

Build Distribution for Maintaining the Famous GCC 4.7

Oliver Reiche <oliver.reiche@huawei.com>

Intelligent Cloud Technologies Lab, Huawei Munich Research Center
www.huaweicloud.com

FOSDEM 2024

February 4, 2024

Agenda

Famous GCC 4.7

“Build Distribution”

Patching GCC 4.7.4

Bootstrap Process

Wrap-up

Famous GCC 4.7

Famous GCC 4.7

The **Bootstrappable Builds** movement strives for

- building all software from source
- with only a minimal *binary seed*

But how to build a C++ compiler without a C++ compiler?

Why is GCC 4.7 famous?

It plays a key role for bootstrappable builds:

- last GCC that can be build with only a C compiler
- stepping stone for C++ and beyond (Rust, Java, etc.)

Software preservation

GCC 4.7 is from 2012–2014

- build on modern systems (glibc, musl)
- build with modern compilers
- build with modern C11 standard
- build reproducibly

Famous GCC 4.7

The **Bootstrappable Builds** movement strives for

- building all software from source
- with only a minimal *binary seed*

But how to build a C++ compiler without a C++ compiler?

Why is GCC 4.7 famous?

It plays a key role for bootstrappable builds:

- last GCC that can be build with only a C compiler
- stepping stone for C++ and beyond (Rust, Java, etc.)

Software preservation

GCC 4.7 is from 2012–2014

- build on modern systems (glibc, musl)
- build with modern compilers
- build with modern C11 standard
- build reproducibly

Famous GCC 4.7

The **Bootstrappable Builds** movement strives for

- building all software from source
- with only a minimal *binary seed*

But how to build a C++ compiler without a C++ compiler?

Why is GCC 4.7 famous?

It plays a key role for bootstrappable builds:

- last GCC that can be build with only a C compiler
- stepping stone for C++ and beyond (Rust, Java, etc.)

Software preservation

GCC 4.7 is from 2012–2014

- build on modern systems (glibc, musl)
- build with modern compilers
- build with modern C11 standard
- build reproducibly

“Build Distribution”

"Build Distribution"

Project: Bootstrappable Toolchain

Using the open source build system [JustBuild](#)

- bootstraps latest compilers (GCC13/Clang17)
- bootstraps latest build tools (make/cmake/python)
- requires only a reduced binary seed
 - Coreutils
 - POSIX shell
 - C compiler (e.g., TinyCC)
- all toolchains are built from source
- uses existing build descriptions (make/cmake)
- on-demand tool bootstrap (minimal Linux distribution)
 - ↪ aka the "Build Distribution" for toolchains

The screenshot shows the GitHub repository page for 'Bootstrappable Toolchain'. At the top, there is a navigation bar with 'master' selected and a 'Code' button. Below the navigation bar, there are tabs for 'README', 'Apache-2.0 license', and 'GPL-2.0 license'. The main content area features the repository name 'Bootstrappable Toolchain' in a large font, followed by a description: 'This repository provides compiler toolchains and additional build tools that are acquired via [Bootstrappable Builds](#). For details about the bootstrap process, see [BOOTSTRAP.md](#).' Below this, it lists 'Available compiler toolchains are:' followed by a list of toolchain names: gcc-latest-native, gcc-13.2.0-native, clang-latest-native, clang-17.0.5-native, clang-16.0.6-native, gcc-latest-musl, gcc-13.2.0-musl, gcc-latest-musl-static, and gcc-13.2.0-musl-static. To the right of the list is a QR code. At the bottom, it says 'For details about how these compilers are built, see [COMPILERS.md](#).'

"Build Distribution"

Project: Bootstrappable Toolchain

Using the open source build system [JustBuild](#)

- bootstraps latest compilers (GCC13/Clang17)
- bootstraps latest build tools (make/cmake/python)
- requires only a reduced binary seed
 - Coreutils
 - POSIX shell
 - C compiler (e.g., TinyCC)
- all toolchains are built from source
- uses existing build descriptions (make/cmake)
- on-demand tool bootstrap (minimal Linux distribution)
 - ↪ aka the "Build Distribution" for toolchains

The screenshot shows the GitHub repository page for 'Bootstrappable Toolchain'. At the top, there is a navigation bar with 'master' selected and a 'Code' button. Below the navigation bar, there are links for 'README', 'Apache-2.0 license', and 'GPL-2.0 license'. The main heading is 'Bootstrappable Toolchain'. The description states: 'This repository provides compiler toolchains and additional build tools that are acquired via [Bootstrappable Builds](#). For details about the bootstrap process, see [BOOTSTRAP.md](#).' Below this, it says 'Available compiler toolchains are:' followed by a list of toolchain names: gcc-latest-native, gcc-13.2.0-native, clang-latest-native, clang-17.0.5-native, clang-16.0.6-native, gcc-latest-musl, gcc-13.2.0-musl, gcc-latest-musl-static, and gcc-13.2.0-musl-static. To the right of the list is a QR code. At the bottom, it says 'For details about how these compilers are built, see [COMPILERS.md](#).'

"Build Distribution"

Project: Bootstrappable Toolchain

Using the open source build system [JustBuild](#)

- bootstraps latest compilers (GCC13/Clang17)
- bootstraps latest build tools (make/cmake/python)
- requires only a reduced binary seed
 - Coreutils
 - POSIX shell
 - C compiler (e.g., TinyCC)
- all toolchains are built from source
- uses existing build descriptions (make/cmake)
- on-demand tool bootstrap (minimal Linux distribution)
 - ↪ aka the "Build Distribution" for toolchains

The screenshot shows the GitHub repository page for 'Bootstrappable Toolchain'. At the top, there is a navigation bar with 'master' selected and a 'Code' button. Below the navigation bar, there are tabs for 'README', 'Apache-2.0 license', and 'GPL-2.0 license'. The main heading is 'Bootstrappable Toolchain'. The description states: 'This repository provides compiler toolchains and additional build tools that are acquired via [Bootstrappable Builds](#). For details about the bootstrap process, see [BOOTSTRAP.md](#).' Below this, it says 'Available compiler toolchains are:' followed by a list of toolchain names: gcc-latest-native, gcc-13.2.0-native, clang-latest-native, clang-17.0.5-native, clang-16.0.6-native, gcc-latest-musl, gcc-13.2.0-musl, gcc-latest-musl-static, and gcc-13.2.0-musl-static. To the right of the list is a QR code. At the bottom, it says 'For details about how these compilers are built, see [COMPILERS.md](#).'

"Build Distribution"

Project: Bootstrappable Toolchain

Using the open source build system [JustBuild](#)

- bootstraps latest compilers (GCC13/Clang17)
- bootstraps latest build tools (make/cmake/python)
- requires only a reduced binary seed
 - Coreutils
 - POSIX shell
 - C compiler (e.g., TinyCC)
- all toolchains are built from source
- uses existing build descriptions (make/cmake)
- on-demand tool bootstrap (minimal Linux distribution)
 - ↪ aka the "Build Distribution" for toolchains

The screenshot shows the GitHub repository page for 'Bootstrappable Toolchain'. At the top, there is a navigation bar with 'master' selected and a 'Code' button. Below the navigation bar, there are tabs for 'README', 'Apache-2.0 license', and 'GPL-2.0 license'. The main heading is 'Bootstrappable Toolchain'. The description states: 'This repository provides compiler toolchains and additional build tools that are acquired via [Bootstrappable Builds](#). For details about the bootstrap process, see [BOOTSTRAP.md](#).' Below this, it says 'Available compiler toolchains are:' followed by a list of toolchain names: gcc-latest-native, gcc-13.2.0-native, clang-latest-native, clang-17.0.5-native, clang-16.0.6-native, gcc-latest-musl, gcc-13.2.0-musl, gcc-latest-musl-static, and gcc-13.2.0-musl-static. To the right of the list is a QR code. At the bottom, it says 'For details about how these compilers are built, see [COMPILERS.md](#).'

"Build Distribution"

Project: Bootstrappable Toolchain

Using the open source build system [JustBuild](#)

- bootstraps latest compilers (GCC13/Clang17)
- bootstraps latest build tools (make/cmake/python)
- requires only a reduced binary seed
 - Coreutils
 - POSIX shell
 - C compiler (e.g., TinyCC)
- all toolchains are built from source
- uses existing build descriptions (make/cmake)
- on-demand tool bootstrap (minimal Linux distribution)
 - ↪ aka the "Build Distribution" for toolchains

The screenshot shows the GitHub repository page for 'Bootstrappable Toolchain'. At the top, there is a navigation bar with 'master' selected and a 'Code' button. Below the navigation bar, there are links for 'README', 'Apache-2.0 license', and 'GPL-2.0 license'. The main heading is 'Bootstrappable Toolchain'. The description states: 'This repository provides compiler toolchains and additional build tools that are acquired via [Bootstrappable Builds](#). For details about the bootstrap process, see [BOOTSTRAP.md](#).' Below this, it says 'Available compiler toolchains are:' followed by a list of toolchain names: gcc-latest-native, gcc-13.2.0-native, clang-latest-native, clang-17.0.5-native, clang-16.0.6-native, gcc-latest-musl, gcc-13.2.0-musl, gcc-latest-musl-static, and gcc-13.2.0-musl-static. To the right of the list is a QR code. At the bottom, it says 'For details about how these compilers are built, see [COMPILERS.md](#).'

Patching GCC 4.7.4

Patching GCC 4.7.4

Maintenance patches and backports

- building with C11 [[Mike Frysinger](#)]
- musl support in `config.sub` [[4.8.0](#)]
- general musl support [[6.1.0](#)]
- musl linker support [[6.1.0](#)]
- libstdc++ support for musl [[6.1.0](#)]
- use new type name `context_t` [[8.1.0](#)]
- musl detection in `config.guess`
[[9.1.0](#),[11.1.0](#),[11.1.0](#)]

Patching GCC 4.7.4

Maintenance patches and backports

- building with C11 [[Mike Frysinger](#)]
- musl support in config.sub [[4.8.0](#)]
- **general musl support** [[6.1.0](#)]
- musl linker support [[6.1.0](#)]
- libstdc++ support for musl [[6.1.0](#)]
- use new type name context_t [[8.1.0](#)]
- musl detection in config.guess [[9.1.0](#),[11.1.0](#),[11.1.0](#)]

```

1 --- a/gcc/config/linux.h
2 +++ b/gcc/config/linux.h
3 @@ -33,10 +33,12 @@ see the files COPYING3 and COPYING.
4 #define OPTION_GLIBC (DEFAULT_LIBC == LIBC_GLIBC)
5 #define OPTION_UCLIBC (DEFAULT_LIBC == LIBC_UCLIBC)
6 #define OPTION_BIONIC (DEFAULT_LIBC == LIBC_BIONIC)
7 +#define OPTION_MUSL (DEFAULT_LIBC == LIBC_MUSL)
8 #else
9 #define OPTION_GLIBC (linux_libc == LIBC_GLIBC)
10 #define OPTION_UCLIBC (linux_libc == LIBC_UCLIBC)
11 #define OPTION_BIONIC (linux_libc == LIBC_BIONIC)
12 +#define OPTION_MUSL (linux_libc == LIBC_MUSL)
13 #endif

```

Patching GCC 4.7.4

Maintenance patches and backports

- building with C11 [[Mike Frysinger](#)]
- musl support in config.sub [[4.8.0](#)]
- general musl support [[6.1.0](#)]
- **musl linker support** [[6.1.0](#)]
- libstdc++ support for musl [[6.1.0](#)]
- use new type name context_t [[8.1.0](#)]
- musl detection in config.guess [[9.1.0](#),[11.1.0](#),[11.1.0](#)]

```

1 --- a/gcc/config/i386/linux64.h
2 +++ b/gcc/config/i386/linux64.h
3 @@ -31,3 +31,10 @@ see the files COPYING3 and COPYING.R
4 #define GLIBC_DYNAMIC_LINKER32 "/lib/ld-linux.so.2"
5 #define GLIBC_DYNAMIC_LINKER64 "/lib64/ld-linux-x86-64
6 #define GLIBC_DYNAMIC_LINKERX32 "/libx32/ld-linux-x32.
7 +
8 +#undef MUSL_DYNAMIC_LINKER32
9 +#define MUSL_DYNAMIC_LINKER32 "/lib/ld-musl-i386.so.1"
10 +#undef MUSL_DYNAMIC_LINKER64
11 +#define MUSL_DYNAMIC_LINKER64 "/lib/ld-musl-x86_64.so
12 +#undef MUSL_DYNAMIC_LINKERX32
13 +#define MUSL_DYNAMIC_LINKERX32 "/lib/ld-musl-x32.so.1"

```


Patching Reproducibility of GCC 4.7.4

JustBuild builds in isolation (i.e., using temp directory, possibly in the user's home).

Problem: `cc1/cc1plus` contain checksums, computed from:

1. the path of the linker that was used
2. the relevant object files

... both contain the build directory.

Compute Reproducible Checksums!

independently of the build directory:

- strip linker path
- strip debug symbols from objects

Patching Reproducibility of GCC 4.7.4

JustBuild builds in isolation (i.e., using temp directory, possibly in the user's home).

Problem: **cc1/cc1plus** contain checksums, computed from:

1. the path of the linker that was used
2. the relevant object files

... both contain the build directory.

Compute Reproducible Checksums!

independently of the build directory:

- strip linker path
- strip debug symbols from objects



Patching Reproducibility of GCC 4.7.4

JustBuild builds in isolation (i.e., using temp directory, possibly in the user's home).

Problem: `cc1/cc1plus` contain checksums, computed from:

1. the path of the linker that was used
2. the relevant object files

... both contain the build directory.

Compute Reproducible Checksums!

independently of the build directory:

- strip linker path
- strip debug symbols from objects

```

1 --- a/gcc/Makefile.in
2 +++ b/gcc/Makefile.in
3 @@ -1804,12 +1804,17 @@ gcc-cross$(exeext): xgcc$(exeext)
4
5 checksum-options:
6     echo "$$(LINKER) $$(ALL_LINKERFLAGS) $$(LDFLAGS)" > checks
7 +   && sed -i 's|'$$${BUILD_ROOT_DIR:-/nonexistent}'|/build|
8     && $(srcdir)/../move-if-change checksum-options.tmp che
9
10 # compute checksum over all object files and the options
11 cc1-checksum.c : build/genchecksum$(build_exeext) checksum
12     $(C_OBJS) $(BACKEND) $(LIBDEPS)
13 -   build/genchecksum$(build_exeext) $(C_OBJS) $(BACKEND) $
14 +   rm -rf stripped_c_checksum_inputs; \
15 +   mkdir stripped_c_checksum_inputs; \
16 +   cp $(C_OBJS) $(BACKEND) $(LIBDEPS) stripped_c_checksum_
17 +   $(STRIP_FOR_TARGET) -g stripped_c_checksum_inputs/*; \
18 +   build/genchecksum$(build_exeext) $$$(ls stripped_c_check
19     checksum-options > cc1-checksum.c.tmp &&
20     $(srcdir)/../move-if-change cc1-checksum.c.tmp cc1-check

```



Patching Reproducibility of GCC 4.7.4

JustBuild builds in isolation (i.e., using temp directory, possibly in the user's home).

Problem: `cc1/cc1plus` contain checksums, computed from:

1. the path of the linker that was used
2. the relevant object files

... both contain the build directory.

Compute Reproducible Checksums!

independently of the build directory:

- strip linker path
- strip debug symbols from objects

```

1 --- a/gcc/Makefile.in
2 +++ b/gcc/Makefile.in
3 @@ -1804,12 +1804,17 @@ gcc-cross$(exeext): xgcc$(exeext)
4
5 checksum-options:
6     echo "$(LINKER) $(ALL_LINKERFLAGS) $(LDFLAGS)" > checksum-opts.tmp
7 +   && sed -i 's|'$$${BUILD_ROOT_DIR}:/nonexistent}'|/build|
8     && $(srcdir)/../move-if-change checksum-options.tmp checksum-options
9
10 # compute checksum over all object files and the options
11 cc1-checksum.c : build/genchecksum$(build_exeext) checksum-opts.tmp
12 $(C_OBJS) $(BACKEND) $(LIBDEPS)
13 - build/genchecksum$(build_exeext) $(C_OBJS) $(BACKEND) $(LIBDEPS)
14 + rm -rf stripped_c_checksum_inputs; \
15 + mkdir stripped_c_checksum_inputs; \
16 + cp $(C_OBJS) $(BACKEND) $(LIBDEPS) stripped_c_checksum_inputs; \
17 + $(STRIP_FOR_TARGET) -g stripped_c_checksum_inputs/*; \
18 + build/genchecksum$(build_exeext) $(ls stripped_c_checksum_inputs)
19     checksum-options > cc1-checksum.c.tmp &&
20 $(srcdir)/../move-if-change cc1-checksum.c.tmp cc1-checksum.c

```

Note that all patches are applied automatically as part of the *bootstrap process*.

Bootstrap Process



Bootstrap Stage-0: *GCC-4.7.4*

Stage-0

toolchain stage-0

coreutils, sh, cc

Bootstrap Stage-0: GCC-4.7.4

Stage-0

toolchain stage-0

```
graph LR; A[toolchain stage-0] --> B[bootstrap busybox];
```

bootstrap busybox

coreutils, sh, cc

+ grep, find,
sed, patch, ...

Bootstrap Stage-0: GCC-4.7.4

Stage-0

toolchain stage-0



bootstrap busybox



bootstrap make

coreutils, sh, cc

+ grep, find,
sed, patch, ...

+ make

Bootstrap Stage-0: GCC-4.7.4

Stage-0

toolchain stage-0

bootstrap busybox

bootstrap make

bootstrap ar

coreutils, sh, cc

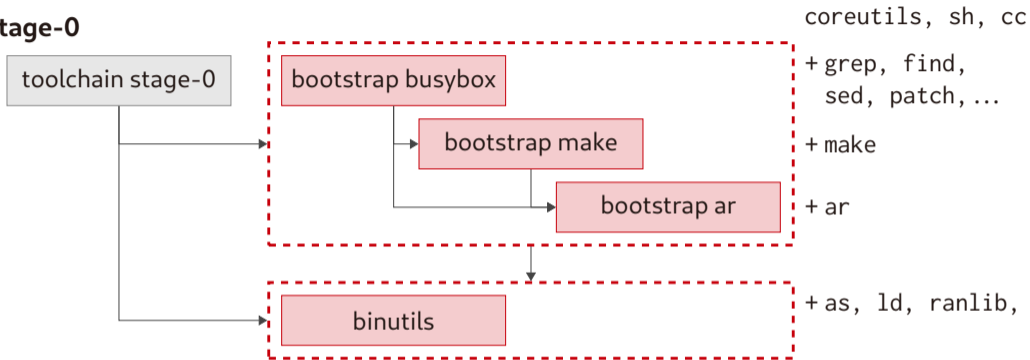
+ grep, find,
sed, patch, ...

+ make

+ ar

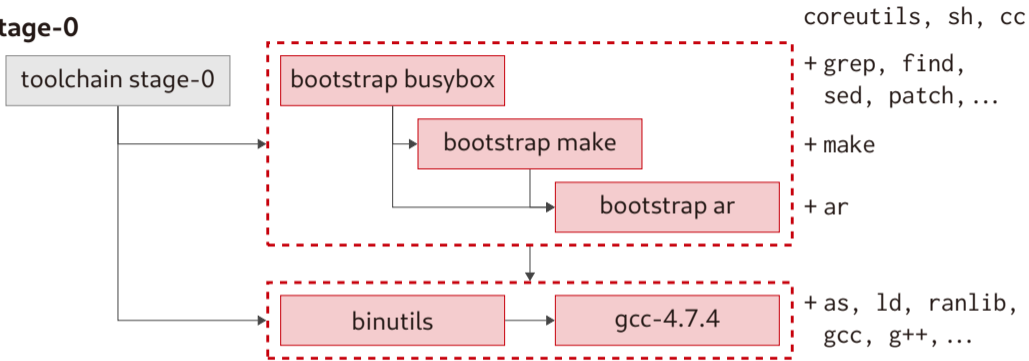
Bootstrap Stage-0: GCC-4.7.4

Stage-0



Bootstrap Stage-0: GCC-4.7.4

Stage-0



Run bootstrap and install with **JustBuild**:

```
just-mr --main stage-0/gcc install toolchain -o /opt/gcc-4.7.4
```

Wrap-up

Wrap-up

Works on any x86_64 Linux system

Tested on ArchLinux, Debian, Fedora, and VoidLinux, but also very different systems:

NixOS only GCC/Clang + glibc

prologic/ulinux TinyCC + musl libc
(≈ 6 MB on [DockerHub](#))

Use as explicit dependency

Make the toolchain a committed project dependency

- easier project setup
- git bisect support
- "predict" binary hashes!
(see our [demo application](#))

Install [JustBuild](#) and try it yourself:

```
1 git clone https://github.com/just-buildsystem/bootstrappable-toolchain.git
2 cd bootstrappable-toolchain
3 just-mr --main gcc-13.2.0-musl install -D'{"ARCH":"x86_64"}' -o /opt/gcc
4 # grab some coffee and wait...
```

Wrap-up

Works on any x86_64 Linux system

Tested on ArchLinux, Debian, Fedora, and VoidLinux, but also very different systems:

NixOS only GCC/Clang + glibc

prologic/ulinux TinyCC + musl libc
(\approx 6 MB on [DockerHub](#))

Use as explicit dependency

Make the toolchain a committed project dependency

- easier project setup
- git bisect support
- "predict" binary hashes!
(see our [demo application](#))

Install [JustBuild](#) and try it yourself:

```
1 git clone https://github.com/just-buildsystem/bootstrappable-toolchain.git
2 cd bootstrappable-toolchain
3 just-mr --main gcc-13.2.0-musl install -D'{"ARCH":"x86_64"}' -o /opt/gcc
4 # grab some coffee and wait...
```

Wrap-up

Works on any x86_64 Linux system

Tested on ArchLinux, Debian, Fedora, and VoidLinux, but also very different systems:

NixOS only GCC/Clang + glibc

prologic/ulinux TinyCC + musl libc
(≈6 MB on [DockerHub](#))

Use as explicit dependency

Make the toolchain a committed project dependency

- easier project setup
- git bisect support
- "predict" binary hashes!
(see our [demo application](#))

Install [JustBuild](#) and try it yourself:

```
1 git clone https://github.com/just-buildsystem/bootstrappable-toolchain.git
2 cd bootstrappable-toolchain
3 just-mr --main gcc-13.2.0-musl install -D'{"ARCH":"x86_64"}' -o /opt/gcc
4 # grab some coffee and wait...
```

Thank you.

Special thanks to the community!

- Rich Felker <https://github.com/richfelker/musl-cross-make>
- <https://bootstrappable.org>
- <https://reproducible-builds.org>
- https://bootstrapping.miraheze.org/wiki/C_compilers