NAME

scan-build — Clang static analyzer

SYNOPSIS

scan-build [**-ohkvV**] [**-analyze-headers**] [**-enable-checker** [*checker_name*]] [**-disable-checker** [*checker_name*]] [**--help**] [**--help-checkers**] [**--html-title** [=*title*]] [**--keep-going**] [**-plist**] [**-plist-html**] [**--status-bugs**] [**--use-c++** [=*compiler_path*]] [**--use-cc** [=*compiler_path*]] [**--view**] [**-constraints** [*model*]] [**-maxloop** *N*] [**-no-failure-reports**] [**-stats**] [**-store** [*model*]] *build_command* [build_options]

DESCRIPTION

scan-build is a Perl script that invokes the Clang static analyzer. Options used by **scan-build** or by the analyzer appear first, followed by the *build_command* and any *build_options* normally used to build the target system.

The static analyzer employs a long list checking algorithms, see **CHECKERS**. Output can be written in standard `.plist` and/or HTML format.

The following options are supported:

-analyze-headers
: Also analyze functions in #included files.

-enable-checker *checker_name*, **-disable-checker** *checker_name*
: Enable/disable *checker_name*. See **CHECKERS**.

-h, **--help**
: Display this message.

--help-checkers
: List default checkers, see **CHECKERS**.

--html-title[=*title*]
: Specify the title used on generated HTML pages. A default title is generated if *title* is not specified.

-k, **--keep-going**
: Add a “keep on going” option to *build_command*. Currently supports make and xcodebuild. This is a convenience option; one can specify this behavior directly using build options.

-o
: Target directory for HTML report files. Subdirectories will be created as needed to represent separate invocations of the analyzer. If this option is not specified, a directory is created in /tmp (TMPDIR on Mac OS X) to store the reports.

-plist
: Output the results as a set of `.plist` files. (By default the output of **scan-build** is a set of HTML files.)

-plist-html
: Output the results as a set of HTML and .plist files

--status-bugs
: Set exit status to 1 if it found potential bugs and 0 otherwise. By default the exit status of **scan-build** is that returned by *build_command*.

--use-c++[=*compiler_path*]
Guess the default compiler for your C++ and Objective-C++ code. Use this option to specify an alternate compiler.

--use-cc[=*compiler_path*]
Guess the default compiler for your C and Objective-C code. Use this option to specify an alternate compiler.

-v Verbose output from **scan-build** and the analyzer. A second and third *v* increases verbosity.

-V, **--view**
View analysis results in a web browser when the build completes.

-constraints [*model*]
Specify the contraint engine used by the analyzer. By default the `range` model is used. Specifying `basic` uses a simpler, less powerful constraint model used by checker-0.160 and earlier.

-maxloop *N*
Specifiy the number of times a block can be visited before giving up. Default is 4. Increase for more comprehensive coverage at a cost of speed.

-no-failure-reports
Do not create a `failures` subdirectory that includes analyzer crash reports and preprocessed source files.

-stats
Generates visitation statistics for the project being analyzed.

-store [*model*]
Specify the store model used by the analyzer. By default, the `region` store model is used. `region` specifies a field- sensitive store model. Users can also specify `basic` which is far less precise but can more quickly analyze code. `basic` was the default store model for checker-0.221 and earlier.

RETURN VALUES

scan-build returns the value returned by *build_command* unless **--status-bugs** or **--keep-going** is used.

CHECKERS

The checkers listed below may be enabled/disabled using the **-enable-checker** and **-disable-checker** options. A default group of checkers is run unless explicitly disabled. Exactly which checkers constitute the default group is a function of the operating system in use; they are listed with **--help-checkers**.

core.AdjustedReturnValue
Check to see if the return value of a function call is different than the caller expects (e.g., from calls through function pointers).

core.AttributeNonNull
Check for null pointers passed as arguments to a function whose arguments are marked with the `nonnull` attribute.

core.CallAndMessage
Check for logical errors for function calls and Objective-C message expressions (e.g., uninitialized arguments, null function pointers).

core.DivideZero
: Check for division by zero.

core.NullDereference
: Check for dereferences of null pointers.

core.StackAddressEscape
: Check that addresses to stack memory do not escape the function.

core.UndefinedBinaryOperatorResult
: Check for undefined results of binary operators.

core.VLASize
: Check for declarations of VLA of undefined or zero size.

core.builtin.BuiltinFunctions
: Evaluate compiler builtin functions, e.g. **`alloca`**().

core.builtin.NoReturnFunctions
: Evaluate `panic` functions that are known to not return to the caller.

core.uninitialized.ArraySubscript
: Check for uninitialized values used as array subscripts.

core.uninitialized.Assign
: Check for assigning uninitialized values.

core.uninitialized.Branch
: Check for uninitialized values used as branch conditions.

core.uninitialized.CapturedBlockVariable
: Check for blocks that capture uninitialized values.

core.uninitialized.UndefReturn
: Check for uninitialized values being returned to the caller.

deadcode.DeadStores
: Check for values stored to variables that are never read afterwards.

debug.DumpCFG
: Display Control-Flow Graphs.

debug.DumpCallGraph
: Display Call Graph.

debug.DumpDominators
: Print the dominance tree for a given Control-Flow Graph.

debug.DumpLiveVars
: Print results of live variable analysis.

debug.Stats
: Emit warnings with analyzer statistics.

debug.TaintTest
: Mark tainted symbols as such.

debug.ViewCFG
: View Control-Flow Graphs using **`GraphViz`**.

debug.ViewCallGraph
: View Call Graph using **GraphViz**.

llvm.Conventions
: Check code for LLVM codebase conventions.

osx.API
: Check for proper uses of various Mac OS X APIs.

osx.AtomicCAS
: Evaluate calls to *OSAtomic* functions.

osx.SecKeychainAPI
: Check for proper uses of Secure Keychain APIs.

osx.cocoa.AtSync
: Check for null pointers used as mutexes for @synchronized.

osx.cocoa.ClassRelease
: Check for sending `retain`, `release`, or `autorelease` directly to a Class.

osx.cocoa.IncompatibleMethodTypes
: Warn about Objective-C method signatures with type incompatibilities.

osx.cocoa.NSAutoreleasePool
: Warn for suboptimal uses of *NSAutoreleasePool* in Objective-C GC mode.

osx.cocoa.NSError
: Check usage of NSError** parameters.

osx.cocoa.NilArg
: Check for prohibited nil arguments to Objective-C method calls.

osx.cocoa.RetainCount
: Check for leaks and improper reference count management.

osx.cocoa.SelfInit
: Check that `self` is properly initialized inside an initializer method.

osx.cocoa.UnusedIvars
: Warn about private ivars that are never used.

osx.cocoa.VariadicMethodTypes
: Check for passing non-Objective-C types to variadic methods that expect only Objective-C types.

osx.coreFoundation.CFError
: Check usage of CFErrorRef* parameters.

osx.coreFoundation.CFNumber
: Check for proper uses of **CFNumberCreate**().

osx.coreFoundation.CFRetainRelease
: Check for null arguments to **CFRetain**() and **CFRelease**().

osx.coreFoundation.containers.OutOfBounds
: Checks for index out-of-bounds when using the *CFArray* API.

osx.coreFoundation.containers.PointerSizedValues
: Warns if *CFArray*, *CFDictionary*, or *CFSet* are created with non-pointer-size values.

security.FloatLoopCounter
: Warn on using a floating point value as a loop counter (CERT: FLP30-C, FLP30-CPP).

security.insecureAPI.UncheckedReturn
: Warn on uses of functions whose return values must be always checked.

security.insecureAPI.getpw
: Warn on uses of **getpw**().

security.insecureAPI.gets
: Warn on uses of **gets**().

security.insecureAPI.mkstemp
: Warn when **mkstemp**() is passed fewer than 6 X's in the format string.

security.insecureAPI.mktemp
: Warn on uses of **mktemp**().

security.insecureAPI.rand
: Warn on uses of **rand**(), **random**(), and related functions.

security.insecureAPI.strcpy
: Warn on uses of **strcpy**() and **strcat**().

security.insecureAPI.vfork
: Warn on uses of **vfork**().

unix.API Check calls to various UNIX/Posix functions.

unix.Malloc
: Check for memory leaks, double free, and use-after-free.

unix.cstring.BadSizeArg
: Check the size argument passed into C string functions for common erroneous patterns.

unix.cstring.NullArg
: Check for null pointers being passed as arguments to C string functions.

EXAMPLE

scan-build -o /tmp/myhtmldir make -j4

The above example causes analysis reports to be deposited into a subdirectory of `/tmp/myhtmldir` and to run **make** with the **-j4** option. A different subdirectory is created each time **scan-build** analyzes a project. The analyzer should support most parallel builds, but not distributed builds.

AUTHORS

scan-build was written by Ted Kremenek. Documentation contributed by James K. Lowden ⟨jklowden@schemamania.org⟩.