



Red Hat Developer Toolset 12

ユーザーガイド

Red Hat Developer Toolset のインストールと使用

Red Hat Developer Toolset 12 ユーザーガイド

Red Hat Developer Toolset のインストールと使用

Jacob Valdez

Red Hat Customer Content Services

jvaldez@redhat.com

Zuzana Zoubková

Red Hat Customer Content Services

Jaromír Hradílek

Red Hat Customer Content Services

Matt Newsome

Red Hat Software Engineering

Robert Krátký

Red Hat Customer Content Services

Vladimír Slávik

Red Hat Customer Content Services

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[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] 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[®] 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

Red Hat Developer Toolset は、Red Hat Enterprise Linux プラットフォームで開発者向けの Red Hat 製品です。Red Hat Developer Toolset User Guide では、本製品の概要、Red Hat Developer Toolset バージョンのツールを起動して使用方法、および詳細な情報を含むリソースへのリンクについて説明しています。

目次

多様性を受け入れるオープンソースの強化	5
パート I. はじめに	6
第1章 RED HAT DEVELOPER TOOLSET	7
1.1. RED HAT DEVELOPER TOOLSET	7
1.2. 主な特長	9
1.3. 互換性	9
1.4. RED HAT DEVELOPER TOOLSET へのアクセスの取得	9
1.5. RED HAT DEVELOPER TOOLSET のインストール	11
1.6. RED HAT DEVELOPER TOOLSET の更新	13
1.7. RED HAT DEVELOPER TOOLSET のアンインストール	13
1.8. RED HAT DEVELOPER TOOLSET コンテナイメージの使用	14
1.9. 関連資料	14
パート II. 開発ツール	16
第2章 GNU コンパイラコレクション (GCC)	17
2.1. GNU C コンパイラ	17
2.2. GNU C++ コンパイラ	18
2.3. GNU FORTRAN コンパイラ	22
2.4. RED HAT DEVELOPER TOOLSET での GCC の詳細	23
2.5. 関連資料	24
第3章 GNU MAKE	26
3.1. MAKE のインストール	26
3.2. MAKE の使用	26
3.3. MAKEFILE の使用	27
3.4. 関連資料	28
第4章 BINUTILS	30
4.1. BINUTILS のインストール	31
4.2. GNU アセンブラの使用	31
4.3. GNU リンカーの使用	31
4.4. その他のバイナリツールの使用	32
4.5. RED HAT DEVELOPER TOOLSET における BINUTILS の詳細	33
4.6. 関連資料	33
第5章 ELFUTILS	35
5.1. ELFUTILS のインストール	36
5.2. ELFUTILS の使用	36
5.3. 関連資料	36
第6章 DWZ	38
6.1. DWZ のインストール	38
6.2. DWZ の使用	38
6.3. 関連資料	38
第7章 ANNOBIN	40
7.1. ANNOBIN のインストール	40
7.2. ANNOBIN プラグインの使用	40
7.3. ANNOCHECK の使用	40
7.4. 関連資料	41

パート III. デバッグツール	42
第8章 GNU デバッガー (GDB)	43
8.1. GNU デバッガーのインストール	43
8.2. デバッグプログラムの準備	43
8.3. GNU デバッガーの実行	44
8.4. ソースコードの一覧表示	45
8.5. ブレークポイントの設定	46
8.6. 実行の開始	48
8.7. 現在の値の表示	48
8.8. 継続実行	48
8.9. 関連資料	49
第9章 STRACE	51
9.1. STRACE のインストール	51
9.2. STRACE の使用	51
9.3. 関連資料	55
第10章 LTRACE	56
10.1. LTRACE のインストール	56
10.2. LTRACE の使用	56
10.3. 関連資料	59
第11章 MEMSTOMP	60
11.1. MEMSTOMP のインストール	61
11.2. MEMSTOMP の使用	61
11.3. 関連資料	63
パート IV. パフォーマンス監視ツール	64
第12章 SYSTEMTAP	65
12.1. SYSTEMTAP のインストール	65
12.2. SYSTEMTAP の使用	66
12.3. 関連資料	66
第13章 VALGRIND	68
13.1. VALGRIND のインストール	68
13.2. VALGRIND の使用	69
13.3. 関連資料	69
第14章 OPROFILE	71
14.1. OPROFILE のインストール	71
14.2. OPROFILE の使用	72
14.3. 関連資料	72
第15章 DYNINST	74
15.1. DYNINST のインストール	74
15.2. DYNINST の使用	74
15.3. 関連資料	79
パート V. コンパイラツールセット	81
第16章 コンパイラツールセットのドキュメント	82
パート VI. ヘルプの取得	83
第17章 RED HAT 製品ドキュメントへのアクセス	84

Red Hat Developer Toolset	84
Red Hat Enterprise Linux	84
第18章 グローバルサポートサービスへの連絡	85
18.1. 必要な情報の収集	85
18.2. 問題のエスカレーション	86
18.3. サービスリクエストの再オープン	86
18.4. 関連資料	87
付録A バージョン 12.0 での変更点	88
A.1. GCC の変更点	88
A.2. ELFUTILS の変更点	90
A.3. GDB の変更点	91
A.4. STRACE の変更点	93
A.5. SYSTEMTAP の変更点	94
A.6. DYNINST の変更点	95
A.7. ANNOBIN の変更点	95
付録B バージョン 12.1 での変更点	96
B.1. GCC の変更点	96
B.2. ANNOBIN の変更点	96
B.3. BINUTILS の変更点	96

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](#) をご覧ください。

パート I. はじめに

第1章 RED HAT DEVELOPER TOOLSET

1.1. RED HAT DEVELOPER TOOLSET

Red Hat Developer Toolsetは、Red Hat Enterprise Linux プラットフォームで開発者向けの Red Hat 製品です。これにより、複数のバージョンの Red Hat Enterprise Linux にインストールされ、使用できる、完全な開発およびパフォーマンス分析ツールが提供されます。また、Red Hat Developer Toolset ツールチェーンで構築された実行ファイルは、複数のバージョンの Red Hat Enterprise Linux にデプロイおよび実行できます。互換性の詳細情報は、「[互換性](#)」を参照してください。

Red Hat Developer Toolset は、これらのプラットフォームにインストールする場合に、Red Hat Enterprise Linux 7で提供されるデフォルトのシステムツールに代わるものではありません。開発者ツールの並列セットは、開発者がオプションで使用するための代替の、新しいバージョンのツールを提供します。デフォルトのコンパイラーとデバッガーは、ベースの Red Hat Enterprise Linux システムが提供するもののままです。

Red Hat Developer Toolset 12.1の新機能

Red Hat Developer Toolset 4.1以降、Red Hat Developer Toolset コンテンツは、他の Red Hat Software Collections コンテンツ (<https://access.redhat.com/downloads>) (特に [Server](#) および [Workstation](#)) とともに ISO 形式で利用できます。「[オプションパッケージのインストール](#)」で説明している **Optional** チャンネルを必要とするパッケージは、ISO イメージからインストールできないことに注意してください。

表1.1 Red Hat Developer Toolset のコンポーネント

名前	バージョン	説明
GCC	12.2.1	C、C++、および Fortran に対応するポータブルなコンパイラスイート。
binutils	2.36.1	オブジェクトファイルおよびバイナリーを検査および操作するためのバイナリーツールおよびその他のユーティリティーのコレクション。
elfutils	0.187	ELF ファイルを検証および操作するためのバイナリーツールおよびその他のユーティリティーのコレクション。
dwz	0.14	ELF 共有ライブラリーおよび ELF 実行ファイルに含まれる DWARF デバッグ情報 (サイズ) を最適化するツール。
GDB	11.2	C、C++、および Fortran で記述されたプログラムのコマンドラインデバッガー。
ltrace	0.7.91	プログラムが作成する動的ライブラリーへの呼び出しを表示するデバッグツール。また、プログラムが実行するシステムコールを監視することもできます。
strace	5.18	プログラムが使用するシステムコールを監視し、受信するシグナルを監視するデバッグツール。
memstomp	0.15	さまざまな標準で使用できないメモリー領域が重複するライブラリー関数への呼び出しを識別するデバッグツール。

名前	バージョン	説明
SystemTap	4.7	インストルメント化、再コンパイル、インストール、および再起動を行わずにシステム全体のアクティビティを監視するトレースおよびプロブのツール。
Valgrind	3.19.0	メモリーエラーを検出したり、メモリー管理問題を特定したり、システムコールで引数が間違っているのを報告するために、アプリケーションのプロファイルを行うインストルメンテーションフレームワークや多数のツールです。
OProfile	1.4.0	システム全体のプロファイラー。プロセッサ上のパフォーマンス監視ハードウェアを使用して、システム上のカーネルと実行ファイルに関する情報を取得します。
Dyninst	12.1.0	実行時にユーザー空間の実行ファイルをインストルメント化し、作業するためのライブラリー。
make	4.3	依存関係を追跡するビルド自動化ツール
annobin	11.08	ビルドセキュリティチェックツール。

Red Hat Developer Toolset は、これまで Red Hat Enterprise Linux で提供されていた [テクノロジープレビュー](#) コンパイラーリリースとは異なります。

1. Red Hat Developer Toolset は、「[互換性](#)」で説明されているように、Red Hat Enterprise Linux の複数のメジャーおよびマイナーリリースで使用できます。
2. 以前の Red Hat Enterprise Linux で提供されるテクノロジープレビューコンパイラーや、Red Hat Developer Toolset は Red Hat Enterprise Linux サブスクリプションレベルアグリーメントで完全にサポートされており、機能的に完全であり、実稼働環境での使用を目的としています。

重要なバグ修正およびセキュリティエラーは、各メジャーバージョンリリースから 2 年間 Red Hat Enterprise Linux と似た方法で Red Hat Developer Toolset サブスクリバラーに発行されます。Red Hat Developer Toolset の新規メジャーバージョンは、年次リリースされ、既存のコンポーネントに重要な更新を提供し、主要な新規コンポーネントを追加します。新しいメジャーリリースバージョンリリースの 6 か月後に発行される新しい単一のマイナーリリースでは、バグ修正、セキュリティエラー、および新しいマイナーリリースの小規模な更新が提供されます。

また、[Red Hat Enterprise Linux アプリケーションの互換性仕様](#) は、Red Hat Developer Toolset にも適用されます（「[C++ 互換性](#)」にも説明されている新しいバージョンの C++11 言語機能を使用する一部の制限に影響を受けます）。



重要

Red Hat Developer Toolset が提供するアプリケーションおよびライブラリーは、Red Hat Enterprise Linux のシステムバージョンを置き換えず、システムバージョンを優先して使用されていません。**Software Collections** というフレームワークを使用すると、追加の開発者ツールセットが `/opt/` ディレクトリーにインストールされ、ユーザーが `scl` ユーティリティーを使用してオンデマンドで明示的に有効にします。

1.2. 主な特長

Red Hat Developer Toolset 12.1 には、以下の変更があります。

- Red Hat Developer Toolset バージョンの **GNU コンパイラコレクション (GCC)** が、多くの新機能およびバグ修正により、バージョン 12.2.1 にアップグレードされました。
- Red Hat Developer Toolset バージョンの **GNU デバッガー (GDB)** は、多くの新機能およびバグ修正により、バージョン 11.2 にアップグレードされました。

本リリースで導入された変更および機能の完全リストは、[付録B バージョン 12.1 での変更点](#) を参照してください。

1.3. 互換性

Red Hat Developer Toolset 12.1 は、数多くのアーキテクチャーで Red Hat Enterprise Linux 7 で利用できます。

ABI の互換性情報は、「[C++ 互換性](#)」を参照してください。

表1.2 Red Hat Developer Toolset 12.1 の互換性

	Red Hat Enterprise Linux 7.7 での実行	Runs on Red Hat Enterprise Linux 7.9
Red Hat Enterprise Linux 7.7 でビルド	サポート対象	サポート対象
Built with Red Hat Enterprise Linux 7.9	サポート対象外	サポート対象

アーキテクチャーのサポート

Red Hat Developer Toolset は、以下のアーキテクチャーで利用できます。

- 64 ビット Intel および AMD アーキテクチャー
- IBM Power Systems、ビッグエンディアン
- IBM Power Systems (リトルエンディアン)
- 64 ビット IBM Z

1.4. RED HAT DEVELOPER TOOLSET へのアクセスの取得

Red Hat Developer Toolset は、Red Hat Software Collections の一部として配布されるオフリングです。

このコンテンツセットは、<https://access.redhat.com/solutions/472793> に記載されている Red Hat Enterprise Linux 7 サブスクリプションをご利用いただけます。

Red Hat Subscription Management を使用して Red Hat Developer Toolset を有効にします。このサブスクリプション管理サービスでシステムを登録する方法は、ガイドの [Red Hat Subscription Management](#) コレクションを参照してください。

1.4.1. Red Hat Software Collections の使用

Red Hat Developer Toolset を含む Red Hat Software Collections のリポジトリへのアクセスを提供するサブスクリプションをアタッチし、そのリポジトリを有効にします。

1. Red Hat Software Collections (つまり Red Hat Developer Toolset) を提供するサブスクリプションのプール ID を確認します。これを行うには、システムで利用できるサブスクリプションの一覧を表示します。

```
# subscription-manager list --available
```

このコマンドは、利用可能な各サブスクリプションの名前、固有の ID、有効期限、およびサブスクリプションに関連するその他の詳細情報を表示します。プール ID は、**Pool ID** で始まる行に一覧表示されます。

Red Hat Developer Toolset へのアクセスを提供するサブスクリプションの完全リストは、<https://access.redhat.com/solutions/472793> を参照してください。

2. 適切なサブスクリプションをシステムに割り当てます。

```
# subscription-manager attach --pool=pool_id
```

pool_id を、直前の手順で確認したプール ID に置き換えます。システムが現在割り当てているサブスクリプションの一覧を、いつでも確認するには、次のコマンドを実行します。

```
# subscription-manager list --consumed
```

3. Red Hat Software Collections リポジトリの正確な名前を確認します。リポジトリメタデータを取得し、利用可能な Yum リポジトリの一覧を表示します。

```
# subscription-manager repos --list
```

リポジトリ名は、使用している Red Hat Enterprise Linux のバージョンによって異なり、以下のフォーマットに基づいています。

```
rhel-variant-rhscl-version-rpms
rhel-variant-rhscl-version-debug-rpms
rhel-variant-rhscl-version-source-rpms
```

さらに、**devtoolset-12-gcc-plugin-devel** などの特定のパッケージは、**Optional** チャンネルでのみ利用可能なパッケージに依存します。これらのパッケージを含むリポジトリ名は、以下の形式を使用します。

```
rhel-version-variant-optional-rpms
rhel-version-variant-optional-debug-rpms
rhel-version-variant-optional-source-rpms
```

通常のリポジトリとオプションのリポジトリの両方で、**variant** を Red Hat Enterprise Linux システムのバリエーション (**server** または **workstation**) に置き換え、**version** を Red Hat Enterprise Linux システムバージョン (**7**) に置き換えます。

4. 手順番号 3 からリポジトリを有効にします。

```
# subscription-manager repos --enable repository
```

`repository` を、有効にするリポジトリの名前に置き換えます。

サブスクリプションがシステムに割り当てられているら、「[Red Hat Developer Toolset のインストール](#)」の説明に従って Red Hat Developer Toolset をインストールできます。Red Hat Subscription Management を使用してシステムを登録し、サブスクリプションに関連付ける方法は、[Red Hat Subscription Management](#) のガイドを参照してください。

1.5. RED HAT DEVELOPER TOOLSET のインストール

Red Hat Developer Toolset は、Red Hat Enterprise Linux に含まれる標準のパッケージ管理ツールを使用してインストール、更新、アンインストール、および検査できる RPM パッケージのコレクションとして配布されています。Red Hat Developer Toolset をシステムにインストールするには、Red Hat Software Collections コンテンツセットへのアクセスを提供する有効なサブスクリプションが必要です。システムを適切なサブスクリプションに関連付け、Red Hat Developer Toolset にアクセスする方法は、9「[Red Hat Developer Toolset へのアクセスの取得](#)」を参照してください。



重要

Red Hat Developer Toolset をインストールする前に、利用可能なすべての Red Hat Enterprise Linux 更新をインストールします。

1.5.1. 利用可能なすべてのコンポーネントのインストール

Red Hat Developer Toolset に含まれるすべてのコンポーネントをインストールするには、`devtoolset-12` パッケージをインストールします。

```
# yum install devtoolset-12
```

これにより、開発、デバッグ、パフォーマンス監視ツール、およびその他の依存パッケージがすべてシステムにインストールされます。「[個別パッケージグループのインストール](#)」に説明されているように、選択したパッケージグループのみをインストールすることを選択できます。



注記

Red Hat Developer Toolset 3.0 以降、`scl-utils` パッケージは Red Hat Developer Toolset の一部ではありません。これは、`scl` ユーティリティーが Red Hat Developer Toolset ソフトウェアコレクションとともにインストールされた以前のバージョンからの変更点です。

1.5.2. 個別パッケージグループのインストール

統合開発環境やソフトウェア開発ツールチェーンなど、特定のコンポーネントのみのインストールを容易にするため、Red Hat Developer Toolset には、[表1.3「Red Hat Developer Toolset メタパッケージ」](#)の説明に従って、選択したパッケージグループをインストールできる複数のメタパッケージが同梱されています。

表1.3 Red Hat Developer Toolset メタパッケージ

パッケージ名	説明	インストール済みコンポーネント
<code>devtoolset-12-perftools</code>	パフォーマンス監視ツール	SystemTap、Valgrind、OProfile、Dyninst

パッケージ名	説明	インストール済みコンポーネント
devtoolset-12-toolchain	開発およびデバッグのツール	gcc, make, GDB, binutils, elfutils, dwz, memstomp, strace, ltrace

これらのメタパッケージをインストールするには、以下を行います。

```
# yum install package_name
```

package_name を、インストールするメタパッケージの一覧に置き換えます。たとえば、開発およびデバッグのツールチェーンおよびそれに依存するパッケージのみをインストールするには、次のコマンドを実行します。

```
# yum install devtoolset-12-toolchain
```

「[利用可能なすべてのコンポーネントのインストール](#)」で説明されているように、利用可能なすべてのコンポーネントをインストールすることを選択できます。

1.5.3. オプションパッケージのインストール

Red Hat Developer Toolset には、デフォルトでインストールされていない複数のオプションパッケージが同梱されています。システムにインストールされていない Red Hat Developer Toolset パッケージの一覧を表示するには、次のコマンドを実行します。

```
$ yum list available devtoolset-12-*
```

これらのオプションパッケージのいずれかをインストールするには、次のコマンドを実行します。

```
# yum install package_name
```

package_name を、インストールするパッケージの一覧に置き換えます。パッケージ名はスペースで区切られます。たとえば、**devtoolset-12-gdb-gdbserver** および **devtoolset-12-gdb-doc** パッケージをインストールするには、次のコマンドを実行します。

```
# yum install devtoolset-12-gdb-gdbserver devtoolset-12-gdb-doc
```

1.5.4. デバッグ情報のインストール

Red Hat Developer Toolset パッケージのデバッグ情報をインストールするには、**yum-utils** パッケージがインストールされ、以下のコマンドを実行します。

```
# debuginfo-install package_name
```

たとえば、**devtoolset-12-dwz** パッケージのデバッグ情報をインストールするには、以下を実行します。

```
# debuginfo-install devtoolset-12-dwz
```

このコマンドを使用するには、このパッケージを使用してリポジトリにアクセスする必要があります。

す。システムが Red Hat Subscription Management に登録されている場合は、「[Red Hat Developer Toolset へのアクセスの取得](#)」の説明に従って `rhel-variant-rhsc-version-debug-rpms` リポジトリを有効にします。debuginfo パッケージへのアクセス方法は、<https://access.redhat.com/site/solutions/9907> を参照してください。



注記

`devtoolset-12-package_name-debuginfo` パッケージは、ベース Red Hat Enterprise Linux システムまたは他のバージョンの Red Hat Developer Toolset からの対応するパッケージと競合する可能性があります。この競合は、64 ビットの debuginfo パッケージが 32 ビットの debuginfo パッケージと競合する multilib 環境でも発生します。

Red Hat Developer Toolset 12.1 をインストールする前に、競合する debuginfo パッケージを手動でアンインストールし、必要に応じて関連する debuginfo パッケージのみをインストールします。

1.6. RED HAT DEVELOPER TOOLSET の更新

1.6.1. マイナーバージョンへの更新

Red Hat Developer Toolset の新規マイナーバージョンが利用できるようになったら、Red Hat Enterprise Linux インストールを更新します。

```
# yum update
```

これにより、Red Hat Developer Toolset バージョンの開発、デバッグ、パフォーマンス監視ツール、その他の依存パッケージなど、Red Hat Enterprise Linux システムのパッケージがすべて更新されます。



重要

Red Hat Developer Toolset を使用するには、リリース前のバージョンをすべて削除する必要があります。また、ベータリリースを含む Red Hat Developer Toolset のプレリリースバージョンから Red Hat Developer Toolset 12.1 に更新することはできません。以前に Red Hat Developer Toolset のプレリリースバージョンをインストールしている場合は、「[Red Hat Developer Toolset のアンインストール](#)」の説明に従ってシステムからアンインストールし、「[Red Hat Developer Toolset のインストール](#)」の説明に従って新規バージョンをインストールします。

1.6.2. メジャーバージョンへの更新

Red Hat Developer Toolset の新規メジャーバージョンが利用可能な場合は、以前のバージョンと並行してインストールできます。システムに Red Hat Developer Toolset をインストールする方法は、「[Red Hat Developer Toolset のインストール](#)」を参照してください。

1.7. RED HAT DEVELOPER TOOLSET のアンインストール

システムから Red Hat Developer Toolset パッケージをアンインストールするには、以下を実行します。

```
# yum remove devtoolset-12* libasan libatomic libcilkrts libitm liblsan libtsan libubsan
```

これにより、**GNU Compiler Collection**、**GNU Debugger**、**binutils**、およびシステムから Red Hat Developer Toolset に含まれるその他のパッケージが削除されます。



注記

Red Hat Developer Toolset 12.1 for Red Hat Enterprise Linux 7 には、上記のコマンドで削除しようとする **libatomic** および **libitm** ライブラリーが含まれなくなりました。これらは、そのシステムで Red Hat Developer Toolset コンポーネントが適切に機能するためには必要ないからです。ただし、上記のコマンドは Red Hat Enterprise Linux 7 でも期待どおりに機能します。

Red Hat Developer Toolset が提供するツールのアンインストールは、これらのツールの Red Hat Enterprise Linux システムバージョンには影響しないことに注意してください。

1.8. RED HAT DEVELOPER TOOLSET コンテナイメージの使用

Docker 形式のコンテナイメージ を使用して、仮想ソフトウェアコンテナ内で Red Hat Developer Toolset コンポーネントを実行することができます。そのため、それらをホストシステムから分離し、迅速なデプロイメントを可能にします。Red Hat Developer Toolset docker 形式のコンテナイメージおよび Red Hat Developer Toolset **Dockerfile** の詳細は、[Using Red Hat Software Collections Container Images](#) を参照してください。



注記

Docker デーモン、コマンドラインツール、および Docker 形式のコンテナイメージを構築および使用するのに必要なコンポーネントが含まれる **docker** パッケージは、現在 Red Hat Enterprise Linux 7 製品の Server バリエーションでのみ利用できます。

[RHEL 7 での Docker の取得](#) の手順に従って、Docker 形式のコンテナイメージを構築して使用する環境を設定します。

1.9. 関連資料

Red Hat Developer Toolset および Red Hat Enterprise Linux の詳細は、以下の資料を参照してください。

オンラインドキュメント

- [Red Hat Subscription Management](#) の一連のガイド: Red Hat Subscription Management のガイドは、Red Hat Enterprise Linux でサブスクリプションを管理する方法の詳細情報を提供します。
- [Red Hat Developer Toolset 12.1 Release Notes](#): Red Hat Developer Toolset 12.1 の [リリースノート](#) には、詳細情報が含まれています。
- [Red Hat Enterprise Linux 7 開発者ガイド](#): Red Hat Enterprise Linux 7 [開発者ガイド](#) は、Eclipse IDE、ライブラリーおよびランタイムサポート、コンパイルおよびビルド、デバッグ、およびプロファイリングに関する詳細情報を提供します。
- [Red Hat Enterprise Linux 7 インストールガイド](#): Red Hat Enterprise Linux 7 の [インストールガイド](#) では、システムの取得、インストール、および更新の方法を説明します。
- [Red Hat Enterprise Linux 7 システム管理者のガイド](#) - Red Hat Enterprise Linux 7 の [システム管理者のガイド](#) では、Red Hat Enterprise Linux 7 のデプロイメント、設定、および管理に関する情報を説明しています。

- [Red Hat Software Collections Container Images の使用](#) : 本ガイドでは、Red Hat Software Collections をベースとしたコンテナイメージを使用する方法を説明します。利用可能なコンテナイメージには、アプリケーション、デーモン、データベース、および Red Hat Developer Toolset コンテナイメージが含まれます。イメージは、Red Hat Enterprise Linux 7 Server および Red Hat Enterprise Linux Atomic Host で実行できます。
- [Getting Started with Containers](#): Red Hat Enterprise Linux 7 および Red Hat Enterprise Linux Atomic Host でのコンテナイメージのビルドおよび使用に関する包括的な概要を説明します。

以下も併せて参照してください。

- [付録B バージョン12.1 での変更点](#): Red Hat Developer Toolset の以前のバージョンにおける Red Hat Developer Toolset ツールのバージョンに対する変更および改善のリスト。

パート II. 開発ツール

第2章 GNU コンパイラーコレクション (GCC)

GNU コンパイラーコレクション (通常は GCC の省略形) は、幅広いプログラミング言語に対応する移植可能なコンパイラースイートです。

Red Hat Developer Toolset には GCC 12.2.1 が同梱されています。このバージョンは、Red Hat Enterprise Linux に含まれるバージョンよりも新しいもので、バグ修正および機能強化が数多く追加されました。

2.1. GNU C コンパイラー

2.1.1. C コンパイラーのインストール

Red Hat Developer Toolset では、GNU C コンパイラーが `devtoolset-12-gcc` パッケージで提供され、「[Red Hat Developer Toolset のインストール](#)」の説明に従って `devtoolset-12-toolchain` で自動的にインストールされます。

2.1.2. C コンパイラーの使用

コマンドラインで C++ プログラムをコンパイルするには、以下のように `gcc` コンパイラーを実行します。

```
$ scl enable devtoolset-12 'gcc -o output_file source_file...'
```

これにより、現在の作業ディレクトリーに `output_file` という名前のバイナリーファイルが作成されます。`-o` オプションを省略すると、コンパイラーはデフォルト `a.out` という名前のファイルを作成します。

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、各ソースファイルのオブジェクトファイルを最初にコンパイルしてから、これらのオブジェクトファイルをリンクすることが一般的です。これにより、単一のソースファイルを変更する場合は、プロジェクト全体をコンパイルせずにこのファイルのみを再コンパイルできます。コマンドラインでオブジェクトファイルをコンパイルするには、以下のコマンドを実行します。

```
$ scl enable devtoolset-12 'gcc -o object_file -c source_file'
```

これにより、`object_file` という名前のオブジェクトファイルが作成されます。`-o` オプションを省略すると、コンパイラーは、ファイル `.o` 拡張子が付いたソースファイルからという名前のファイルを作成します。オブジェクトファイルをリンクし、バイナリーファイルを作成します。

```
$ scl enable devtoolset-12 'gcc -o output_file object_file...'
```

この `scl` ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、デフォルトで Red Hat Developer Toolset `gcc` でシェルセッションを実行できます。

```
$ scl enable devtoolset-12 'bash'
```



注記

使用中の **gcc** のバージョンを確認するには、以下を行います。

```
$ which gcc
```

Red Hat Developer Toolset の **gcc** 実行可能なパスは、**/opt** で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset **gcc** と一致することを確認することができます。

```
$ gcc -v
```

例2.1 コマンドラインでの C プログラムのコンパイル

以下の内容を含むソースファイル **hello.c** について考えてみましょう。

```
#include <stdio.h>

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

Red Hat Developer Toolset の **gcc** コンパイラーを使用して、このソースコードをコマンドラインでコンパイルします。

```
$ scl enable devtoolset-12 'gcc -o hello hello.c'
```

これにより、現在の作業ディレクトリーに **hello** という名前のバイナリーファイルが作成されます。

2.1.3. C プログラムの実行

プログラムを **gcc** コンパイルすると、実行可能なバイナリーファイルが作成されます。コマンドラインでこのプログラムを実行するには、実行ファイルがあるディレクトリーに移動し、これを実行します。

```
$ ./file_name
```

例2.2 コマンドラインでの C プログラムの実行

例2.1「コマンドラインでの C プログラムのコンパイル」にあるように **hello** バイナリーファイルを正常にコンパイルしたと仮定して、シェルプロンプトで次のコマンドを実行します。

```
$ ./hello
Hello, World!
```

2.2. GNU C++ コンパイラー

2.2.1. C++ コンパイラーのインストール

Red Hat Developer Toolset では、GNU C++ コンパイラーは `devtoolset-12-gcc-c++` パッケージで提供され、「[Red Hat Developer Toolset のインストール](#)」の説明に従って `devtoolset-12-toolchain` パッケージで自動的にインストールされます。

2.2.2. C++ コンパイラーの使用

コマンドラインで C++ プログラムをコンパイルするには、以下のように `g++` コンパイラーを実行します。

```
$ scl enable devtoolset-12 'g++ -o output_file source_file...'
```

これにより、現在の作業ディレクトリーに `output_file` という名前のバイナリーファイルが作成されます。`-o` オプションを省略すると、`g++` コンパイラーはデフォルト `a.out` でという名前のファイルを作成します。

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、各ソースファイルのオブジェクトファイルを最初にコンパイルしてから、これらのオブジェクトファイルをリンクすることが一般的です。これにより、単一のソースファイルを変更する場合は、プロジェクト全体をコンパイルせずにこのファイルのみを再コンパイルできます。コマンドラインでオブジェクトファイルをコンパイルするには、以下のコマンドを実行します。

```
$ scl enable devtoolset-12 'g++ -o object_file -c source_file'
```

これにより、`object_file` という名前のオブジェクトファイルが作成されます。`-o` オプションを省略すると、`g++` コンパイラーは、ファイル `.o` 拡張子が付いたソースファイルからという名前のファイルを作成します。オブジェクトファイルをリンクし、バイナリーファイルを作成します。

```
$ scl enable devtoolset-12 'g++ -o output_file object_file...'
```

この `scl` ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、デフォルトで Red Hat Developer Toolset `g++` でシェルセッションを実行できます。

```
$ scl enable devtoolset-12 'bash'
```

注記

使用中の `g++` のバージョンを確認するには、以下を行います。

```
$ which g++
```

Red Hat Developer Toolset の `g++` 実行可能なパスは、`/opt` で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset `g++` と一致を確認することができます。

```
$ g++ -v
```

例2.3 コマンドラインでの C++ プログラムのコンパイル

以下の内容を含むソースファイル **hello.cpp** について考えてみましょう。

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[]) {
    cout << "Hello, World!" << endl;
    return 0;
}
```

Red Hat Developer Toolset の **g++** コンパイラーを使用して、このソースコードをコマンドラインでコンパイルします。

```
$ scl enable devtoolset-12 'g++ -o hello hello.cpp'
```

これにより、現在の作業ディレクトリーに **hello** という名前のバイナリーファイルが作成されます。

2.2.3. C++ プログラムの実行

プログラムを **g++** コンパイルすると、実行可能なバイナリーファイルが作成されます。コマンドラインでこのプログラムを実行するには、実行ファイルがあるディレクトリーに移動し、これを実行します。

```
$ ./file_name
```

例2.4 コマンドラインでの C++ プログラムの実行

例2.3「コマンドラインでの C++ プログラムのコンパイル」にあるように、**hello** バイナリーファイルを正常にコンパイルしたと仮定して実行できます。

```
$ ./hello
Hello, World!
```

2.2.4. C++ 互換性

すべての **-std** モードの Red Hat Enterprise Linux バージョン 5、6、7、および Red Hat Developer Toolset バージョン 1 から 10 のすべてのコンパイラーは、C++98 モードの他のコンパイラーと互換性があります。

C++11、C++14、または C++17 モードのコンパイラーは、同じリリースシリーズであれば、同じモードでの別のコンパイラーとの互換性が保証されています。

サポートされる例:

- Red Hat Developer Toolset 6.x からの C++11 および C++11
- Red Hat Developer Toolset 6.x からの C++14 および C++14
- Red Hat Developer Toolset 10.x からの C++17 および C++17



重要

- Red Hat Developer Toolset 10.x の GCC コンパイラーは、C++20 を使用してコードを構築することは可能ですが、この機能は Red Hat では実験的で、サポートされていません。
- 本セクションで説明されている互換性情報はすべて、GCC C++ コンパイラーの Red Hat が提供するバージョンにのみ関連します。

2.2.4.1. C++ ABI

-std=c++98 または **-std=gnu++98** で Red Hat Developer Toolset ツールチェーンが明示的にビルドした C++98 準拠のバイナリーまたはライブラリーは、Red Hat Enterprise Linux 5、6、または 7 システム GCC が構築したバイナリーおよび共有ライブラリーと自由に混在できます。

Red Hat Developer Toolset 12.1 のデフォルトの言語標準設定は、オプション **-std=gnu++17** を明示的に使用する場合と同等の GNU 拡張機能を備えた C++17 です。

Red Hat Developer Toolset では、それぞれのフラグでコンパイルされたすべての C++ オブジェクトが Red Hat Developer Toolset 6 以降を使用してビルドされている場合、C++14 言語バージョンの使用がサポートされます。デフォルトモードの C++98 でシステム GCC によってコンパイルされたオブジェクトには互換性がありますが、C++11 モードまたは C++14 モードのシステム GCC でコンパイルされたオブジェクトには互換性がありません。

Red Hat Developer Toolset 10.x 以降では、C++17 言語バージョンの使用は実験的なものになり、Red Hat によってサポートされます。C++17 でコンパイルされたすべての C++ オブジェクトは、Red Hat Developer Toolset 10.x 以降を使用してビルドする必要があります。



重要

ご使用のアプリケーションで C++11、C++14、および C++17 機能を使用するには、上記の ABI 互換性情報を慎重に検討する必要があります。

-std=c++0x または **-std=gnu++0x** フラグを使用して Red Hat Enterprise Linux 7 システムツールチェーン GCC によってビルドされたオブジェクト、バイナリー、およびライブラリーと、Red Hat Developer Toolset の GCC を使用して C++11 以降の言語バージョンでビルドされたものを混合することは、明示的にサポートされていません。

上記で説明した C++11、C++14、および C++17 ABI のほかに、Red Hat Developer Toolset では [Red Hat Enterprise Linux アプリケーションの互換性仕様](#) は変更されません。Red Hat Developer Toolset で構築されたオブジェクトを、Red Hat Enterprise Linux 7 ツールチェーン (特に **.o/.a** ファイル) で構築したオブジェクトと混在する場合、Red Hat Developer Toolset ツールチェーンはどのリンクにも使用してください。これにより、Red Hat Developer Toolset が提供する新しいライブラリー機能は、リンク時に解決されます。

ベクトルの長さを持つシステムで名前が競合しないように、SIMD ベクトルタイプの新しい標準マングリングが追加されました。Red Hat Developer Toolset のコンパイラーは、デフォルトで新しいマングリングを使用します。GCC C++ コンパイラーコールに **-fabi-version=2** または **-fabi-version=3** オプションを追加して、以前の標準マングリングを使用できます。以前の mangling を使用するコードの警告を表示するには、**-Wabi** オプションを使用します。

Red Hat Enterprise Linux 7 では、GCC C++ コンパイラーはデフォルトで古い mangling を使用しますが、強固なエイリアスに対応するターゲットに新しいマングリングを持つエイリアスをエミュレートします。コンパイラーコールに **-fabi-version=4** オプションを追加して、新しい標準マングリングを使用できます。以前の mangling を使用するコードの警告を表示するには、**-Wabi** オプションを使用します。

2.3. GNU FORTRAN コンパイラー

2.3.1. Fortran コンパイラーのインストール

Red Hat Developer Toolset では、GNU Fortran コンパイラーは `devtoolset-12-gcc-gfortran` パッケージで提供され、「[Red Hat Developer Toolset のインストール](#)」の説明に従って `devtoolset-12-toolchain` で自動的にインストールされます。

2.3.2. Fortran コンパイラーの使用

コマンドラインで Fortran プログラムをコンパイルするには、以下のように `gfortran` コンパイラーを実行します。

```
$ scl enable devtoolset-12 'gfortran -o output_file source_file...'
```

これにより、現在の作業ディレクトリーに `output_file` という名前のバイナリーファイルが作成されます。`-o` オプションを省略すると、コンパイラーはデフォルト `a.out` という名前のファイルを作成します。

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、各ソースファイルのオブジェクトファイルを最初にコンパイルしてから、これらのオブジェクトファイルをリンクすることが一般的です。これにより、単一のソースファイルを変更する場合は、プロジェクト全体をコンパイルせずにこのファイルのみを再コンパイルできます。コマンドラインでオブジェクトファイルをコンパイルするには、以下のコマンドを実行します。

```
$ scl enable devtoolset-12 'gfortran -o object_file -c source_file'
```

これにより、`object_file` という名前のオブジェクトファイルが作成されます。`-o` オプションを省略すると、コンパイラーは、ファイル `.o` 拡張子が付いたソースファイルからという名前のファイルを作成します。オブジェクトファイルをリンクし、バイナリーファイルを作成します。

```
$ scl enable devtoolset-12 'gfortran -o output_file object_file...'
```

この `scl` ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、デフォルトで Red Hat Developer Toolset `gfortran` でシェルセッションを実行できます。

```
$ scl enable devtoolset-12 'bash'
```

注記

使用中の `gfortran` のバージョンを確認するには、以下を行います。

```
$ which gfortran
```

Red Hat Developer Toolset の `gfortran` 実行可能なパスは、`/opt` で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset `gfortran` と一致することを確認することができます。

```
$ gfortran -v
```

例2.5 コマンドラインで Fortran プログラムのコンパイル

以下の内容を含むソースファイル **hello.f** について考えてみましょう。

```
program hello
  print *, "Hello, World!"
end program hello
```

Red Hat Developer Toolset の **gfortran** コンパイラーを使用して、このソースコードをコマンドラインでコンパイルします。

```
$ scl enable devtoolset-12 'gfortran -o hello hello.f'
```

これにより、現在の作業ディレクトリーに **hello** という名前のバイナリーファイルが作成されます。

2.3.3. Fortran プログラムの実行

プログラムを **gfortran** コンパイルすると、実行可能なバイナリーファイルが作成されます。コマンドラインでこのプログラムを実行するには、実行ファイルがあるディレクトリーに移動し、これを実行します。

```
$/file_name
```

例2.6 コマンドラインでの Fortran プログラムの実行

例2.5「コマンドラインで Fortran プログラムのコンパイル」にあるように、**hello** バイナリーファイルを正常にコンパイルしたと仮定して実行できます。

```
$/hello
Hello, World!
```

2.4. RED HAT DEVELOPER TOOLSET での GCC の詳細

ライブラリーの静的リンク

最新のライブラリー機能の一部は、複数のバージョンの Red Hat Enterprise Linux での実行に対応するために、Red Hat Developer 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。標準の Red Hat Enterprise Linux エラータではこのコードが変更されないため、これにより、重要性が高くないセキュリティーリスクが発生します。Red Hat は、このリスクにより、開発者がアプリケーションを再構築する必要がある場合でも、セキュリティーエラータを使用してこのアプリケーションと通信します。



重要

このようなセキュリティーリスクが発生するため、開発者は同じ理由によりアプリケーション全体を静的にリンクしないことが強く推奨されます。

連結時に、オブジェクトファイルの後にライブラリーを指定

Red Hat Developer Toolset では、ライブラリーは、静的アーカイブで一部のシンボルを指定できるリンカースクリプトを使用してリンクされます。これは、Red Hat Enterprise Linux の複数のバージョンとの互換性を確保するために必要になります。ただし、リンカーのスクリプトは、対応する共有オブジェクトファイルの名前を使用します。したがって、リンカーは、オブジェクトファイルを指定するオプションの前に、ライブラリーを追加するオプションを指定する際に、想定とは異なるシンボル処理ルールを使用して、オブジェクトファイルが必要とするシンボルを認識しません。

```
$ scl enable devtoolset-12 'gcc -lsomelib objfile.o'
```

このように、GCC Toolset のライブラリーを使用すると、リンカーのエラーメッセージで、**undefined reference to symbol** になります。この問題を回避するには、標準のリンクプラクティスに従い、オブジェクトファイルを指定するオプションの後に、ライブラリーを追加するオプションを指定します。

```
$ scl enable devtoolset-12 'gcc objfile.o -lsomelib'
```

この推奨事項は、Red Hat Enterprise Linux のベースバージョンの **GCC** を使用する場合にも適用されることに注意してください。

2.5. 関連資料

GNU Compiler Collections およびその機能の詳細は、以下の資料を参照してください。

インストールされているドキュメント

- **gcc(1): gcc** コンパイラーの man ページでは、その使用方法に関する詳細情報が提供されます。一部の例外を除き、**g++** は **gcc** と同じコマンドラインオプションを受け付けます。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man gcc'
```

- **gfortran(1): gfortran** コンパイラーの man ページは、その使用方法の詳細情報を提供します。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man gfortran'
```

- **C++ Standard Library Documentation: C++ 標準ライブラリーのドキュメント**は、オプションでインストールできます。

```
# yum install devtoolset-12-libstdc++-docs
```

インストールが完了すると、HTML ドキュメントは **/opt/rh/devtoolset-12/root/usr/share/doc/devtoolset-12-libstdC++-docs-12.2.1/html/index.html** で利用可能になります。

オンラインドキュメント

- [Red Hat Enterprise Linux 7 Developer Guide](#) : Red Hat Enterprise Linux 7 の **Developer Guide** では、**GCC** に関する詳細な情報を提供します。
- [GNU コンパイラーコレクションの使用](#): アップストリームの GCC のマニュアルでは、GNU コンパイラーとその使用方法を詳細に説明します。

- [GNU C++ ライブラリー](#): GNU C++ ライブラリーのドキュメントでは、標準の C++ ライブラリーの GNU 実装に関する詳細情報が提供されています。
- [GNU Fortran コンパイラ](#): GNU Fortran コンパイラのドキュメントでは、**gfortran** の使用方法に関する詳細情報が提供されています。

以下も併せて参照してください。

- [1章 Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。
- [4章 binutils](#): **binutils** を使用する手順。オブジェクトファイルおよびバイナリーを検査および操作するためのバイナリーツールのコレクションです。
- [5章 elfutils](#): **elfutils** を使用する手順。ELF ファイルを検査および操作するためのバイナリーツールのコレクションです。
- [6章 dwz](#): **dwz** ツールを使用して、ELF 共有ライブラリーや、サイズの ELF 実行ファイルに含まれる DWARF デバッグ情報を最適化する説明。
- [8章 GNU デバッガー \(GDB\)](#): C、C++、および Fortran で書かれたデバッグプログラムの手順。

第3章 GNU MAKE

GNU make ユーティリティ (通常は **make**) は、ソースファイルからの実行可能ファイルの生成を制御するツールです。**make** は自動的に複雑なプログラムのどの部分に変更されたかを判断し、再コンパイルする必要があります。**make** は **Makefile** と呼ばれる設定ファイルを使用して、プログラムを構築する方法を制御します。

Red Hat Developer Toolset には **make 4.3** が同梱されています。このバージョンは、Red Hat Enterprise Linux に含まれるバージョンよりも新しいもので、バグ修正および機能強化が数多く追加されました。

3.1. MAKE のインストール

Red Hat Developer Toolset では、**devtoolset-12-make** パッケージにより **GNU make** が提供され、「[Red Hat Developer Toolset のインストール](#)」の説明通りに **devtoolset-12-toolchain** で自動的にインストールされます。

3.2. MAKE の使用

Makefile を使用せずにプログラムを構築するには、以下のように **make** ツールを実行します。

```
$ scl enable devtoolset-12 'make source_file_without_extension'
```

このコマンドは、C、C++、Fortran を含む多数のプログラミング言語に定義される暗黙的なルールを利用します。結果は、現在の作業ディレクトリーの **source_file_without_extension** という名前のバイナリーファイルです。

この **scl** ユーティリティを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、デフォルトで Red Hat Developer Toolset **make** でシェルセッションを実行できます。

```
$ scl enable devtoolset-12 'bash'
```

注記

使用中の **make** のバージョンを確認するには、以下を行います。

```
$ which make
```

Red Hat Developer Toolset の **make** 実行可能なパスは、**/opt** で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset **make** と一致を確認することができます。

```
$ make -v
```

例3.1 make を使用した C プログラムの構築

以下の内容を含むソースファイル **hello.c** について考えてみましょう。

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

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

Red Hat Developer Toolset の **make** ユーティリティーで定義されている暗黙的なルールを使用して、このソースコードを構築します。

```
$ scl enable devtoolset-12 'make hello'
cc hello.c -o hello
```

これにより、現在の作業ディレクトリーに **hello** という名前のバイナリーファイルが作成されます。

3.3. MAKEFILE の使用

複数のソースファイルで設定される複雑なプログラムを構築するには、**make** は **Makefile** と呼ばれる設定ファイルを使用して、プログラムのコンポーネントをコンパイルし、最終的な実行可能ファイルを構築する方法を制御します。Makefile には、作業ディレクトリーのクリーニング、プログラムファイルのインストールおよびアンインストール、およびその他の操作の指示を含めることもできます。

make は、現在のディレクトリーの **GNUmakefile**、**makefile**、または **Makefile** というファイルを自動的に使用します。別のファイル名を指定するには、**-f** オプションを使用します。

```
$ make -f make_file
```

Makefile 構文の詳細の説明は、本ガイドの対象外です。[GNU make](#)、アップストリームの **GNU make** マニュアルを参照してください。これは、**GNU make** ユーティリティー、Makefile 構文、その使用方法を詳細に説明しています。

完全な **make** マニュアルは、インストールの一部として Texinfo 形式でも利用できます。このマニュアルを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'info make'
```

例3.2 例: Makefile を使用した C プログラムの構築

例3.1「[make を使用した C プログラムの構築](#)」で紹介されている簡単な C プログラムを構築するために **Makefile** という名前が付けられた以下の汎用 Makefile について考えてみましょう。Makefile は変数を定義し、**ターゲット**と**レシピ**で設定される 4 つの **ルール** を指定します。レシピの行は、TAB 文字で開始する必要があることに注意してください。

```
CC=gcc
CFLAGS=-c -Wall
SOURCE=hello.c
OBJ=$(SOURCE:.c=.o)
EXE=hello

all: $(SOURCE) $(EXE)

$(EXE): $(OBJ)
    $(CC) $(OBJ) -o $@
```

```
.O: .c
    $(CC) $(CFLAGS) $< -o $@

clean:
    rm -rf $(OBJ) $(EXE)
```

この Makefile を使用して **hello.c** プログラムを構築するには、**make** ユーティリティーを実行します。

```
$ scl enable devtoolset-12 'make'
gcc -c -Wall hello.c -o hello.o
gcc hello.o -o hello
```

これにより、現在の作業ディレクトリーに、新しいオブジェクトファイル **hello.c** と新しいバイナリーファイル **hello** が作成されます。

作業ディレクトリーを消去するには、以下のコマンドを実行します。

```
$ scl enable devtoolset-12 'make clean'
rm -rf hello.o hello
```

これにより、作業ディレクトリーからオブジェクトおよびバイナリーファイルが削除されます。

3.4. 関連資料

GNU make ツールとその機能の詳細は、以下の資料を参照してください。

インストールされているドキュメント

- **make(1): make** ユーティリティーの man ページでは、その使用方法に関する情報が提供されません。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man make'
```

- Makefile 構文の詳細情報を含む完全な **make** マニュアルも Texinfo 形式から入手できます。Red Hat Developer Toolset に含まれるバージョンの情報マニュアルを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'info make'
```

オンラインドキュメント

- **GNU make**: アップストリームの GNU make マニュアルでは、GNU make ユーティリティー、Makefile 構文、およびその使用方法が詳細に説明されています。

以下も併せて参照してください。

- [1章 Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。

- [2章GNU コンパイラーコレクション\(GCC\)](#): 幅広いプログラミング言語に対応する移植可能なコンパイラスイートである **GNU コンパイラーコレクション** を使用する手順。
- [4章binutils](#): **binutils** を使用する手順。オブジェクトファイルおよびバイナリーを検査および操作するためのバイナリーツールのコレクションです。
- [5章elfutils](#): **elfutils** を使用する手順。ELF ファイルを検査および操作するためのバイナリーツールのコレクションです。
- [6章dwz](#): **dwz** ツールを使用して、ELF 共有ライブラリーや、サイズの ELF 実行ファイルに含まれる DWARF デバッグ情報を最適化する説明。
- [8章GNU デバッガー\(GDB\)](#): C、C++、および Fortran で書かれたデバッグプログラムの手順。

第4章 BINUTILS

binutils は、**GNU リンカー**、**GNU アセンブラ**、およびオブジェクトファイルおよびバイナリーを検査および操作できるその他のユーティリティなど、さまざまなバイナリーツールの集合です。Red Hat Developer Toolset バージョンの **binutils** で配布されるバイナリーツールの完全リストは、[表 4.1 「Red Hat Developer Toolset の binutils に含まれるツール」](#) を参照してください。

Red Hat Developer Toolset には **binutils 2.36.1** が同梱されています。このバージョンは、Red Hat Enterprise Linux および Red Hat Developer Toolset の以前のリリースに含まれるバージョンよりも新しいもので、バグ修正および機能強化を提供します。

表4.1 Red Hat Developer Toolset の binutils に含まれるツール

名前	説明
addr2line	アドレスをファイル名および行番号に変換します。
ar	アーカイブからファイルを作成、変更、および抽出します。
as	GNU アセンブラー。
c++filt	mangled C++ シンボルをデコードします。
dwp	DWARF オブジェクトファイルを1つの DWARF パッケージファイルに統合します。
elfedit	ELF ファイルを検査および編集します。
gprof	プロファイリング情報を表示します。
ld	GNU リンカー。
ld.bfd	GNU リンカーの代替。
ld.gold	GNU リンカーの代替手段
nm	オブジェクトファイルのシンボルを一覧表示します。
objcopy	オブジェクトファイルをコピーして変換します。
objdump	オブジェクトファイルの情報を表示します。
ranlib	このアーカイブにアクセスするために、アーカイブのコンテンツにインデックスを生成します。
readelf	ELF ファイルに関する情報を表示します。

名前	説明
size	オブジェクトまたはアーカイブファイルのセクションサイズを一覧表示します。
strings	ファイルの印刷可能な文字シーケンスを表示します。
strip	オブジェクトファイルからのすべてのシンボルを破棄します。

4.1. BINUTILS のインストール

Red Hat Developer Toolset では、**devtoolset-12-binutils** パッケージで **binutils** が提供され、「[Red Hat Developer Toolset のインストール](#)」で説明されているように、**devtoolset-12-toolchain** で自動的にインストールされます。

4.2. GNU アセンブラーの使用

アセンブリ言語プログラムからオブジェクトファイルを生成するには、以下のように **as** ツールを実行します。

```
$ scl enable devtoolset-12 'as option ... -o object_file source_file'
```

これにより、現在の作業ディレクトリーに **object_file** という名前のオブジェクトファイルが作成されます。

この **scl** ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、デフォルトで Red Hat Developer Toolset **as** でシェルセッションを実行できます。

```
$ scl enable devtoolset-12 'bash'
```



注記

使用中の **as** のバージョンを確認するには、以下を行います。

```
$ which as
```

Red Hat Developer Toolset の **as** 実行可能なパスは、**/opt** で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset **as** と一致を確認することができます。

```
$ as -v
```

4.3. GNU リンカーの使用

オブジェクトファイルから実行可能なバイナリーファイルまたはライブラリーを作成するには、以下のように ld ツールを実行します。

```
$ scl enable devtoolset-12 'ld option ... -o output_file object_file ...'
```

これにより、現在の作業ディレクトリーに **output_file** という名前のバイナリーファイルが作成されます。**-o** オプションを省略すると、コンパイラーはデフォルト **a.out** でという名前のファイルを作成します。

この **scl** ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、デフォルトで Red Hat Developer Toolset **ld** でシェルセッションを実行できます。

```
$ scl enable devtoolset-12 'bash'
```

注記

使用中の **ld** のバージョンを確認するには、以下を行います。

```
$ which ld
```

Red Hat Developer Toolset の **ld** 実行可能なパスは、**/opt** で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset **ld** と一致を確認することができます。

```
$ ld -v
```

4.4. その他のバイナリーツールの使用

binutils は、リンカーやアセンブラー以外の多くのバイナリーツールを提供します。これらのツールの一覧は、[表4.1「Red Hat Developer Toolset の binutils に含まれるツール」](#) を参照してください。

binutils の一部であるツールを実行するには、次のコマンドを実行します。

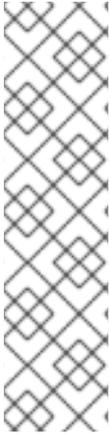
```
$ scl enable devtoolset-12 'tool option ... file_name'
```

binutils で配布されるツールの一覧は、[表4.1「Red Hat Developer Toolset の binutils に含まれるツール」](#) を参照してください。たとえば、**objdump** ツールを使用してオブジェクトファイルを検査するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'objdump option ... object_file'
```

この **scl** ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、Red Hat Developer Toolset バイナリーツールでシェルセッションをデフォルトとして実行できます。

```
$ scl enable devtoolset-12 'bash'
```



注記

`binutils` のバージョンを確認するには、以下のコマンドを使用します。

```
$ which objdump
```

Red Hat Developer Toolset の `objdump` 実行可能なパスは、`/opt` で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset `objdump` と一致することを確認することができます。

```
$ objdump -v
```

4.5. RED HAT DEVELOPER TOOLSET における BINUTILS の詳細

ライブラリーの静的リンク

最新のライブラリー機能の一部は、複数のバージョンの Red Hat Enterprise Linux での実行に対応するために、Red Hat Developer 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。標準の Red Hat Enterprise Linux エラータではこのコードが変更されないため、これにより、重要性が高くないセキュリティーリスクが発生します。Red Hat は、このリスクにより、開発者がアプリケーションを再構築する必要がある場合でも、セキュリティーエラータを使用してこのアプリケーションと通信します。



重要

このようなセキュリティーリスクが発生するため、開発者は同じ理由によりアプリケーション全体を静的にリンクしないことが強く推奨されます。

連結時に、オブジェクトファイルの後にライブラリーを指定

Red Hat Developer Toolset では、ライブラリーは、静的アーカイブで一部のシンボルを指定できるリンカースクリプトを使用してリンクされます。これは、Red Hat Enterprise Linux の複数のバージョンとの互換性を確保するために必要になります。ただし、リンカーのスクリプトは、対応する共有オブジェクトファイルの名前を使用します。したがって、リンカーは、オブジェクトファイルを指定するオプションの前に、ライブラリーを追加するオプションを指定する際に、想定とは異なるシンボル処理ルールを使用して、オブジェクトファイルが必要とするシンボルを認識しません。

```
$ scl enable devtoolset-12 'ld -lsoamelib objfile.o'
```

このように、GCC Toolset のライブラリーを使用すると、リンカーのエラーメッセージで、**undefined reference to symbol** になります。この問題を回避するには、標準のリンクプラクティスに従い、オブジェクトファイルを指定するオプションの後に、ライブラリーを追加するオプションを指定します。

```
$ scl enable devtoolset-12 'ld objfile.o -lsoamelib'
```

また、この推奨事項は、Red Hat Enterprise Linux のベースバージョンの `binutils` を使用している場合にも適用されることに注意してください。

4.6. 関連資料

`binutils` の詳細は、以下に記載のドキュメントを参照してください。

インストールされているドキュメント

- `as(1)`、`ld(1)`、`addr2line(1)`、`ar(1)`、`c++filt(1)`、`dwp(1)`、`elfedit(1)`、`gprof(1)`、`nm(1)`、`objcopy(1)`、`runlib(1)`、`readelf(1)`、`size(1)`、`string(1)`、`strip(1)`: さまざまな `binutils` ツールの `man` ページは、それぞれの使用方法に関する詳細情報を提供します。Red Hat Developer Toolset に含まれるバージョンの `man` ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man tool'
```

オンラインドキュメント

- [binutils のドキュメント](#): `binutils` ドキュメントでは、バイナリーツールとその使用方法に関する詳細な説明が記載されています。

以下も併せて参照してください。

- [1章 Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。
- [5章 elfutils](#): ELF ファイルを検査および操作するためのバイナリーツールのコレクションである `elfutils` の使用方法。
- [2章 GNU コンパイラーコレクション \(GCC\)](#): C、C++、および Fortran で書かれたプログラムをコンパイルする方法。

第5章 ELFUTILS

elfutils は、**eu-objdump**、**eu-readelf**、および ELF ファイルの検査と操作を行うことができる他のユーティリティーなどの一連のバイナリーツールです。Red Hat Developer Toolset バージョンの elfutils で配布されるバイナリーツールの完全リストは、[表5.1「Red Hat Developer Toolset の elfutils に含まれるツール」](#) を参照してください。

Red Hat Developer Toolset には **elfutils 0.187** が同梱されています。このバージョンは、Red Hat Developer Toolset の以前のリリースに含まれていたバージョンよりも新しいもので、バグ修正および機能強化が提供されています。

表5.1 Red Hat Developer Toolset の elfutils に含まれるツール

名前	説明
eu-addr2line	アドレスをファイル名および行番号に変換します。
eu-ar	アーカイブからファイルを作成、変更、および抽出します。
eu-elfcmp	2つの ELF ファイルの関連する部分を等価性と比較します。
eu-elflint	ELF ファイルは汎用 ABI (gABI) およびプロセッサ固有の補完 ABI (psABI) 仕様と互換性があります。
eu-findtextrel	ファイルでテキストの再配置のソースを見つけます。
eu-make-debug-archive	デバッグ用のオフラインアーカイブを作成します。
eu-nm	オブジェクトファイルのシンボルを一覧表示します。
eu-objdump	オブジェクトファイルの情報を表示します。
eu-ranlib	このアーカイブにアクセスするために、アーカイブのコンテンツにインデックスを生成します。
eu-readelf	ELF ファイルに関する情報を表示します。
eu-size	オブジェクトまたはアーカイブファイルのセクションサイズを一覧表示します。
eu-stack	プロセスとコアをアンバウンドする新しいユーティリティー。
eu-strings	ファイルの印刷可能な文字シーケンスを表示します。

名前	説明
eu-strip	オブジェクトファイルからのすべてのシンボルを破棄します。
eu-unstrip	ストライプ化されたファイルと個別のシンボルおよびデバッグ情報を統合します。

5.1. ELFUTILS のインストール

Red Hat Developer Toolset では、`devtoolset-12-elfutils` パッケージにより `elfutils` が提供され、「[Red Hat Developer Toolset のインストール](#)」で説明されているように、`devtoolset-12-toolchain` で自動的にインストールされます。

5.2. ELFUTILS の使用

`elfutils` の一部であるツールのいずれかを実行するには、以下のようにツールを実行します。

```
$ scl enable devtoolset-12 'tool option ... file_name'
```

`elfutils` で配布されるツールの一覧は、[表5.1「Red Hat Developer Toolset の elfutils に含まれるツール」](#)を参照してください。たとえば、`eu-objdump` ツールを使用してオブジェクトファイルを検査するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'eu-objdump option ... object_file'
```

この `scl` ユーティリティを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、Red Hat Developer Toolset バイナリーツールでシェルセッションをデフォルトとして実行できます。

```
$ scl enable devtoolset-12 'bash'
```

注記

`elfutils` のバージョンを確認するには、以下のコマンドを使用します。

```
$ which eu-objdump
```

Red Hat Developer Toolset の `eu-objdump` 実行可能なパスは、`/opt` で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset `eu-objdump` と一致することを確認することができます。

```
$ eu-objdump -V
```

5.3. 関連資料

`elfutils` の詳細は、以下に記載のドキュメントを参照してください。

以下も併せて参照してください。

- [1章Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。
- [2章GNU コンパイラコレクション\(GCC\)](#): C、C++、および Fortran で書かれたプログラムのコンパイル手順。
- [4章binutils](#): **binutils** を使用する手順。オブジェクトファイルおよびバイナリーを検査および操作するためのバイナリーツールのコレクションです。
- [6章dwz](#): **dwz** ツールを使用して、**ELF** 共有ライブラリーや、サイズの **ELF** 実行ファイルに含まれる **DWARF** デバッグ情報を最適化する説明。

第6章 DWZ

dwz は、ELF 共有ライブラリー および ELF 実行可能ファイルに含まれる DWARF デバッグ情報のサイズの最適化を試みるコマンドラインツールです。そのためには、**dwz** が DWARF 情報表現を、可能な場合は同等の小規模な表現に置き換え、**DWARF Standard** の Appendix E からの手法を使用して重複の量を減らします。

Red Hat Developer Toolset には **dwz 0.14** が同梱されています。

6.1. DWZ のインストール

Red Hat Developer Toolset では、**devtoolset-12-dwz** パッケージにより **dwz** ユーティリティーが提供され、「[Red Hat Developer Toolset のインストール](#)」で説明されているように、**devtoolset-12-toolchain** で自動的にインストールされます。

6.2. DWZ の使用

バイナリーファイルで DWARF デバッグ情報を最適化するには、以下のように **dwz** ツールを実行します。

```
$ scl enable devtoolset-12 'dwz option... file_name'
```

この **scl** ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、デフォルトで Red Hat Developer Toolset **dwz** でシェルセッションを実行できます。

```
$ scl enable devtoolset-12 'bash'
```

注記

使用中の **dwz** のバージョンを確認するには、以下を行います。

```
$ which dwz
```

Red Hat Developer Toolset の **dwz** 実行可能なパスは、**/opt** で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset **dwz** と一致を確認することができます。

```
$ dwz -v
```

6.3. 関連資料

dwz とその機能の詳細は、以下に挙げるリソースを参照してください。

インストールされているドキュメント

- **dwz(1)**: **dwz** ユーティリティーの man ページは、その使用方法の詳細情報を提供します。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man dwz'
```

-

以下も併せて参照してください。

- [1章Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。
- [2章GNU コンパイラコレクション\(GCC\)](#): C、C++、および Fortran で書かれたプログラムのコンパイル手順。
- [4章binutils](#): **binutils** を使用する手順。オブジェクトファイルおよびバイナリーを検査および操作するためのバイナリーツールのコレクションです。
- [5章elfutils](#): **elfutils** を使用する手順。ELF ファイルを検査および操作するためのバイナリーツールのコレクションです。

第7章 ANNOBIN

Annobin プロジェクトは、**annobin** プラグインと **annockeck** プログラムで設定されます。

annobin プラグインは、GNU コンパイラコレクション (GCC) コマンドライン、コンパイル状態、およびコンパイルプロセスをスキャンし、ELF ノートを生成します。ELF ノートでは、バイナリーの構築方法を記録し、セキュリティ強化チェックを実行する **annockeck** プログラムの情報を得ることができます。

セキュリティ強化チェッカーは **annockeck** プログラムの一部で、デフォルトで有効になっています。バイナリーファイルをチェックして、必要なセキュリティ強化オプションでプログラムが構築されていて、正しくコンパイルされているかを判断します。**annockeck** は、ELF オブジェクトファイルのディレクトリー、アーカイブ、および RPM パッケージを再帰的にスキャンできます。



注記

ファイルは ELF 形式である必要があります。**annockeck** は、他のバイナリーファイルタイプの処理に対応していません。

7.1. ANNOBIN のインストール

Red Hat Developer Toolset では、**devtoolset-12-gcc** パッケージにより、**annobin** プラグインおよび **annockeck** プログラムが提供され、「[オプションパッケージのインストール](#)」の説明に従ってインストールされます。

7.2. ANNOBIN プラグインの使用

gcc を使用して **annobin** プラグインにオプションを渡すには、以下を使用します。

```
$ scl enable devtoolset-12 'gcc -fplugin=annobin -fplugin-arg-annobin-option file-name'
```

この **scl** ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、デフォルトで Red Hat Developer Toolset **as** でシェルセッションを実行できます。

```
$ scl enable devtoolset-12 'bash'
```

7.3. ANNOCHECK の使用

annockeck プログラムでファイル、ディレクトリー、または RPM パッケージをスキャンするには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'annockeck file-name'
```



注記

annockeck は ELF ファイルのみを検索します。他のファイルタイプは無視されます。

この **scl** ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意して

ください。これにより、デフォルトで Red Hat Developer Toolset **as** でシェルセッションを実行できます。

```
$ scl enable devtoolset-12 'bash'
```

注記

使用中の **annockeck** のバージョンを確認するには、以下を行います。

```
$ which annockeck
```

Red Hat Developer Toolset の **annockeck** 実行可能なパスは、**/opt** で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset **annockeck** と一致することを確認することができます。

```
$ annockeck --version
```

7.4. 関連資料

annockeck、**annobin**、および その機能の詳細は、以下の資料を参照してください。

インストールされているドキュメント

- **annockeck(1)**: **annockeck** ユーティリティーの man ページには、その使用方法に関する詳細情報が記載されています。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man annockeck'
```

- **annobin(1)**: **annobin** ユーティリティーの man ページには、その使用方法に関する詳細情報が記載されています。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man annobin'
```

パート III. デバッグツール

第8章 GNU デバッガー (GDB)

GNU デバッガー (通常は **GDB** として省略) は、さまざまなプログラミング言語で書かれたプログラムのデバッグに使用できるコマンドラインツールです。これにより、デバッグするコード内でメモリーを検査したり、コードの実行状態を制御したり、コードの特定セクションの実行を検出したりできます。

Red Hat Developer Toolset には **GDB 11.2** が同梱されています。このバージョンは、Red Hat Enterprise Linux に同梱されていたバージョンと Red Hat Developer Toolset の以前のリリースよりも新しいもので、機能拡張やバグ修正が数多く追加されました。

8.1. GNU デバッガーのインストール

Red Hat Developer Toolset では、**devtoolset-12-gdb** パッケージにより **GNU Debugger** が提供され、「[Red Hat Developer Toolset のインストール](#)」の説明に従って **devtoolset-12-toolchain** で自動的にインストールされます。

8.2. デバッグプログラムの準備

デバッグ情報を使用したプログラムのコンパイル

GNU デバッガー が読み取り可能なデバッグ情報で C プログラムをコンパイルするには、**-g** オプションを指定して **gcc** コンパイラーが実行されていることを確認します。

```
$ scl enable devtoolset-12 'gcc -g -o output_file input_file...'
```

同様に、デバッグ情報を使用して C++ プログラムをコンパイルするには、以下を実行します。

```
$ scl enable devtoolset-12 'g++ -g -o output_file input_file...'
```

例8.1 デバッグ情報を使用した C プログラムのコンパイル

以下の内容を **fibonacci.c** 含むという名前のソースファイルについて考えてみましょう。

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

int main (int argc, char *argv[]) {
    unsigned long int a = 0;
    unsigned long int b = 1;
    unsigned long int sum;

    while (b < LONG_MAX) {
        printf("%ld ", b);
        sum = a + b;
        a = b;
        b = sum;
    }

    return 0;
}
```

GNU デバッガー のデバッグ情報とともに、Red Hat Developer Toolset の **GCC** を使用してコマンドラインでこのプログラムをコンパイルします。

```
$ scl enable devtoolset-12 'gcc -g -o fibonacci fibonacci.c'
```

これにより、現在の作業ディレクトリーに **fibonacci** という名前のバイナリーファイルが作成されます。

既存パッケージのデバッグ情報のインストール

システムにすでにインストールされているパッケージのデバッグ情報をインストールするには、次のコマンドを実行します。

```
# debuginfo-install package_name
```

debuginfo-install ユーティリティーをシステムで使用できるようにするには、**yum-utils** パッケージがインストールされている必要があることに注意してください。

例8.2 glibc パッケージのデバッグ情報のインストール

glibc パッケージのデバッグ情報をインストールします。

```
# debuginfo-install glibc
Loaded plugins: product-id, refresh-packagekit, subscription-manager
--> Running transaction check
---> Package glibc-debuginfo.x86_64 0:2.17-105.el7 will be installed
...
```

8.3. GNU デバッガーの実行

デバッグするプログラムで **GNU デバッガー** を実行するには、以下を実行します。

```
$ scl enable devtoolset-12 'gdb file_name'
```

これにより、インタラクティブモードで **gdb** デバッガーが開始され、デフォルトのプロンプト (**gdb**) が表示されます。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、いつでも以下のコマンドを実行します。

```
(gdb) quit
```

この **scl** ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、デフォルトで Red Hat Developer Toolset **gdb** でシェルセッションを実行できます。

```
$ scl enable devtoolset-12 'bash'
```



注記

使用中の **gdb** のバージョンを確認するには、以下を行います。

\$ which gdb

Red Hat Developer Toolset の **gdb** 実行可能なパスは、**/opt** で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset **gdb** と一致を確認することができます。

\$ gdb -v

例8.3 fiacci バイナリーファイルでの gdb ユーティリティーの実行

この例では、例8.1「デバッグ情報を使用した C プログラムのコンパイル」の説明に従って **fibonacci** バイナリーファイルを正常にコンパイルしていることを前提としています。

gdb でデバッグ **fibonacci** を起動します。

```
$ scl enable devtoolset-12 'gdb fibonacci'
GNU gdb (GDB) Red Hat Enterprise Linux 8.2-2.el7
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from fibonacci...done.
(gdb)
```

8.4. ソースコードの一覧表示

デバッグしているプログラムのソースコードを表示するには、次のコマンドを実行します。

(gdb) list

デバッグしているプログラムの実行を開始する前に、**gdb** はソースコードの最初の 10 行を表示し、その後にも使用しても別の 10 行が一覧表示されます。実行を開始すると、**gdb** は通常はブレークポイントを設定する際に実行が停止した行を取ります。

特定の行を取り囲むコードを表示することもできます。

(gdb) list file_name:line_number

同様に、特定の関数の開始点を囲むコードを表示するには、次のコマンドを実行します。

```
(gdb) list file_name:function_name
```

list コマンドが表示する行数を変更できます。

```
(gdb) set listsize number
```

例8.4 fiacci バイナリーファイルのソースコードの一覧表示

例8.1「デバッグ情報を使用した C プログラムのコンパイル」に記載されている **fibonacci.c** ファイルには、正確に 17 行があります。デバッグ情報を使用してコンパイルし、**gdb** ユーティリティーにソースコード全体を一覧表示できるようにする場合は、以下のコマンドを実行して、一覧表示されている行の数を 20 に変更します。

```
(gdb) set listsize 20
```

引数なしで **list** コマンドを実行して、デバッグするファイルのソースコード全体を表示できるようになりました。

```
(gdb) list
1  #include <stdio.h>
2  #include <limits.h>
3
4  int main (int argc, char *argv[]) {
5      unsigned long int a = 0;
6      unsigned long int b = 1;
7      unsigned long int sum;
8
9      while (b < LONG_MAX) {
10         printf("%ld ", b);
11         sum = a + b;
12         a = b;
13         b = sum;
14     }
15
16     return 0;
17 }
```

8.5. ブレークポイントの設定

新しいブレークポイントの設定

特定の行に新しいブレークポイントを設定するには、以下を行います。

```
(gdb) break file_name:line_number
```

特定の関数でブレークポイントを設定することもできます。

```
(gdb) break file_name:function_name
```

例8.5 新しいブレークポイントの設定

この例では、例8.1「デバッグ情報を使用したCプログラムのコンパイル」に一覧表示されている **fibonacci.c** ファイルをデバッグ情報を使用してコンパイルしていることを前提としています。

10 行目で新しいブレークポイントを設定します。

```
(gdb) break 10
Breakpoint 1 at 0x4004e5: file fibonacci.c, line 10.
```

ブレークポイントの一覧表示

現在設定されているブレークポイントの一覧を表示するには、次のコマンドを実行します。

```
(gdb) info breakpoints
```

例8.6 ブレークポイントの一覧表示

この例では、例8.5「新しいブレークポイントの設定」の手順に従っていることを前提としています。

現在設定されているブレークポイントの一覧を表示します。

```
(gdb) info breakpoints
Num   Type      Disp Enb Address          What
1     breakpoint keep y  0x00000000004004e5 in main at fibonacci.c:10
```

既存のブレークポイントの削除

特定の行に設定されている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。

```
(gdb) clear line_number
```

同様に、特定の関数に設定した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。

```
(gdb) clear function_name
```

例8.7 既存のブレークポイントの削除

この例では、例8.1「デバッグ情報を使用したCプログラムのコンパイル」に一覧表示されている **fibonacci.c** ファイルをデバッグ情報を使用してコンパイルしていることを前提としています。

7 行目で新しいブレークポイントを設定します。

```
(gdb) break 7
Breakpoint 2 at 0x4004e3: file fibonacci.c, line 7.
```

このブレークポイントを削除します。

```
(gdb) clear 7
Deleted breakpoint 2
```

8.6. 実行の開始

デバッグしているプログラムの実行を開始するには、次のコマンドを実行します。

```
(gdb) run
```

プログラムがコマンドライン引数を許可する場合は、**run** コマンドに引数として指定できます。

```
(gdb) run argument...
```

最初のブレークポイント (存在する場合) 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実行が停止します。

例8.8 フィボナッチバイナリーファイルの実行

この例では、[例8.5「新しいブレークポイントの設定」](#)の手順に従っていることを前提としています。

fibonacci バイナリーファイルの実行

```
(gdb) run
Starting program: /home/john/fibonacci

Breakpoint 1, main (argc=1, argv=0x7fffffff4d8) at fibonacci.c:10
10      printf("%ld ", b);
```

8.7. 現在の値の表示

この **gdb** ユーティリティーを使用すると、複雑で、有効な式やライブラリー機能まで、プログラムに関連するほぼすべての値を表示できます。ただし、変数の値を表示するのが最も一般的なタスクです。

特定の変数の現在の値を表示するには、次のコマンドを実行します。

```
(gdb) print variable_name
```

例8.9 変数の現在の値の表示

この例では、[例8.8「フィボナッチバイナリーファイルの実行」](#)の指示に従い、10行目でブレークポイントに到達した後に、**fibonacci** バイナリーの実行が停止していることを前提としています。

変数 **a** および **b** の現在の値を表示します。

```
(gdb) print a
$1 = 0
(gdb) print b
$2 = 1
```

8.8. 継続実行

ブレークポイントに達した後にデバッグするプログラムの実行を再開するには、次のコマンドを実行します。

```
(gdb) continue
```

別のブレークポイントに到達すると、実行が再び停止します。特定の数のブレークポイントをスキップするには (通常はループをデバッグする場合)、以下を実行します。

```
(gdb) continue number
```

この **gdb** ユーティリティーでは、1行のコードを実行した後に実行を停止することもできます。

```
(gdb) step
```

最後に、特定の数行を実行できます。

```
(gdb) step number
```

例8.10 フィボナッチバイナリーファイルの実行の継続

この例では、[例8.8「フィボナッチバイナリーファイルの実行」](#)の指示に従い、10行目でブレークポイントに到達した後に、**fibonacci** バイナリーの実行が停止していることを前提としています。

実行を再開します。

```
(gdb) continue
Continuing.
```

```
Breakpoint 1, main (argc=1, argv=0x7ffffffe4d8) at fibonacci.c:10
10      printf("%ld ", b);
```

ブレークポイントに到達すると、実行が停止します。

次の3行コードを実行します。

```
(gdb) step 3
13      b = sum;
```

これにより、**b** に変数を割り当てる前に、**sum** 変数の現在の値を確認することができます。

```
(gdb) print sum
$3 = 2
```

8.9. 関連資料

GNU デバッガー およびその機能の詳細は、以下の資料を参照してください。

インストールされているドキュメント

devtoolset-12-gdb-doc パッケージをインストールすると、次のドキュメントが HTML および PDF 形式で `/opt/rh/devtoolset-12/root/usr/share/doc/devtoolset-12-gdb-doc-11.2` ディレクトリーに提供されます。

- **GDB ガイドを使用したデバッグ**。これは、同じ名前のアップストリーム資料のコピーです。本書のバージョンは、Red Hat Developer Toolset で利用可能な **GDB** のバージョンに完全に対応します。
- **GDB の Obsolete Annotations** ドキュメント。これは廃止された **GDB** レベル 2 アノテーションを一覧表示します。

オンラインドキュメント

- [Red Hat Enterprise Linux 7 開発者ガイド](#) : Red Hat Enterprise Linux 7 **開発者ガイド** では、**GNU デバッガー** およびデバッグに関する詳細情報が提供されています。
- [GDB ドキュメント](#): アップストリームの **GDB** ドキュメントには、**GDB User Manual** およびその他の参考資料が含まれています。

以下も併せて参照してください。

- [1章 Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。
- [2章 GNU コンパイラーコレクション \(GCC\)](#): C、C++、および Fortran で書かれたプログラムをコンパイルする方法の詳細。
- [9章 strace](#): **strace** ユーティリティーを使用して、プログラムが使用するシステムコールを監視し、受信するシグナル。
- [11章 memstomp](#): **memstomp** ユーティリティーを使用する手順で、さまざまな標準で使用できないメモリー領域が重複しているライブラリー関数への呼び出しを特定します。

第9章 STRACE

strace は、実行中のプロセスによって確立および受信されたシステムコールを追跡するために使用できるコマンドラインの診断およびデバッグツールです。これは、各システムコールの名前、引数、および戻り値と、プロセスによって受信したシグナルとそのカーネルとの対話を記録し、このレコードを標準エラー出力または選択したファイルに出力します。

Red Hat Developer Toolset には **strace 5.18** が同梱されています。

9.1. STRACE のインストール

Red Hat Enterprise Linux では、**devtoolset-12-strace** パッケージにより **strace** ユーティリティーが提供され、「[Red Hat Developer Toolset のインストール](#)」の説明に従って **devtoolset-12-toolchain** で自動的にインストールされます。

9.2. STRACE の使用

分析するプログラムで **strace** ユーティリティーを実行するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'strace program argument...'
```

program を、分析するプログラムの名前に置き換え、**argument** を、このプログラムに指定するコマンドラインオプションと引数に置き換えます。以下の例では、**-p** コマンドラインオプションとプロセス ID を使用して、実行中のプロセスでユーティリティーを実行できます。

```
$ scl enable devtoolset-12 'strace -p process_id'
```

この **scl** ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、デフォルトで Red Hat Developer Toolset **strace** でシェルセッションを実行できます。

```
$ scl enable devtoolset-12 'bash'
```

注記

任意の時点で使用している **strace** のバージョンを確認するには、以下のコマンドを実行します。

```
$ which strace
```

Red Hat Developer Toolset の **strace** 実行可能なパスは、**/opt** で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset **strace** と一致することを確認することができます。

```
$ strace -V
```

9.2.1. 出力のファイルへのリダイレクト

9.2.1. 出力のファイルへのリダイレクト

デフォルトでは **strace**、各システムコールの名前、引数、および戻り値を標準エラー出力に出力します。この出力をファイルにリダイレクトするには、**-o** コマンドラインオプションの後にファイル名を指定します。

```
$ scl enable devtoolset-12 'strace -o file_name program argument...'
```

`file_name` をファイル名に置き換えます。

例9.1 出力のファイルへのリダイレクト

例8.1「デバッグ情報を使用した C プログラムのコンパイル」から **fibonacci** ファイルのバージョンを若干変更したことを検討してください。この実行可能ファイルには、Fibonacci シーケンスが表示され、オプションでこのシーケンスのメンバー数を指定することができます。このファイルで **strace** ユーティリティーを実行し、トレース出力を **fibonacci.log** リダイレクトします。

```
$ scl enable devtoolset-12 'strace -o fibonacci.log ./fibonacci 20'
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

これにより、現在の作業ディレクトリーに、**fibonacci.log** という新しいプレーンテキストファイルが作成されます。

9.2.2. 選択したシステム呼び出しの追跡

選択したシステムコールのセットのみを追跡するには、**strace** コマンドラインオプションを指定して **-e** ユーティリティーを実行します。

```
$ scl enable devtoolset-12 'strace -e expression program argument...'
```

`expression` を、トレースするシステムコールのコンマ区切りリスト、または表9.1「**-e** オプションで一般的に使用される値」にリストされているにキーワードに置き換えます。使用できるすべての値の説明は、**strace(1)** の man ページを参照してください。

表9.1 **-e** オプションで一般的に使用される値

値	説明
%file	ファイル名を引数として受け入れるシステムコール。
%process	プロセス管理に関連するシステムコール。
%network	ネットワークに関連するシステムコール。
%signal	シグナル管理に関連するシステムコール。
%ipc	IPC (inter-process communication) に関連するシステムコール。
%desc	ファイル記述子に関連するシステムコール。

構文 **-e 式** は、完全な形式の **-e trace=式** です。

例9.2 選択したシステム呼び出しの追跡

例11.1「[memstomp の使用](#)」の **employee** ファイルを考慮します。この実行可能ファイルで **strace** ユーティリティーを実行し、**mmap** および **munmap** システムコールのみをトレースします。

```
$ scl enable devtoolset-12 'strace -e mmap,munmap ./employee'
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f896c744000
mmap(NULL, 61239, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f896c735000
mmap(0x3146a00000, 3745960, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x3146a00000
mmap(0x3146d89000, 20480, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x189000) = 0x3146d89000
mmap(0x3146d8e000, 18600, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x3146d8e000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f896c734000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f896c733000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f896c732000
munmap(0x7f896c735000, 61239) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f896c743000
John,john@example.comDoe,
+++ exited with 0 +++
```

9.2.3. タイムスタンプの表示

トレースの各行を、時間、分、および秒で正確な時刻に接頭辞するには、**-t** コマンドラインオプションを指定して **strace** ユーティリティーを実行します。

```
$ scl enable devtoolset-12 'strace -t program argument...'
```

ミリ秒を表示するには、**-t** オプションを 2 回指定します。

```
$ scl enable devtoolset-12 'strace -tt program argument...'
```

トレースの各行を、各システムコールの実行に必要な時間に接頭辞を付けるには、**-r** コマンドラインオプションを使用します。

```
$ scl enable devtoolset-12 'strace -r program argument...'
```

例9.3 タイムスタンプの表示

pwd という名前の実行ファイルを考慮してください。このファイルで **strace** ユーティリティーを実行し、出力にタイムスタンプを含めます。

```
$ scl enable devtoolset-12 'strace -tt pwd'
19:43:28.011815 execve("./pwd", ["/.pwd"], [/* 36 vars */]) = 0
```

```

19:43:28.012128 brk(0)          = 0xcd3000
19:43:28.012174 mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc869cb0000
19:43:28.012427 open("/etc/ld.so.cache", O_RDONLY) = 3
19:43:28.012446 fstat(3, {st_mode=S_IFREG|0644, st_size=61239, ...}) = 0
19:43:28.012464 mmap(NULL, 61239, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc869ca1000
19:43:28.012483 close(3)          = 0
...
19:43:28.013410 +++ exited with 0 +++

```

9.2.4. サマリーの表示

各システムコールの実行に必要な時間、これらのシステムコールが実行した回数、実行中に発生したエラー数の概要を表示するには、**-c** コマンドラインオプションを指定して **strace** ユーティリティーを実行します。

```
$ scl enable devtoolset-12 'strace -c program argument...'
```

例9.4 サマリーの表示

lsblk という名前の実行ファイルを考慮してください。このファイルで **strace** ユーティリティーを実行し、トレースの概要を表示します。

```

$ scl enable devtoolset-12 'strace -c lsblk > /dev/null'
% time  seconds  usecs/call  calls  errors syscall
-----
 80.88  0.000055      1    106    16 open
 19.12  0.000013      0    140     0 munmap
  0.00  0.000000      0    148     0 read
  0.00  0.000000      0     1     0 write
  0.00  0.000000      0    258     0 close
  0.00  0.000000      0     37     2 stat
...
-----
100.00  0.000068      0   1790    35 total

```

9.2.5. システムコール結果の改ざん

システムコールから返されたエラーをシミュレートすると、プログラムで不足しているエラー処理の特定に役立ちます。

特定のシステムコールの結果としてプログラムが一般的なエラーを受け取るには、**-e fault=** オプションを指定して **strace** ユーティリティーを実行し、システムコールを指定します。

```
$ scl enable devtoolset-12 'strace -e fault=syscall program argument...'
```

エラーの種類または戻り値を指定するには、**-e inject=** オプションを使用します。

```

$ scl enable devtoolset-12 'strace -e inject=syscall:error=error-type program argument'
$ scl enable devtoolset-12 'strace -e inject=syscall:retval=return-value program argument'

```

エラータイプと戻り値は相互排他的であることに注意してください。

例9.5 システムコール結果の改ざん

lsblk という名前の実行ファイルを考慮してください。このファイルで **strace** ユーティリティーを実行すると、**mmap()** システムコールがエラーを返すようにします。

```
$ scl enable devtoolset-12 'strace -e fault=mmap:error=EPERM lsblk > /dev/null'
execve("/usr/bin/lsblk", ["lsblk"], 0x7fff1c0e02a0 /* 54 vars */) = 0
brk(NULL) = 0x55d9e8b43000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = -
1 EPERM (Operation not permitted) (INJECTED)
writev(2, [{iov_base="lsblk", iov_len=5}, {iov_base=": ", iov_len=2}, {iov_base="error while loading
shared libra...", iov_len=36}, {iov_base=": ", iov_len=2}, {iov_base="", iov_len=0}, {iov_base="",
iov_len=0}, {iov_base="cannot create cache for search p...", iov_len=35}, {iov_base=": ",
iov_len=2}, {iov_base="Cannot allocate memory", iov_len=22}, {iov_base="\n", iov_len=1}],
10lsblk: error while loading shared libraries: cannot create cache for search path: Cannot allocate
memory
) = 105
exit_group(127) = ?
+++ exited with 127 +++
```

9.3. 関連資料

strace とその機能の詳細は、以下に挙げるリソースを参照してください。

インストールされているドキュメント

- **strace(1): strace** ユーティリティーの man ページは、その使用方法に関する詳細情報を提供します。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man strace'
```

以下も併せて参照してください。

- [1章 Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。
- [10章 ltrace](#): **ltrace** ツールを使用して、プログラムライブラリー呼び出しを追跡する手順。
- [8章 GNU デバッガー \(GDB\)](#): C、C++、および Fortran で書かれたデバッグプログラムの手順。
- [11章 memstomp](#): **memstomp** ユーティリティーを使用する手順で、さまざまな標準で使用できないメモリー領域が重複しているライブラリー関数への呼び出しを特定します。

第10章 LTRACE

ltrace は、コマンドラインの診断およびデバッグのツールです。これは、共有ライブラリーに追加された呼び出しの表示に使用できます。これは動的ライブラリーフックメカニズムを使用し、静的にリンクされたライブラリーへの呼び出しを追跡できなくなります。**ltrace** は、ライブラリーコールの戻り値も表示します。出力は、標準エラー出力または選択したファイルに出力されます。

Red Hat Developer Toolset には **ltrace 0.7.91** が同梱されています。ベースバージョンの **ltrace** は、以前のリリースの Red Hat Developer Toolset と同じですが、さまざまな機能強化およびバグ修正がポートされています。

10.1. LTRACE のインストール

Red Hat Enterprise Linux では、**devtoolset-12-ltrace** パッケージにより **ltrace** ユーティリティーが提供され、「[Red Hat Developer Toolset のインストール](#)」の説明通りに **devtoolset-12-toolchain** で自動的にインストールされます。

10.2. LTRACE の使用

分析するプログラムで **ltrace** ユーティリティーを実行するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'ltrace program argument...'
```

program を、分析するプログラムの名前に置き換え、**argument** を、このプログラムに指定するコマンドラインオプションと引数に置き換えます。以下の例では、**-p** コマンドラインオプションとプロセス ID を使用して、実行中のプロセスでユーティリティーを実行できます。

```
$ scl enable devtoolset-12 'ltrace -p process_id'
```

この **scl** ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、デフォルトで Red Hat Developer Toolset **ltrace** でシェルセッションを実行できます。

```
$ scl enable devtoolset-12 'bash'
```

注記

任意の時点で使用している **ltrace** のバージョンを確認するには、以下のコマンドを実行します。

```
$ which ltrace
```

Red Hat Developer Toolset の **ltrace** 実行可能なパスは、**/opt** で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset **ltrace** と一致することを確認することができます。

```
$ ltrace -V
```

10.2.1. 出力のファイルへのリダイレクト

デフォルトでは **ltrace**、各システムコールの名前、引数、および戻り値を標準エラー出力に出力します。この出力をファイルにリダイレクトするには、**-o** コマンドラインオプションの後にファイル名を指定します。

```
$ scl enable devtoolset-12 'ltrace -o file_name program argument...'
```

`file_name` をファイル名に置き換えます。

例10.1 出力のファイルへのリダイレクト

例8.1「デバッグ情報を使用したCプログラムのコンパイル」から **fibonacci** ファイルのバージョンを若干変更したことを検討してください。この実行可能ファイルには、Fibonacci シーケンスが表示され、オプションでこのシーケンスのメンバー数を指定することができます。このファイルで **ltrace** ユーティリティを実行し、トレース出力を **fibonacci.log** にリダイレクトします。

```
$ scl enable devtoolset-12 'ltrace -o fibonacci.log ./fibonacci 20'
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

これにより、現在の作業ディレクトリーに、**fibonacci.log** という新しいプレーンテキストファイルが作成されます。

10.2.2. 選択したライブラリー呼び出しの追跡

選択したライブラリーコールのセットのみを追跡するには、**-e** コマンドラインオプションを指定して **ltrace** ユーティリティを実行します。

```
$ scl enable devtoolset-12 'ltrace -e expression program argument...'
```

`expression` をルールチェーンに置き換え、トレースするライブラリー呼び出しを指定します。ルールは、シンボル名 (**malloc** または **free** など) と、ライブラリー SONAME (**libc.so** など) を識別するパターンで設定されています。たとえば、**malloc** および **free** 関数への呼び出しを追跡し、**libc** ライブラリーが実行したものを省略するには、以下のコマンドを実行します。

```
$ scl enable devtoolset-12 'ltrace -e malloc+free-@libc.so* program'
```

例10.2 選択したライブラリー呼び出しの追跡

ls コマンドを考慮します。このプログラムで **ltrace** ユーティリティを実行し、**opendir**、**readdir**、および **closedir** 関数呼び出しでのみトレースします。

```
$ scl enable devtoolset-12 'ltrace -e opendir+readdir+closedir ls'
ls->opendir(".") = { 3 }
ls->readdir({ 3 }) = { 61533, "." }
ls->readdir({ 3 }) = { 131, ".." }
ls->readdir({ 3 }) = { 67185100, "BUILDROOT" }
ls->readdir({ 3 }) = { 202390772, "SOURCES" }
ls->readdir({ 3 }) = { 60249, "SPECS" }
ls->readdir({ 3 }) = { 67130110, "BUILD" }
ls->readdir({ 3 }) = { 136599168, "RPMS" }
ls->readdir({ 3 }) = { 202383274, "SRPMS" }
ls->readdir({ 3 }) = nil
```

```
ls->closedir({ 3 }) = 0
BUILD BUILDROOT RPMS SOURCES SPECS SRPMS
+++ exited (status 0) +++
```

利用可能なフィルター式の詳細は、**ltrace(1)** man ページを参照してください。

10.2.3. タイムスタンプの表示

トレースの各行を、時間、分、および秒で正確な時刻に接頭辞するには、**-t** コマンドラインオプションを指定して **ltrace** ユーティリティを実行します。

```
$ scl enable devtoolset-12 'ltrace -t program argument...'
```

ミリ秒を表示するには、**-t** オプションを 2 回指定します。

```
$ scl enable devtoolset-12 'ltrace -tt program argument...'
```

トレースの各行を、各システムコールの実行に必要な時間に接頭辞を付けるには、**-r** コマンドラインオプションを使用します。

```
$ scl enable devtoolset-12 'ltrace -r program argument...'
```

例10.3 タイムスタンプの表示

pwd コマンドを考慮します。このプログラムで **ltrace** ユーティリティを実行し、出力にタイムスタンプを含めます。

```
$ scl enable devtoolset-12 'ltrace -tt pwd'
13:27:19.631371 __libc_start_main([ "pwd" ] <unfinished ...>
13:27:19.632240 getenv("POSIXLY_CORRECT") = nil
13:27:19.632520 strrchr("pwd", '/') = nil
13:27:19.632786 setlocale(LC_ALL, "") = "en_US.UTF-8"
13:27:19.633220 bindtextdomain("coreutils", "/usr/share/locale") = "/usr/share/locale"
13:27:19.633471 textdomain("coreutils") = "coreutils"
(...)
13:27:19.637110 exited (status 0)
```

10.2.4. サマリーの表示

各システムコールの実行に必要な時間と、これらのシステムコールを実行した回数の概要を表示するには、**-c** コマンドラインオプションを指定して **ltrace** ユーティリティを実行します。

```
$ scl enable devtoolset-12 'ltrace -c program argument...'
```

例10.4 サマリーの表示

lsblk コマンドについて考えてみましょう。このプログラムで **ltrace** ユーティリティを実行し、トレースの概要を表示します。

```
$ scl enable devtoolset-12 'ltrace -c lsblk > /dev/null'
```

```

% time  seconds  usecs/call  calls  function
-----
53.60  0.261644   261644     1  __libc_start_main
4.48   0.021848    58       374  mbrtowc
4.41   0.021524    57       374  wcwidth
4.39   0.021409    57       374  __ctype_get_mb_cur_max
4.38   0.021359    57       374  iswprint
4.06   0.019838    74       266  readdir64
3.21   0.015652    69       224  strlen
...
-----
100.00  0.488135           3482  total

```

10.3. 関連資料

`ltrace` とその機能の詳細は、以下に挙げるリソースを参照してください。

インストールされているドキュメント

- **ltrace(1): ltrace** ユーティリティーの man ページは、その使用方法に関する詳細情報を提供します。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man ltrace'
```

オンラインドキュメント

- [RHEL 6 および 7 の ltrace](#): Red Hat Developer Blog の記事では、アプリケーションのデバッグに `ltrace` を使用する方法について、詳細な情報 (実用的な例を含む) が提供されます。

以下も併せて参照してください。

- [1章 Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。
- [9章 strace](#): `strace` ツールを使用して、プログラムシステムコールを追跡する手順。
- [8章 GNU デバッガー \(GDB\)](#): C、C++、および Fortran で書かれたデバッグプログラムの手順。
- [11章 memstomp](#): `memstomp` ユーティリティーを使用する手順で、さまざまな標準で使用できないメモリー領域が重複しているライブラリー関数への呼び出しを特定します。

第11章 MEMSTOMP

memstomp は、このような重複がさまざまな標準で許可されていない場合に、重複するメモリー領域で関数呼び出しを特定するために使用できます。表11.1「**memstomp** によって検査された関数呼び出し」に一覧表示されているライブラリー関数への呼び出しをインターセプトし、それぞれのメモリーの重複に対して詳細なバックトレースが表示され、問題のデバッグに役立つ詳細なバックトレースが表示されます。

Valgrind と同様に、**memstomp** ユーティリティーはアプリケーションを再コンパイルせずに検査します。ただし、このツールよりも高速であるため、便利な代替手段となります。

Red Hat Developer Toolset には、**memstomp 0.1.5** が同梱されています。

表11.1 **memstomp** によって検査された関数呼び出し

機能	説明
memcpy	n バイトをメモリー領域から別のメモリー領域にコピーし、2 番目のメモリー領域にポインターを返します。
memccpy	最大 n バイトを1つのメモリー領域から別のメモリー領域にコピーし、特定の文字が見つかったときに停止します。最後に書き込まれたバイトの後にポインターを返すか、指定の文字が見つからない場合は NULL を返します。
mempcpy	n バイトを1つのメモリー領域から別のメモリー領域にコピーし、最後に書き込まれたバイトの後にポインターをバイトに戻します。
strcpy	あるメモリー領域から別のメモリー領域に文字列をコピーし、2 番目の文字列にポインターを返します。
stpcpy	あるメモリー領域から別のメモリー領域に文字列をコピーし、2 番目の文字列の null 終端文字にポインターを返します。
strncpy	最大 n 文字を1つの文字列から別の文字列にコピーし、2 番目の文字列にポインターを返します。
stpncpy	最大 n 文字を1つの文字列から別の文字列にコピーします。2 番目の文字列の終端 null バイトへのポインターを返します。または、文字列が null 終端ではない場合、最後に書き込まれたバイトの後にバイトへのポインターを返します。
strcat	1つの文字列を別の文字列に追加し、2 番目の文字列の終了点を上書きし、その最後に新しい文字列を追加します。新しい文字列へのポインターを返します。
strncat	1つの文字列から別の文字列に最大 n 文字を追加し、2 番目の文字列の終了点を上書きして、その最後に新しい文字列を追加します。新しい文字列へのポインターを返します。
wmemcpy	n ワイド文字を、あるアレイから別のアレイにコピーし、2 番目のアレイにポインターを返す memcpy() 関数に相当するワイド文字。
wmempcpy	n ワイド文字を、あるアレイから別のアレイにコピーし、書き込まれたワイド文字のアレイに続くバイトにポインターを返す mempcpy() 関数に相当するワイド文字。

機能	説明
wcscpy	ワイド文字の文字列を、あるアレイから別のアレイにコピーし、2番目のアレイにポインタを返す strncpy() 関数に相当するワイド文字。
wcsncpy	最大 n ワイド文字を、あるアレイから別のアレイにコピーし、2番目の文字列にポインタを返す strncpy() 関数に相当するワイド文字。
wcscat	1つのワイド文字の文字列を別の文字列に追加し、2番目の文字列の null 終端文字を上書きして、その最後に新しい文字列を追加する、 strcat() 関数のワイド文字。新しい文字列へのポインタを返します。
wcsncat	最大 n のワイド文字を、あるアレイから別のアレイに追加し、2つ目のワイド文字の文字列の null 終端文字を上書きして、新しいものを最後に追加する strncat() 関数のワイド文字。新しい文字列へのポインタを返します。

11.1. MEMSTOMP のインストール

Red Hat Developer Toolset では、**devtoolset-12-memstomp** パッケージにより **memstomp** ユーティリティーが提供され、「[Red Hat Developer Toolset のインストール](#)」で説明されているように **devtoolset-12-toolchain** で自動的にインストールされます。

11.2. MEMSTOMP の使用

分析するプログラムで **memstomp** ユーティリティーを実行するには、以下を実行します。

```
$ scl enable devtoolset-12 'memstomp program argument...'
```

問題が検出されたときに、解析されたプログラムをすぐに終了するには、**--kill** (または **-k**) コマンドラインオプションを指定してユーティリティーを実行します。

```
$ scl enable devtoolset-12 'memstomp --kill program argument...'
```

マルチスレッドプログラムを分析する場合は、**--kill** オプションの使用が特に推奨されます。バックトレースの内部実装はスレッドセーフではなく、このオプションなしでマルチスレッドプログラムで **memstomp** ユーティリティーを実行すると信頼できない結果が発生する可能性があります。

さらに、解析しているプログラムをデバッグ情報でコンパイルした場合や、このデバッグ情報が利用できる場合は、**--debug-fnfo** (または **-d**) コマンドラインオプションを使用して、より詳細なバックトレースを作成できます。

```
$ scl enable devtoolset-12 'memstomp --debug-info program argument...'
```

バイナリーファイルにビルドされたデバッグ情報を使用してプログラムをコンパイルする方法は、「[デバッグプログラムの準備](#)」を参照してください。Red Hat Developer Toolset パッケージのデバッグ情報をインストールする方法は、「[デバッグ情報のインストール](#)」を参照してください。

この **scl** ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、Red Hat Developer Toolset **memstomp** をデフォルトとして設定して、シェルセッションを実行することができます。

```
$ scl enable devtoolset-12 'bash'
```

例11.1 memstomp の使用

現在の作業ディレクトリーに、以下の内容を含む **employee.c** という名前のソースファイルを作成します。

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

#define BUFSIZE 80

int main(int argc, char *argv[]) {
    char employee[BUFSIZE] = "John,Doe,john@example.com";
    char name[BUFSIZE] = {0};
    char surname[BUFSIZE] = {0};
    char *email;
    size_t length;

    /* Extract the information: */
    memccpy(name, employee, ',', BUFSIZE);
    length = strlen(name);
    memccpy(surname, employee + length, ',', BUFSIZE);
    length += strlen(surname);
    email = employee + length;

    /* Compose the new entry: */
    strcat(employee, surname);
    strcpy(employee, name);
    strcat(employee, email);

    /* Print the result: */
    puts(employee);

    return 0;
}
```

このプログラムを、**employee** という名前のバイナリーファイルにコンパイルします。

```
$ scl enable devtoolset-12 'gcc -rdynamic -g -o employee employee.c'
```

メモリー領域が重複する誤った関数呼び出しを特定するには、以下を実行します。

```
$ scl enable devtoolset-12 'memstomp --debug-info ./employee'
```

```
memstomp: 0.1.4 successfully initialized for process employee (pid 14887).
```

```
strcat(dest=0x7fff13afc265, src=0x7fff13afc269, bytes=21) overlap for employee(14887)
??:0 strcpy()
??:0 strcpy()
??:0 _Exit()
??:0 strcat()
employee.c:26 main()
```

```
??:0 __libc_start_main()
??:0 _start()
John,john@example.comDoe,
```

11.3. 関連資料

memstomp とその機能の詳細は、以下に挙げるリソースを参照してください。

インストールされているドキュメント

- **memstomp(1): memstomp** ユーティリティーの man ページは、その使用方法に関する詳細情報を提供します。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man memstomp'
```

以下も併せて参照してください。

- [1章 Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。
- [8章 GNU デバッガー \(GDB\)](#): C、C++、および Fortran で書かれたデバッグプログラムの手順。
- [9章 strace](#): strace ユーティリティーを使用して、プログラムが使用するシステムコールを監視し、受信するシグナル。
- [13章 Valgrind](#): Valgrind ツールを使用してアプリケーションのプロファイルを作成して、初期化されていないメモリの使用、メモリの割り当ておよびメモリの解放、システムコールでの不適切な引数の使用など、メモリエラーやメモリ管理問題を検出します。

パート IV. パフォーマンス監視ツール

第12章 SYSTEMTAP

SystemTap は、ユーザーがインストルメント化、再コンパイル、インストール、および再起動を行わずに、システム全体のアクティビティを監視できる追跡およびプロービングツールです。これはカスタムスクリプト言語でプログラミングできます。これにより、(トレース、フィルターリング、分析)、到達 (実行中のカーネルおよびアプリケーションを調べるために) 表現性を提供します。

SystemTap は、カーネルまたはアプリケーション内の関数呼び出し、タイマー、トレースポイント、パフォーマンスカウンターなど、さまざまなタイプのイベントを監視できます。一部のサンプルスクリプトは、Some included example scripts produce output similar to **netstat**, **ps**, **top**, **iostat** などの出力を生成するものもあります。これには、pretty-printed function callgraph トレースや、セキュリティーバグを操作するツールが含まれます。

Red Hat Developer Toolset には **SystemTap 4.7** が同梱されています。このバージョンは、以前のリリースの Red Hat Developer Toolset に含まれるバージョンよりも新しいもので、バグ修正および機能拡張が追加されています

表12.1 Red Hat Developer Toolset の SystemTap に分散したツール

名前	説明
stap	プローブ命令を C コードに変換し、カーネルモジュールを構築して、実行中の Linux カーネルに読み込みます。
stapdyn	SystemTap 用の Dyninst バックエンド。
staprun	stap ユーティリティーでビルドされたカーネルモジュールをロード、アンロード、アタッチ、切断します。
stapsh	SystemTap のリモートシェルとして機能します。
stap-prep	SystemTap の実行に必要なカーネル情報パッケージを判断して、可能であればダウンロードします。
stap-merge	CPU ファイルごとのマージ。このスクリプトは、 -b コマンドラインオプションで stap ユーティリティーを実行すると自動的に実行されます。
stap-report	SystemTap のバグを報告するために、システムに関する重要な情報を収集します。
stap-server	stap クライアントからのリクエストをリッスンするコンパイルサーバー。

12.1. SYSTEMTAP のインストール

Red Hat Developer Toolset では、**devtoolset-12-systemtap** パッケージにより **SystemTap** が提供され、「[Red Hat Developer Toolset のインストール](#)」の説明に従って **devtoolset-12-perftools** で自動的にインストールされます。

インストルメンテーションを Linux カーネルに配置するために、**SystemTap** ではデバッグ情報を含む追加パッケージをインストールする必要がある場合があります。インストールするパッケージを確認するには、以下のように **stap-prep** ユーティリティーを実行します。

\$ scli enable devtoolset-12 'stap-prep'

このコマンドを **root** ユーザーとして実行すると、このユーティリティーが自動的にインストール用のパッケージを提供することに注意してください。システムにこのパッケージをインストールする方法は、[Red Hat Enterprise Linux 7 SystemTap Beginners Guide](#)を参照してください。

12.2. SYSTEMTAP の使用

SystemTap に含まれるツールのいずれかを実行するには、以下を実行します。

\$ scli enable devtoolset-12 'tool option...'

SystemTap で配布されるツールの一覧は、[表12.1 「Red Hat Developer Toolset の SystemTap に分散したツール」](#)を参照してください。たとえば、**stap** ツールを実行してインストールメンテーションモジュールを構築するには、次のコマンドを実行します。

\$ scli enable devtoolset-12 'stap option... argument...'

この **scli** ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、Red Hat Developer Toolset **SystemTap** でシェルセッションをデフォルトとして実行できます。

\$ scli enable devtoolset-12 'bash'**注記**

任意の時点で使用している **SystemTap** のバージョンを確認するには、次のコマンドを実行します。

\$ which stap

Red Hat Developer Toolset の **stap** 実行可能なパスは、**/opt** で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset **SystemTap** と一致することを確認することができます。

\$ stap -V

12.3. 関連資料

SystemTap とその機能の詳細は、以下に挙げるリソースを参照してください。

インストールされているドキュメント

- **stap(1): stap** コマンドの man ページでは、その使用方法や、他の関連する man ページへの参照が提供されています。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

\$ scli enable devtoolset-12 'man stap'

- **staprun(8)**: **staprun** コマンドの man ページは、その使用方法の詳細情報を提供します。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man staprun'
```

オンラインドキュメント

- [Red Hat Enterprise Linux 7 SystemTap ビギナーズガイド](#) : Red Hat Enterprise Linux 7 の **SystemTap ビギナーズガイド** では、**SystemTap** とその使用方法を紹介します。
- [Red Hat Enterprise Linux 7 SystemTap Tapset Reference](#) : Red Hat Enterprise Linux 7 の **SystemTap Tapset Reference** は、**SystemTap** に関する詳細を提供します。
- [SystemTap ドキュメント](#): **SystemTap** ドキュメントでは、**SystemTap** に関するドキュメントや、**SystemTap** スクリプトの例を多数提供しています。

以下も併せて参照してください。

- [1章 Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。
- [13章 Valgrind](#): **Valgrind** ツールを使用してアプリケーションのプロファイルを作成して、初期化されていないメモリの使用、メモリの割り当ておよびメモリの解放、システムコールでの不適切な引数の使用など、メモリエラーやメモリ管理問題を検出します。
- [14章 OProfile](#): コードのどのセクションが最も多くの CPU 時間および理由を消費するかを決定するために **OProfile** ツールを使用する命令。
- [15章 Dyninst](#): **Dyninst** ライブラリーを使用してユーザー空間の実行ファイルをインストルメント化するための手順

第13章 VALGRIND

Valgrind は、プロファイリングアプリケーション用のツールが多数同梱されるインストールメンテーションフレームワークです。これは、初期化されていないメモリーの使用、メモリーの不適切な割り当ておよび解放など、さまざまなメモリーエラーやメモリー管理の問題を検出するために使用できます。Red Hat Developer Toolset バージョンの **Valgrind** で配布されるプロファイリングツールの完全リストは、[表13.1「Red Hat Developer Toolset の Valgrind で配布されるツール」](#) を参照してください。

Valgrind は、アプリケーションを書き換え、書き換えたバイナリーをインストール化して、アプリケーションのプロファイルを作成します。これにより、アプリケーションを再コンパイルせずにプロファイリングできますが、**Valgrind** は特に非常に詳細な実行を実行する場合など、他のプロファイラーよりも大幅に遅くなります。したがって、これは時間固有の問題のデバッグや、カーネルスペースのデバッグには適していません。

Red Hat Developer Toolset には **Valgrind 3.19.0** が同梱されています。このバージョンは、以前のリリースの Red Hat Developer Toolset に含まれるバージョンよりも新しいもので、バグ修正および機能拡張が追加されています

表13.1 Red Hat Developer Toolset の Valgrind で配布されるツール

名前	説明
Memcheck	システムコールを傍受し、読み書き操作をすべてチェックして、メモリー管理の問題を検出します。
Cachegrind	レベル1命令キャッシュ (I1)、レベル1データキャッシュ (D1)、および統一レベル2キャッシュ (L2) をシミュレートして、キャッシュミスのソースを特定します。
Callgrind	関数呼び出し履歴を表す呼び出しグラフを生成します。
Helgrind	POSIX スレッドのプリミティブを使用するマルチスレッド C、C++、および Fortran プログラムで同期エラーを検出します。
DRD	POSIX スレッドのプリミティブまたは、これらの POSIX スレッドのプリミティブ上に構築された他のスレッド概念を使用するマルチスレッド C および C++ プログラムのエラーを検出します。
Massif	ヒープおよびスタックの使用状況を監視します。

13.1. VALGRIND のインストール

Red Hat Developer Toolset では、**devtoolset-12-valgrind** パッケージにより **Valgrind** が提供され、**devtoolset-12-perftools** で自動的にインストールされます。

Red Hat Developer Toolset および関連パッケージをシステムにインストールする方法は、[「Red Hat Developer Toolset のインストール」](#) を参照してください。



注記

GNU デバッガー と **Valgrind** を組み合わせて使用する場合は、Red Hat Developer Toolset に含まれる **GDB** のバージョンを使用して、すべての機能が完全にサポートされることを確認することが推奨されます。

13.2. VALGRIND の使用

プロファイルするプログラムで Valgrind ツールを実行するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'valgrind --tool=tool program argument...'
```

Valgrind で配布されるツールの一覧は、[表13.1「Red Hat Developer Toolset の Valgrind で配布されるツール」](#)を参照してください。--tool コマンドラインオプションの引数は小文字で指定する必要があります。このオプションを省略すると、Valgrind はデフォルトで Memcheck を使用します。たとえば、プログラムで Cachegrind を実行して、キャッシュミスのソースを特定するには、以下を実行します。

```
$ scl enable devtoolset-12 'valgrind --tool=cachegrind program argument...'
```

この scl ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、Red Hat Developer Toolset Valgrind でシェルセッションをデフォルトとして実行できます。

```
$ scl enable devtoolset-12 'bash'
```

注記

いずれかの時点で使用している Valgrind のバージョンを確認するには、次のコマンドを実行します。

```
$ which valgrind
```

Red Hat Developer Toolset の valgrind 実行可能なパスは、/opt で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset Valgrind と一致することを確認することができます。

```
$ valgrind --version
```

13.3. 関連資料

Valgrind およびその機能の詳細は、以下に挙げるリソースを参照してください。

インストールされているドキュメント

- **valgrind(1): valgrind ユーティリティーの man ページ**では、Valgrind の使用方法の詳細情報が提供されています。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man valgrind'
```

- **Valgrind ドキュメント**: Valgrind の HTML ドキュメントは、`/opt/rh/devtoolset-12/root/usr/share/doc/devtoolset-12-valgrind-3.19.0/html/index.html` にあります。

オンラインドキュメント

- [Red Hat Enterprise Linux 7 Developer Guide](#) : Red Hat Enterprise Linux 7 の **Developer Guide** では、Valgrind およびその Eclipse プラグインについての詳細情報が記載されています。

- [Red Hat Enterprise Linux 7 パフォーマンスチューニングガイド](#) : Red Hat Enterprise Linux 7 の [パフォーマンスチューニングガイド](#) では、アプリケーションのプロファイルに **Valgrind** の使用に関する詳細情報が記載されています。

以下も併せて参照してください。

- [1章 Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。
- [11章 memstomp](#): **memstomp** ユーティリティーを使用する手順で、さまざまな標準で使用できないメモリー領域が重複しているライブラリー関数への呼び出しを特定します。
- [12章 SystemTap](#): **SystemTap** ツールの概要と、そのツールを使用して稼働中のシステムの動作を監視する方法を紹介します。
- [14章 OProfile](#): コードのどのセクションが最も多くの CPU 時間および理由を消費するかを決定するために **OProfile** ツールを使用する命令。
- [15章 Dyninst](#): **Dyninst** ライブラリーを使用してユーザー空間の実行ファイルをインストルメント化するための手順

第14章 OPROFILE

OProfile はオーバーヘッドが低く、システム全体のプロファイラーで、プロセッサ上のパフォーマンス監視ハードウェアを使用して、メモリーの参照時、レベル2 キャッシュ (L2) 要求の数、ハードウェア割り込みの受信回数など、システム上のカーネルおよび実行可能ファイルに関する情報を取得します。これは、設定ユーティリティー、データ収集用のデーモン、および人間が判読可能なフォームに変換するのに使用できるツールで設定されます。OProfile の Red Hat Developer Toolset バージョンで配布されるツールの完全リストは、[表14.1 「Red Hat Developer Toolset の OProfile と配布されるツール」](#) を参照してください。

OProfile は、すべての n 番目のイベントの詳細を記録して、インストールメンテーションを追加せずにアプリケーションをプロファイルします。これにより、Valgrind よりも少ないリソースを消費できますが、そのサンプルの正確性も低くなります。Valgrind とは異なり、単一プロセスとユーザー空間内の子データのデータのみを収集するため、OProfile はユーザー空間およびカーネル空間プロセスの両方でシステム全体のデータを収集するのに適しており、実行する **root** 権限が必要になります。

Red Hat Developer Toolset には OProfile 1.4.0 が同梱されています。

表14.1 Red Hat Developer Toolset の OProfile と配布されるツール

名前	説明
opperf	Linux Performance Events サブシステムを使用して、単一プロセスまたはシステム全体のサンプルを記録します。
opannotate	プロファイリングデータからアノテーション付きのソースファイルまたはアセンブリーの一覧を生成します。
oparchive	実行ファイル、デバッグファイル、およびサンプルファイルを含むディレクトリーを生成します。
opgprof	gprof と互換性のある形式でプロファイリングセッションの概要を生成します。
ophelp	利用できるイベントの一覧を表示します。
opimport	サンプルデータベースファイルをネイティブバイナリー形式からネイティブ形式に変換します。
opjitconv	just-in-time (JIT) ダンプファイルを Executable および Linkable Format (ELF) に変換します。
opreport	プロファイリングセッションのイメージおよびシンボルの概要を生成します。
ocount	監視対象のコマンドの実行中に特定のイベントが発生する回数をカウントするための新しいツール。

14.1. OPROFILE のインストール

Red Hat Developer Toolset では、**devtoolset-12-oprofile** パッケージにより OProfile が提供され、「[Red Hat Developer Toolset のインストール](#)」で説明されているように **devtoolset-12-perftools** で自動的にインストールされます。

14.2. OPROFILE の使用

OProfile で配布されるツールのいずれかを実行するには、以下を実行します。

```
# scl enable devtoolset-12 'tool option...'
```

OProfile で配布されるツールの一覧は、[表14.1「Red Hat Developer Toolset の OProfile と配布されるツール」](#)を参照してください。たとえば、**ophelp** コマンドを使用して、XML 形式で利用可能なイベントを一覧表示します。

```
$ scl enable devtoolset-12 'ophelp -X'
```

この **scl** ユーティリティーを使用してコマンドを実行すると、これを Red Hat Enterprise Linux システムに優先して使用する Red Hat Developer Toolset バイナリーで実行することができることに注意してください。これにより、Red Hat Developer Toolset **OProfile** でシェルセッションをデフォルトとして実行できます。

```
$ scl enable devtoolset-12 'bash'
```

注記

.30 任意の時点で使用している OProfile のバージョンを確認するには、次のコマンドを実行します。

```
$ which operf
```

Red Hat Developer Toolset の **operf** 実行可能なパスは、**/opt** で始まります。以下のコマンドを使用して、バージョン番号が Red Hat Developer Toolset **OProfile** と一致することを確認することができます。

```
# operf --version
```

14.3. 関連資料

OProfile とその機能の詳細は、以下に挙げるリソースを参照してください。

インストールされているドキュメント

- **oprofile(1)**: **oprofile** という名前の man ページでは、OProfile および利用可能なツールの概要が記載されています。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
$ scl enable devtoolset-12 'man oprofile'
```

- **opannotate(1)**、**oparchive(1)**、**operf(1)**、**opgprof(1)**、**ophelp(1)** **opimport(1)**、**opreport(1)**: OProfile で配布される各種ツールの man ページは、それぞれの使用方法に関する詳細情報を提供します。Red Hat Developer Toolset に含まれるバージョンの man ページを表示するには、次のコマンドを実行します。

```
scl enable devtoolset-12 'man tool'
```

オンラインドキュメント

- [Red Hat Enterprise Linux 7 Developer Guide](#) : Red Hat Enterprise Linux 7 の **Developer Guide** では、**OProfile** に関する詳細な情報を提供します。
- [Red Hat Enterprise Linux 7 システム管理者のガイド](#) : Red Hat Enterprise Linux 7 の**システム管理者ガイド** では、この **operf** ツールの使用方法を説明しています。

以下も併せて参照してください。

- [1章Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。
- [12章SystemTap](#): **SystemTap** の概要と、そのツールを使用して稼働中のシステムの動作を監視する方法を紹介します。
- [13章Valgrind](#): **Valgrind** ツールを使用してアプリケーションのプロファイルを作成して、初期化されていないメモリの使用、メモリの割り当ておよびメモリの解放、システムコールでの不適切な引数の使用など、メモリエラーやメモリ管理問題を検出します。
- [15章Dyninst](#): **Dyninst** ライブラリーを使用してユーザー空間の実行ファイルをインストルメント化するための手順

第15章 DYNINST

Dyninst ライブラリーは、実行時にユーザー空間の実行ファイルをインストルメント化し、操作するための **アプリケーションプログラミングインターフェイス (API)** を提供します。実行中のプログラムへのコードを挿入したり、特定のサブルーチン呼び出しを変更したり、プログラムから削除したりするために使用できます。これは、有用なデバッグおよびパフォーマンス監視ツールとして機能します。**Dyninst API** は、一般的に **SystemTap** とともに使用され、**root** ユーザー以外の遊座一空間実行可能ファイルをインストルメント化できます。

Red Hat Developer Toolset には **Dyninst 12.1.0** が同梱されています。

15.1. DYNINST のインストール

Red Hat Developer Toolset では、**devtoolset-12-dyninst** パッケージにより **Dyninst** ライブラリーが提供され、「[Red Hat Developer Toolset のインストール](#)」で説明されているように **devtoolset-12-perftools** で自動的にインストールされます。さらに、**devtoolset-12-toolchain** パッケージで提供される **GNU コンパイラーコレクション** をインストールすることも推奨されます。

バイナリー用のカスタムインストルメンテーションを作成する場合は、関連するヘッダーファイルをインストールします。

```
# yum install devtoolset-12-dyninst-devel
```

このライブラリーの API ドキュメントをインストールすることもできます。

```
# yum install devtoolset-12-dyninst-doc
```

devtoolset-12-dyninst-doc パッケージに含まれるドキュメントの完全なリストは、「[関連資料](#)」を参照してください。システムにオプションパッケージをインストールする方法の詳細は、「[Red Hat Developer Toolset のインストール](#)」を参照してください。

15.2. DYNINST の使用

15.2.1. SystemTap での Dyninst の使用

SystemTap とともに **Dyninst** を使用して **root** ユーザー以外がユーザー空間の実行ファイルをインストルメント化できるようにするには、**--dyninst** (または **--runtime=dyninst**) コマンドラインオプション **stap** を指定してコマンドを実行します。これは、**SystemTap** スクリプト **stap** を、**Dyninst** ライブラリーを使用する C コードに変換し、この C コードを共有ライブラリーにコンパイルしてから、共有ライブラリーを読み込み、スクリプトを実行します。このように実行する場合は、**stap** コマンドに **-c** または **-x** コマンドラインオプションも指定する必要があることに注意してください。

Dyninst ランタイムを使用して実行ファイルをインストルメント化するには、以下を実行します。

```
$ scl enable devtoolset-12 "stap --dyninst -c 'command' option... argument..."
```

同様に、**Dyninst** ランタイムを使用してユーザーのプロセスをインストルメント化するには、以下を実行します。

```
$ scl enable devtoolset-12 "stap --dyninst -x process_id option... argument..."
```

SystemTap の Red Hat Developer Toolset バージョンの詳細は、[12章SystemTap](#) を参照してください。SystemTap とその使用方法の概要は、Red Hat Enterprise Linux 7 の [SystemTap ビギナーズガイド](#) を参照してください。

例15.1 SystemTap での Dyninst の使用

以下の内容を **exercise.C** 含むという名前のソースファイルについて考えてみましょう。

```
#include <stdio.h>

void print_iteration(int value) {
    printf("Iteration number %d\n", value);
}

int main(int argc, char **argv) {
    int i;
    printf("Enter the starting number: ");
    scanf("%d", &i);
    for(; i>0; --i)
        print_iteration(i);
    return 0;
}
```

このプログラムは、開始番号の入力をユーザー要求し、1までのカウントダウンを行います。これは、標準出力に番号を出力するために各反復に対して **print_iteration()** 関数を呼び出します。Red Hat Developer Toolset の **g++** コンパイラーを使用して、このプログラムをコマンドラインでコンパイルします。

```
$ scl enable devtoolset-12 'g++ -g -o exercise exercise.C'
```

ここ **count.stp** で、以下の内容を含む別のソースファイルを考慮します。

```
#!/usr/bin/stap

global count = 0

probe process.function("print_iteration") {
    count++
}

probe end {
    printf("Function executed %d times.\n", count)
}
```

この SystemTap スクリプトは、プロセスの実行中に **print_iteration()** 関数が呼び出された回数を出力します。このスクリプトは、**exercise** バイナリーファイルで実行します。

```
$ scl enable devtoolset-12 "stap --dyninst -c './exercise' count.stp"
Enter the starting number: 5
Iteration number 5
Iteration number 4
Iteration number 3
Iteration number 2
Iteration number 1
Function executed 5 times.
```

15.2.2. Dyninst をスタンドアロンライブラリーとして使用

Dyninst ライブラリーをアプリケーションの一部として使用する前に、`DYNINSTAPI_RT_LIB` 環境変数の値をランタイムライブラリーファイルへのパスに設定します。

```
$ export DYNINSTAPI_RT_LIB=/opt/rh/devtoolset-12/root/usr/lib64/dyninst/libdyninstAPI_RT.so
```

これにより、現在のシェルセッションで `DYNINSTAPI_RT_LIB` 環境変数が設定されます。

例15.2 「Dyninst をスタンドアロンアプリケーションとして使用する」 は、ユーザー空間プロセスの実行を監視するプログラムを作成およびビルドする方法を示しています。Dyninst の使用方法に関する詳細は、「[関連資料](#)」に記載されているリソースを参照してください。

例15.2 Dyninst をスタンドアロンアプリケーションとして使用する

例15.1 「[SystemTap での Dyninst の使用](#)」 の `exercise.C` ソースファイルを考慮します。このプログラムにより、ユーザーは開始番号を入力し、1までカウントして各反復の `print_iteration()` 関数をカウントし、標準出力に番号を出力します。

ここ `cout.C` で、以下の内容を含む別のソースファイルを考慮します。

```
#include <stdio.h>
#include <fcntl.h>
#include "BPatch.h"
#include "BPatch_process.h"
#include "BPatch_function.h"
#include "BPatch_Vector.h"
#include "BPatch_thread.h"
#include "BPatch_point.h"

void usage() {
    fprintf(stderr, "Usage: count <process_id> <function>\n");
}

// Global information for counter
BPatch_variableExpr *counter = NULL;

void createCounter(BPatch_process *app, BPatch_image *applImage) {
    int zero = 0;
    counter = app->malloc(*applImage->findType("int"));
    counter->writeValue(&zero);
}

bool interceptfunc(BPatch_process *app,
                  BPatch_image *applImage,
                  char *funcName) {
    BPatch_Vector<BPatch_function*> func;
    applImage->findFunction(funcName, func);
    if(func.size() == 0) {
        fprintf(stderr, "Unable to find function to instrument()\n");
        exit (-1);
    }
}
```

```

BPatch_Vector<BPatch_snippet *> incCount;
BPatch_Vector<BPatch_point *> *points;
points = func[0]->findPoint(BPatch_entry);
if ((*points).size() == 0) {
    exit (-1);
}

BPatch_arithExpr counterPlusOne(BPatch_plus, *counter, BPatch_constExpr(1));
BPatch_arithExpr addCounter(BPatch_assign, *counter, counterPlusOne);

return app->insertSnippet(addCounter, *points);
}

void printCount(BPatch_thread *thread, BPatch_exitType) {
    int val = 0;
    counter->readValue(&val, sizeof(int));
    fprintf(stderr, "Function executed %d times.\n", val);
}

int main(int argc, char *argv[]) {
    int pid;
    BPatch bpatch;
    if (argc != 3) {
        usage();
        exit(1);
    }
    pid = atoi(argv[1]);
    BPatch_process *app = bpatch.processAttach(NULL, pid);
    if (!app) exit (-1);
    BPatch_image *applImage = app->getImage();
    createCounter(app, applImage);
    fprintf(stderr, "Finding function %s(): ", argv[2]);
    BPatch_Vector<BPatch_function*> countFuncs;
    fprintf(stderr, "OK\nInstrumenting function %s(): ", argv[2]);
    interceptfunc(app, applImage, argv[2]);
    bpatch.registerExitCallback(printCount);
    fprintf(stderr, "OK\nWaiting for process %d to exit...\n", pid);
    app->continueExecution();
    while (!app->isTerminated())
        bpatch.waitForStatusChange();
    return 0;
}

```

Dyninst ライブラリーのデストラクターが呼び出される前に、クライアントアプリケーションがすべての **Bpatch** オブジェクトを破棄することが期待されることに注意してください。それ以外の場合は、セグメンテーションフォールトでミューターが突然終了する可能性があります。この問題を回避するには、**main()** 関数で mutator の **BPatch** オブジェクトをローカル変数として設定します。または、グローバル変数として **BPatch** 使用する必要がある場合は、ミューターの終了前にすべての変更プロセスを手作業でデタッチします。

このプログラムは、プロセス ID および関数名をコマンドライン引数として受け入れ、プロセスの実行中に呼び出された関数の合計回数を出力します。これらの2つのファイル **Makefile** を構築するには、以下を使用します。

```

DTS    = /opt/rh/devtoolset-12/root
CXXFLAGS = -g -I$(DTS)/usr/include/dyninst

```

```

LBITS := $(shell getconf LONG_BIT)

ifeq ($(LBITS),64)
  DYNINSTLIBS = $(DTS)/usr/lib64/dyninst
else
  DYNINSTLIBS = $(DTS)/usr/lib/dyninst
endif

.PHONY: all
all: count exercise

count: count.C
g++ $(CXXFLAGS) count.C -I /usr/include/dyninst -c
g++ $(CXXFLAGS) count.o -L $(DYNINSTLIBS) -ldyninstAPI -o count

exercise: exercise.C
g++ $(CXXFLAGS) exercise.C -o exercise

.PHONY: clean
clean:
rm -rf *~ *.o count exercise

```

Red Hat Developer Toolset の **g++** コンパイラーを使用してコマンドラインで2つのプログラムをコンパイルするには、**make** ユーティリティーを実行します。

```

$ scl enable devtoolset-12 make
g++ -g -I/opt/rh/devtoolset-12/root/usr/include/dyninst count.C -c
g++ -g -I/opt/rh/devtoolset-12/root/usr/include/dyninst count.o -L /opt/rh/devtoolset-12/root/usr/lib64/dyninst -ldyninstAPI -o count
g++ -g -I/opt/rh/devtoolset-12/root/usr/include/dyninst exercise.C -o exercise

```

これにより、**exercise** と **count** という名前のバイナリーファイルが、現在の作業ディレクトリーに作成されます。

あるシェルセッションで、以下のように **exercise** バイナリーファイルを実行し、開始番号の入力を求めるプロンプトを待ちます。

```

$ ./exercise
Enter the starting number:

```

この番号は入力しないでください。代わりに別のシェルセッションを開始し、プロンプトに以下のコマンドを入力します。 **DYNINSTAPI_RT_LIB** 環境変数を設定して、**count** バイナリーファイルを実行します。

```

$ export DYNINSTAPI_RT_LIB=/opt/rh/devtoolset-12/root/usr/lib64/dyninst/libdyninstAPI_RT.so
$ ./count `pidof exercise` print_iteration
Finding function print_iteration(): OK
Instrumenting function print_iteration(): OK
Waiting for process 8607 to exit...

```

最初のシェルセッションに切り替え、**exercise** プログラムで要求される開始番号を入力します。以下に例を示します。

```

Enter the starting number: 5

```

```
Iteration number 5
Iteration number 4
Iteration number 3
Iteration number 2
Iteration number 1
```

exercise プログラムが終了すると、**count** プログラムにより、**print_iteration()** 関数の実行回数が表示されます。

```
Function executed 5 times.
```

15.3. 関連資料

Dyninst およびその機能の詳細は、以下の資料を参照してください。

インストールされているドキュメント

devtoolset-12-dyninst-doc パッケージは、以下のドキュメントを **/opt/rh/devtoolset-12/root/usr/share/doc/devtoolset-12-dyninst-doc-12.1.0/** ディレクトリーにインストールします。

- **Dyninst Programmer's Guide API** の Dyninst の詳細な説明は、**API** ファイルに保存されます。
- **DynC API Programmer's Guide** DynC API の紹介は **dynC_API.pdf** ファイルに保存されています。
- **ParseAPI Programmer's Guide**: ParseAPI の紹介は、**ParseAPI.pdf** ファイルに保存されます。
- **PatchAPI Programmer's Guide**: DynC API の紹介は **PatchAPI.pdf** ファイルに保存されています。
- **ProcControlAPI Programmer's Guide** ProcControlAPI に関する詳細な説明は、**ProcControlAPI.pdf** ファイルに保存されています。
- **StackwalkerAPI Programmer's Guide**: StackwalkerAPI に関する詳細な説明は、**stackwalker.pdf** ファイルに保存されています。
- **SymtabAPI Programmer's Guide**: SymtabAPI の紹介は、**SymtabAPI.pdf** ファイルに保存されています。
- **InstructionAPI Reference Manual**: InstructionAPI に関する詳細な説明は、**InstructionAPI.pdf** ファイルに保存されています。

システムにこのパッケージをインストールする方法は、「[Dyninst のインストール](#)」を参照してください。

オンラインドキュメント

- [Dyninst ホームページ](#): プロジェクトのホームページでは、追加のドキュメントや関連文書へのリンクが記載されています。
- [Red Hat Enterprise Linux 7 SystemTap ビギナーズガイド](#) : Red Hat Enterprise Linux 7 の **SystemTap ビギナーズガイド** では、SystemTap とその使用方法を紹介します。
- [Red Hat Enterprise Linux 7 SystemTap Tapset Reference](#) : Red Hat Enterprise Linux 7 の **SystemTap Tapset Reference** は、SystemTap に関する詳細を提供します。

以下も併せて参照してください。

- [1章Red Hat Developer Toolset](#): Red Hat Developer Toolset の概要およびそのシステムへのインストール方法の詳細。
- [12章SystemTap](#): **SystemTap** の概要と、そのツールを使用して稼働中のシステムの動作を監視する方法を紹介します。
- [13章Valgrind](#): **Valgrind** ツールを使用してアプリケーションのプロファイルを作成して、初期化されていないメモリの使用、メモリの割り当ておよびメモリの解放、システムコールでの不適切な引数の使用など、メモリエラーやメモリ管理問題を検出します。
- [14章OProfile](#): コードのどのセクションが最も多くの CPU 時間および理由を消費するかを決定するために **OProfile** ツールを使用する命令。

パート V. コンパイラーツールセット

第16章 コンパイラーツールセットのドキュメント

3つのコンパイラーツールセットの説明。

- LLVM Toolset
- Go Toolset
- Rust Toolset

[Red Hat Developer Tools](#) の下にある別のドキュメントセットに移動しました。

パート VI. ヘルプの取得

第17章 RED HAT 製品ドキュメントへのアクセス

<https://access.redhat.com/site/documentation/> の Red Hat Product Documentation は、中心的な情報源です。現在、これは現在 23 言語に翻訳されています。各製品では、HTML、PDF、および EPUB 形式のインストール、ユーザー、およびリファレンスガイドなど、リリースや技術ノートとはさまざまな要素を提供しています。

直接的または間接的に関連するドキュメントの一覧を以下に示します。

Red Hat Developer Toolset

- [Red Hat Developer Toolset 12.1 Release Notes](#) : Red Hat Developer Toolset 12.1 の **リリースノート** には、詳細情報が含まれています。
- [Using Red Hat Software Collections Container Images: Using Red Hat Software Collections Container Images](#) では、Red Hat Developer Toolset コンテナイメージを含む、Red Hat Software Collections に同梱されているコンテナイメージを取得、設定、および使用する手順を説明しています。
- [Red Hat Software Collections Packaging Guide : Software Collections Packaging Guide](#) では、Software Collections の概念を説明し、それらを作成、構築、および拡張する方法を文書化しています。

Red Hat Enterprise Linux

- [Red Hat Enterprise Linux 7 開発者ガイド](#) : Red Hat Enterprise Linux 7 **開発者ガイド** は、ライブラリーおよびランタイムサポート、コンパイルおよびビルド、デバッグ、およびプロファイリングに関する詳細情報を提供します。
- [Red Hat Enterprise Linux 7 インストールガイド](#) : Red Hat Enterprise Linux 7 の **インストールガイド** では、システムの取得、インストール、および更新の方法を説明します。
- [Red Hat Enterprise Linux 7 システム管理者のガイド](#) - Red Hat Enterprise Linux 7 の **システム管理者のガイド** では、Red Hat Enterprise Linux 7 のデプロイメント、設定、および管理に関する情報を説明しています。

第18章 グローバルサポートサービスへの連絡

Self-Support サブスクリプションをお持ちでない限り、Red Hat ドキュメント Web サイトとカスタマーポータルの方がご質問への回答がない場合は、**グローバルサポートサービス (GSS)** にご連絡ください。

18.1. 必要な情報の収集

GSS と通信する前に、いくつかの情報を取得する必要があります。

背景情報

GSS を呼び出す前に、以下の背景情報があることを確認してください。

- 製品が実行するハードウェアの種類、製造元、およびモデル
- ソフトウェアバージョン
- 最新のアップグレード
- システムへの最近の変更
- 問題および症状の説明
- 問題に関するメッセージまたは重要な情報



注記

Red Hat のログイン情報を忘れていた場合は、<https://access.redhat.com/site/help/LoginAssistance.html> で復元できます。

診断

Red Hat Enterprise Linux の診断レポートも必要です。このレポートは **sosreport** と呼ばれ、レポートを作成するプログラムは **sos** パッケージで提供されます。**sos** パッケージとそのすべての依存関係をシステムにインストールするには、以下を実行します。

```
# yum install sos
```

レポートを生成するには、次のコマンドを実行します。

```
# sosreport
```

詳細は、ナレッジベースの記事 <https://access.redhat.com/kb/docs/DOC-3593> を参照してください。

アカウントおよび連絡先情報

お客様にヘルプを提供するため、**GSS** は、サポートのニーズに応じた調整を行い、連絡を行うためにアカウント情報が必要です。**GSS** にアクセスするには、以下の点を確認してください。

- Red Hat のカスタマー番号または Red Hat Network (RHN) のログイン名
- 会社名
- 連絡先
- 希望する連絡先方法 (電話またはメール) および連絡先情報 (電話番号またはメールアドレス)

問題の重大度

GSS チームが作業の優先順位を決定できるようにするには、問題の重大度を判断することが重要です。重大度は 4 つあります。

重大度 1 (緊急)

実稼働環境でのソフトウェアの使用に深刻な影響を与える問題。ビジネス運営が停止され、手順の回避策はありません。

重大度 2 (高)

ソフトウェアが機能しているが、実稼働環境が著しく減少している問題。ビジネス操作に大きく影響し、回避策は発生しません。

重大度 3 (中)

ソフトウェアの使用に関して、部分的に、非クリティカルで失われる問題。ビジネスには、小～中程度の影響があります。また、対応策を用いてビジネスを継続できます。

重大度 4 (低)

一般的な使用に関する質問、ドキュメントエラーの報告、または将来の製品改善に関する推奨事項。

問題の重大度を判断する方法は、<https://access.redhat.com/support/policy/severity> を参照してください。

問題の重大度が判断されたら、カスタマーポータルからサービスリクエストを **Connect** オプションまたは <https://access.redhat.com/support/contact/technicalSupport.html> から提出します。サービスリクエストを送信するには、Red Hat ログインの詳細が必要なことに注意してください。

重大度がレベル 1 または 2 の場合は、呼び出し先のサービスリクエストに従います。連絡先情報と営業時間は <https://access.redhat.com/support/contact/technicalSupport.html> を参照してください。

プレミアムサブスクリプションをお持ちの場合は、重大度 1 と 2 ケースで、営業時間後にサポートが利用可能になります。

プレミアムサブスクリプションと標準のサブスクリプションの両方については、<https://access.redhat.com/support/offerings/production/sla.html> に掲載されています。

18.2. 問題のエスカレーション

問題が適切に処理されていない、または適切に処理されていない場合は、エスカレートできます。エスカレーションには 2 つのタイプがあります。

技術的なエスカレーション

問題が適切に解決されていない場合や、それを行うのにより多くのリソースが必要な場合。

管理エスカレーション

問題がより深刻な場合や、より高い優先順位を持つと思われる場合。

コンタクトを含むエスカレーションの詳細

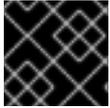
は、https://access.redhat.com/support/policy/mgt_escalation.html から入手できます。

18.3. サービスリクエストの再オープン

終了したサービス要求 (問題の再発生など) に関する詳細情報がある場合

は、https://access.redhat.com/support/policy/mgt_escalation.html の Red Hat カスタマーポータルにアクセスするか、ローカルサポートセンターまでお電話いただくことで、リクエストを再度開くことが

できます。詳細は <https://access.redhat.com/support/contact/technicalSupport.html> を参照してください。



重要

サービスリクエストを再度開くには、元の service-request 番号が必要です。

18.4. 関連資料

詳細は、以下の資料を参照してください。

オンラインドキュメント

- [Getting Started: Getting Started](#) ページは、Red Hat サブスクリプションを購入されたお客様のスタート地点として機能し、ダウンロードを行うための **Red Hat Welcome Kit** および **Quick Guide to Red Hat Support** を提供しています。
- [How can a RHEL Self-Support subscription be used?](#) : Self-Support サブスクリプションサブスクリプションをご利用のお客様向けナレッジベース生地。
- [Red Hat Global Support Services および公開メーリングリスト](#) : 公開 Red Hat メーリングリストに関するよくある質問に回答するナレッジベースの記事です。

付録A バージョン 12.0 での変更点

以下のセクションでは、Red Hat Developer Toolset 12.1 で導入された機能および互換性の変更点について説明しています。リストは完全ではなく、更新されます。

A.1. GCC の変更点

Red Hat Developer Toolset 12.1 には **GCC 12.2.1** が同梱されています。

以下の機能は、Red Hat Developer Toolset の以前のリリース以降に追加または変更されています。

一般的な改善

- ベクトル化は **-O2** で有効化され、元の **-O2 -ftree-vectorize -fvect-cost-model=very-cheap** と同等になりました。

言語固有の改善

OpenMP

- OpenMP 5.0 のサポートが拡張されました。 **close map** 修飾子と **affinity** 句がサポートされるようになりました。
- 新規実装された OpenMP 5.1 機能が追加されました。たとえば、OpenMP ディレクティブを C++11 属性として表現するためのサポート、 **masked** および **scope** コンストラクト、 **nothing** および **error** ディレクティブ、 **proc_bind** 句および **OMP_PROC_BIND** 環境変数での **primary** の使用、 **order** 句に対する **reproducible** および **unconstrained** 修飾子、および C と C++ のみの場合は、 **allocate** 句の **align** と **allocator** 修飾子、およびアトミック拡張。

C ファミリー

- 新しい機能と属性:
 - clang 言語拡張と互換性のある **__builtin_shufflevector** ビルトインのサポート。
 - unavailable** 属性のサポート。
 - 新しい組み込み関数 **__builtin_assoc_barrier** は、浮動小数点式の再関連付けを禁止するために使用されます。
 - clang 言語拡張と互換性のある **__builtin_dynamic_object_size** ビルトインのサポート。
- 新しい警告:
 - Wbidi-chars** は、誤解を招く可能性のある UTF-8 双方向制御文字について警告します。デフォルトは **-Wbidi-chars=unpaired** です。
 - Warray-compare** は、配列型の 2 つのオペランド間の比較について警告します。
- 警告の強化:
 - Wattributes** が拡張されたため、 **-Wno-attributes=ns::attr** または **-Wno-attributes=ns::** を使用して、C++11 および C2X で不明なスコープ属性に関する警告を抑制することができるようになりました。同様に、 **#pragma GCC diagnostic ignored_attributes "vendor::attr"** を使用して同じことを行うことができます。

C

- C2X では、C++ のような桁区切り記号がサポートされています。
- **#elifdef** および **#elifndef** 前処理ディレクティブがサポートされるようになりました。
- **-Wformat** でチェックする **printf** および **scanf** フォーマットは、2 進整数に対して C2X によって指定された **%b** フォーマット、および **printf** に対して C2X によって推奨される **%B** フォーマットをサポートするようになりました。

C++

- 以下の C++23 機能が実装されました。
 - P1938R3: **if consteval** の実装
 - P0849R8: **auto(x)**: 言語の **decay-copy**
 - P2242R3: **constexpr** 関数の非リテラル変数 (およびラベルと goto)
 - P2334R1: 前処理ディレクティブ **elifdef** および **elifndef** のサポート
 - P2360R0: **init-statement** を拡張して **alias-declaration** を許可する
 - P2128R6: マルチディメンション添字演算子
 - DR 2397: 配列へのポインターと参照の自動指定子
- 新しいコマンドラインオプション **-fimplicit-constexpr** は、インライン関数を暗黙的に **constexpr** にするために使用されます。
- 新しいコマンドラインオプション **-ffold-simple-inlines** は、**std::move**、**std::forward**、**std::addressof**、および **std::as_const** などの特定の単純なインライン関数への呼び出しを折りたたむために使用されます。
- 演繹ガイドはクラススコープで宣言できます
- **-Wuninitialized** は、メンバー初期化子リストで初期化されていない変数を使用することについて警告するようになりました。
- 新しい警告:
 - 次の C++ 言語の不一致に対する新しい警告オプション: **-Wc11-extensions**、**-Wc14-extensions**、**-Wc17-extensions**、**-Wc20-extensions**、および **-Wc++23-extensions**。これらはデフォルトで有効になっており、古い C++ 標準方言を使用するコードでの新しい C++ 構造の発生に関する既存の警告を制御するために使用されます。
 - 新しい警告 **-Wmissing-requires** は、欠落している **requires** について警告します。
 - 既存の **std::is_constant_evaluated** 警告は、より多くの場合に警告するように拡張されています。

Red Hat Developer Toolset 12.1 の新しい C++ 機能の詳細については、[Red Hat Developers の記事](#) を参照してください。

ランタイムライブラリー libstdC++

- 以下の実験的な C++20 サポート機能が改善されました。
 - **std::vector**、**std::basic_string**、**std::optional**、および **std::variant** を **constexpr** 関数で使えようになりました。

使用してごみ処理を行いました。

- デフォルトの初期化を伴う配列の **std::make_shared**、および **std::atomic<std::shared_ptr<T>>**。
- レイアウトの互換性とポインタの相互転換可能性の特性。
- 以下の実験的な C++23 サポート機能が改善されました。
 - **std::optional** のモナド操作。
 - **std::move_only_function**
 - **<spanstream>**
 - **std::basic_string::resize_and_overwrite**
 - **std::unique_ptr** を **constexpr** 関数で使用できるようになりました。
 - **<stacktrace>** (デフォルトではビルドされません。追加のライブラリーへのリンクが必要です)。
 - **<stdatomic.h>**
 - **std::invoke_r**
 - **constexpr std::type_info::operator==**

アーキテクチャー固有の改善

64 ビット ARM アーキテクチャー

- **-mcpu** および **-mtune** オプションにより、多数の新しい CPU がサポートされます。
- 2020 Arm アーキテクチャー拡張からのアクセラレーターメモリーの 64 バイトアトミックロード/ストア組み込み関数は、**+ls64** オプション拡張によってサポートされます。
- 初期コード生成サポートは、**memcpy**、**memmove**、および **memset** 標準関数を高速化するハードウェア命令でサポートされています。**+mopsoption** 拡張機能を使用してコンパイルするときに、これらの命令を生成します。
- **arm_neon.h** ヘッダーからアクセスできる Arm C Language Extensions (ACLE) Advanced Single Instruction Multiple Data (SIMD) 組み込み関数が大幅に再実装され、以前の GCC バージョンよりもパフォーマンスの高いコードが生成されます。

AMD アーキテクチャーおよび Intel 64 ビットアーキテクチャー

- Intel AVX512-FP16 の新しい ISA 拡張サポートが追加されました。AVX512FP16 組み込み関数は、**-mavx512fp16** コンパイラスイッチを介して利用できます。
- C と C++ の両方で、**SSE2** が有効になっているシステムで **_Float16 type** がサポートされません。
- **-mharden-sls=[none|all|return|indirect-jmp]** を介して、関数リターンと間接ジャンプの Straight Line Speculation (SLS) に対する緩和がサポートされます。

A.2. ELFUTILS の変更点

Red Hat Developer Toolset 12.1 には **elfutils 0.187** が同梱されています。

以下の機能は、Red Hat Developer Toolset の以前のリリース以降に追加または変更されています。

- **libdw** ライブラリーでの NVIDIA Cuda ラインマップ拡張機能のサポート。
- **eu-addr2line** 実行可能ファイルは、デフォルトで **-A**、**--absolute** オプションになり、完全なコンパイルディレクトリーを含むファイル名が表示されます。新しい **--relative** オプションを使用して、古い動作に戻します。
- **eu-readelf** および **eu-elflint** 実行可能ファイルは、FDO パッケージ化メタデータ ELF ノートを認識するようになりました。
- **debuginfod** 実行可能ファイルは、以下をサポートするようになりました。
 - インメモリーデータベース用の新しい **-d :memory:** オプション。
 - 古いファイルのグルーミングに **-I** または **-X** 正規表現を使用する新しい **-r** オプション。
 - 接続スレッドプーリング用の新しい **-C** オプション。
 - 読み取り専用データベースからサービスを提供するための新しい **--passive** モード。
- **debuginfod-client** ディレクトリーが、**DEBUGINFOD_MAXSIZE**、**DEBUGINFOD_MAXTIME**、および **DEBUGINFOD_RETRY** の環境変数の設定をサポートするようになりました。ネガティブキャッシュファイルのサイズがゼロになり、ファイルのスキャンやバックアップの作成時に、許可のないファイルよりも適切に機能します。

A.3. GDB の変更点

Red Hat Developer Toolset 12.1 には **GDB 11.2** が同梱されています。

以下の機能は、Red Hat Developer Toolset の以前のリリース以降に追加または変更されています。

新機能

- Aarch64 MTE の新しいサポート。"memory-tag" プレフィックスが付いた新しいコマンドを参照してください。
- **-break-insert** および **-dprintf-insert** の **--qualified** オプション。このオプションは、全スコープで検索するのではなく、ユーザーのイベントの場所と完全に一致するものを探します。たとえば、**break --qualified foo** は、グローバルスコープで **foo** という名前のシンボルを探します。**--qualified** を指定しないと、GDB はすべてのスコープでその名前のシンボルを検索します。
- **--force-condition**: 現在無効になっていても、指定された条件が定義されます。
- **-break-condition --force**: MI コマンドと同様。
- **-file-list-exec-source-files** は、出力を制限するオプションの **REGEXP** を受け入れます。
- **.gdbinit** 検索パスには config ディレクトリーが含まれます。順序は次のとおりです。
 - a. \$XDG_CONFIG_HOME/gdb/gdbinit
 - b. \$HOME/.config/gdb/gdbinit
 - c. \$HOME/.gdbinit.

- `~/.config/gdb/gdbearlyinit` または `~/.gdbearlyinit` のサポート。
- `-eix` および `-eieux` 早期初期化ファイルオプション。

ターミナルユーザーインターフェイス (TUI)

- ターミナルユーザーインターフェイス (TUI) ウィンドウ内でのマウス操作がサポートされません。
- フォーカスされたウィンドウで機能しないキーの組み合わせが GDB に渡されるようになりました。

新規コマンドおよび改善されたコマンド

- 新しい `show print memory-tag-violations` コマンド。
- 新しい `set print memory-tag-violations` コマンド。
- 新しい `memory-tag show-logical-tag` コマンド。
- 新しい `memory-tag with-logical-tag` コマンド。
- 新しい `memory-tag show-allocation-tag` コマンド。
- 新しい `memory-tag check` コマンド。
- 新しい `show startup-quietly` および `set startup-quietly` コマンド: GDB スクリプトで `-q` または `-quiet` を指定する方法。早期初期化ファイルでのみ有効です。
- 新しい `show print type hex` および `set print type hex` コマンド: 構造体メンバーのサイズまたはオフセットを 10 進数ではなく 16 進数で出力するように GDB に指示します。
- 新しい `show python ignore-environment` コマンドと `set python ignore-environment` コマンド: 有効にすると、GDB の Python インタープリターは、Python 実行可能ファイルに `-E` を渡すのと同じように、Python 環境変数を無視します。早期初期化ファイルでのみ有効です。
- 新しい `show python dont-write-bytecode` コマンドと `set python dont-write-bytecode` コマンド: 'off' の場合、これらのコマンドは、Python 実行可能ファイルに `-B` を渡すのと同様に、GDB の Python インタープリターがインポートされたモジュールのバイトコードコンパイル済みオブジェクトを書き込むことを抑制します。早期初期化ファイルでのみ有効です。
- `break LOCATION if CONDITION`: `CONDITION` が無効な場合、GDB はブレークポイントの設定を拒否します。 `-force-condition` オプションはこれをオーバーライドします。
- `CONDITION -force N COND`: 前のコマンドと同じ。
- `Inferior ID`: `ID` が省略されている場合、このコマンドは現在の下位関連情報を出力します。それ以外の場合は、変更されません。
- `ptype[/FLAGS] TYPE | EXPRESSION`: `'/x'` フラグを使用して、struct メンバーのサイズとオフセットを出力するときに 16 進法での表記を使用します。 `'/d'` フラグを使用して同じことを行いますが、10 進法を使用します。
- `info sources`: 出力が再構築されました。

Python API

- 下位オブジェクトには、読み取り専用の `connection_num` 属性が含まれています。

- 新しい `gdb.Frame.level()` メソッド。
- 新しい `gdb.PendingFrame.level()` メソッド。
- `gdb.Stop` の代わりに `gdb.BreakpointEvent` が出力されます。

A.4. STRACE の変更点

Red Hat Developer Toolset 12.1 には `strace 5.18` が同梱されています。

以下の機能は、Red Hat Developer Toolset の以前のリリース以降に追加または変更されています。

動作の変更

- 変更された `%process` クラスには、プロセスのライフサイクル (作成、実行、終了) に関連するシステムコールが含まれています。
 - 新しい呼び出し: `kill`、`tkill`、`tgkill`、`pidfd_send_signal`、および `rt_sigqueueinfo`
 - 削除された呼び出し: `arch_prctl` および `unshare`

改良

- 新しい `--secontext=mismatch` オプションは、SELinux コンテキストで不一致を検出します。
- 出力文字列のエスケープシーケンスで 8 進数文字の代わりに 16 進数を使用するための新しい `--strings-in-hex=non-ascii-chars` オプション。
- 新しい `--decode-pids=comm` オプションとそのエイリアス `-Y` は、PID のコマンド名を出力します。
- `--pidns-translation` オプションの新しい `--decode-pids=pidns` エイリアス。
- `--decode-fds=path` オプションを有効にして `AT_FDCWD` 定数を使用した場合の現在の作業ディレクトリーの出力が実装されました。
- 新しい `--tips` オプションは、トレースセッションの最後に `strace` のヒント、トリック、微調整を出力します。
- `all@pers` および `%class@pers` をサポートするための、システムコール仕様式のパーソナリティー指定構文が拡張されました。
- `syscall` 仕様式の無効なシステムコール番号の拒否を強化しました。
- 次のシステムコールのデコードが実装されています:
`futex_waitv`、`memfd_secret`、`process_mrelease`、`quotactl_fd`、および `set_mempolicy_home_node`。
- 次のシステムコールのデコードが強化されました:
`bpf`、`io_uring_register`、`prctl`、`seccomp`、および `times`。
- 次の `ioctl` コマンドのデコードが実装されています:
`COUNTER_*`、`HDIO_*`、`KD*`、`LIRC*`、`RTC_PARAM_GET`、`RTC_PARAM_SET`、および `SECCOMP_*`。
- `BTRFS_IOC_FS_INFO ioctl` コマンドのデコードが強化されました。

- 次の netlink メッセージのデコードが実装されています:
RTM_NEWCACHEREPORT、**RTM_{NEW,DEL,GET}NEXTHOP**、および
RTM_{NEW,GET}STATS NETLINK_ROUTE。
- 次の netlink 属性のデコードが強化されました:
IFLA_*、**MDBA_ROUTER_PATTR_***、**TCA_ACT_IN_HW_COUNT**、***_INTVL**、および
***_TIMER**。
- **siginfo_t.si_pkey** フィールドのデコードが実装されています。
- 関連する **AUDIT_ARCH_*** 値 (ptrace **PTRACE_GET_SYSCALL_INFO** 要求、**SIGSYS siginfo_t**) が存在する場所でのシステムコール名の出力が改善されました。
- 次のソケットアドレスのデコードが実装されています:
AF_ALG、**AF_IEEE802154**、**AF_MCTP**、**AF_NFC**、**AF_QIPCRTR**、**AF_RRPC**、**AF_VSOCK**、
および **AF_XDP**。
- **AF_IPX** および **AF_NETLINK** ソケットアドレスのデコードが強化されました。
- 次の定数の一覧が更新されました:
AF_*、**ARPHRD_***、**AUDIT_***、**BPF_***、**BR_***、**BTRFS_***、**DEVCONF_***、**DM_***、**ETH_P_***、
FAN_*、**FAN_REPORT_***、**IFA_***、**IFLA_***、**IORING_***、**IOSQE_***、**IPV4_DEVCONF_***、**KEY_***、
KVM_*、**MADV_***、**MODULE_INIT_***、**MOUNT_ATTR_***、**MOVE_MOUNT_***、**MPOL_***、
NDA_*、**PACKET_***、**RTM_***、**SCTP_***、**SO_***、**TCA_ACT_***、**UFFD_***、**V4L2_***、**XFRM_M**
SG_*、および ***_MAGIC**。
- Linux 5.18 カーネルに合わせて **ioctl** コマンドのリストが更新されました。

バグ修正

- **bpf(BPF_*_GET_NEXT_ID)** 呼び出しの終了時の **union bpf_attr.next_id** の更新された値の出力が修正されました。
- **struct bpf_prog_info.map_ids** 配列の出力が修正されました。
- **--print-fds** オプションの **dev**、**pidfd**、および **socket** 引数は、**path** 引数を暗示しなくなりました。

A.5. SYSTEMTAP の変更点

Red Hat Developer Toolset 12.1 には **SystemTap 4.7** が同梱されています。

以下の機能は、Red Hat Developer Toolset の以前のリリース以降に追加または変更されています。

- xMOK キーを使用してモジュールに手動で署名するための新しい **--sign-module** オプション。SecureBoot で使用します。
- アノテーション付きソースコードのシステム全体のプロファイルを作成するための新しい **stap-profile-annotate** ツール。
- 関数のエントリーとリターンをプローブするための新しい一般的な Python タップセット。
- ユーザー空間にある可能性がある文字列のカーネル空間プローブの **\$ foo \$** の拡張処理。
- 非キャプチャグループの正規表現言語の拡張。
- 最近追加されたいくつかのカーネルシステムコールのタップセットサポートの追加。

主な変更の詳細は、アップストリームの [SystemTap 4.7 リリースノート](#) を参照してください。

A.6. DYNINST の変更点

Red Hat Developer Toolset 12.1 には **Dyninst 12.1.0** が同梱されています。

以下の機能は、Red Hat Developer Toolset 12.1 の以前のリリース以降に追加されています。

- **glibc-2.35** の複数名前空間に対する初期サポート
- DWARF 並列解析の並列処理性を修正
- **CUDA** および **CDNA2** GPU バイナリーのサポートを改善
- IBM POWER Systems (リトルエンディアン) レジスターアクセスのサポートを改善
- PIE バイナリーのサポートを改善
- キャッチブロックの解析を修正
- 64 ビット Arm (**aarch64**) 浮動小数点レジスターへのアクセスを修正

A.7. ANNOBIN の変更点

Red Hat Developer Toolset 12.1 には **Annobin 11.08** が同梱されています。

以下の機能は、Red Hat Developer Toolset の以前のリリース以降に追加または変更されています。

- annocheck の新しいコマンドラインオプションは、別の方法でデバッグ情報を見つけることができない場合、**debuginfod** サービスの使用を避けるように指示します。**debuginfod** を使用すると annocheck で多くの情報が提供されますが、**debuginfod** サーバーが使用できない場合、annocheck のパフォーマンスが大幅に低下する可能性もあります。
- Annobin ソースは、必要に応じて設定および作成するのではなく、**meson** および **ninja** を使用してビルドできるようになりました。
- Annocheck は、Rust 1.18 コンパイラーによってビルドされたバイナリーをサポートするようになりました。

付録B バージョン 12.1 での変更点

以下のセクションでは、Red Hat Developer Toolset 12.1 で導入された機能およびバグ修正について説明します。

B.1. GCC の変更点

Red Hat Developer Toolset 12.1 の **GCC** は **GCC 12.2.1** に更新され、アップストリームの **GCC** で利用可能な多くのバグ修正および拡張機能が提供されています。

B.2. ANNOBIN の変更点

Red Hat Developer Toolset 12.1 の **Annobin** が **Annobin 11.08** に更新になりました。

B.3. BINUTILS の変更点

Red Hat Developer Toolset 12.1 のリリースにより、**binutils 2.36.1** のセキュリティー更新が利用可能になりました。