NAME

llvm-cov - emit coverage information

SYNOPSIS

llvm-cov *command* [*args...*]

DESCRIPTION

The **llvm-cov** tool shows code coverage information for programs that are instrumented to emit profile data. It can be used to work with **gcov**-style coverage or with **clang**'s instrumentation based profiling.

If the program is invoked with a base name of **gcov**, it will behave as if the **llvm-cov gcov** command were called. Otherwise, a command should be provided.

COMMANDS

- *gcov*
- *show*
- *report*
- *export*

GCOV COMMAND

SYNOPSIS

llvm-cov gcov [*options*] *SOURCEFILE*

DESCRIPTION

The **llvm-cov gcov** tool reads code coverage data files and displays the coverage information for a specified source file. It is compatible with the **gcov** tool from version 4.2 of **GCC** and may also be compatible with some later versions of **gcov**.

To use **llvm-cov gcov**, you must first build an instrumented version of your application that collects coverage data as it runs. Compile with the **-fprofile-arcs** and **-ftest-coverage** options to add the instrumentation. (Alternatively, you can use the **--coverage** option, which includes both of those other options.)

At the time you compile the instrumented code, a **.gcno** data file will be generated for each object file. These **.gcno** files contain half of the coverage data. The other half of the data comes from **.gcda** files that are generated when you run the instrumented program, with a separate **.gcda** file for each object

file. Each time you run the program, the execution counts are summed into any existing **.gcda** files, so be sure to remove any old files if you do not want their contents to be included.

By default, the **.gcda** files are written into the same directory as the object files, but you can override that by setting the **GCOV_PREFIX** and **GCOV_PREFIX_STRIP** environment variables. The **GCOV_PREFIX_STRIP** variable specifies a number of directory components to be removed from the start of the absolute path to the object file directory. After stripping those directories, the prefix from the **GCOV_PREFIX** variable is added. These environment variables allow you to run the instrumented program on a machine where the original object file directories are not accessible, but you will then need to copy the **.gcda** files back to the object file directories where **llvm-cov gcov** expects to find them.

Once you have generated the coverage data files, run **llvm-cov gcov** for each main source file where you want to examine the coverage results. This should be run from the same directory where you previously ran the compiler. The results for the specified source file are written to a file named by appending a **.gcov** suffix. A separate output file is also created for each file included by the main source file, also with a **.gcov** suffix added.

The basic content of an **.gcov** output file is a copy of the source file with an execution count and line number prepended to every line. The execution count is shown as **-** if a line does not contain any executable code. If a line contains code but that code was never executed, the count is displayed as #####.

OPTIONS

-a, --all-blocks
Display all basic blocks. If there are multiple blocks for a single line of source code, this option causes llvm-cov to show the count for each block instead of just one count for the entire line.

-b, --branch-probabilities
Display conditional branch probabilities and a summary of branch information.

-c, --branch-counts
Display branch counts instead of probabilities (requires -b).

-m, --demangled-names
Demangle function names.

-f, --function-summaries
Show a summary of coverage for each function instead of just one summary for an entire source

file.

--help
Display available options (--help-hidden for more).

-l, --long-file-names
For coverage output of files included from the main source file, add the main file name followed by ## as a prefix to the output file names. This can be combined with the --preserve-paths option to use complete paths for both the main file and the included file.

-n, --no-output
Do not output any **.gcov** files. Summary information is still displayed.

-o <DIR|FILE>, --object-directory=<DIR>, --object-file=<FILE>
Find objects in DIR or based on FILE's path. If you specify a particular object file, the coverage data files are expected to have the same base name with **.gcno** and **.gcda** extensions. If you specify a directory, the files are expected in that directory with the same base name as the source file.

-p, --preserve-paths
Preserve path components when naming the coverage output files. In addition to the source file name, include the directories from the path to that file. The directories are separate by **#** characters, with **.** directories removed and **..** directories replaced by ^ characters. When used with the --long-file-names option, this applies to both the main file name and the included file name.

-r Only dump files with relative paths or absolute paths with the prefix specified by **-s**.

-s <string>
Source prefix to elide.

-t, --stdout
Print to stdout instead of producing **.gcov** files.

-u, --unconditional-branches
Include unconditional branches in the output for the --branch-probabilities option.

-version
Display the version of llvm-cov.

-x, --hash-filenames
Use md5 hash of file name when naming the coverage output files. The source file name will be

suffixed by ## followed by MD5 hash calculated for it.

EXIT STATUS

llvm-cov gcov returns 1 if it cannot read input files. Otherwise, it exits with zero.

SHOW COMMAND

SYNOPSIS

llvm-cov show [*options*] -instr-profile *PROFILE* [*BIN*] [-*object BIN*]... [-*sources*] [*SOURCE*]...

DESCRIPTION

The **llvm-cov show** command shows line by line coverage of the binaries *BIN*... using the profile data *PROFILE*. It can optionally be filtered to only show the coverage for the files listed in *SOURCE*....

BIN may be an executable, object file, dynamic library, or archive (thin or otherwise).

To use **llvm-cov show**, you need a program that is compiled with instrumentation to emit profile and coverage data. To build such a program with **clang** use the **-fprofile-instr-generate** and **-fcoverage-mapping** flags. If linking with the **clang** driver, pass **-fprofile-instr-generate** to the link stage to make sure the necessary runtime libraries are linked in.

The coverage information is stored in the built executable or library itself, and this is what you should pass to **llvm-cov show** as a *BIN* argument. The profile data is generated by running this instrumented program normally. When the program exits it will write out a raw profile file, typically called **default.profraw**, which can be converted to a format that is suitable for the *PROFILE* argument using the **llvm-profdata merge** tool.

OPTIONS

-show-branches=<VIEW>

Show coverage for branch conditions in terms of either count or percentage. The supported views are: "count", "percent".

-show-line-counts

Show the execution counts for each line. Defaults to true, unless another **-show** option is used.

-show-expansions

Expand inclusions, such as preprocessor macros or textual inclusions, inline in the display of the source file. Defaults to false.

-show-instantiations

For source regions that are instantiated multiple times, such as templates in **C++**, show each instantiation separately as well as the combined summary. Defaults to true.

-show-regions
Show the execution counts for each region by displaying a caret that points to the character where the region starts. Defaults to false.

-show-line-counts-or-regions
Show the execution counts for each line if there is only one region on the line, but show the individual regions if there are multiple on the line. Defaults to false.

-use-color
Enable or disable color output. By default this is autodetected.

-arch=[*NAMES*]
Specify a list of architectures such that the Nth entry in the list corresponds to the Nth specified binary. If the covered object is a universal binary, this specifies the architecture to use. It is an error to specify an architecture that is not included in the universal binary or to use an architecture that does not match a non-universal binary.

-name=<NAME>
Show code coverage only for functions with the given name.

-name-allowlist=<FILE>
Show code coverage only for functions listed in the given file. Each line in the file should start with *allowlist_fun:*, immediately followed by the name of the function to accept. This name can be a wildcard expression.

-name-regex=<PATTERN>
Show code coverage only for functions that match the given regular expression.

-ignore-filename-regex=<PATTERN>
Skip source code files with file paths that match the given regular expression.

-format=<FORMAT>
Use the specified output format. The supported formats are: "text", "html".

-tab-size=<TABSIZE>
Replace tabs with <TABSIZE> spaces when preparing reports. Currently, this is only supported for the html format.

-output-dir=PATH

Specify a directory to write coverage reports into. If the directory does not exist, it is created. When used in function view mode (i.e when -name or -name-regex are used to select specific functions), the report is written to PATH/functions.EXTENSION. When used in file view mode, a report for each file is written to PATH/REL_PATH_TO_FILE.EXTENSION.

-Xdemangler=<TOOL>|<TOOL-OPTION>

Specify a symbol demangler. This can be used to make reports more human-readable. This option can be specified multiple times to supply arguments to the demangler (e.g *-Xdemangler c++filt -Xdemangler -n* for C++). The demangler is expected to read a newline-separated list of symbols from stdin and write a newline-separated list of the same length to stdout.

-num-threads=N, -j=N

Use N threads to write file reports (only applicable when -output-dir is specified). When N=0, llvm-cov auto-detects an appropriate number of threads to use. This is the default.

-compilation-dir=<dir>

Directory used as a base for relative coverage mapping paths. Only applicable when binaries have been compiled with one of *-fcoverage-prefix-map -fcoverage-compilation-dir*, or *-ffile-compilation-dir*.

-line-coverage-gt=<N>

Show code coverage only for functions with line coverage greater than the given threshold.

-line-coverage-lt=<N>

Show code coverage only for functions with line coverage less than the given threshold.

-region-coverage-gt=<N>

Show code coverage only for functions with region coverage greater than the given threshold.

-region-coverage-lt=<N>

Show code coverage only for functions with region coverage less than the given threshold.

-path-equivalence=<from>,<to>

Map the paths in the coverage data to local source file paths. This allows you to generate the coverage data on one machine, and then use llvm-cov on a different machine where you have the same files on a different path.

-coverage-watermark=<high>,<low>

Set high and low watermarks for coverage in html format output. This allows you to set the high

and low watermark of coverage as desired, green when coverage >= high, red when coverage < low, and yellow otherwise. Both high and low should be between 0-100 and high > low.

-debuginfod

Use debuginfod to look up coverage mapping for binary IDs present in the profile but not in any object given on the command line. Defaults to true if debuginfod is compiled in and configured via the DEBUGINFOD_URLS environment variable.

-debug-file-directory=<dir>

Provides local directories to search for objects corresponding to binary IDs in the profile (as with debuginfod). Defaults to system build ID directories.

REPORT COMMAND

SYNOPSIS

llvm-cov report [*options*] -instr-profile *PROFILE* [*BIN*] [-*object BIN*]... [-*sources*] [*SOURCE*]...

DESCRIPTION

The **llvm-cov report** command displays a summary of the coverage of the binaries *BIN*... using the profile data *PROFILE*. It can optionally be filtered to only show the coverage for the files listed in *SOURCE*....

BIN may be an executable, object file, dynamic library, or archive (thin or otherwise).

If no source files are provided, a summary line is printed for each file in the coverage data. If any files are provided, summaries can be shown for each function in the listed files if the **-show-functions** option is enabled.

For information on compiling programs for coverage and generating profile data, see *SHOW COMMAND*.

OPTIONS

-use-color[=VALUE]

Enable or disable color output. By default this is autodetected.

-arch=<name>

If the covered binary is a universal binary, select the architecture to use. It is an error to specify an architecture that is not included in the universal binary or to use an architecture that does not match

a non-universal binary.

-show-region-summary
Show statistics for all regions. Defaults to true.

-show-branch-summary
Show statistics for all branch conditions. Defaults to true.

-show-functions
Show coverage summaries for each function. Defaults to false.

-show-instantiation-summary
Show statistics for all function instantiations. Defaults to false.

-ignore-filename-regex=<PATTERN>
Skip source code files with file paths that match the given regular expression.

-compilation-dir=<dir>
Directory used as a base for relative coverage mapping paths. Only applicable when binaries have been compiled with one of *-fcoverage-prefix-map -fcoverage-compilation-dir*, or *-ffile-compilation-dir*.

-debuginfod

Attempt to look up coverage mapping from objects using debuginfod. This is attempted by default for binary IDs present in the profile but not provided on the command line, so long as debuginfod is compiled in and configured via DEBUGINFOD_URLS.

-debug-file-directory=<dir>

Provides a directory to search for objects corresponding to binary IDs in the profile.

EXPORT COMMAND

SYNOPSIS

llvm-cov export [*options*] -instr-profile *PROFILE* [*BIN*] [*-object BIN*]... [*-sources*] [*SOURCE*]...

DESCRIPTION

The **llvm-cov export** command exports coverage data of the binaries *BIN*... using the profile data *PROFILE* in either JSON or lcov trace file format.

When exporting JSON, the regions, functions, branches, expansions, and summaries of the coverage data will be exported. When exporting an lcov trace file, the line-based coverage, branch coverage, and summaries will be exported.

The exported data can optionally be filtered to only export the coverage for the files listed in *SOURCE...*.

For information on compiling programs for coverage and generating profile data, see *SHOW COMMAND*.

OPTIONS

-arch=<name>
If the covered binary is a universal binary, select the architecture to use. It is an error to specify an architecture that is not included in the universal binary or to use an architecture that does not match a non-universal binary.

-format=<FORMAT>
Use the specified output format. The supported formats are: "text" (JSON), "lcov".

-summary-only
Export only summary information for each file in the coverage data. This mode will not export coverage information for smaller units such as individual functions or regions. The result will contain the same information as produced by the **llvm-cov report** command, but presented in JSON or lcov format rather than text.

-ignore-filename-regex=<PATTERN>
Skip source code files with file paths that match the given regular expression.

-skip-expansions

Skip exporting macro expansion coverage data.

-skip-functions

Skip exporting per-function coverage data.

-num-threads=N, -j=N

Use N threads to export coverage data. When N=0, llvm-cov auto-detects an

appropriate number of threads to use. This is the default.

-compilation-dir=<dir>
Directory used as a base for relative coverage mapping paths. Only applicable when binaries have been compiled with one of *-fcoverage-prefix-map -fcoverage-compilation-dir*, or *-ffile-compilation-dir*.

-debuginfod

Attempt to look up coverage mapping from objects using debuginfod. This is attempted by default for binary IDs present in the profile but not provided on the command line, so long as debuginfod is compiled in and configured via DEBUGINFOD_URLS.

-debug-file-directory=<dir>

Provides a directory to search for objects corresponding to binary IDs in the profile.

AUTHOR

Maintained by the LLVM Team (https://llvm.org/).

COPYRIGHT