Red Hat Developer Tools 1

Using LLVM 17.0.6 Toolset

Installing and using LLVM 17.0.6 Toolset

Last Updated: 2024-05-23

Red Hat Developer Tools 1 Using LLVM 17.0.6 Toolset

Installing and using LLVM 17.0.6 Toolset

Jacob Valdez
jvaldez@redhat.com

Legal Notice

Abstract

LLVM Toolset is a Red Hat offering for developers on the Red Hat Enterprise Linux (RHEL) operating system. Use this guide for an overview of LLVM Toolset, to learn how to invoke and use different versions of LLVM tools, and to find resources with more in-depth information.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. LLVM TOOLSET

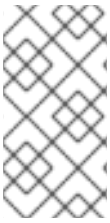
LLVM Toolset is a Red Hat offering for developers on Red Hat Enterprise Linux (RHEL). It provides the LLVM compiler infrastructure framework, the Clang compiler for the C and C++ languages, the LLDB debugger, and related tools for code analysis.

For Red Hat Enterprise Linux 8, LLVM Toolset is available as a module. LLVM Toolset is available as packages for Red Hat Enterprise Linux 9.

1.1. LLVM TOOLSET COMPONENTS

The following components are available as a part of LLVM Toolset:

Name	Version	Description
clang	17.0.6	An LLVM compiler front end for C and C++.
lldb	17.0.6	A C and C++ debugger using portions of LLVM.
compiler-rt	17.0.6	Runtime libraries for LLVM and Clang.
llvm	17.0.6	A collection of modular and reusable compiler and toolchain technologies.
libomp	17.0.6	A library for using Open MP API specification for parallel programming.
lld	17.0.6	An LLVM linker.
python-lit	17.0.6	A software testing tool for LLVM- and Clang-based test suites.



NOTE

The CMake build manager is not part of LLVM Toolset. On Red Hat Enterprise Linux 8, CMake is available in the system repository. On Red Hat Enterprise Linux 9, CMake is available in the system repository. For more information on how to install CMake, see Installing CMake on Red Hat Enterprise Linux .

1.2. LLVM TOOLSET COMPATIBILITY

LLVM Toolset is available for Red Hat Enterprise Linux 8 and Red Hat Enterprise Linux 9 on the following architectures:

- AMD and Intel 64-bit

- 64-bit ARM
- IBM Power Systems, Little Endian
- 64-bit IBM Z

1.3. INSTALLING LLVM TOOLSET

Complete the following steps to install LLVM Toolset including all development and debugging tools as well as dependent packages.

Prerequisites

- All available Red Hat Enterprise Linux updates are installed.

Procedure

On Red Hat Enterprise Linux 8, install the **llvm-toolset** module by running:

```
# yum module install llvm-toolset
```

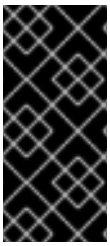
IMPORTANT

This does not install the LLDB debugger or the **python3-lit** package on Red Hat Enterprise Linux 8. To install the LLDB debugger and the **python3-lit** package, run:

```
# yum install lldb python3-lit
```

On Red Hat Enterprise Linux 9, install the **llvm-toolset** package by running:

```
# dnf install llvm-toolset
```



IMPORTANT

This does not install the LLDB debugger or the **python3-lit** package on Red Hat Enterprise Linux 9. To install the LLDB debugger and the **python3-lit** package, run:

```
# dnf install lldb python3-lit
```

1.4. INSTALLING THE CMAKE BUILD MANAGER

The CMake build manager is a tool that manages the build process of your source code independently from your compiler. CMake can generate a native build environment to compile source code, create libraries, generate wrappers, and build executable files.
Complete the following steps to install the CMake build manager.

Prerequisites

- LLVM Toolset is installed.
  For more information, see Installing LLVM Toolset.

Procedure

To install CMake, run the following command:

- On Red Hat Enterprise Linux 8:

```
# yum install cmake
```

- On Red Hat Enterprise Linux 9:

```
# dnf install cmake
```

Additional resources

- For more information on the CMake build manager, see the official CMake documentation overview About CMake.
- For an introduction to using the CMake build manager, see:
  - The CMake Reference Documentation Introduction.
  - The official CMake documentation CMake Tutorial.

1.5. INSTALLING LLVM TOOLSET DOCUMENTATION

You can install documentation for LLVM Toolset on your local system.

Prerequisites

- LLVM Toolset is installed.
  For more information, see Installing LLVM Toolset.

Procedure

To install the **llvm-doc** package, run the following command:

- On Red Hat Enterprise Linux 8:

  ```
  # yum install llvm-doc
  ```

  You can find the documentation under the following path:
  /usr/share/doc/llvm/html/index.html.

- On Red Hat Enterprise Linux 9:

  ```
  # dnf install llvm-doc
  ```

  You can find the documentation under the following path:
  /usr/share/doc/llvm/html/index.html.

1.6. INSTALLING CMAKE DOCUMENTATION

You can install documentation for the CMake build manager on your local system.

Prerequisites

- CMake is installed.
  For more information, see Installing the CMake build manager .

Procedure

To install the **cmake-doc** package, run the following command:

- On Red Hat Enterprise Linux 8:

  ```
  # yum install cmake-doc
  ```

  You can find the documentation under the following path:
  /usr/share/doc/cmake/html/index.html.

- On Red Hat Enterprise Linux 9:

  ```
  # dnf install cmake-doc
  ```

  You can find the documentation under the following path:
  /usr/share/doc/cmake/html/index.html.

1.7. ADDITIONAL RESOURCES

- For more information on LLVM Toolset, see the official LLVM documentation.

CHAPTER 2. THE CLANG COMPILER

Clang is an LLVM compiler front end for the C-based languages C, C++, Objective C/C++, OpenCL, and Cuda.

LLVM Toolset is distributed with Clang 17.0.6.

2.1. PREREQUISITES

- LLVM Toolset is installed.
  For more information, see Installing LLVM Toolset.

2.2. COMPILING A SOURCE FILE

Use the Clang compiler to compile source files as well as assembly language source files. Clang creates an executable binary file as a result of compiling. To be able to debug your code, enable debug information by adding the **-g** flag to your Clang commands.

NOTE

To compile a C++ program, use **clang++** instead of **clang**.

Procedure

To compile your program, run the following command:

- On Red Hat Enterprise Linux 8:

```
$ clang -o -g <binary_file> <source_file>
```

  - Replace ***<binary_file>*** with the desired name of your output file and ***<source_file>*** with the name of your source file.

- On Red Hat Enterprise Linux 9:

```
$ clang -o -g <binary_file> <source_file>
```

  - Replace ***<binary_file>*** with the desired name of your output file and ***<source_file>*** with the name of your source file.

2.3. RUNNING A PROGRAM

The Clang compiler creates an executable binary file as a result of compiling. Complete the following steps to execute this file and run your program.

Prerequisites

- Your program is compiled.
  For more information on how to compile your program, see Compiling a source file.

Procedure

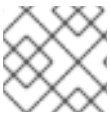
To run your program, run in the directory containing the executable file:

```
$ ./<binary_file>
```

- Replace ***<binary_file>*** with the name of your executable file.

2.4. LINKING OBJECT FILES TOGETHER

By linking object files together, you can compile only source files that contain changes instead of your entire project.
When you are working on a project that consists of several source files, use the Clang compiler to compile an object file for each of the source files. As a next step, link those object files together. Clang automatically generates an executable file containing your linked object files. After compilation, link your object files together again.



NOTE

To compile a C++ program, use **clang++** instead of **clang**.

Procedure

1. To compile a source file to an object file, run the following command:
   - On Red Hat Enterprise Linux 8:

     ```
     $ clang -o <object_file> -c <source_file>
     ```

     - Replace ***<object_file>*** with the desired name of your object file and ***<source_file>*** with the name of your source file.
   - On Red Hat Enterprise Linux 9:

     ```
     $ clang -o <object_file> -c <source_file>
     ```

     - Replace ***<object_file>*** with the desired name of your object file and ***<source_file>*** with the name of your source file.
2. To link object files together, run the following command:
   - On Red Hat Enterprise Linux 8:

     ```
     $ clang -o <output_file> <object_file_0> <object_file_1>
     ```

     - Replace ***<output_file>*** with the desired name of your output file and ***<object_file>*** with the names of the object files you want to link.
   - On Red Hat Enterprise Linux 9:

     ```
     $ clang -o <output_file> <object_file_0> <object_file_1>
     ```

     - Replace ***<output_file>*** with the desired name of your output file and ***<object_file>*** with the names of the object files you want to link.

IMPORTANT

At the moment, certain library features are statically linked into applications built with LLVM Toolset to support their execution on multiple versions of Red Hat Enterprise Linux. This creates a small security risk. Red Hat will issue a security erratum in case you need to rebuild your applications due to this risk.

Red Hat advises to not statically link your entire application.

2.5. ADDITIONAL RESOURCES

- For more information on the Clang compiler, see the official Clang compiler documentation .
- To display the manual page included in LLVM Toolset, run:

NOTE

To compile a C++ program, use **clang++** instead of **clang**.

- On Red Hat Enterprise Linux 8:

```
$ man clang
```

- On Red Hat Enterprise Linux 9:

```
$ man clang
```

CHAPTER 3. THE LLDB DEBUGGER

The LLDB debugger is a command-line tool for debugging C and C++ programs. Use LLDB to inspect memory within the code being debugged, control the execution state of the code, and detect the execution of particular sections of code.

LLVM Toolset is distributed with LLDB 17.0.6.

3.1. PREREQUISITES

- LLVM Toolset is installed.
  For more information, see Installing LLVM Toolset.

- Your compiler is configured to create debug information.
  For instructions on configuring the Clang compiler, see Controlling Debug Information in the Clang Compiler User's Manual.
  For instructions on configuring the GCC compiler, see Preparing a Program for Debugging in the Red Hat Developer Toolset User Guide.

3.2. STARTING A DEBUGGING SESSION

Use LLDB to start an interactive debugging session.

Procedure

- To run LLDB on a program you want to debug, use the following command:
  - On Red Hat Enterprise Linux 8:

    ```
    $ lldb <binary_file_name>
    ```

    - Replace ***<binary_file>*** with the name of your compiled program.
      You have started your LLDB debugging session in interactive mode. Your command-line terminal now displays the default prompt **(lldb)**.

  - On Red Hat Enterprise Linux 9:

    ```
    $ lldb <binary_file>
    ```

    - Replace ***<binary_file>*** with the name of your compiled program.
      You have started your LLDB debugging session in interactive mode. Your command-line terminal now displays the default prompt **(lldb)**.

- To quit the debugging session and return to the shell prompt, run the following command:

```
(lldb) quit
```

3.3. EXECUTING YOUR PROGRAM DURING A DEBUGGING SESSION

Use LLDB to execute your program during your debugging session. The execution of your program stops when the first breakpoint is reached, when an error occurs, or when the program terminates.

Prerequisites

- You have started an interactive debugging session.
  For more information, see Starting a debugging session with LLDB .

Procedure

- To execute the program you are debugging, run:

```
(lldb) run
```

- To execute the program you are debugging using a specific argument, run:

```
(lldb) run <argument>
```

  - Replace ***<argument>*** with the command-line argument you want to use.

3.4. USING BREAKPOINTS

Use breakpoints to pause the execution of your program at a set point in your source code.

Prerequisites

- You have started an interactive debugging session.
  For more information, see Starting a debugging session with LLDB .

Procedure

- To set a new breakpoint on a specific line, run the following command:

```
(lldb) breakpoint set --file <source_file_name> --line <line_number>
```

  - Replace ***<source_file_name>*** with the name of your source file and ***<line_number>*** with the line number you want to set your breakpoint at.

- To set a breakpoint on a specific function, run the following command:

```
(lldb) breakpoint set --name <function_name>
```

  - Replace ***<function_name>*** with the name of the function you want to set your breakpoint at.

- To display a list of currently set breakpoints, run the following command:

```
(lldb) breakpoint list
```

- To delete a breakpoint, run:

```
(lldb) breakpoint clear -f <source_file_name> -l <line_number>
```

  - Replace ***<source_file_name>*** with the name of your source file and ***<line_number>*** with line number of the breakpoint you want to delete.

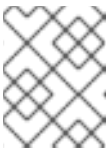
- To resume the execution of your program after it reached a breakpoint, run:

```
(lldb) continue
```

- To skip a specific number of breakpoints, run the following command:

```
(lldb) continue -i <breakpoints_to_skip>
```

  - Replace ***<breakpoints_to_skip>*** with the number of breakpoints you want to skip.



NOTE

To skip a loop, set the ***<breakpoints_to_skip>*** to match the loop iteration count.

3.5. STEPPING THROUGH CODE

You can use LLDB to step through the code of your program to execute only one line of code after the line pointer.

Prerequisites

- You have started an interactive debugging session.
  For more information, see Starting a debugging session with LLDB .

Procedure

- To step through one line of code:
  1. Set your line pointer to the line you want to execute.
  2. Run the following command:

```
(lldb) step
```

- To step through a specific number of lines of code:
  1. Set your line pointer to the line you want to execute.
  2. Run the following command:

```
(lldb) step -c <number>
```

     - Replace ***<number>*** with the number of lines you want to execute.

3.6. LISTING SOURCE CODE

Before you execute the program you are debugging, the LLDB debugger automatically displays the first 10 lines of source code. Each time the execution of the program is stopped, LLDB displays the line of source code on which it stopped as well as its surrounding lines. You can use LLDB to manually trigger the display of source code during your debugging session.

Prerequisites

- You have started an interactive debugging session.
  For more information, see Starting a debugging session with LLDB .

Procedure

- To list the first 10 lines of the source code of the program you are debugging, run:

```
(lldb) list
```

- To display the source code from a specific line, run:

```
(lldb) list <source_file_name>:<line_number>
```

  - Replace ***<source_file_name>*** with the name of your source file and ***<line_number>*** with the number of the line you want to display.

3.7. DISPLAYING CURRENT PROGRAM DATA

The LLDB debugger provides data on variables of any complexity, any valid expressions, and function call return values. You can use LLDB to display data relevant to the program state.

Prerequisites

- You have started an interactive debugging session.
  For more information, see Starting a debugging session with LLDB .

Procedure

To display the current value of a certain variable, expression, or return value, run:

```
(lldb) print <data_name>
```

- Replace ***<data_name>*** with data you want to display.

3.8. ADDITIONAL RESOURCES

- For more information on the LLDB debugger, see the official LLDB documentation LLDB Tutorial.
- For a list of GDB commands and their LLDB equivalents, see the GDB to LLDB Command Map .

CHAPTER 4. CONTAINER IMAGES WITH LLVM TOOLSET ON RHEL 8

On RHEL 8, you can build your own LLVM Toolset container images on top of Red Hat Universal Base Images (UBI) containers using Containerfiles.

4.1. CREATING A CONTAINER IMAGE OF LLVM TOOLSET ON RHEL 8

On RHEL 8, LLVM Toolset packages are part of the Red Hat Universal Base Images (UBIs) repositories. To keep the container image size small, install only individual packages instead of the entire LLVM Toolset.

Prerequisites

- An existing Containerfile.
  For information on creating Containerfiles, see the Dockerfile reference page.

Procedure

- Visit the Red Hat Container Catalog .
- Select a UBI.
- Click **Get this image** and follow the instructions.
- To create a container image containing LLVM Toolset, add the following lines to your Containerfile:

```
FROM registry.access.redhat.com/ubi8/ubi:latest

RUN yum module install -y llvm-toolset
```

- To create a container image containing an individual package only, add the following lines to your Containerfile:

```
RUN yum install -y <package-name>
```

  - Replace ***<package-name>*** with the name of the package you want to install.

4.2. ADDITIONAL RESOURCES

- For more information on Red Hat UBI images, see Working with Container Images.
- For more information on Red Hat UBI repositories, see Universal Base Images (UBI): Images, repositories, packages, and source code.

CHAPTER 5. CHANGES IN LLVM TOOLSET

LLVM Toolset has been updated from version 16.0.1 to 17.0.6 on RHEL 8 and RHEL 9. Notable changes include:

- The opaque pointers migration is now completed.
- Removed support for the legacy pass manager in middle-end optimization.

Clang changes:

- C++20 coroutines are no longer considered experimental.
- Improved code generation for the **std::move** function and similar in unoptimized builds.

For detailed information regarding the updates, see LLVM and Clang upstream release notes.