



# Red Hat Developer Tools 1

## LLVM 15.0.7 Toolset の使用

LLVM 15.0.7 Toolset のインストールおよび使用



# Red Hat Developer Tools 1 LLVM 15.0.7 Toolset の使用

---

LLVM 15.0.7 Toolset のインストールおよび使用

Jacob Valdez

[jvaldez@redhat.com](mailto:jvaldez@redhat.com)

## 法律上の通知

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

LLVM Toolset は、Red Hat Enterprise Linux (RHEL) オペレーティングシステム上の開発者向けの Red Hat 製品です。本書では、LLVM Toolset の概要、さまざまなバージョンの LLVM ツールを起動して使用する方法、および詳細な情報を含むリソースを検索する方法を説明します。

## 目次

多様性を受け入れるオープンソースの強化 .....	3
RED HAT ドキュメントへのフィードバック (英語のみ) .....	4
<b>第1章 LLVM TOOLSET .....</b>	<b>5</b>
1.1. LLVM TOOLSET コンポーネント .....	5
1.2. LLVM TOOLSET の互換性 .....	5
1.3. LLVM TOOLSET のインストール .....	6
1.4. CMAKE ビルドマネージャーのインストール .....	6
1.5. LLVM TOOLSET ドキュメントのインストール .....	7
1.6. CMAKE ドキュメントのインストール .....	7
1.7. 関連情報 .....	8
<b>第2章 CLANG コンパイラー .....</b>	<b>9</b>
2.1. 前提条件 .....	9
2.2. ソースファイルのコンパイル .....	9
2.3. プログラムの実行 .....	9
2.4. オブジェクトファイルの連結 .....	10
2.5. 関連情報 .....	11
<b>第3章 LLDB デバッガー .....</b>	<b>12</b>
3.1. 前提条件 .....	12
3.2. デバッグセッションの開始 .....	12
3.3. デバッグセッション中のプログラムの実行 .....	12
3.4. ブレークポイントの使用 .....	13
3.5. コードのステップ実行 .....	14
3.6. ソースコードのリスト表示 .....	14
3.7. 現在のプログラムデータの表示 .....	15
3.8. 関連情報 .....	15
<b>第4章 RHEL 8 で LLVM TOOLSET を使用したコンテナイメージ .....</b>	<b>16</b>
4.1. RHEL 8 での LLVM TOOLSET のコンテナイメージの作成 .....	16
4.2. 関連情報 .....	16
<b>第5章 LLVM TOOLSET の変更点 .....</b>	<b>17</b>



## 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、[Red Hat CTO である Chris Wright のメッセージ](#) を参照してください。

## RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

### 特定の文章に関するコメントの送信

1. **Multi-page HTML** 形式でドキュメントを表示し、ページが完全にロードされてから右上隅に **Feedback** ボタンが表示されていることを確認します。
2. カーソルを使用して、コメントを追加するテキスト部分を強調表示します。
3. 強調表示されたテキストの近くに表示される **Add Feedback** ボタンをクリックします。
4. フィードバックを追加し、**Submit** をクリックします。

### Jira からのフィードバック送信 (アカウントが必要)

1. [Jira](#) の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。



# 第1章 LLVM TOOLSET

LLVM Toolset は、Red Hat Enterprise Linux (RHEL) 上の開発者向けの Red Hat 製品です。これは、LLVM コンパイラインフラストラクチャーフレームワーク、C 言語および C++ 言語用の Clang コンパイラ、LLDB デバッガー、コード解析の関連ツールを提供します。

Red Hat Enterprise Linux 8 では、LLVM Toolset はモジュールとして利用できます。LLVM Toolset は、Red Hat Enterprise Linux 9 のパッケージとして利用できます。

## 1.1. LLVM TOOLSET コンポーネント

以下のコンポーネントは、LLVM Toolset の一部として利用できます。

Name	バージョン	説明
clang	16.0.6	C および C++ の LLVM コンパイラフロントエンド。
lldb	16.0.6	LLVM の一部を使用した C および C++ デバッガー。
compiler-rt	16.0.6	LLVM および Clang のランタイムライブラリー。
llvm	16.0.6	一連のモジュールおよび再利用可能なコンパイラおよびツールチェーン技術。
libomp	16.0.6	並列プログラミングに Open MP API 仕様を使用するためのライブラリー。
lld	16.0.6	LLVM リンカー。
python-lit	16.0.6	LLVM および Clang ベースのテストスイート用のソフトウェアテストツール。



### 注記

CMake ビルドマネージャーは、LLVM Toolset の一部ではありません。Red Hat Enterprise Linux 8 では、CMake がシステムリポジトリから利用できます。Red Hat Enterprise Linux 9 では、CMake がシステムリポジトリから利用できます。CMake のインストール方法は、[CMake ビルドマネージャーのインストール](#) を参照してください。

## 1.2. LLVM TOOLSET の互換性

LLVM Toolset は、Red Hat Enterprise Linux 7 および Red Hat Enterprise Linux 8 では、以下のアーキテクチャーで利用できます。

- AMD および Intel 64 ビット

- 64 ビット ARM
- IBM Power Systems、リトルエンディアン
- 64 ビット IBM Z

### 1.3. LLVM TOOLSET のインストール

すべての開発ツールおよびデバッグツールと依存パッケージを含む LLVM Toolset をインストールするには、以下の手順を実行します。

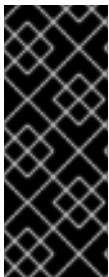
#### 前提条件

- 利用可能な Red Hat Enterprise Linux のすべての更新がインストールされている。

#### 手順

Red Hat Enterprise Linux 8 で、以下のコマンドを実行して **llvm-toolset** モジュールをインストールします。

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



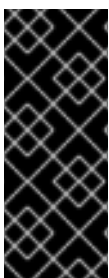
#### 重要

LLDB デバッガーまたは **python3-lit** パッケージは Red Hat Enterprise Linux 8 にインストールされません。LLDB デバッガーと **python3-lit** パッケージをインストールするには、次を実行します。

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

Red Hat Enterprise Linux 9 で、以下のコマンドを実行して **llvm-toolset** パッケージをインストールします。

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



#### 重要

LLDB デバッガーまたは **python3-lit** パッケージは Red Hat Enterprise Linux 9 にインストールされません。LLDB デバッガーと **python3-lit** パッケージをインストールするには、次を実行します。

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

### 1.4. CMAKE ビルドマネージャーのインストール

CMake ビルドマネージャーは、ソースコードのビルドプロセスをコンパイラとは別に管理するツールです。CMake は、ソースコードのコンパイル、ライブラリーの作成、ラッパーの生成、実行可能なファイルの構築を行うネイティブビルド環境を生成できます。

CMake ビルドマネージャーをインストールするには、以下の手順を実行します。

#### 前提条件

- LLVM Toolset がインストールされている。  
詳細は、[LLVM Toolset のインストール](#) を参照してください。

## 手順

CMake をインストールするには、以下のコマンドを実行します。

- Red Hat Enterprise Linux 8 の場合

```
# yum install cmake
```

- Red Hat Enterprise Linux 9 の場合

```
# dnf install cmake
```

## 関連情報

- CMake ビルドマネージャーの詳細は、CMake の公式ドキュメント [About CMake](#) を参照してください。
- CMake ビルドマネージャーの使用の概要は、以下を参照してください。
  - CMake リファレンスドキュメントの [Introduction](#)
  - CMake の公式ドキュメント [CMake Tutorial](#)

## 1.5. LLVM TOOLSET ドキュメントのインストール

LLVM Toolset のドキュメントは、ローカルシステムにインストールできます。

### 前提条件

- LLVM Toolset がインストールされている。  
詳細は、[LLVM Toolset のインストール](#) を参照してください。

## 手順

**llvm-doc** パッケージをインストールするには、以下のコマンドを実行します。

- Red Hat Enterprise Linux 8 の場合

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

ドキュメントは、**/usr/share/doc/llvm/html/index.html** の下にあります。

- Red Hat Enterprise Linux 9 の場合

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

ドキュメントは、**/usr/share/doc/llvm/html/index.html** の下にあります。

## 1.6. CMAKE ドキュメントのインストール

CMake ビルドマネージャーのドキュメントをローカルシステムにインストールできます。

## 前提条件

- Cmake がインストールされている。  
詳細は、[CMake ビルドマネージャーのインストール](#) を参照してください。

## 手順

**cmake-doc** パッケージをインストールするには、以下のコマンドを実行します。

- Red Hat Enterprise Linux 8 の場合

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

ドキュメントは、**/usr/share/doc/cmake/html/index.html** の下にあります。

- Red Hat Enterprise Linux 9 の場合:

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

ドキュメントは、**/usr/share/doc/cmake/html/index.html** の下にあります。

## 1.7. 関連情報

- LLVM Toolset の詳細は、[LLVM の公式ドキュメント](#) を参照してください。

## 第2章 CLANG コンパイラー

Clang は、C ベースの言語 C、C ++、Objective C/C++、OpenCL、および Cuda 用の LLVM コンパイラフロントエンドです。

LLVM Toolset には、Clang 15.0.7 が同梱されています。

### 2.1. 前提条件

- LLVM Toolset がインストールされている。  
詳細は、[LLVM Toolset のインストール](#) を参照してください。

### 2.2. ソースファイルのコンパイル

Clang コンパイラーを使用して、アセンブリ言語ソースファイルとソースファイルをコンパイルします。Clang は、コンパイルの結果として実行可能なバイナリファイルを作成します。コードをデバッグできるようにするには、**-g** フラグを Clang コマンドに追加してデバッグ情報を有効にします。



#### 注記

C++ プログラムをコンパイルするには、**clang** の代わりに **clang++** を使用します。

#### 手順

プログラムをコンパイルするには、以下のコマンドを実行します。

- Red Hat Enterprise Linux 8 の場合

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

- **<binary\_file>** を出力ファイルの必要な名前に、**<source\_file>** をソースファイルの名前に置き換えます。

- Red Hat Enterprise Linux 9 の場合

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

- **<binary\_file>** を出力ファイルの必要な名前に、**<source\_file>** をソースファイルの名前に置き換えます。

### 2.3. プログラムの実行

Clang コンパイラーは、コンパイルの結果として、実行可能なバイナリファイルを作成します。このファイルを実行し、プログラムを実行するには、以下の手順に従います。

#### 前提条件

- プログラムがコンパイルされている。  
プログラムをコンパイルする方法は、[ソースファイルのコンパイル](#) を参照してください。

#### 手順

プログラムを実行するには、実行ファイルが含まれるディレクトリーで実行します。

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

- **<binary\_file>** は、実行可能ファイルの名前に置き換えます。

## 2.4. オブジェクトファイルの連結

オブジェクトファイルを連結して、プロジェクト全体ではなく、変更を含むソースファイルのみをコンパイルできます。

複数のソースファイルで設定されるプロジェクトで作業する場合は、Clang コンパイラを使用して各ソースファイルのオブジェクトファイルをコンパイルします。次のステップとして、これらのオブジェクトファイルを連結します。Clang は、リンクされたオブジェクトファイルを含む実行ファイルを自動的に生成します。コンパイル後に、オブジェクトファイルを再度リンクします。



### 注記

C++ プログラムをコンパイルするには、**clang** の代わりに **clang++** を使用します。

### 手順

1. ソースファイルをオブジェクトファイルにコンパイルするには、以下のコマンドを実行します。

- Red Hat Enterprise Linux 8 の場合

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

- **<object\_file>** をオブジェクトファイルの名前に置き換え、**<source\_file>** をソースファイルの名前に置き換えます。

- Red Hat Enterprise Linux 9 の場合

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

- **<object\_file>** をオブジェクトファイルの名前に置き換え、**<source\_file>** をソースファイルの名前に置き換えます。

2. オブジェクトファイルをリンクするには、以下のコマンドを実行します。

- Red Hat Enterprise Linux 8 の場合

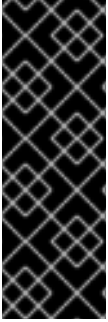
```
$ clang -o <output_file> <object_file_0> <object_file_1>
```

- **<output\_file>** を出力ファイルの必要な名前に置き換え、**<object\_file>** をリンクするオブジェクトファイルの名前に置き換えます。

- Red Hat Enterprise Linux 9 の場合

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

- **<output\_file>** を出力ファイルの必要な名前に置き換え、**<object\_file>** をリンクするオブジェクトファイルの名前に置き換えます。



### 重要

現時点では、Red Hat Enterprise Linux の複数のバージョンでの実行に対応するために、特定のライブラリー機能は LLVM Toolset で構築されたアプリケーションに静的にリンクされています。これにより、セキュリティリスクが小さくなります。このリスクによりアプリケーションを再構築する必要がある場合、Red Hat はセキュリティエラーを発行します。

Red Hat は、アプリケーション全体を静的にリンクしないことを推奨します。

## 2.5. 関連情報

- Clang コンパイラーの詳細は、[Clang コンパイラーの公式ドキュメント](#) を参照してください。
- LLVM Toolset に含まれる man ページを表示するには、以下を実行します。



### 注記

C++ プログラムをコンパイルするには、**clang** の代わりに **clang++** を使用します。

- Red Hat Enterprise Linux 8 の場合

```
$ man clang
```

- Red Hat Enterprise Linux 9 の場合

```
$ man clang
```

## 第3章 LLDB デバッガー

LLDB デバッガーは、C および C++ プログラムのデバッグを行うためのコマンドラインツールです。LLDB を使用してデバッグするコード内でメモリーを検査し、コードの実行状態を制御して、コードの特定セクションの実行を検出します。

LLVM Toolset には、LLDB 15.0.7 が同梱されています。

### 3.1. 前提条件

- LLVM Toolset がインストールされている。  
詳細は、[LLVM Toolset のインストール](#) を参照してください。
- コンパイラーがデバッグ情報を作成するように設定されている。  
Clang コンパイラーの設定方法は、Clang Compiler User's Manual の [Controlling Debug Information](#) を参照してください。  
GCC コンパイラーの設定方法は、Red Hat Developer Toolset User Guide の [デバッグプログラムの準備](#) を参照してください。

### 3.2. デバッグセッションの開始

LLDB を使用して対話式のデバッグセッションを開始します。

#### 手順

- デバッグするプログラムで LLDB を実行するには、以下のコマンドを使用します。
  - Red Hat Enterprise Linux 8 の場合

```
$ lldb <binary_file_name>
```

- **<binary\_file>** をコンパイルしたプログラムの名前に置き換えます。  
対話モードで LLDB デバッグセッションを開始しました。コマンドラインターミナルに、デフォルトのプロンプト (**lldb**) が表示されるようになりました。

- Red Hat Enterprise Linux 9 の場合

```
$ lldb <binary_file>
```

- **<binary\_file>** をコンパイルしたプログラムの名前に置き換えます。  
対話モードで LLDB デバッグセッションを開始しました。コマンドラインターミナルに、デフォルトのプロンプト (**lldb**) が表示されるようになりました。

- デバッグセッションを終了してシェルプロンプトに戻るには、以下のコマンドを実行します。

```
(lldb) quit
```

### 3.3. デバッグセッション中のプログラムの実行

LLDB を使用して、デバッグセッション中にプログラムを実行します。最初のブレークポイントに達するか、エラーが発生した場合、またはプログラムが終了したときにプログラムの実行が停止します。

#### 前提条件



- インタラクティブなデバッグセッションを開始している。  
詳細は、[デバッグセッションの開始](#) を参照してください。

#### 手順

- デバッグしているプログラムを実行するには、次のコマンドを実行します。

```
(lldb) run
```

- 特定の引数を使用してデバッグするプログラムを実行するには、以下を実行します。

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

- **<argument>** を使用するコマンドライン引数に置き換えます。

### 3.4. ブレークポイントの使用

ブレークポイントを使用して、ソースコードの設定ポイントでプログラムの実行を一時停止します。

#### 前提条件

- インタラクティブなデバッグセッションを開始している。  
詳細は、[デバッグセッションの開始](#) を参照してください。

#### 手順

- 特定の行に新しいブレークポイントを設定するには、以下のコマンドを実行します。

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

- **<source\_file\_name>** をソースファイルの名前に置き換え、**<line\_number>** をブレークポイントを設定する行番号に置き換えます。

- 特定の関数にブレークポイントを設定するには、以下のコマンドを実行します。

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

- **<function\_name>** をブレークポイントを設定する関数の名前に置き換えます。

- 現在設定されているブレークポイントのリストを表示するには、以下のコマンドを実行します。

```
(lldb) breakpoint list
```

- ブレークポイントを削除するには、次のコマンドを実行します。

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

- **<source\_file\_name>** をソースファイルの名前に置き換え、**<line\_number>** は削除するブレークポイントの行番号に置き換えます。

- ブレークポイントに達した後にプログラムの実行を再開するには、以下のコマンドを実行します。

```
(lldb) continue
```

- 特定の数のブレークポイントを省略するには、以下のコマンドを実行します。

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

- **<breakpoints\_to\_skip>** をスキップするブレークポイントの数に置き換えます。



#### 注記

ループを省略するには、**<breakpoints\_to\_skip>** をループの反復数に一致するように設定します。

## 3.5. コードのステップ実行

LLDB を使用すると、プログラムのコードをステップ実行して、行ポインターの後のコードを 1 行だけ実行できます。

### 前提条件

- インタラクティブなデバッグセッションを開始している。  
詳細は、[デバッグセッションの開始](#) を参照してください。

### 手順

- コードの 1 行でステップ実行するには、以下を実行します。
  1. 行ポインターを実行する行に設定します。
  2. 以下のコマンドを実行します。

```
(lldb) step
```

- コードの特定の数行をステップ実行するには、以下を行います。
  1. 行ポインターを実行する行に設定します。
  2. 以下のコマンドを実行します。

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

- **<number>** を実行する行数に置き換えます。

## 3.6. ソースコードのリスト表示

デバッグしているプログラムを実行する前に、LLDB デバッガーにソースコードの最初の 10 行が自動的に表示されます。プログラムの実行が停止するたびに、LLDB は停止するソースコードの行と、周りの行を表示します。LLDB を使用すると、デバッグセッションでソースコードの表示を手動でトリガーできます。

### 前提条件

- インタラクティブなデバッグセッションを開始している。  
詳細は、[デバッグセッションの開始](#) を参照してください。

#### 手順

- デバッグしているプログラムのソースコードの最初の 10 行をリスト表示するには、次のコマンドを実行します。

```
(lldb) list
```

- 特定の行からソースコードを表示するには、以下を実行します。

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

- **<source\_file\_name>** をソースファイルの名前に置き換え、**<line\_number>** は表示する行数に置き換えます。

### 3.7. 現在のプログラムデータの表示

LLDB デバッガーは、複雑さ、有効な式、および関数呼び出しの戻り値の変数にデータを提供します。LLDB を使用して、プログラムの状態に関連するデータを表示できます。

#### 前提条件

- インタラクティブなデバッグセッションを開始している。  
詳細は、[デバッグセッションの開始](#) を参照してください。

#### 手順

特定の変数、式、または戻り値の現在の値を表示するには、以下を実行します。

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

- **<data\_name>** を表示するデータに置き換えます。

### 3.8. 関連情報

- LLDB デバッガーの詳細は、LLDB の公式ドキュメント [LLDB Tutorial](#) を参照してください。
- GDB コマンドとそれに相当する LLDB のリストについては、[GDB to LLDB command map](#) を参照してください。

## 第4章 RHEL 8 で LLVM TOOLSET を使用したコンテナイメージ

RHEL 8 では、Containerfiles を使用して、独自の LLVM Toolset コンテナイメージを Red Hat Universal Base Images (UBI) コンテナ上にビルドできます。

### 4.1. RHEL 8 での LLVM TOOLSET のコンテナイメージの作成

RHEL 8 では、LLVM Toolset パッケージは、Red Hat Universal Base Images (UBI) リポジトリの一部です。コンテナイメージのサイズを小さくするには、LLVM Toolset 全体ではなく、個々のパッケージのみをインストールします。

#### 前提条件

- 既存の Containerfile がある。  
Containerfiles 作成の詳細は、[Dockerfile reference](#) のページを参照してください。

#### 手順

- [Red Hat Container Catalog](#) にアクセスします。
- UBI を選択します。
- **Get this image** をクリックして、指示に従います。
- LLVM Toolset を含むコンテナイメージを作成するには、以下の行を Containerfile に追加します。

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

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

- 個々のパッケージのみを含むコンテナイメージを作成するには、以下の行を Containerfile に追加します。

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

- **<package-name>** をインストールするパッケージの名前に置き換えます。

### 4.2. 関連情報

- Red Hat UBI イメージの詳細は、[コンテナイメージの使用](#) を参照してください。
- Red Hat UBI リポジトリの詳細は、[Universal Base Images \(UBI\): イメージ、リポジトリ、パッケージ、およびソースコード](#) を参照してください。

## 第5章 LLVM TOOLSET の変更点

LLVM Toolset が、RHEL 7 および RHEL 8 のバージョン 11.0.0 から 12.0.1 に更新されました。主な変更点は、以下のとおりです。

- 最適化の改善
- 新しい CPU 拡張のサポート
- 新しい C++ バージョンへのサポートの改善

後方互換性のない注目すべき変更は次のとおりです。

- Clang のデフォルトの C++ 標準は、**gnu++14** ではなく **gnu++17** になりました。
- **-Wimplicit-function-declaration**、**-Wimplicit-int**、および **-Wincomplete-function-pointer-types** オプションは、C コードのデフォルトでエラーになるようになりました。これは、configure スクリプトの動作に影響を与える可能性があります。

デフォルトでは、Clang 16 は、GCC Toolset 13 によって提供される **libstdc++** ライブラリーバージョン 13 と **binutils 2.40** を使用します。