Rocky Enterprise Linux 9.2 Manual Pages on command 'x86_64-linux-gnu-objcopy.1'

C:\>man x86_64-linux-gnu-objcopy.1

OBJCOPY(1) GNU Development Tools OBJCOPY(1)

NAME

objcopy - copy and translate object files

SYNOPSIS

objcopy [-F bfdname|--target=bfdname]

[-I bfdname|--input-target=bfdname]

[-O bfdname|--output-target=bfdname]

[-B bfdarch|--binary-architecture=bfdarch]

[-S|--strip-all]

[-g|--strip-debug]

[--strip-unneeded]

[-K symbolname|--keep-symbol=symbolname]

[-N symbolname|--strip-symbol=symbolname]

[--strip-unneeded-symbol=symbolname]

[-G symbolname|--keep-global-symbol=symbolname]

[--localize-hidden]

[-L symbolname|--localize-symbol=symbolname]

[--globalize-symbol=symbolname]

[--globalize-symbols=filename]

[-W symbolname|--weaken-symbol=symbolname]

[-w|--wildcard]

[-x|--discard-all]

[-X|--discard-locals]

[-b byte|--byte=byte]

[-i [breadth]|--interleave[=breadth]]

[--interleave-width=width]

[-j sectionpattern|--only-section=sectionpattern]

[-R sectionpattern|--remove-section=sectionpattern]

[--keep-section=sectionpattern]

[--remove-relocations=sectionpattern]

[-p|--preserve-dates]

[-D|--enable-deterministic-archives]

[-U|--disable-deterministic-archives]

[--debugging]

[--gap-fill=val]

[--pad-to=address]

[--set-start=val]

[--adjust-start=incr]

[--change-addresses=incr]

[--change-section-address sectionpattern{=,+,-}val]

[--change-section-lma sectionpattern{=,+,-}val]

[--change-section-vma sectionpattern{=,+,-}val]

[--change-warnings] [--no-change-warnings]

[--set-section-flags sectionpattern=flags]

[--set-section-alignment sectionpattern=align]

[--add-section sectionname=filename]

[--dump-section sectionname=filename]

[--update-section sectionname=filename]

[--rename-section oldname=newname[,flags]]

[--long-section-names {enable,disable,keep}]

[--change-leading-char] [--remove-leading-char]

[--reverse-bytes=num]

[--srec-len=ival] [--srec-forceS3]

[--redefine-sym old=new]

[--redefine-syms=filename]

[--weaken]

[--keep-symbols=filename]

[--strip-symbols=filename]

[--strip-unneeded-symbols=filename]

[--keep-global-symbols=filename]

[--localize-symbols=filename]

[--weaken-symbols=filename]

[--add-symbol name=[section:]value[,flags]]

[--alt-machine-code=index]

[--prefix-symbols=string]

[--prefix-sections=string]

[--prefix-alloc-sections=string]

[--add-gnu-debuglink=path-to-file]

[--keep-file-symbols]

[--only-keep-debug]

[--strip-dwo]

[--extract-dwo]

[--extract-symbol]

[--writable-text]

[--readonly-text]

[--pure]

[--impure]

[--file-alignment=num]

[--heap=size]

[--image-base=address]

[--section-alignment=num]

[--stack=size]

[--subsystem=which:major.minor]

[--compress-debug-sections]

[--decompress-debug-sections]

[--elf-stt-common=val]

[--merge-notes]

[--no-merge-notes]

[--verilog-data-width=val]

[-v|--verbose]

[-V|--version]

[--help] [--info]

infile [outfile]

DESCRIPTION

The GNU objcopy utility copies the contents of an object file to another. objcopy uses the GNU BFD Library to read and write the object files. It can write the destination object file in a format different from that of the source object file.

The exact behavior of objcopy is controlled by command-line options. Note that objcopy should be able to copy a fully linked file between any two formats. However, copying a relocatable object file between any two formats may not work as expected.

objcopy creates temporary files to do its translations and deletes them afterward.

objcopy uses BFD to do all its translation work; it has access to all the formats described in BFD and thus is able to recognize most formats without being told explicitly.

objcopy can be used to generate S-records by using an output target of srec (e.g., use -O srec).

objcopy can be used to generate a raw binary file by using an output target of binary (e.g., use -O binary). When objcopy generates a raw binary file, it will essentially produce a memory dump of the contents of the input object file. All symbols and relocation information will be discarded. The memory dump will start at the load address of the lowest section copied into the output file.

When generating an S-record or a raw binary file, it may be helpful to use -S to remove sections containing debugging information. In some cases -R will be useful to remove sections which contain information that is not needed by the binary file.

Note---objcopy is not able to change the endianness of its input files. If the input format has an endianness (some formats do not), objcopy can only copy the inputs into file formats that have the same endianness or which have no endianness (e.g., srec). (However, see the --reverse-bytes option.)

OPTIONS

infile

outfile

The input and output files, respectively. If you do not specify outfile, objcopy creates a temporary file and destructively renames the result with the name of infile.

-I bfdname

--input-target=bfdname

Consider the source file's object format to be bfdname, rather than attempting to deduce it.

-O bfdname

--output-target=bfdname

Write the output file using the object format bfdname.

-F bfdname

--target=bfdname

Use bfdname as the object format for both the input and the output file; i.e., simply transfer data from source to destination with no translation.

-B bfdarch

--binary-architecture=bfdarch

Useful when transforming a architecture-less input file into an object file. In this case the output architecture can be set to bfdarch. This option will be ignored if the input file has a known bfdarch. You can access this binary data inside a program by referencing the special symbols that are created by the conversion process. These symbols are called _binary_objfile_start, _binary_objfile_end and _binary_objfile_size. e.g. you can transform a picture file into an object file and then access it in your code using these symbols.

-j sectionpattern

--only-section=sectionpattern

Copy only the indicated sections from the input file to the output file. This option may be given more than once. Note that using this option inappropriately may make the output file unusable. Wildcard characters are accepted in sectionpattern.

If the first character of sectionpattern is the exclamation point (!) then matching sections will not be copied, even if earlier use of --only-section on the same command line would otherwise copy it. For example:

--only-section=.text.* --only-section=!.text.foo

will copy all sectinos maching '.text.*' but not the section '.text.foo'.

-R sectionpattern

--remove-section=sectionpattern

Remove any section matching sectionpattern from the output file. This option may be given more than once. Note that using this option inappropriately may make the output file unusable. Wildcard characters are accepted in sectionpattern. Using both the -j and -R options together results in undefined behaviour.

If the first character of sectionpattern is the exclamation point (!) then matching sections will not be removed even if an earlier use of --remove-section on the same command line would otherwise remove it. For example:

--remove-section=.text.* --remove-section=!.text.foo

will remove all sections matching the pattern '.text.*', but will not remove the section '.text.foo'.

--keep-section=sectionpattern

When removing sections from the output file, keep sections that match sectionpattern.

--remove-relocations=sectionpattern

Remove non-dynamic relocations from the output file for any section matching sectionpattern. This option may be given more than once. Note that using this option inappropriately may make the output file unusable, and attempting to remove a dynamic relocation section such as .rela.plt from an executable or shared library with --remove-relocations=.plt will not work. Wildcard characters are accepted in sectionpattern. For example:

--remove-relocations=.text.*

will remove the relocations for all sections matching the pattern '.text.*'.

If the first character of sectionpattern is the exclamation point (!) then matching sections will not have their relocation removed even if an earlier use of --remove-relocations on the same command line would otherwise cause the relocations to be removed. For example:

--remove-relocations=.text.* --remove-relocations=!.text.foo

will remove all relocations for sections matching the pattern '.text.*', but will not remove relocations for the section '.text.foo'.

-S

--strip-all

Do not copy relocation and symbol information from the source file.

-g

--strip-debug

Do not copy debugging symbols or sections from the source file.

--strip-unneeded

Strip all symbols that are not needed for relocation processing.

-K symbolname

--keep-symbol=symbolname

When stripping symbols, keep symbol symbolname even if it would normally be stripped. This option may be given more than once.

-N symbolname

--strip-symbol=symbolname

Do not copy symbol symbolname from the source file. This option may be given more than once.

--strip-unneeded-symbol=symbolname

Do not copy symbol symbolname from the source file unless it is needed by a relocation. This option may be given more than once.

-G symbolname

--keep-global-symbol=symbolname

Keep only symbol symbolname global. Make all other symbols local to the file, so that they are not visible externally. This option may be given more than once. Note: this option cannot be used in conjunction with the --globalize-symbol or --globalize-symbols options.

--localize-hidden

In an ELF object, mark all symbols that have hidden or internal visibility as local. This option applies on top of symbol-specific localization options such as -L.

-L symbolname

--localize-symbol=symbolname

Convert a global or weak symbol called symbolname into a local symbol, so that it is not visible externally. This option may be given more than once. Note - unique symbols are not converted.

-W symbolname

--weaken-symbol=symbolname

Make symbol symbolname weak. This option may be given more than once.

--globalize-symbol=symbolname

Give symbol symbolname global scoping so that it is visible outside of the file in which it is defined. This option may be given more than once. Note: this option cannot be used in conjunction with the -G or --keep-global-symbol options.

-w

--wildcard

Permit regular expressions in symbolnames used in other command line options. The question mark (?), asterisk (*), backslash (\) and square brackets ([]) operators can be used anywhere in the symbol name. If the first character of the symbol name is the exclamation point (!) then the sense of the switch is reversed for that symbol. For example:

```
-w -W !foo -W fo*
```

would cause objcopy to weaken all symbols that start with "fo" except for the symbol "foo".

-x

--discard-all

Do not copy non-global symbols from the source file.

-X

--discard-locals

Do not copy compiler-generated local symbols. (These usually start with L or ..)

-b byte

--byte=byte

If interleaving has been enabled via the --interleave option then start the range of bytes to keep at the byteth byte. byte can be in the range from 0 to breadth-1, where breadth is the value given by the --interleave option.

-i [breadth]

--interleave[=breadth]

Only copy a range out of every breadth bytes. (Header data is not affected). Select which byte in the range begins the copy with the --byte option. Select the width of the range with the --interleave-width option.

This option is useful for creating files to program ROM. It is typically used with an "srec" output target. Note that objcopy will complain if you do not specify the --byte option as well.

The default interleave breadth is 4, so with --byte set to 0, objcopy would copy the first byte out of every four bytes from the input to the output.

--interleave-width=width

When used with the --interleave option, copy width bytes at a time. The start of the range of bytes to be copied is set by the --byte option, and the extent of the range is set with the --interleave option.

The default value for this option is 1. The value of width plus the byte value set by the --byte option must not exceed the interleave breadth set by the --interleave option.

This option can be used to create images for two 16-bit flashes interleaved in a 32-bit bus by passing -b 0 -i 4 --interleave-width=2 and -b 2 -i 4 --interleave-width=2 to two objcopy commands. If the input was '12345678' then the outputs would be '1256' and '3478' respectively.

-p

--preserve-dates

Set the access and modification dates of the output file to be the same as those of the input file.

-D

--enable-deterministic-archives

Operate in deterministic mode. When copying archive members and writing the archive index, use zero for UIDs, GIDs, timestamps, and use consistent file modes for all files.

If binutils was configured with --enable-deterministic-archives, then this mode is on by default. It can be disabled with the -U option, below.

-U

--disable-deterministic-archives

Do not operate in deterministic mode. This is the inverse of the -D option, above: when copying archive members and writing the archive index, use their actual UID, GID, timestamp, and file mode values.

This is the default unless binutils was configured with --enable-deterministic-archives.

--debugging

Convert debugging information, if possible. This is not the default because only certain debugging formats are supported, and the conversion process can be time consuming.

--gap-fill val

Fill gaps between sections with val. This operation applies to the load address (LMA) of the sections. It is done by increasing the size of the section with the lower address, and filling in the extra space created with val.

--pad-to address

Pad the output file up to the load address address. This is done by increasing the size of the last section. The extra space is filled in with the value specified by --gap-fill (default zero).

--set-start val

Set the start address of the new file to val. Not all object file formats support setting the start address.

--change-start incr

--adjust-start incr

Change the start address by adding incr. Not all object file formats support setting the start address.

--change-addresses incr

--adjust-vma incr

Change the VMA and LMA addresses of all sections, as well as the start address, by adding incr. Some object file formats do not permit section addresses to be changed arbitrarily. Note that this does not relocate the sections; if the program expects sections to be loaded at a certain address, and this option is used to change the sections such that they are loaded at a different address,

the program may fail.

--change-section-address sectionpattern{=,+,-}val

--adjust-section-vma sectionpattern{=,+,-}val

Set or change both the VMA address and the LMA address of any section matching sectionpattern. If = is used, the section address is set to val. Otherwise, val is added to or subtracted from the section address. See the comments under --change-addresses, above. If sectionpattern does not match any sections in the input file, a warning will be issued, unless --no-change-warnings is used.

--change-section-lma sectionpattern{=,+,-}val

Set or change the LMA address of any sections matching sectionpattern. The LMA address is the address where the section will be loaded into memory at program load time. Normally this is the same as the VMA address, which is the address of the section at program run time, but on some systems, especially those where a program is held in ROM, the two can be different. If = is used, the section address is set to val. Otherwise, val is added to or subtracted from the section address. See the comments under --change-addresses, above. If sectionpattern does not match any sections in the input file, a warning will be issued, unless --no-change-warnings is used.

--change-section-vma sectionpattern{=,+,-}val

Set or change the VMA address of any section matching sectionpattern. The VMA address is the address where the section will be located once the program has started executing. Normally this is the same as the LMA address, which is the address where the section will be loaded into memory, but on some systems, especially those where a program is held in ROM, the two can be different. If = is used, the section address is set to val. Otherwise, val is added to or subtracted from the section address. See the comments under --change-addresses, above. If sectionpattern does not match any sections in the input file, a warning will be issued, unless --no-change-warnings is used.

--change-warnings

--adjust-warnings

If --change-section-address or --change-section-lma or --change-section-vma is used, and the section pattern does not match any sections, issue a warning. This is the default.

--no-change-warnings

--no-adjust-warnings

Do not issue a warning if --change-section-address or --adjust-section-lma or --adjust-section-vma is used, even if the section pattern does not match any sections.

--set-section-flags sectionpattern=flags

Set the flags for any sections matching sectionpattern. The flags argument is a comma separated string of flag names. The recognized names are alloc, contents, load, noload, readonly, code, data, rom, share, and debug. You can set the contents flag for a section which does not have contents, but it is not meaningful to clear the contents flag of a section which does have contents--just remove the section instead. Not all flags are meaningful for all object file formats.

--set-section-alignment sectionpattern=align

Set the alignment for any sections matching sectionpattern. align specifies the alignment in bytes and must be a power of two, i.e. 1, 2, 4, 8....

--add-section sectionname=filename

Add a new section named sectionname while copying the file. The contents of the new section are taken from the file filename. The size of the section will be the size of the file. This option only works on file formats which can support sections with arbitrary names. Note - it may be necessary to use the --set-section-flags option to set the attributes of the newly created section.

--dump-section sectionname=filename

Place the contents of section named sectionname into the file filename, overwriting any contents that may have been there previously. This option is the inverse of --add-section. This option is similar to the --only-section option except that it does not create a formatted file, it just dumps the contents as raw binary data, without applying any relocations. The option can be specified more than once.

--update-section sectionname=filename

Replace the existing contents of a section named sectionname with the contents of file filename. The size of the section will be adjusted to the size of the file. The section flags for sectionname will be unchanged. For ELF format

files the section to segment mapping will also remain unchanged, something which is not possible using --remove-section followed by --add-section. The option can be specified more than once.

Note - it is possible to use --rename-section and --update-section to both update and rename a section from one command line. In this case, pass the original section name to --update-section, and the original and new section names to --rename-section.

--add-symbol name=[section:]value[,flags]

Add a new symbol named name while copying the file. This option may be specified multiple times. If the section is given, the symbol will be associated with and relative to that section, otherwise it will be an ABS symbol. Specifying an undefined section will result in a fatal error. There is no check for the value, it will be taken as specified. Symbol flags can be specified and not all flags will be meaningful for all object file formats. By default, the symbol will be global. The special flag 'before=othersym' will insert the new symbol in front of the specified othersym, otherwise the symbol(s) will be added at the end of the symbol table in the order they appear.

--rename-section oldname=newname[,flags]

Rename a section from oldname to newname, optionally changing the section's flags to flags in the process. This has the advantage over using a linker script to perform the rename in that the output stays as an object file and does not become a linked executable.

This option is particularly helpful when the input format is binary, since this will always create a section called .data. If for example, you wanted instead to create a section called .rodata containing binary data you could use the following command line to achieve it:

```
objcopy -I binary -O <output_format> -B <architecture> \
 --rename-section .data=.rodata,alloc,load,readonly,data,contents \
 <input_binary_file> <output_object_file>
```

--long-section-names {enable,disable,keep}

Controls the handling of long section names when processing "COFF" and "PE-COFF" object formats. The default behaviour, keep, is to preserve long

section names if any are present in the input file. The enable and disable options forcibly enable or disable the use of long section names in the output object; when disable is in effect, any long section names in the input object will be truncated. The enable option will only emit long section names if any are present in the inputs; this is mostly the same as keep, but it is left undefined whether the enable option might force the creation of an empty string table in the output file.

--change-leading-char

Some object file formats use special characters at the start of symbols. The most common such character is underscore, which compilers often add before every symbol. This option tells objcopy to change the leading character of every symbol when it converts between object file formats. If the object file formats use the same leading character, this option has no effect. Otherwise, it will add a character, or remove a character, or change a character, as appropriate.

--remove-leading-char

If the first character of a global symbol is a special symbol leading character used by the object file format, remove the character. The most common symbol leading character is underscore. This option will remove a leading underscore from all global symbols. This can be useful if you want to link together objects of different file formats with different conventions for symbol names. This is different from --change-leading-char because it always changes the symbol name when appropriate, regardless of the object file format of the output file.

--reverse-bytes=num

Reverse the bytes in a section with output contents. A section length must be evenly divisible by the value given in order for the swap to be able to take place. Reversing takes place before the interleaving is performed.

This option is used typically in generating ROM images for problematic target systems. For example, on some target boards, the 32-bit words fetched from 8-bit ROMs are re-assembled in little-endian byte order regardless of the CPU byte order. Depending on the programming model, the endianness of the ROM may need to be modified.

Consider a simple file with a section containing the following eight bytes: 12345678.

Using --reverse-bytes=2 for the above example, the bytes in the output file would be ordered 21436587.

Using --reverse-bytes=4 for the above example, the bytes in the output file would be ordered 43218765.

By using --reverse-bytes=2 for the above example, followed by --reverse-bytes=4 on the output file, the bytes in the second output file would be ordered 34127856.

--srec-len=ival

Meaningful only for srec output. Set the maximum length of the Srecords being produced to ival. This length covers both address, data and crc fields.

--srec-forceS3

Meaningful only for srec output. Avoid generation of S1/S2 records, creating S3-only record format.

--redefine-sym old=new

Change the name of a symbol old, to new. This can be useful when one is trying link two things together for which you have no source, and there are name collisions.

--redefine-syms=filename

Apply --redefine-sym to each symbol pair "old new" listed in the file filename. filename is simply a flat file, with one symbol pair per line. Line comments may be introduced by the hash character. This option may be given more than once.

--weaken

Change all global symbols in the file to be weak. This can be useful when building an object which will be linked against other objects using the -R option to the linker. This option is only effective when using an object file format which supports weak symbols.

--keep-symbols=filename

Apply --keep-symbol option to each symbol listed in the file filename. filename is simply a flat file, with one symbol name per line. Line comments may be introduced by the hash character. This option may be given more than

once.

--strip-symbols=filename

Apply --strip-symbol option to each symbol listed in the file filename. filename is simply a flat file, with one symbol name per line. Line comments may be introduced by the hash character. This option may be given more than once.

--strip-unneeded-symbols=filename

Apply --strip-unneeded-symbol option to each symbol listed in the file filename. filename is simply a flat file, with one symbol name per line. Line comments may be introduced by the hash character. This option may be given more than once.

--keep-global-symbols=filename

Apply --keep-global-symbol option to each symbol listed in the file filename. filename is simply a flat file, with one symbol name per line. Line comments may be introduced by the hash character. This option may be given more than once.

--localize-symbols=filename

Apply --localize-symbol option to each symbol listed in the file filename. filename is simply a flat file, with one symbol name per line. Line comments may be introduced by the hash character. This option may be given more than once.

--globalize-symbols=filename

Apply --globalize-symbol option to each symbol listed in the file filename. filename is simply a flat file, with one symbol name per line. Line comments may be introduced by the hash character. This option may be given more than once. Note: this option cannot be used in conjunction with the -G or --keep-global-symbol options.

--weaken-symbols=filename

Apply --weaken-symbol option to each symbol listed in the file filename. filename is simply a flat file, with one symbol name per line. Line comments may be introduced by the hash character. This option may be given more than once.

--alt-machine-code=index

If the output architecture has alternate machine codes, use the indexth code instead of the default one. This is useful in case a machine is assigned an official code and the tool-chain adopts the new code, but other applications still depend on the original code being used. For ELF based architectures if the index alternative does not exist then the value is treated as an absolute number to be stored in the e_machine field of the ELF header.

--writable-text

Mark the output text as writable. This option isn't meaningful for all object file formats.

--readonly-text

Make the output text write protected. This option isn't meaningful for all object file formats.

--pure

Mark the output file as demand paged. This option isn't meaningful for all object file formats.

--impure

Mark the output file as impure. This option isn't meaningful for all object file formats.

--prefix-symbols=string

Prefix all symbols in the output file with string.

--prefix-sections=string

Prefix all section names in the output file with string.

--prefix-alloc-sections=string

Prefix all the names of all allocated sections in the output file with string.

--add-gnu-debuglink=path-to-file

Creates a .gnu_debuglink section which contains a reference to path-to-file and adds it to the output file. Note: the file at path-to-file must exist. Part of the process of adding the .gnu_debuglink section involves embedding a checksum of the contents of the debug info file into the section.

If the debug info file is built in one location but it is going to be installed at a later time into a different location then do not use the path to the installed location. The --add-gnu-debuglink option will fail because the installed file does not exist yet. Instead put the debug info file in the

current directory and use the --add-gnu-debuglink option without any directory components, like this:

objcopy --add-gnu-debuglink=foo.debug

At debug time the debugger will attempt to look for the separate debug info file in a set of known locations. The exact set of these locations varies depending upon the distribution being used, but it typically includes:

"* The same directory as the executable."

"* A sub-directory of the directory containing the executable" called .debug

"* A global debug directory such as /usr/lib/debug."

As long as the debug info file has been installed into one of these locations before the debugger is run everything should work correctly.

--keep-file-symbols

When stripping a file, perhaps with --strip-debug or --strip-unneeded, retain any symbols specifying source file names, which would otherwise get stripped.

--only-keep-debug

Strip a file, removing contents of any sections that would not be stripped by --strip-debug and leaving the debugging sections intact. In ELF files, this preserves all note sections in the output.

Note - the section headers of the stripped sections are preserved, including their sizes, but the contents of the section are discarded. The section headers are preserved so that other tools can match up the debuginfo file with the real executable, even if that executable has been relocated to a different address space.

The intention is that this option will be used in conjunction with --add-gnu-debuglink to create a two part executable. One a stripped binary which will occupy less space in RAM and in a distribution and the second a debugging information file which is only needed if debugging abilities are required. The suggested procedure to create these files is as follows:

1.<Link the executable as normal. Assuming that it is called> "foo" then...

1.<Run "objcopy --only-keep-debug foo foo.dbg" to> create a file containing the debugging info.

1.<Run "objcopy --strip-debug foo" to create a>

stripped executable.

1.<Run "objcopy --add-gnu-debuglink=foo.dbg foo">

to add a link to the debugging info into the stripped executable.

Note---the choice of ".dbg" as an extension for the debug info file is arbitrary. Also the "--only-keep-debug" step is optional. You could instead do this:

1.<Link the executable as normal.>

1.<Copy "foo" to "foo.full">

1.<Run "objcopy --strip-debug foo">

1.<Run "objcopy --add-gnu-debuglink=foo.full foo">

i.e., the file pointed to by the --add-gnu-debuglink can be the full executable. It does not have to be a file created by the --only-keep-debug switch.

Note---this switch is only intended for use on fully linked files. It does not make sense to use it on object files where the debugging information may be incomplete. Besides the gnu_debuglink feature currently only supports the presence of one filename containing debugging information, not multiple filenames on a one-per-object-file basis.

--strip-dwo

Remove the contents of all DWARF .dwo sections, leaving the remaining debugging sections and all symbols intact. This option is intended for use by the compiler as part of the -gsplit-dwarf option, which splits debug information between the .o file and a separate .dwo file. The compiler generates all debug information in the same file, then uses the --extract-dwo option to copy the .dwo sections to the .dwo file, then the --strip-dwo option to remove those sections from the original .o file.

--extract-dwo

Extract the contents of all DWARF .dwo sections. See the --strip-dwo option for more information.

--file-alignment num

Specify the file alignment. Sections in the file will always begin at file offsets which are multiples of this number. This defaults to 512. [This

option is specific to PE targets.]

--heap reserve

--heap reserve,commit

Specify the number of bytes of memory to reserve (and optionally commit) to be used as heap for this program. [This option is specific to PE targets.]

--image-base value

Use value as the base address of your program or dll. This is the lowest memory location that will be used when your program or dll is loaded. To reduce the need to relocate and improve performance of your dlls, each should have a unique base address and not overlap any other dlls. The default is 0x400000 for executables, and 0x10000000 for dlls. [This option is specific to PE targets.]

--section-alignment num

Sets the section alignment field in the PE header. Sections in memory will always begin at addresses which are a multiple of this number. Defaults to 0x1000. [This option is specific to PE targets.]

--stack reserve

--stack reserve,commit

Specify the number of bytes of memory to reserve (and optionally commit) to be used as stack for this program. [This option is specific to PE targets.]

--subsystem which

--subsystem which:major

--subsystem which:major.minor

Specifies the subsystem under which your program will execute. The legal values for which are "native", "windows", "console", "posix", "efi-app", "efi-bsd", "efi-rtd", "sal-rtd", and "xbox". You may optionally set the subsystem version also. Numeric values are also accepted for which. [This option is specific to PE targets.]

--extract-symbol

Keep the file's section flags and symbols but remove all section data.

Specifically, the option:

*<removes the contents of all sections;>

*<sets the size of every section to zero; and>

*<sets the file's start address to zero.>

This option is used to build a .sym file for a VxWorks kernel. It can also be a useful way of reducing the size of a --just-symbols linker input file.

--compress-debug-sections

Compress DWARF debug sections using zlib with SHF_COMPRESSED from the ELF ABI. Note - if compression would actually make a section larger, then it is not compressed.

--compress-debug-sections=none

--compress-debug-sections=zlib

--compress-debug-sections=zlib-gnu

--compress-debug-sections=zlib-gabi

For ELF files, these options control how DWARF debug sections are compressed. --compress-debug-sections=none is equivalent to --decompress-debug-sections. --compress-debug-sections=zlib and --compress-debug-sections=zlib-gabi are equivalent to --compress-debug-sections. --compress-debug-sections=zlib-gnu compresses DWARF debug sections using zlib. The debug sections are renamed to begin with .zdebug instead of .debug. Note - if compression would actually make a section larger, then it is not compressed nor renamed.

--decompress-debug-sections

Decompress DWARF debug sections using zlib. The original section names of the compressed sections are restored.

--elf-stt-common=yes

--elf-stt-common=no

For ELF files, these options control whether common symbols should be converted to the "STT_COMMON" or "STT_OBJECT" type. --elf-stt-common=yes converts common symbol type to "STT_COMMON". --elf-stt-common=no converts common symbol type to "STT_OBJECT".

--merge-notes

--no-merge-notes

For ELF files, attempt (or do not attempt) to reduce the size of any SHT_NOTE type sections by removing duplicate notes.

-V

--version

Show the version number of objcopy.

--verilog-data-width=bytes

For Verilog output, this options controls the number of bytes converted for each output data element. The input target controls the endianness of the conversion.

-v

--verbose

Verbose output: list all object files modified. In the case of archives, objcopy -V lists all members of the archive.

--help

Show a summary of the options to objcopy.

--info

Display a list showing all architectures and object formats available.

@file

Read command-line options from file. The options read are inserted in place of the original @file option. If file does not exist, or cannot be read, then the option will be treated literally, and not removed.

Options in file are separated by whitespace. A whitespace character may be included in an option by surrounding the entire option in either single or double quotes. Any character (including a backslash) may be included by prefixing the character to be included with a backslash. The file may itself contain additional @file options; any such options will be processed recursively.

SEE ALSO

ld(1), objdump(1), and the Info entries for binutils.

COPYRIGHT