The **LLVM** Compiler Infrastructure

Site Map:

Download!

Download now: **LLVM 5.0.0**
All Releases
APT Packages
Win Installer

View the open-source **license**

Search this Site

Search!

Useful Links

Mailing Lists:
LLVM-announce
LLVM-dev
LLVM-bugs
LLVM-commits
LLVM-branch-commits
LLVM-testresults

LLVM Overview

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

LLVM began as a research project at the University of Illinois, with the goal of providing a modern, SSA-based compilation strategy capable of supporting both static and dynamic compilation of arbitrary programming languages. Since then, LLVM has grown to be an umbrella project consisting of a number of subprojects, many of which are being used in production by a wide variety of commercial and open source projects as well as being widely used in academic research. Code in the LLVM project is licensed

Latest LLVM Release!

7 Sep 2017: LLVM 5.0.0 is now **available for download**! LLVM is publicly available under an open source License. Also, you might want to check out **the new features** in SVN that will appear in the next LLVM release. If you want them early, download LLVM through anonymous SVN.

ACM Software System Award!

LLVM has been awarded the **2012 ACM Software System Award**! This award is given by ACM to *one* software system worldwide every year. LLVM is in highly distinguished company! Click on any of the individual recipients' names on that page for the detailed citation describing the award.

Upcoming Releases

LLVM Release Schedule:

- 5.0.1:
  - To be decided.

Developer Meetings

Upcoming: October 18-19, 2017

Proceedings from past meetings:

IRC Channel:
irc.oftc.net #llvm

Dev. Resources:
doxygen ViewVC
Blog Bugzilla
Buildbot LNT Coverage
Scan-build llvm-cov

Release Emails

5.0.0: Sep 2017
4.0.1: Jul 2017
4.0.0: Mar 2017
3.9.1: Dec 2016
3.9.0: Sep 2016
3.8.1: July 2016
3.8.0: Mar 2016
3.7.1: Jan 2016
3.7.0: Sep 2015
3.6.2: Jul 2015
3.6.1: May 2015
3.5.2: Apr 2015
3.6: Feb 2015
3.5.1: Jan 2015
3.5: Sep 2014
3.4.2: June 2014
3.4.1: May 2014
3.4: Jan 2014
3.3: Jun 2013
3.2: Dec 2012
3.1: May 2012
3.0: Dec 2011
2.9: Apr 2011
2.8: Oct 2010
2.7: Apr 2010
2.6: Oct 2009
2.5: Mar 2009
2.4: Nov 2008
2.3: Jun 2008
2.2: Feb 2008
2.1: Sep 2007
2.0: May 2007
Older News

Maintained by the
llvm-admin team

under the "UIUC" BSD-Style license.

The primary sub-projects of LLVM are:

1. The **LLVM Core** libraries provide a modern source- and target-independent optimizer, along with code generation support for many popular CPUs (as well as some less common ones!) These libraries are built around a well specified code representation known as the LLVM intermediate representation ("LLVM IR"). The LLVM Core libraries are well documented, and it is particularly easy to invent your own language (or port an existing compiler) to use LLVM as an optimizer and code generator.

2. **Clang** is an "LLVM native" C/C++/Objective-C compiler, which aims to deliver amazingly fast compiles (e.g. about 3x faster than GCC when

- March 27-28, 2017
- November 3-4, 2016
- March 17-18, 2016
- October 29-30, 2015
- April 13-14, 2015
- October 28-29, 2014
- April 7-8, 2014
- Nov 6-7, 2013
- April 29-30, 2013
- November 7-8, 2012
- April 12, 2012
- November 18, 2011
- September 2011
- November 2010
- October 2009
- August 2008
- May 2007

compiling Objective-C code in a debug configuration), extremely useful error and warning messages and to provide a platform for building great source level tools. The Clang Static Analyzer is a tool that automatically finds bugs in your code, and is a great example of the sort of tool that can be built using the Clang frontend as a library to parse C/C++ code.

3. The **LLDB** project builds on libraries provided by LLVM and Clang to provide a great native debugger. It uses the Clang ASTs and expression parser, LLVM JIT, LLVM disassembler, etc so that it provides an experience that "just works". It is also blazing fast and much more memory efficient than GDB at loading symbols.

4. The **libc++** and **libc++ ABI** projects provide a standard conformant and

high-performance implementation of the C++ Standard Library, including full support for C++11.

5. The **compiler-rt** project provides highly tuned implementations of the low-level code generator support routines like "`__fixunsdfdi`" and other calls generated when a target doesn't have a short sequence of native instructions to implement a core IR operation. It also provides implementations of run-time libraries for dynamic testing tools such as AddressSanitizer, ThreadSanitizer, MemorySanitizer, and DataFlowSanitizer.

6. The **OpenMP** subproject provides an OpenMP runtime for use with the OpenMP implementation in Clang.

7. The **polly** project implements a suite of cache-locality optimizations as

well as auto-parallelism and vectorization using a polyhedral model.

8. The **libclc** project aims to implement the OpenCL standard library.

9. The **klee** project implements a "symbolic virtual machine" which uses a theorem prover to try to evaluate all dynamic paths through a program in an effort to find bugs and to prove properties of functions. A major feature of klee is that it can produce a testcase in the event that it detects a bug.

10. The **SAFECode** project is a memory safety compiler for C/C++ programs. It instruments code with run-time checks to detect memory safety errors (e.g., buffer overflows) at run-time. It can be used to protect software from security attacks and can also be used as a memory safety error

debugging tool like Valgrind.

11. The **lld** project aims to be the built-in linker for clang/llvm. Currently, clang must invoke the system linker to produce executables.

In addition to official subprojects of LLVM, there are a broad variety of other projects that use components of LLVM for various tasks. Through these external projects you can use LLVM to compile Ruby, Python, Haskell, Java, D, PHP, Pure, Lua, and a number of other languages. A major strength of LLVM is its versatility, flexibility, and reusability, which is why it is being used for such a wide variety of different tasks: everything from doing light-weight JIT compiles of embedded languages like Lua to compiling Fortran code for massive super computers.

As much as everything else, LLVM has a broad and friendly community of people who are interested in building great low-level tools. If you are interested in getting involved, a good

first place is to skim the LLVM Blog and to sign up for the LLVM Developer mailing list. For information on how to send in a patch, get commit access, and copyright and license topics, please see the LLVM Developer Policy.