

# GNU 開発ツール・コマンド一覧

## GNU 開発ツールのバージョン情報

<code>ld -v</code>	binutils のバージョン番号を表示
<code>gcc -dumpversion</code>	GCC のバージョン番号を表示
<code>sudo ldconfig -V</code>	GNU C ライブラリ (glibc) のバージョン番号を表示 ldconfig を実行するためには、スーパーユーザの権限が必要 (小文字の v を指定すると、キャッシュファイルの再構築が行われてしまうため注意)

## gcc (GCC ドライバ)

<code>xxxxxx.c</code>	C 言語ソースファイルから実行可能ファイルを自動的にビルド (出力ファイル名を指定しない場合、a.out という名前の実行可能ファイルが作業ディレクトリ上に生成される)
<code>-Wall</code>	すべての警告を表示
<code>-Werror</code>	警告をエラーとみなす
<code>-v</code>	ビルドの過程を詳細に表示
<code>-save-temps</code>	中間ファイルを削除せずに保存
<code>-o</code>	出力ファイル名の指定
<code>-c</code>	オブジェクトファイル (*.o) を出力後、終了
<code>-E</code>	プリプロセッサ結果を標準出力へ出力後、終了
<code>-S</code>	アセンブリ言語ソースファイル (*.s) を出力後、終了
<code>-print-prog-name=xxx</code>	指定された外部コマンドの絶対パスを表示
<code>-print-file-name=xxx</code>	指定された外部ファイル (crt, ライブラリ) の絶対パスを表示
<code>-print-libgcc-file-name</code>	GCC ランタイムライブラリの絶対パスを表示
<code>-static</code>	静的リンクにより実行可能ファイルを生成
<code>-shared</code>	共有オブジェクトを出力
<code>-fPIC</code>	位置独立コードを出力

---

## cc1 (C コンパイラ)

---

**-E** プリプロセスを実行 (GCC バージョン3以降)  
**-quiet** コンパイル済みの関数名や処理時間情報を表示しない  
**-o** 出力ファイル名の指定

## cpp (C プリプロセッサ)

---

**-Dxxxxx=yyy** コマンドライン上でマクロ定義を行う  
例) `-DMESSAGE="Hello, world!"` は、ソースファイル中の  
`#define MESSAGE "Hello, world!"` 文に相当。

**-v** インクルードディレクトリの探索パスを表示

**-H** インデント形式でインクルードの状況を表示

**-M** Makefile 形式で外部依存ファイルを表示

**-dM** すべてのマクロ定義文を表示

**-nostdinc** ヘッダーファイル探索パスを初期化する  
システムヘッダーファイル：探索なし  
ローカルヘッダーファイル：カレントディレクトリのみ

**-I** ディレクトリ名 ヘッダーファイル探索パスを追加する  
(`-I`とディレクトリ名の間にはスペースを挿入しない)

**-I-** セパレータ  
前方指定 `-I`：ローカルヘッダーファイルのみを探索  
後方指定 `-I`：全ヘッダーファイルを探索

システムヘッダーファイル  
探索順序 (デフォルト)

1. `/usr/local/include`
2. `/usr/lib/gcc-lib/i486-linux/3.3.5/include`
3. `/usr/i486-linux/include`
4. `/usr/include`

## as (アセンブラ)

---

**-v** バージョン番号の表示  
**-o** 出力ファイル名の指定

## ar (アーカイバ)

---

<b>t</b>	アーカイブファイル中のテーブルリストを表示
<b>x</b> メンバ名	アーカイブファイル中の指定されたメンバを抽出

## crt ファイル (C RunTime startup)

---

<b>crt1.o</b>	ELF 実行開始アドレスである <code>_start</code> エントリを持ち、実行環境の初期化を行った後に、 <code>main</code> 関数を呼び出す (格納ディレクトリ <code>/usr/lib</code> )
<b>crti.o</b> <b>crtn.o</b>	<code>.init</code> , <code>.fini</code> セクションを含み、それぞれ初期化処理と終了時処理を担当 (格納ディレクトリ <code>/usr/lib</code> )
<b>crtbegin.o</b> <b>crtend.o</b>	<code>.ctors</code> , <code>.dtors</code> などのセクションを含んでおり、コンストラクタ・デコンストラクタ処理を担当 (格納ディレクトリ <code>/usr/lib/gcc-lib/i486-linux/3.3.5</code> )

## ld (リンカ)

---

<b>--verbose</b>	内蔵リンカスクリプトを表示
<b>-t</b>	リンクに使用された入力ファイルを追跡表示する
<b>-L</b> ディレクトリ名	ライブラリの探索ディレクトリを追加
<b>-l</b> xxxxx	ライブラリを指定 (ベースネームから <code>lib</code> を削除した省略名を指定)
<b>-dynamic-linker</b> xxxxx	ダイナミック・リンカローダ (ELF ローダ) の指定
ライブラリ探索順序	1. <code>/usr/i386-linux/lib</code> 2. <code>/usr/local/lib</code> 3. <code>/lib</code> 4. <code>/usr/lib</code>

## ldd (List Dynamic Dependencies)

---

指定された実行可能ファイルの外部依存ファイルを表示  
 ・ 共有ライブラリ: ダイナミックセクションの情報に基づき表示  
 ・ ELF ローダ: プログラムヘッダの `INTERP` フィールドを表示

---

## ld-linux.so.2 (ELF ロード)

---

### 制御環境変数

<code>LD_TRACE_LOADED_OBJECTS</code>	外部依存状況を表示 (ldd コマンドと同一機能)
<code>LD_DEBUG=help</code>	デバッグ用オプションの一覧表示
<code>LD_DEBUG=libs</code>	ライブラリの探索状況を表示
<code>LD_LIBRARY_PATH</code>	ライブラリ探索パスを設定

ライブラリ探索順序	1. <code>LD_LIBRARY_PATH</code> 2. <code>/etc/ld.so.cache</code> 3. <code>/lib</code> 4. <code>/usr/lib</code>
-----------	---

---

## ldconfig (ELF ロード実行環境設定)

---

オプションなし	<code>/etc/ld.so.cache</code> キャッシュファイルおよび関連シンボリックリンクを再構築
<code>-v</code>	再構築作業中に、探索中のディレクトリ名や処理中のライブラリ名を表示
<code>-p</code>	キャッシュファイルの内容をテキスト形式で表示
共有ライブラリ探索順序	1. <code>/lib</code> 2. <code>/usr/lib</code>

---

## objdump (オブジェクト・ダンプ)

---

<code>-d</code>	<code>.text</code> セクションの逆アセンブルリストを表示
<code>-j</code> セクション名	解析対象セクションを指定
<code>-s</code>	セクション内容をダンプ

---

## readelf (ELF 解析)

---

<code>-S</code>	セクションヘッダーテーブルを表示
<code>-l</code>	プログラムヘッダーを表示
<code>-s</code>	シンボル情報を表示
<code>-d</code>	ダイナミックセクションの情報を表示