROSS_COMPILE}gfortran" F77="${CROSS_COMPILE}gfortran"
RANLIB="${CROSS_COMPILE}gcc-ranlib" READELF="${CROSS_COMPILE}readelf"
STRIP="${CROSS_COMPILE}strip" OBJCOPY="${CROSS_COMPILE}objcopy"
OBJDUMP="${CROSS_COMPILE}objdump" AR_FOR_BUILD="/usr/bin/ar" AS_FOR_BUILD="/usr/bin/as"
CC_FOR_BUILD="/usr/bin/gcc" GCC_FOR_BUILD="/usr/bin/gcc" CXX_FOR_BUILD="/usr/bin/g++"
LD_FOR_BUILD="/usr/bin/ld" CPPFLAGS_FOR_BUILD="-I${HOST_DIR}/include" CFLAGS_FOR_BUILD="-
O2 -I${HOST_DIR}/include" CXXFLAGS_FOR_BUILD="-O2 -I${HOST_DIR}/include"
LDFLAGS_FOR_BUILD="-L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib" FCFLAGS_FOR_BUILD=""
DEFAULT_ASSEMBLER="${CROSS_COMPILE}as" DEFAULT_LINKER="${CROSS_COMPILE}ld"
CPPFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64"
CFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -
D_FORTIFY_SOURCE=1" CXXFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1" LDFLAGS="" FCFLAGS=" -O2 -g0"
FFLAGS=" -O2 -g0" PKG_CONFIG="${HOST_DIR}/bin/pkg-config" STAGING_DIR="${STAGING_DIR}"
INTLTOOL_PERL=/usr/bin/perl
```

Glibc - 制作步骤

配置

```
eval "${TARGET_CONFIGURE_OPTS} CXX=no CFLAGS=\"-O2 -fno-lto\" CPPFLAGS=\"\" CXXFLAGS=\"-
O2 -fno-lto\" ac_cv_path_BA
SHELL=/bin/sh libc_cv_forced_unwind=yes libc_cv_ssp=no
libc_cv_slibdir=/lib64 libc_cv_r
cv_prog_MAKE=\"/usr/bin/make -j9\" /bin/bash
${PKGBUILD_DIR}/configure --ta
NAME} --host=${GNU_TARGET_NAME} --
build=x86_64-pc-lin
--with-headers=${S

mkdir -p ${STAGIN

touch ${STAGING_
```

```
GIT_DIR=. PATH="${HOST_DIR}/bin:${HOST_DIR}/sbin:${PATH}" AR="${CROSS_COMPILE}gcc-ar"
AS="${CROSS_COMPILE}as" LD="${CROSS_COMPILE}ld" NM="${CROSS_COMPILE}gcc-nm"
CC="${CROSS_COMPILE}gcc" GCC="${CROSS_COMPILE}gcc" CPP="${CROSS_COMPILE}cpp"
CXX="${CROSS_COMPILE}g++" FC="${CROSS_COMPILE}gfortran" F77="${CROSS_COMPILE}gfortran"
RANLIB="${CROSS_COMPILE}gcc-ranlib" READELF="${CROSS_COMPILE}readelf"
STRIP="${CROSS_COMPILE}strip" OBJCOPY="${CROSS_COMPILE}objcopy"
OBJDUMP="${CROSS_COMPILE}objdump" AR_FOR_BUILD="/usr/bin/ar" AS_FOR_BUILD="/usr/bin/as"
CC_FOR_BUILD="/usr/bin/gcc" GCC_FOR_BUILD="/usr/bin/gcc" CXX_FOR_BUILD="/usr/bin/g++"
LD_FOR_BUILD="/usr/bin/ld" CPPFLAGS_FOR_BUILD="-I${HOST_DIR}/include" CFLAGS_FOR_BUILD="-
O2 -I${HOST_DIR}/include" CXXFLAGS_FOR_BUILD="-O2 -I${HOST_DIR}/include"
LDFLAGS_FOR_BUILD="-L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib" FCFLAGS_FOR_BUILD=""
DEFAULT_ASSEMBLER="${CROSS_COMPILE}as" DEFAULT_LINKER="${CROSS_COMPILE}ld"
CPPFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64"
CFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -
D_FORTIFY_SOURCE=1" CXXFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1" LDFLAGS="" FCFLAGS=" -O2 -g0"
FFLAGS=" -O2 -g0" PKG_CONFIG="${HOST_DIR}/bin/pkg-config" STAGING_DIR="${STAGING_DIR}"
INTLTOOL_PERL=/usr/bin/perl
```

Glibc - 制作步骤

配置

```
eval "${TARGET_CONFIGURE_OPTS} CXX=no CFLAGS=\"-O2 -fno-lto\" CPPFLAGS=\"\" CXXFLAGS=\"-
O2 -fno-lto\" ac_cv_path_BA
libc_cv_slibdir=/lib64 libc_cv_r
${PKGBUILD_DIR}/configure --ta
build=x86_64-pc-lin
--with-headers=${S

mkdir -p ${STAGIN

touch ${STAGING_
```

```
GIT_DIR=. PATH="${HOST_DIR}/bin:${HOST_DIR}/sbin:${PATH}" AR="${CROSS_COMPILE}gcc-ar"
AS="${CROSS_COMPILE}as" LD="${CROSS_COMPILE}ld" NM="${CROSS_COMPILE}gcc-nm"
CC="${CROSS_COMPILE}gcc" GCC="${CROSS_COMPILE}gcc" CPP="${CROSS_COMPILE}cpp"
CXX="${CROSS_COMPILE}g++" FC="${CROSS_COMPILE}gfortran" F77="${CROSS_COMPILE}gfortran"
RANLIB="${CROSS_COMPILE}gcc-ranlib" READELF="${CROSS_COMPILE}readelf"
STRIP="${CROSS_COMPILE}strip" OBJCOPY="${CROSS_COMPILE}objcopy"
OBJDUMP="${CROSS_COMPILE}objdump" AR_FOR_BUILD="/usr/bin/ar" AS_FOR_BUILD="/usr/bin/as"
CC_FOR_BUILD="/usr/bin/gcc" GCC_FOR_BUILD="/usr/bin/gcc" CXX_FOR_BUILD="/usr/bin/g++"
LD_FOR_BUILD="/usr/bin/ld" CPPFLAGS_FOR_BUILD="-I${HOST_DIR}/include" CFLAGS_FOR_BUILD="-
O2 -I${HOST_DIR}/include" CXXFLAGS_FOR_BUILD="-O2 -I${HOST_DIR}/include"
LDFLAGS_FOR_BUILD="-L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib" FCFLAGS_FOR_BUILD=""
DEFAULT_ASSEMBLER="${CROSS_COMPILE}as" DEFAULT_LINKER="${CROSS_COMPILE}ld"
CPPFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64"
CFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -
D_FORTIFY_SOURCE=1" CXXFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1" LDFLAGS="" FCFLAGS=" -O2 -g0"
FFLAGS=" -O2 -g0" PKG_CONFIG="${HOST_DIR}/bin/pkg-config" STAGING_DIR="${STAGING_DIR}"
INTLTOOL_PERL=/usr/bin/perl
```

Glibc - 制作步骤

配置

```
eval "${TARGET_CONFIGURE_OPTS} CXX=no CFLAGS=\"-O2 -fno-lto\" CPPFLAGS=\"\" CXXFLAGS=\"-
O2 -fno-lto\" ac_cv_path_BASH_SHELL=/bin/sh libc_cv_forced_unwind=yes libc_cv_ssp=no
libc_cv_slibdir=/lib64 libc_cv_rtlddir=/lib ac_cv_prog_MAKE=\"/usr/bin/make -j9\" /bin/bash
${PKGBUILD_DIR}/configure --target=${GNU_TARGET_NAME} --host=${GNU_TARGET_NAME} --
build=x86_64-pc-linux-gnu --prefix=/usr --disable-shared --disable-profile --disable-werror --without-gd
--with-headers=${STAGING_DIR}/usr/incl                    kernel=6.12")

mkdir -p ${STAGING_DIR}/usr

touch ${STAGING_DIR}/usr/ir
```

选项	含义/用途
--target=riscv64-unknown-linux-gnu	定义目标系统。对于 glibc（以及绝大多数除了 gcc/binutils 的软件包来说），--target 没有实际意义，可以省略或让它默认等于 --host 即可，这里只是统一列出。
--host=riscv64-unknown-linux-gnu	定义了我们编译出来的 C 库将要在 riscv64 的 linux 上运行
--build=x86_64-pc-linux-gnu	定义了构建系统，即运行 configure 和 make 的机器是 x86_64 的 linux

Glibc - 制作步骤

配置

```
eval "${TARGET_CONFIGURE_OPTS} CXX=no CFLAGS=\"-O2 -fno-lto\" CPPFLAGS=\"\" CXXFLAGS=\"-O2 -fno-lto\" ac_cv_path_BASH_SHELL=/bin/sh libc_cv_forced_unwind=yes libc_cv_ssp=no libc_cv_slibdir=/lib64 libc_cv_rtlddir=/lib ac_cv_prog_MAKE=\"/usr/bin/make -j9\" /bin/bash ${PKGBUILD_DIR}/configure --target=${GNU_TARGET_NAME} --host=${GNU_TARGET_NAME} --build=x86_64-pc-linux-gnu --prefix=/usr --enable-shared --disable-profile --disable-werror --without-gd --with-headers=${STAGING_DIR}/usr/include  --enable-kernel=6.12")

mkdir -p ${STAGING_DIR}/usr/include

touch ${STAGING_DIR}/
```

选项	含义/用途
--prefix=/usr	指定安装目录，相对于 install_root
--enable-shared	除了静态库外还生成动态库
--disable-profile	禁用 profiling（性能分析）库的构建。
--disable-werror	编译警告（warnings）不视为错误（errors）
--without-gd	禁用 gd 图形库支持
--with-headers=${STAGING_DIR}/usr/include	指定内核头文件的位置
--enable-kernel=6.12	指定 glibc 支持的最低内核版本

Glibc - 制作步骤

配置

```
eval "${TARGET_CONFIGURE_OPTS} CXX=no CFLAGS=\"-O2 -fno-lto\" CPPFLAGS=\"\" CXXFLAGS=\"-O2 -fno-lto\" ac_cv_path_BASH_SHELL=/bin/sh libc_cv_forced_unwind=yes libc_cv_ssp=no libc_cv_slibdir=/lib64 libc_cv_rtlddir=/lib ac_cv_prog_MAKE=\"/usr/bin/make -j9\" /bin/bash ${PKGBUILD_DIR}/configure --target=${GNU_TARGET_NAME} --host=${GNU_TARGET_NAME} --build=x86_64-pc-linux-gnu --prefix=/usr --enable-shared  --with-pkgversion=\"Buildroot\" --disable-profile --disable-werror --without-gd --with-headers=${STAGING_DIR}/usr/include  --enable-kernel=6.12")

mkdir -p ${STAGING_DIR}/usr/include/gnu

touch ${STAGING_DIR}/usr/include/gnu/stubs.h
```

host-gcc-final 编译过程中依赖于 ${STAGING_DIR}/usr/include/gnu 目录必须存在，所以我们手动创建一个，参考 http://gcc.gnu.org/ml/gcc/2002-01/msg00900.html

Glibc – 制作步骤

构建

```
eval "${TARGET_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR}/build"
```

安装 （install-staging）

```
eval "${TARGET_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install_root=${STAGING_DIR}
install -C ${PKGBUILD_DIR}/build"
```

安装 （install-target）

```
for libpattern in ld*.so.* libanl.so.* libc.so.* libdl.so.* libgcc_s.so.* libm.so.* libpthread.so.*
libresolv.so.* librt.so.* libutil.so.* libnss_files.so.* libnss_dns.so.* libmvec.so.*; do
        copy_toolchain_lib_root ${libpattern}
done
```

Glibc – 制作步骤

构建

```
eval "${TARGET_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR}/build"
```

安装 （install-staging）

```
eval "${TARGET_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install_root=${STAGING_DIR}
install -C ${PKGBUILD_DIR}/build"
```

安装 （install-target）

```
for libpattern in ld*.so.* libanl.so.* libc.so.* libdl.so.* libgcc_s.so.
libresolv.so.* librt.so.* libutil.so.* libnss_files.so.* libnss_dns.so.*
        copy_toolchain_lib_root ${libpattern}
done
```

install_root 参数用于在不修改原始配置的情况下，将编译好的文件安装到指定的临时根目录，也就是在配置中的 $prefix 之前统一加个前缀。

Glibc - 制作步骤

构建

```
eval "${TARGET_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} -C ${PKGBUILD_DIR}/build"
```

安装 （install-staging）

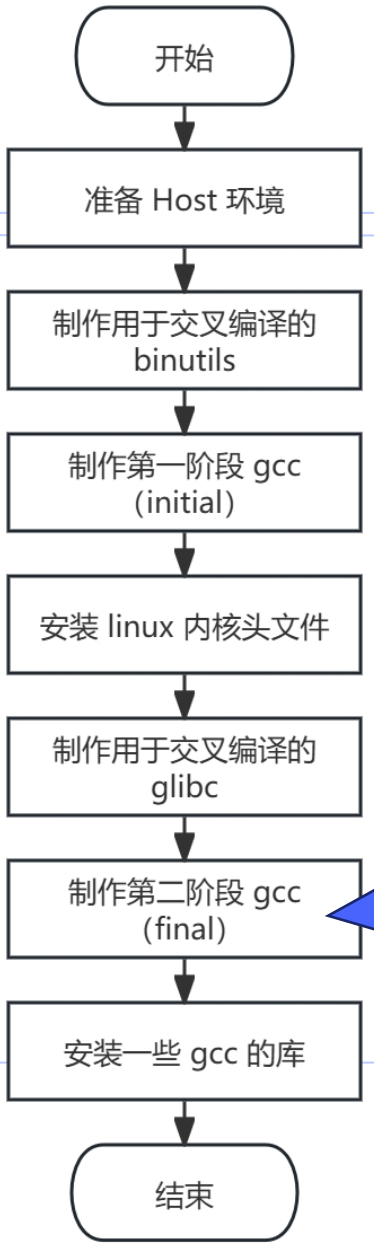
```
eval "${TARGET_MAKE_ENV} /usr/bin/make -j${M
install -C ${PKGBUILD_DIR}/build"
```

将我们安装在 ${STAGING_DIR} 下的动态链接器以及共享库复制到 ${TARGET_DIR}。注意我们只复制运行时必需的文件，而不是重复 install 的操作。

安装 （install-target）

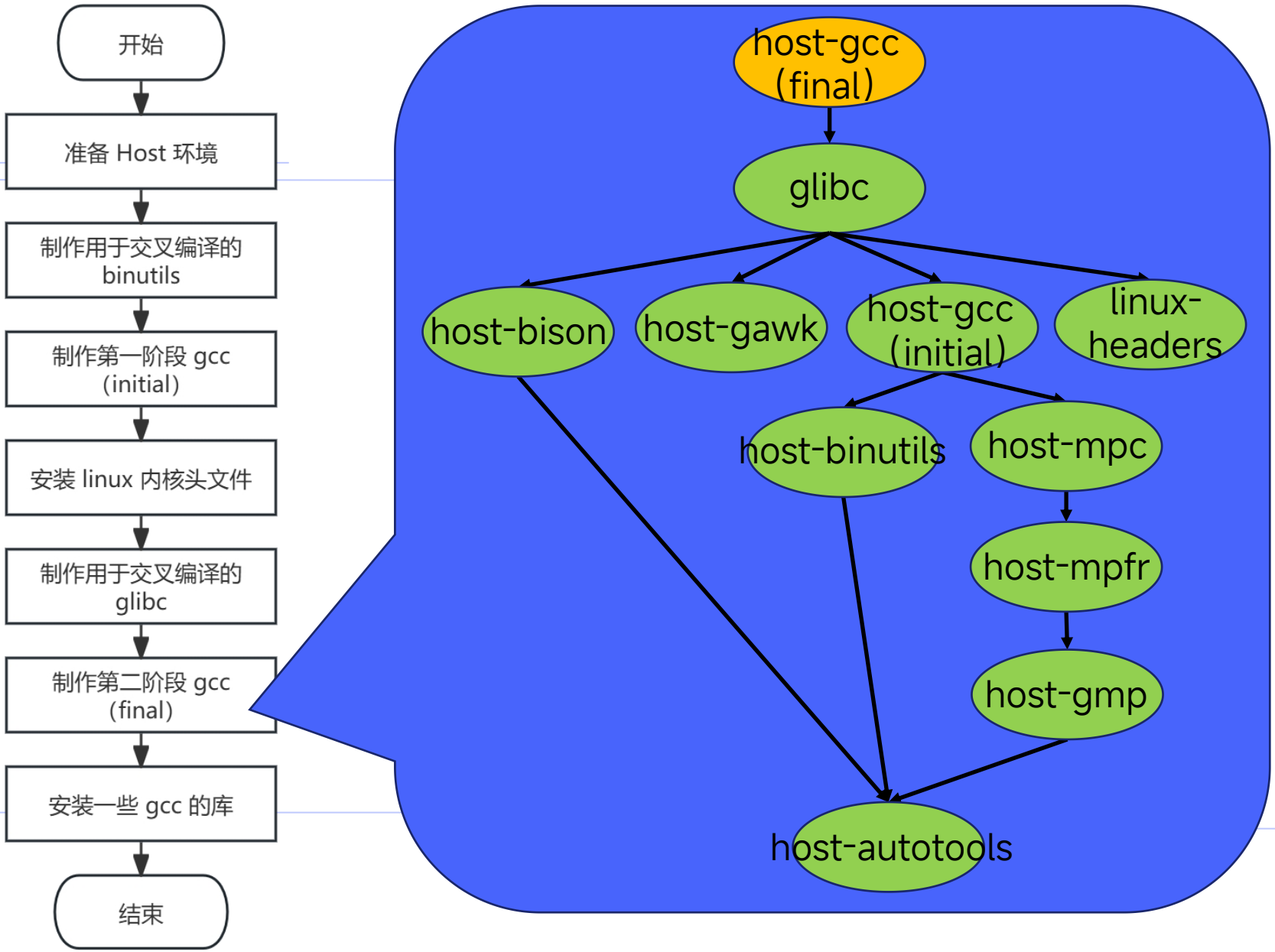
```
for libpattern in ld*.so.* libanl.so.* libc.so.* libdl*.so.* libgcc_s.so.* libm.so.* libpthread.so.*
libresolv.so.* librt.so.* libutil.so.* libnss_files.so.* libnss_dns.so.* libmvec.so.*; do
        copy_toolchain_lib_root ${libpattern}
done
```

制作交叉工具链的步骤



- 制作一个完全功能的、“final”版本的，支持 **交叉编译** 的 GCC。
- 制作 final GCC 使用 **Build 系统** 的 **本地工具链**。
- 制作 final GCC 依赖于“准备 Host 环境”阶段在 **${HOST_DIR}** 下安装的一些工具。
- 制作好后安装到 **${HOST_DIR}** **覆盖** “initial”版本的 GCC。

制作交叉工具链的步骤


开始
准备 Host 环境
制作用于交叉编译的 binutils
制作第一阶段 gcc (initial)
安装 linux 内核头文件
制作用于交叉编译的 glibc
制作第二阶段 gcc (final)
安装一些 gcc 的库
结束
host-gcc (final)
glibc
host-bison
host-gawk
host-gcc (initial)
linux-headers
host-binutils
host-mpc
host-mpfr
host-gmp
host-autotools

GCC(final) – 制作步驟

配置

```
eval "${HOST_CONFIGURE_OPTS} CFLAGS=\"-O2 -I${HOST_DIR}/include\" LDFLAGS=\"-
L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" MAKEINFO=missing CFLAGS_FOR_TARGET=\"-
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -
D_FORTIFY_SOURCE=1 \" CXXFLAGS_FOR_TARGET=\"-D_LARGEFILE_SOURCE -
D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1 \"
LDFLAGS_FOR_TARGET=\" \" AR_FOR_TARGET=gcc-ar NM_FOR_TARGET=gcc-nm
RANLIB_FOR_TARGET=gcc-ranlib ./configure --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/etc\" --enable-static  --target=${GNU_TARGET_NAME} --with-
sysroot=${STAGING_DIR} --enable-__cxa_atexit --with-gnu-ld --disable-libssp --disable-multilib --
disable-decimal-float --enable-plugins --enable-lto --with-gmp=${HOST_DIR} --with-mpc=${HOST_DIR}
--with-mpfr=${HOST_DIR} --without-zstd --disable-libquadmath --disable-libquadmath-support --
enable-tls --enable-threads --without-isl --without-cloog --with-arch=\"rv64imafd_zicsr_zifencei\" --
with-abi=\"lp64d\" --enable-languages=c --with-build-time-
tools=${HOST_DIR}/${GNU_TARGET_NAME}/bin --enable-shared --disable-libgomp"
```

GCC(final) – 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} CFLAGS=\"-O2 -I${HOST_DIR}/include\" LDFLAGS=\"-
L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" MAKEINFO=missing CFLAGS_FOR_TARGET=\"-
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -
D_FORTIFY_SOURCE=1 \" CXXFLAGS_FOR_TARGET=\"-D_LARGEFILE_SOURCE -
D_LARGEFILE64_SOURCE -D_FILE_OF                 BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1 \"
LDFLAGS_FOR_TARGET=\" \" AR_FOR                                  NM_FOR_TARGET=gcc-nm
RANLIB_FOR_TARGET=gcc
sysconfdir=\"${HOST_DIR}
sysroot=${STAGING_DIR}
disable-decimal-float --en                                                    R}
--with-mpfr=${HOST_DIR}
enable-tls --enable-threac
with-abi=\"lp64d\" --enabl
tools=${HOST_DIR}/${GNU
```

选项	含义/用途
MAKEINFO=missing	跳过执行 makeinfo
CFLAGS_FOR_TARGET="....."	构建工具链时才会指定，其值会记录在这里新构建出来的交叉编译器 gcc 中，当我们使用它去构建 Target 的 C 程序（如 libgcc）时作为默认编译选项影响 cc1 的行为。
CXXFLAGS_FOR_TARGET="....."	构建工具链时才会指定，其值会记录在这里新构建出来的交叉编译器 gcc 中，当我们使用它去构建 Target 的 C++ 程序（如 libstdc++）时作为默认编译选项影响 cc1plus 的行为。
LDFLAGS_FOR_TARGET=""	构建工具链时才会指定，其值会记录在这里新构建出来的交叉编译器 gcc 中，当我们使用它去链接 Target 的 程序时作为默认链接选项影响 ld 的行为。
AR_FOR_TARGET=gcc-ar	设置处理交叉构建时所使用的运行时库的归档工具 ar
NM_FOR_TARGET=gcc-nm	设置处理交叉构建时所使用的运行时库的符号列表工具 nm
RANLIB_FOR_TARGET=gcc-ranlib	设置处理交叉构建时所使用的运行时库的创建归档索引工具 ranlib

GCC(final) – 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} CFLAGS=\"-O2 -I${HOST_DIR}/include\" LDFLAGS=\"-
L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" MAKEINFO=missing CFLAGS_FOR_TARGET=\"-
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -
D_FORTIFY_SOURCE=1 \" CXXFLAGS_FOR_TARGET=\"-D_LARGEFILE_SOURCE -
D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64  -O2 -g0 -D_FORTIFY_SOURCE=1 \"
LDFLAGS_FOR_TARGET=\" \" AR_FOR_TARGET=gcc-ar NM_FOR_TARGET=gcc-nm
RANLIB_FOR_TARGET=gcc-ranlib ./configure --prefix=\"${HOST_DIR}\" --
sysconfdir=\"${HOST_DIR}/etc\" --enable-static  --target=${GNU_TARGET_NAME} --with-
sysroot=${STAGING_DIR} --enable-__cxa_atexit --with-gnu-ld --disable-libssp --disable-multilib --
disable-decimal-float --enable-plugins --enable-lto --with-gmp=${HOST_DIR} --with-mpc=${HOST_DIR}
--with-mpfr=${HOST_DIR} --without-zstd --disable-libquadmath --disable-libquadmath-support --
enable-tls --enable-threads --without-isl --without-cloog --with-arch=\"rv64imafd_zicsr_zifencei\" --
with-abi=\"lp64d\" --enable-languages=c --with-build-time-
tools=${HOST_DIR}/${GNU_TARGET_NAME}/bin --ena
```

和 host-gcc-initial 几乎是一致，只是少了

- ✓ --disable-shared
- ✓ --without-headers
- ✓ --disable-threads
- ✓ --with-newlib
- ✓ --disable-largefile

GCC(final) – 制作步骤

配置

```
eval "${HOST_CONFIGURE_OPTS} CFLAGS=\"-O2 -I${HOST_DIR}/include\" LDFLAGS=\"-
L${HOST_DIR}/lib -Wl,-rpath,${HOST_DIR}/lib\" MAKEINFO=missing CFLAGS_FOR_TARGET=\"-
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -O2 -g0 -
D_FORTIFY_SOURCE=1 \" CXXFL
D_LARGEFILE64_SOURCE -D_FIL
LDFLAGS_FOR_TARGET=\" \" AR
RANLIB_FOR_TARGET=gcc-ranli
sysconfdir=\"${HOST_DIR}/etc\" -
sysroot=${STAGING_DIR} --enab
disable-decimal-float --enable-pl
--with-mpfr=${HOST_DIR} --witho
enable-tls --enable-threads --without-isl --with
--with-arch=\"rv64imafd_zicsr_zifencei\" --
with-abi=\"lp64d\" --enable-languages=c --with-build-time-
tools=${HOST_DIR}/${GNU_TARGET_NAME}/bin --enable-shared --disable-libgomp"
```

选项	含义/用途
--with-build-time-tools=${HOST_DIR}/${GNU_TARGET_NAME}/bin	构建 host-gcc-final 过程中隐含了还要构建针对 Target 的诸如 libgcc 等库，所以需要使用前面阶段制作的支持交叉构建的 binutils 和 gcc。
--enable-shared	因为前面也指定了 --enable-static，所以 host-gcc-final 中既会生成静态库，也会生成动态库。
--disable-libgomp	禁止构建 libgomp 库

GCC(final) – 制作步驟

構建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} gcc_cv_prog_makeinfo_modern=no gcc_cv_libc_provides_ssp=yes -C ${PKGBUILD_DIR}/build"

/usr/bin/gcc -O2 -I${HOST_DIR}/include -DBR_CROSS_PATH_SUFFIX='".br_real"' -DBR_SYSROOT="\"${STAGING_SUBDIR}\"" -DBR_ADDITIONAL_CFLAGS='"-fstack-protector-strong",' -DBR2_PIC_PIE -DBR2_RELRO_FULL -s -Wl,--hash-style=both ${PROJECT_DIR}/toolchain/toolchain-wrapper.c -o ${PKGBUILD_DIR}/toolchain-wrapper
```

GCC(final) – 制作步骤

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} gcc_cv_prog_makeinfo_modern=no
gcc_cv_libc_provides_ssp=yes -C ${PKGBUILD_DIR}/build"

/usr/bin/gcc -O2 -I${HOST_DIR}/include -DBR_CRO[illegible]PATH_SUFFIX="".br_real"" -
DBR_SYSROOT="\"${STAGING_SUBDIR}\"" -DBR_ADDITIO[illegible]FLAGS=""-fstack-protector-strong",' -
DBR2_PIC_PIE -DBR2_RELRO_FULL -s -Wl,--hash-style[illegible]
wrapper.c -o ${PKGBUILD_DIR}/toolchain-wrapper
```

和 host-gcc-initial 不同，这里是完整的构建，不再细分为 all-gcc all-target-libgcc

GCC(final) - 制作步骤

构建

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} gcc_cv_prog_makeinfo_modern=no
gcc_cv_libc_provides_ssp=yes -C ${PKGBUILD_DIR}/build"

/usr/bin/gcc -O2 -I${HOST_DIR}/include -DBR_CROSS_PATH_SUFFIX='".br_real"' -
DBR_SYSROOT="\"${STAGING_SUBDIR}\"" -DBR_ADDITIONAL_CFLAGS='"-fstack-protector-strong",' -
DBR2_PIC_PIE -DBR2_RELRO_FULL -s -Wl,--hash-style=both ${PROJECT_DIR}/toolchain/toolchain-
wrapper.c -o ${PKGBUILD_DIR}/toolchain-wrapper
```

和 host-gcc-initial 一样，制作 toolchain-wrapper 这个工具链包装。

GCC(final) – 制作步骤

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install -C ${PKGBUILD_DIR}/build"

if [ ! -e ${HOST_DIR}/bin/${GNU_TARGET_NAME}-cc ]; then
        ln -f ${HOST_DIR}/bin/${GNU_TARGET_NAME}-gcc ${HOST_DIR}/bin/${GNU_TARGET_NAME}-cc;
fi

/usr/bin/install -D -m 0755 ${PKGBUILD_DIR}/toolchain-wrapper ${HOST_DIR}/bin/toolchain-wrapper

cd ${HOST_DIR}/bin;

for i in ${GNU_TARGET_NAME}-*; do
        case "$i" in
        *.br_real)
                ;;
        *-ar|*-ranlib|*-nm)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        *cc|*cc-*|*++|*++-*|*cpp|*-gfortran|*-gdc)
                rm -f $i.br_real;
                mv $i $i.br_real;
                ln -sf toolchain-wrapper $i;
                ln -sf toolchain-wrapper ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ln -snf $i.br_real ${ARCH}-linux${i##${GNU_TARGET_NAME}}.br_real;
                ;;
        *)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        esac;
done
```

GCC(final) – 制作步骤

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install -C ${PKGBUILD_DIR}/build"

if [ ! -e ${HOST_DIR}/bin/${GNU_TARGET_NAME}-cc ]; then
        ln -f ${HOST_DIR}/bin/${GNU_TARGET_NAME}-gcc ${HOST                 ${GNU_TARGET_NAME}-cc;
fi

/usr/bin/install -D -m 0755 ${PKGBUILD_DIR}/toolc

cd ${HOST_DIR}/bin;

for i in ${GNU_TARGET_NAME}-*; do
        case "$i" in
        *.br_real)
                ;;
        *-ar|*-ranlib|*-nm)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        *cc|*cc-*|*++|*++-*|*cpp|*-gfortran|*-gdc)
                rm -f $i.br_real;
                mv $i $i.br_real;
                ln -sf toolchain-wrapper $i;
                ln -sf toolchain-wrapper ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ln -snf $i.br_real ${ARCH}-linux${i##${GNU_TARGET_NAME}}.br_real;
                ;;
        *)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        esac;
done
```

- 和 host-gcc-initial 的不同之处是这里是完整的安装，而不是仅仅安装 install-gcc 和 install-target-libgcc。
- 安装完后，将覆盖原先 host-gcc-initial 的安装。

GCC(final) – 制作步骤

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install -C ${PKGBUILD_DIR}/build"

if [ ! -e ${HOST_DIR}/bin/${GNU_TARGET_NAME}-cc ]; then
        ln -f ${HOST_DIR}/bin/${GNU_TARGET_NAME}-gcc ${HOST_DIR}/bin/${GNU_TARGET_NAME}-cc;
fi

/usr/bin/install -D -m 0755 ${PKGBUILD_DIR}/toolchain-                *{HOST_DIR}/bin/toolchain-wrapper

cd ${HOST_DIR}/bin;

for i in ${GNU_TARGET_NAME}-*; do
        case "$i" in
        *.br_real)
                ;;
        *-ar|*-ranlib|*-nm)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        *cc|*cc-*|*++|*++-*|*cpp|*-gfortran|*-gdc)
                rm -f $i.br_real;
                mv $i $i.br_real;
                ln -sf toolchain-wrapper $i;
                ln -sf toolchain-wrapper ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ln -snf $i.br_real ${ARCH}-linux${i##${GNU_TARGET_NAME}}.br_real;
                ;;
        *)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        esac;
done
```

确保我们有 riscv64-unknown-linux-gnu-cc，虽然只是一个指向 riscv64-unknown-linux-gnu-cc 的符号链接。

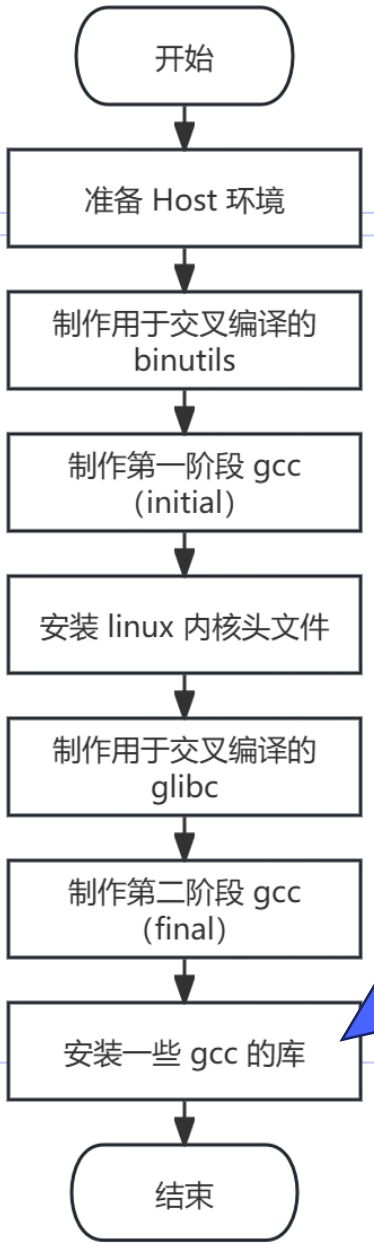
GCC(final) – 制作步骤

安装 （install-host）

```
eval "${HOST_MAKE_ENV} /usr/bin/make -j${MAXNUM_CPUS} install -C ${PKGBUILD_DIR}/build"

if [ ! -e ${HOST_DIR}/bin/${GNU_TARGET_NAME}-cc ]; then
        ln -f ${HOST_DIR}/bin/${GNU_TARGET_NAME}-gcc ${HOST_DIR}/bin/${GNU_TARGET_NAME}-cc;
fi

/usr/bin/install -D -m 0755 ${PKGBUILD_DIR}/toolchain-wrapper ${HOST_DIR}/bin/toolchain-wrapper

cd ${HOST_DIR}/bin;

for i in ${GNU_TARGET_NAME}-*; do
        case "$i" in
        *.br_real)
                ;;
        *-ar|*-ranlib|*-nm)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        *cc|*cc-*|*++|*++-*|*cpp|*-gfortran|*-gdc)
                rm -f $i.br_real;
                mv $i $i.br_real;
                ln -sf toolchain-wrapper $i;
                ln -sf toolchain-wrapper ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ln -snf $i.br_real ${ARCH}-linux${i##${GNU_TARGET_NAME}}.br_real;
                ;;
        *)
                ln -snf $i ${ARCH}-linux${i##${GNU_TARGET_NAME}};
                ;;
        esac;
done
```

和 host-gcc-initial 一样的处理

制作交叉工具链的步骤



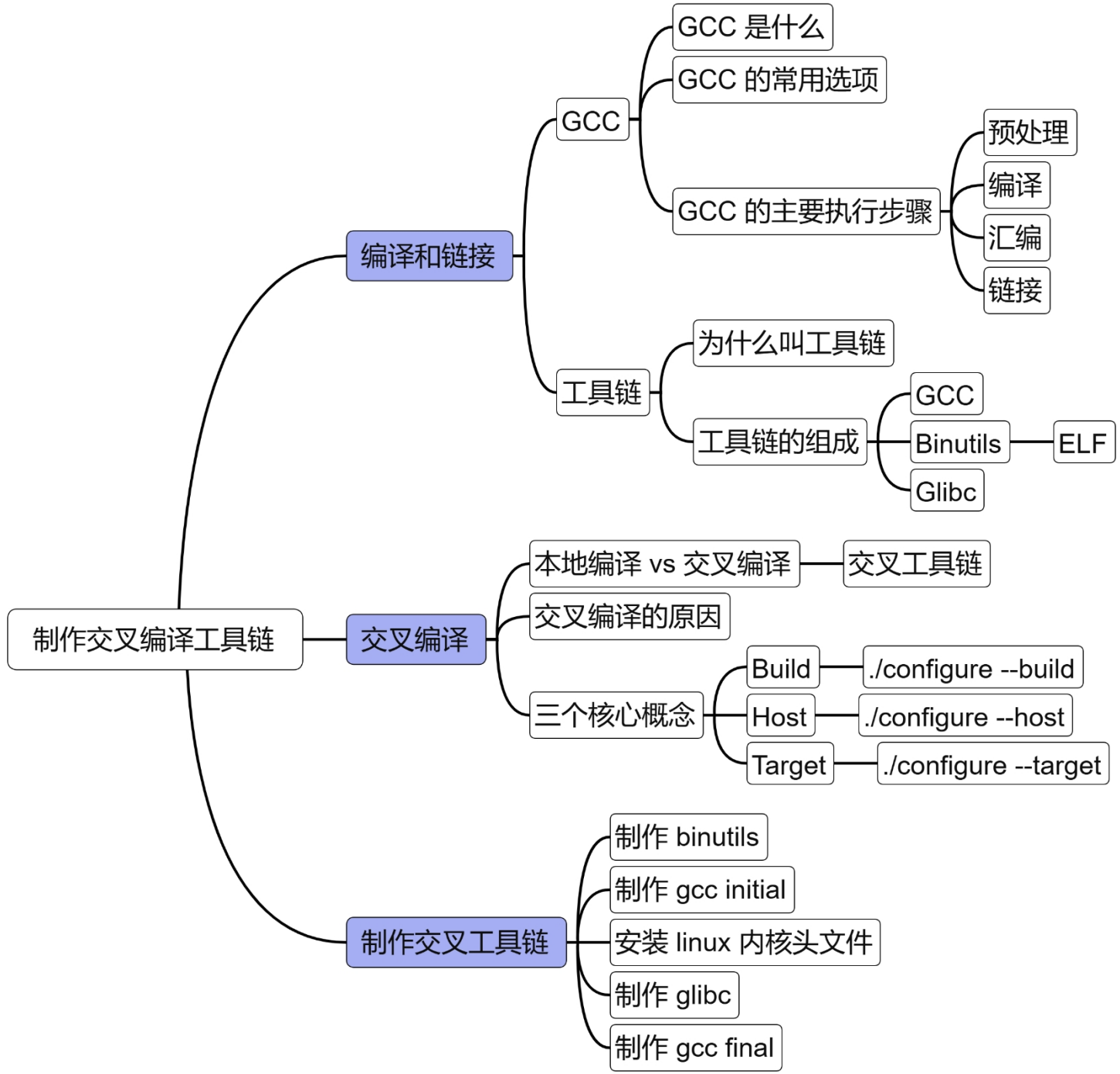

```
cp -dpf ${HOST_DIR}/${GNU_TARGET_NAME}/lib*/libatomic* ${STAGING_DIR}/lib/
cp -dpf ${HOST_DIR}/${GNU_TARGET_NAME}/lib*/libgcc_s* ${STAGING_DIR}/lib/


mkdir -p ${TARGET_DIR}/lib ${TARGET_DIR}/usr/lib
cp -dpf ${HOST_DIR}/${GNU_TARGET_NAME}/lib*/libatomic.so* ${TARGET_DIR}/lib/
cp -dpf ${HOST_DIR}/${GNU_TARGET_NAME}/lib*/libgcc_s.so* ${TARGET_DIR}/lib/
```

制作交叉工具链的步骤

区别在于 STAGING_DIR 下除了 *.so 外还包括 *.a

```
cp -dpf
${HOST_DIR}/${GNU_TARGET_NAME}/lib*/libatomic*
${STAGING_DIR}/lib/
cp -dpf
${HOST_DIR}/${GNU_TA                _NAME}/lib*/libgcc_s*
${STAGING_DI
                                    {TARGET_DIR}/usr/lib
cp -dpf
${HOST_DIR}/${GNU_TARGET_NAME}/lib*/libatomic.so*
${TARGET_DIR}/lib/
cp -dpf
${HOST_DIR}/${GNU_TARGET_NAME}/lib*/libgcc_s.so*
${TARGET_DIR}/lib/
```

本章总结


制作交叉编译工具链
编译和链接
GCC
GCC 是什么
GCC 的常用选项
GCC 的主要执行步骤
预处理
编译
汇编
链接
工具链
为什么叫工具链
工具链的组成
GCC
Binutils
ELF
Glibc
交叉编译
本地编译 vs 交叉编译
交叉工具链
交叉编译的原因
三个核心概念
Build
./configure --build
Host
./configure --host
Target
./configure --target
制作交叉工具链
制作 binutils
制作 gcc initial
安装 linux 内核头文件
制作 glibc
制作 gcc final

谢谢