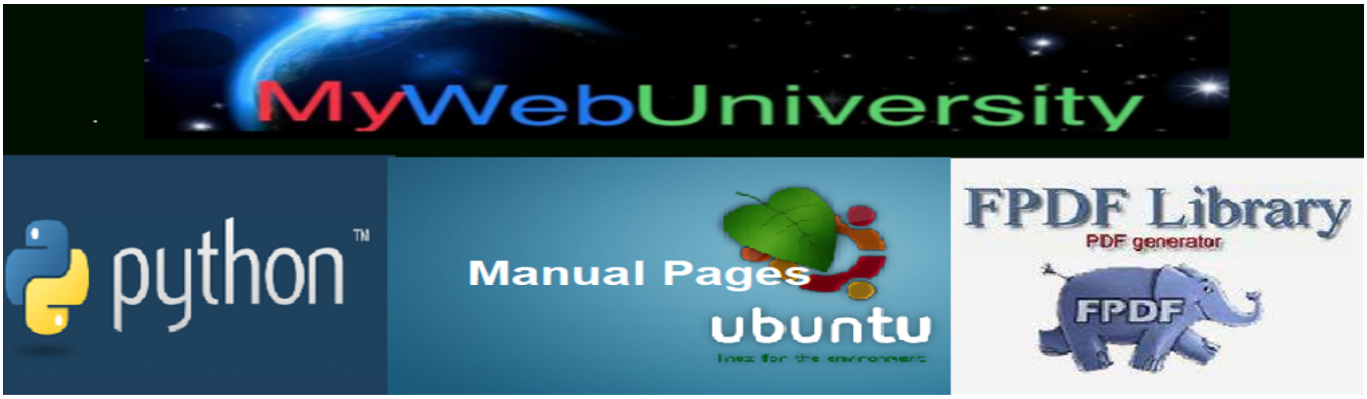

Rocky Enterprise Linux 9.2 Manual Pages on command 'iucode-tool.8'

$ man iucode-tool.8

IUCODE_TOOL(8) iucode_tool manual IUCODE_TOOL(8)

NAME

iucode_tool - Tool to manipulate Intel? IA?32/X86?64 microcode bundles

SYNOPSIS

iucode_tool [options] [[-ttype] filename|dirname] ...

DESCRIPTION

iucode_tool is an utility that can load Intel? processor microcode data from files in both text and binary microcode bundle formats.

It can output a list of the microcodes in these files, merge them, upload them to the ker? nel (to upgrade the microcode in the system processor cores) or write some of them out to a file in binary format for later use.

iucode_tool will load all microcodes in the specified files and directories to memory, in order to process them. Duplicated and outdated microcodes will be discarded. It can read microcode data from standard input (stdin), by specifying a file name of ?-? (minus sign).

Microcode data files are assumed to be in .dat text format if they have a .dat suffix, and to be in binary format otherwise. Standard input (stdin) is assumed to be in .dat text format. The -t option can be used to change the type of the files specified after it, in? cluding for stdin.

If a directory is specified, all files whose names do not begin with a dot will be loaded, in unspecified order. Nested directories are skipped.

Empty files and directories are ignored, and will be skipped.

You can select which microcodes should be written out, listed or uploaded to the kernel using the -S, -s, --date-before and --date-after options. Should none of those options be

specified, all microcodes will be selected.

You can upload the selected microcodes to the kernel, write them out to a file (in binary format), to a Linux early initramfs archive, to per?processor?signature files in a direc? tory, or to per?microcode files in a directory using the -w, --write-earlyfw, -k, -K, and -W options.

iucode_tool will identify microcodes in its output and error messages using a ?n/k? nota? tion, where ?n? is the bundle number, and ?k? is the microcode number within that bundle.

The output of the --list-all option when processing multiple input files is the best exam? ple of how it works.

For more information about Intel processor microcodes, please read the included documenta? tion and the Intel manuals listed in the SEE ALSO section.

OPTIONS

iucode_tool accepts the following options:

-q, --quiet

Inhibit usual output.

-v, --verbose

Print more information. Use more than once for added verbosity.

-h, -?, --help

List all available options and their meanings.

--usage

Show summary of options.

-V, --version

Show version of program.

-t type

Sets the file type of the following files. type can be:

b binary format. This is the same format used by the kernel driver and the BIOS/EFI, which is described in detail by the Intel 64 and IA?32 Architec? tures Software Developer's Manual, Volume 3A, section 9.11.

d Intel microcode .dat text format. This is the format normally used by Intel to distribute microcode data files.

r recover microcode in binary format. Search uncompressed generic binary files for microcodes in Intel microcode binary format to recover. Note: It can find microcode that will not pass strict checks, and thus cause iu?

code_tool to exit if the --no-strict-checks or --ignore-broken options are not in effect.

a (default) iucode_tool will use the suffix of the file name to select the file type: .dat text format for files that have a .dat suffix, and binary type otherwise. Note that for stdin, .dat text format is assumed.

--downgrade

When multiple versions of the microcode for a specific processor are available from different files, keep the one from the file loaded last, regardless of revision levels. Files are always loaded in the order they were specified in the command line. This option has no effect when just one file has been loaded.

--no-downgrade

When multiple versions of the microcode for a specific processor are available from different files, keep the one with the highest revision level. This is the default mode of operation.

--strict-checks

Perform strict checks on the microcode data. It will refuse to load microcodes and microcode data files with unexpected size and metadata. It will also refuse to load microcode entries that have the same metadata, but different payload. This is the default mode of operation.

--no-strict-checks

Perform less strict checks on the microcode data. Use only if you happen to come across a microcode data file that has microcodes with weird sizes or incorrect non? critical metadata (such as invalid dates), which you want to retain. If you just want to skip those, use the --ignore-broken option.

--ignore-broken

Skip broken microcode entries when loading a microcode data file, instead of abort? ing program execution. If the microcode entry has an unsupported format or had its header severely corrupted, all remaining data in the file will have to be ignored. In that case, using a file type of recover microcode in binary format (-tr option) is recommended, as it can skip over badly mangled microcode data.

--no-ignore-broken

Abort program execution if a broken microcode is found while loading a microcode data file. This is the default mode of operation.

-s ! | [!]signature[,[pf_mask][,[lt:|eq:|gt:]revision]]

Select microcodes by the specified signature, processor flags mask (pf_mask), and revision.

If the processor flags mask is specified, it will select only microcodes that are suitable for at least one of the processor flag combinations present in the mask.

If the revision is specified, optionally prefixed by one of the ?eq:?, ?lt:? or ?gt:? operators, it will select only microcodes that have that same revision (if no operator, or if the ?eq:? operator is used), or microcodes that have a revision that is less than (?lt:? operator), or greater than (?gt:? operator), the one spec? ified.

Specify more than once to select more microcodes. This option can be combined with the --scan-system option to select more microcodes. If signature is prefixed with a ?!? (exclamation mark), it will deselect microcodes instead. Ordering matters, with later -s options overriding earlier ones, including --scan-system.

When specifying signature and pf_mask, hexadecimal numbers must be prefixed with ?0x?, and octal numbers with ?0?. Decimal numbers must not have leading zeros, otherwise they would be interpreted as octal numbers.

The special notation -s! (with no signature parameter) instructs iucode_tool to re? quire explicit inclusion of microcode signatures (using the non-negated form of -s, or using --scan-system).

-S, --scan-system[=mode]

Select microcodes by scanning online processors on this system for their signa? tures.

This option can be used only once, and it can be combined with the -s option to se? lect more microcodes. The microcodes selected by --scan-system can also be dese? lected by a later -s !signature option.

The optional mode argument (accepted only by the long version of the option) se? lects the strategy used to scan processors:

0 or auto

Currently the same as fast, but this might change in future versions if In? tel ever deploys multi?signature systems that go beyond mixed?stepping. This is the default mode of operation, for backwards compatibility with pre? vious versions of iucode_tool.

1 or fast

Uses the cpuid instruction to detect the signature of the processor iu? code_tool is running on, and selects all steppings for that processor's type, family and model. Supports mixed?stepping systems.

2 or exact

Uses kernel drivers to scan the signature of every online processor di? rectly. This mode supports multi?signature systems. This scan mode will be slow on large systems with many processors, and likely requires special per? missions (such as running as the root user). Should the scan fail for any reason, as a fail?safe measure, it will issue an warning and consider all possible steppings for every signature it did manage to scan successfully.

--date-before=YYYY-MM-DD and --date-after=YYYY-MM-DD

Limit the selected microcodes by a date range. The date must be given in ISO for? mat, with four digits for the year and two digits for the month and day and ?-? (minus sign) for the separator. Dates are not range?checked, so you can use --date-after=2000-00-00 to select all microcodes dated since January 1st, 2000.

--loose-date-filtering

When a date range is specified, all revisions of the microcode will be considered for selection (ignoring just the date range, all other filters still apply) should any of the microcode's revisions be within the date range.

--strict-date-filtering

When a date range is specified, select only microcodes which are within the date range. This is the default mode of operation.

-l, --list

List selected microcode signatures to standard output (stdout).

-L, --list-all

List all microcode signatures while they're being processed to standard output (stdout).

-k[device], --kernel[=device]

Upload selected microcodes to the kernel. Optionally, the device path can be spec? ified (default: /dev/cpu/microcode). This update method is deprecated: it will be removed eventually from the kernel and from iucode_tool.

-K[directory], --write-firmware[=directory]

Write selected microcodes with the file names expected by the Linux kernel firmware loader. Optionally, the destination directory can be specified (default: /lib/firmware/intel?ucode).

-wfile, --write-to=file

Write selected microcodes to a file in binary format.

--write-earlyfw=file

Write selected microcodes to an early initramfs archive, which should be prepended to the regular initramfs to allow the kernel to update processor microcode very early during system boot.

-Wdirectory, --write-named-to=directory

Write selected microcodes to the specified directory, one microcode per file, in binary format. The file names reflect the microcode signature, processor flags mask and revision.

--write-all-named-to=directory

Write every microcode to the specified directory, one microcode per file, in binary format. The file names reflect the microcode signature, processor flags mask and revision. This is the only way to write out every revision of the same microcode.

--overwrite

Remove the destination file before writing, if it exists and is not a directory. The destination file is not overwritten in?place. Hardlinks will be severed, and any existing access permissions, ACLs and other extended attributes of the old des? tination file will be lost.

--no-overwrite

Abort if the destination file already exists. This is the default mode of opera? tion. Do note that iucode_tool does not follow non?directory symlinks when writing files.

--mini-earlyfw

Optimize the early initramfs cpio container for minimal size. It will change the cpio block size to 16 bytes, and remove header entries for the parent directories of the microcode data file. As a result, the microcode data file will not be available to the regular initramfs, and tools might complain about the non?standard cpio block size.

This will typically reduce the early initramfs size by 736 bytes.

--normal-earlyfw

Optimize the early initramfs size for tool compatibility. This is the default mode of operation. The microcode data file will be available inside the regular initramfs as well.

NOTES

iucode_tool reads all data to memory before doing any processing. It enforces a sanity limit of a maximum of 1GiB worth of binary microcode data per microcode data file.

All informational and error messages are sent to standard error (stderr), while user?re? quested output (such as output generated by the list options) is sent to standard output (stdout).

iucode_tool creates files with permissions 0644 (rw-r--r--), modified by the current umask.

iucode_tool's selected microcode listing and microcode output files are sorted first by processor signature (in ascending order), and then by processor flags mask (in descending order).

When multiple revisions of a microcode are selected, the older ones will be skipped. Only the newest selected revision of a microcode (or the last one in load order when the --downgrade option is active) will be written to a file or uploaded to the kernel.

Intel microcode data files, both in binary and text formats, can be concatenated to gener? ate a bigger and still valid microcode data file.

iucode_tool does not follow symlinks when writing microcode data files. It will either refuse to write the file and abort (default mode of operation), or (when the --overwrite option is active) it will remove the target symlink or file (and therefore breaking hardlinks) before writing the new file.

iucode_tool does follow directory symlinks to locate the directory to write files into.

Linux Notes

Before Linux v4.4, the microcode update driver was split in two parts: the early microcode update driver (which gets microcode data from the initramfs) and the late microcode update driver, which could be a module and got microcode data from the firmware subsystem. The two drivers were unified in Linux v4.4.

The microcode update driver needs to be present in the system at all times to ensure mi? crocode updates are reapplied on resume from suspend and CPU hotplug. Do not unload the microcode module, unless you really know better. Since Linux v4.4, the late microcode

driver cannot be a module anymore and will always be present in the system when enabled.

Updating microcode early is safer. It can only be done at boot and it requires an initramfs, but it is strongly recommended: late microcode updates (which read microcode data from /lib/firmware) cannot safely change visible processor features.

Early microcode updates are available since Linux v3.9. They can safely change visible processor features (such as the microcode updates that disabled Intel TSX instructions on Intel Haswell cores do). They require an uncompressed initramfs image with the microcode update data in /kernel/x86/microcode/GenuineIntel.bin. This uncompressed initramfs image must come before any compressed initramfs image(s), and it has an special name: early initramfs.

The microcode update data inside the early initramfs image must be aligned to a 16?byte boundary due to a bug in several versions of the Linux kernel early microcode update driver. This requires special steps when creating the initramfs archive with the mi? crocode data, and will be handled automatically by the iucode_tool --write-earlyfw option.

Since Linux v4.2, it is also possible to build a kernel with the microcode update data as built?in firmware, using the CONFIG_FIRMWARE_IN_KERNEL facility. This feature is not yet mature as of Linux v4.2.8, v4.4.11, v4.5.5 and v4.6, and might not work in every case.

The /dev/cpu/microcode update interface has been deprecated and should not be used. It has one special requirement: each write syscall must contain whole microcode(s). It can be accessed through iucode_tool --kernel.

Up to Linux v3.5, late microcode updates were required to be triggered per?core, by writ? ing the number 1 to /sys/devices/system/cpu/*/microcode/reload for every cpu. Depending on kernel version, you must either trigger it on every core to avoid a dangerous situation where some cores are using outdated microcode, or the kernel will accept the request only for the boot processor and use it to trigger an update on all system processor cores.

Since Linux v3.6, the late microcode update driver has a new interface that explicitly triggers an update for every core at once when the number 1 is written to /sys/de? vices/system/cpu/microcode/reload.

EXAMPLES

Updating files in /lib/firmware/intel?ucode:

```
iucode_tool -K/lib/firmware/intel?ucode \
        /lib/firmware/intel?ucode \
        /tmp/file-with-new-microcodes.bin
```

Processing several compressed files at once:

zcat intel-microcode*.dat.gz | iucode_tool -l -

zcat intel-microcode*.bin.gz | iucode_tool -l -tb -

Selecting microcodes and creating an early initramfs:

```
iucode_tool --scan-system \
        --write-earlyfw=/tmp/early.cpio \
        /lib/firmware/intel-ucode
iucode_tool -s 0x106a5 -s 0x106a4 -l /lib/firmware/intel-ucode
```

Using the recovery loader to load and to update microcode in an early initramfs:

```
iucode_tool -L -tr /boot/intel-ucode.img
iucode_tool -Ll -S --write-earlyfw=/boot/intel-ucode.img.new \
        -tr /boot/intel-ucode.img -tb /lib/firmware/intel-ucode && \
    mv /boot/intel-ucode.img.new /boot/intel-ucode.img
```

BUGS

Microcode with negative revision numbers is not special?cased, and will not be preferred over regular microcode.

The downgrade mode should be used only for microcodes with the same processor flags mask. It cannot handle the corner cases where modifying a processor flags mask would be required to force the kernel to load a lower revision of a microcode, and iucode_tool will issue an warning when that happens. So far, this has not proved to be a relevant limitation as changes to the processor flags mask of post?launch, production microcode updates are very rare.

The loader version microcode metadata field is ignored by iucode_tool. This shouldn't cause problems as long as the same signature never needs more than a single type of loader.

Files are not replaced atomically: if iucode_tool is interrupted while writing to a file, that file will be corrupted.

SEE ALSO

The Intel 64 and IA?32 Architectures Software Developer's Manual, Volume 3A: System Pro?gramming Guide, Part 1 (order number 253668), section 9.11.

AUTHOR

Henrique de Moraes Holschuh <hmh@hmh.eng.br>