Android Developers

Simpleperf command and options reference

This page describes the most commonly used Simpleperf commands and options. You can use the `simpleperf --help` command to see the full list of options and subcommands.

Syntax

```
simpleperf [--version] [--help] command [command options]
```

Global options

Global Options	Description
`--help`	Display usage help.
`--log severity`	Set the minimum severity of logging. Valid severity values are: • `verbose` • `debug` • `warning` • `info`

	• `fatal` Defaults to `info`.
`--version`	Display version information.

Key commands and options

Commands and Options	Description
`list [hw|sw|cache|tracepoint]`	List all available performance events on this machine.
`record [`***`options`***`] [`***`command`*** `[`***`command-args`***`]]`	Records samples of the profiled process within a time period. If you specify a ***`command`*** in the argument, Simpleperf gathers sampling information only for the process running ***`command`***. It then writes the sampling data to a `perf.data` file. You can use the `-a/-p/-t` record options to change the target of the sampling. • `-a`: System-wide collection from all CPUs. • `-b`: Enable taken branch stack sampling. This is the same as using the `'-j any'` option. See the documentation for that option for more information. • `-c` : Set event sample period. It means recording one sample when ***`count`*** events happen. This option cannot be used with the `-f/-F` option. For tracepoint events, the default option is `-c 1`. • `--call-graph fp | dwarf[,` ***`dump_stack_size`***`]`: Enable call graph recording. You can use either the frame pointer or dwarf debug frame as the method to collect the information used to show the call graphs. The default is setting is `dwarf,65528`. • `--cpu` ***`cpu_item1`***`,`***`cpu_item2`***`,...`: Collect samples only on the selected CPUs. You can specify `[`***`cpu_item`***`]` as a CPU number like 1, or as a CPU range like 0-3.

- `--dump-symbols`: Dump symbols in a `perf.data` file. By default, `perf.data` does not contain symbol information for samples. This option is used when there is no symbol information in report environment.
- `--duration` ***`time_in_sec`***: Monitor threads for ***`time_in_sec`*** seconds instead of running `command`. Here ***`time_in_sec`*** may be any positive floating point number.
- `-e` ***`event1`***`[:`***`modifier1`***`],`***`event2`***`[:`***`modifier2`***`],...` : Select the event list to sample. Use `` `simpleperf list` `` to find all possible event names. You can add modifiers to define how to monitor the event. Possible modifiers are:
  - `u`: monitor user space events only
  - `k`: monitor kernel space events only
- `-f` ***`freq`***: Set event sample frequency. It means recording at most ***`freq`***samples every second. For non-tracepoint events, the default option is `-f 4000`.
- `-F freq`: Same as '-f freq'.
- `-g`: Same as '`--call-graph dwarf`'.
- `--group` ***`event1`***`[:`***`modifier`***`],`***`event2`***`[:`***`modifier2`***`], ...`: Similar to the `-e` option, but events specified in the same `--group` option are monitored as a group, and scheduled in and out at the same time.
- `-j` ***`branch_filter1`***`,`***`branch_filter2`***`,...`: Enable taken branch stack sampling. Each sample captures a series of consecutive taken branches. You can specify the following filters:
  - `any`: Any type of branch.
  - `any_call`: Any function call or system call.
  - `any_ret`: Any function return or system call return.
  - `ind_call`: Any indirect branch.
  - `u`: Only when the branch target is at the user level.

	◦ `k`: Only when the branch target is in the kernel. This option requires at least one branch type among `any`, `any_call`, `any_ret`, or `ind_call`. • `-m mmap_pages`: Set the size of the buffer used to receiving sample data from the kernel. It should be a power of 2. If not set, the max possible value `<=1024` will be used. • `--no-dump-kernel-symbols`: Don't dump kernel symbols in `perf.data`. By default, Simpleperf dumps kernel symbols when needed. • `--no-inherit`: Don't record created child threads or processes. • `--no-unwind`: By default, the user's stack is unwound if the `` `--call-graph dwarf` `` option is used. Use this option to disable the unwinding of the user's stack. • `-o record_file_name`: Set the record file name; the default name is `perf.data`. • `-p pid1,pid2,...`: Record events on existing processes. Mutually exclusive with `-a`. • `--post-unwind`: By default, the user's stack is unwound while recording, if the `` `--call-graph dwarf` `` option is used. Records may be lost as stacking unwinding can be time consuming. Use this option to unwind the user's stack after recording. • `--start_profiling_fd fd_no`: After starting profiling, write "`STARTED`" to `fd_no`, then close `fd_no`. • `--symfs dir`: Look for files with symbols relative to this directory. Use this option to provide files with symbol table and debug information, which are used by `--dump-symbols` and `-g`. • `-t tid1,tid2,...`: Record events on existing threads. Mutually exclusive with `-a`.
`report [options]`	Read a `perf.data` file (created by `simpleperf record`) and display a report showing where time was spent. • `-b`: Use the branch-to addresses in sampled taken branches instead of the instruction addresses. Only valid for `perf.data` recorded with `-b/-j` option. • `--brief-callgraph`: Print brief call graph.

- `--comms comm1,comm2,...`: Report only for selected commands.
- `--dsos dso1,dso2,...`: Report only for selected dynamic shared objects.
- `-g [callee|caller]`: Print call graph. If callee mode is used, the graph shows how functions are called from others. Otherwise, the graph shows how functions call others. The default mode is `caller`.
- `-i file`: Specify path of the record file; the default path is `perf.data`.
- `--kallsyms file`: Set the file to read kernel symbols.
- `--max-stack frames`: Set max stack frames shown when printing call graph.
- `-n`: Print the sample count for each item.
- `--no-demangle`: Don't demangle symbol names.
- `--no-show-ip`: Don't show vaddr in file for unknown symbols.
- `-o report_file_name`: Set the report file name, the default file name is `stdout`.
- `--percent-limit percent`: Set min percentage shown when printing call graph.
- `--pids pid1,pid2,...`: Report only for selected process IDs.
- `--raw-period`: Report period count instead of period percentage.
- `--sort key1,key2,...`: Select the keys used to sort and print the report. The order you specify the keys determines the order of keys used to sort and print the report. Possible keys include:
  - `pid`: Process id.
  - `tid`: Thread id.
  - `comm`: Thread name (can be changed during the lifetime of a thread).
  - `dso`: Shared library.

	◦ `vaddr_in_file`: Virtual address in the shared library. Keys can only be used with the `-b` option: ◦ `dso_from`: Shared library branched from. ◦ `dso_to`: Shared library branched to. ◦ `symbol_from`: Name of function branched from. ◦ `symbol_to`: Name of function branched to. The default sort keys are: `comm`, `pid`, `tid`, `dso`, and `symbol`. • `--symbols symbol1;symbol2;...`: Report only for the selected symbols. • `--symfs dir`: Look for files with symbols relative to this directory. • `--tids tid1,tid2,...`: Report only for the selected thread IDs. • `--vmlinux file`: Parse kernel symbols from `file`.
`stat [options] [command [command-args]]`	Get a summary of how many events have happened in a profiled process within a time period. If you specify a *command* in the argument, Simpleperf gathers performance counter information only for the process running . You can use the `-a/-p/-t` options to change the target of the counter information gathering. • `-a`: Collect system-wide information. • `--cpu cpu_item1,cpu_item2,...`: Collect information only on the selected CPUs. *cpu_item* can be a CPU number like 1, or a CPU range like 0-3. • `--csv`: Write report in comma separate form. • `--duration time_in_sec`: Monitor for *time_in_sec* seconds instead of running *command*. Here *time_in_sec* may be any positive floating point number. • `--interval time_in_ms`: Print stat for every *time_in_ms* milliseconds. Here *time_in_ms* may be any positive floating point number.

	• `-e event1[:modifier1],event2[:modifier2],...` : Select the event list to count. Use `simpleperf list` to find all possible event names. Modifiers can be added to define how the event should be monitored. Possible modifiers are:
	  ◦ `u`: Monitor user space events only.
	  ◦ `k`: Monitor kernel space events only.
	• `--group event1[:modifier],event2[:modifier2] ,...`: Similar to the `-e` option. Events specified in the same `--group` option are monitored as a group, and scheduled in and out at the same time.
	• `--no-inherit`: Don't stat created child threads or processes.
	• `-o output_filename`: Write report to `output_filename` instead of standard output.
	• `-p pid1,pid2,...`: Stat events on existing processes. Mutually exclusive with `-a`.
	• `-t tid1,tid2,...`: Stat events on existing threads. Mutually exclusive with `-a`.
	• `--verbose`: Show result in verbose mode.

Python scripts

The Simpleperf tool includes several Python scripts, located in the `ndk-location/simpleperf/` directory, that help simplify the recording and reporting of performance data on an Android device. You run these scripts on a host development machine that is connected to the target Android device. To see an example of how to use these scripts, refer to the Simpleperf Demo (https://android.googlesource.com/platform/system/extras/+/master/simpleperf/demo/) sample on GitHub.

Script	Description
`annotate.py`	Annotate source files based on profiling data from `perf.data`. You can configure the behavior of this script through the `annotate.config` file.

`app_profiler.py`	Profile Android apps. It prepares the profiling environment, pushes the Simpleperf executable to an Android device, generates the `perf.data` file on the device, then returns the generated file to the host. You can configure the behavior of this script through the `app_profiler.config` file.
`binary_cache_builder.py`	Pull libraries from Android devices. This script is used by `app_profiler.py`.
`pprof_proto_generator.py`	Convert profiling data to the format used by `pprof (https://github.com/google/pprof)` .
`report.py`	Launches a graphical user interface to report profiling results.
`report_sample.py`	Generates `flamegraph (https://github.com/brendangregg/FlameGraph)` output.
`simpleperf_report_lib.py`	Provides a Python interface for parsing profiling data. You can use this script to transform profiling data in `perf.data` to other formats. The `report_sample.py` provided in Simpleperf shows an example of how to do this.