zip(1) - Linux man page

Name

zip - package and compress (archive) files

Synopsis

zip [-**aABcdDeEfFghjklLmoqrRSTuvVwXyz!@$**] [--longoption ...] [-**b** path] [-**n** suffixes] [-**t** date] [-**tt** date] [*zipfile* [*file* ...]] [**-xi** list]

zipcloak (see separate man page)

zipnote (see separate man page)

zipsplit (see separate man page)

Note: Command line processing in *zip* has been changed to support long options and handle all options and arguments more consistently. Some old command lines that depend on command line inconsistencies may no longer work.

Description

zip is a compression and file packaging utility for Unix, VMS, MSDOS, OS/2, Windows 9x/NT/XP, Minix, Atari, Macintosh, Amiga, and Acorn RISC OS. It is analogous to a combination of the Unix commands **tar**(1) and **compress**(1) and is compatible with PKZIP (Phil Katz's ZIP for MSDOS systems).

A companion program (**unzip**(1L)) unpacks *zip* archives. The *zip* and **unzip**(1L) programs can work with archives produced by PKZIP (supporting most PKZIP features up to PKZIP version 4.6), and PKZIP and PKUNZIP can work with archives produced by *zip* (with some exceptions, notably streamed archives, but recent changes in the zip file standard may facilitate better compatibility). *zip* version 3.0 is compatible with PKZIP 2.04 and also supports the Zip64 extensions of PKZIP 4.5 which allow archives as well as files to exceed the previous 2 GB limit (4 GB in some cases). *zip* also now supports **bzip2** compression if the **bzip2** library is included when *zip* is compiled. Note that PKUNZIP 1.10 cannot extract files produced by PKZIP 2.04 or *zip 3.0*. You must use PKUNZIP 2.04g or *unzip 5.0p1* (or later versions) to extract them.

See the **EXAMPLES** section at the bottom of this page for examples of some typical uses of *zip*.

Large Archives and Zip64. *zip* automatically uses the Zip64 extensions when files larger than 4 GB are added to an archive, an archive containing Zip64 entries is updated (if the resulting archive still needs Zip64), the size of the archive will exceed 4 GB, or when the number of entries in the archive will exceed about 64K. Zip64 is also used for archives streamed from standard input as the size of such archives are not known in advance, but the option **-fz-** can be used to force *zip* to create PKZIP 2 compatible

archives (as long as Zip64 extensions are not needed). You must use a PKZIP 4.5 compatible unzip, such as *unzip 6.0* or later, to extract files using the Zip64 extensions.

In addition, streamed archives, entries encrypted with standard encryption, or split archives created with the pause option may not be compatible with PKZIP as data descriptors are used and PKZIP at the time of this writing does not support data descriptors (but recent changes in the PKWare published zip standard now include some support for the data descriptor format *zip* uses).

Mac OS X. Though previous Mac versions had their own *zip* port, *zip* supports Mac OS X as part of the Unix port and most Unix features apply. References to "MacOS" below generally refer to MacOS versions older than OS X. Support for some Mac OS features in the Unix Mac OS X port, such as resource forks, is expected in the next *zip* release.

For a brief help on *zip* and *unzip*, run each without specifying any parameters on the command line.

Use

The program is useful for packaging a set of files for distribution; for archiving files; and for saving disk space by temporarily compressing unused files or directories.

The *zip* program puts one or more compressed files into a single *zip* archive, along with information about the files (name, path, date, time of last modification, protection, and check information to verify file integrity). An entire directory structure can be packed into a *zip* archive with a single command. Compression ratios of 2:1 to 3:1 are common for text files. *zip* has one compression method (deflation) and can also store files without compression. (If **bzip2** support is added, *zip* can also compress using **bzip2** compression, but such entries require a reasonably modern unzip to decompress. When **bzip2** compression is selected, it replaces deflation as the default method.) *zip* automatically chooses the better of the two (deflation or store or, if **bzip2** is selected, **bzip2** or store) for each file to be compressed.

Command format. The basic command format is

    zip options archive inpath inpath ...

where **archive** is a new or existing *zip* archive and **inpath** is a directory or file path optionally including wildcards. When given the name of an existing *zip* archive, *zip* will replace identically named entries in the *zip* archive (matching the relative names as stored in the archive) or add entries for new names. For example, if *foo.zip* exists and contains *foo/file1* and *foo/file2*, and the directory *foo* contains the files *foo/file1* and *foo/file3*, then:

```
zip –r foo.zip foo
```

or more concisely

```
zip –r foo foo
```

will replace *foo/file1* in *foo.zip* and add *foo/file3* to *foo.zip*. After this, *foo.zip* contains *foo/file1*, *foo/file2*, and *foo/file3*, with *foo/file2* unchanged from before.

So if before the zip command is executed *foo.zip* has:

```
foo/file1 foo/file2
```
and directory foo has:
```
file1 file3
```
then *foo.zip* will have:
```
foo/file1 foo/file2 foo/file3
```
where *foo/file1* is replaced and *foo/file3* is new.

-@ file lists. If a file list is specified as **-@** [Not on MacOS], *zip* takes the list of input files from standard input instead of from the command line. For example,

```
zip –@ foo
```
will store the files listed one per line on stdin in *foo.zip*.

Under Unix, this option can be used to powerful effect in conjunction with the *find* (1) command. For example, to archive all the C source files in the current directory and its subdirectories:

```
find . –name "*.[ch]" –print | zip source –@
```
(note that the pattern must be quoted to keep the shell from expanding it).

Streaming input and output. *zip* will also accept a single dash ("-") as the zip file name, in which case it will write the zip file to standard output, allowing the output to be piped to another program. For example:

```
zip –r – . | dd of=/dev/nrst0 obs=16k
```
would write the zip output directly to a tape with the specified block size for the purpose of backing up the current directory.

zip also accepts a single dash ("-") as the name of a file to be compressed, in which case it will read the file from standard input, allowing zip to take input from another program. For example:

```
tar cf – . | zip backup –
```
would compress the output of the tar command for the purpose of backing up the current directory. This generally produces better compression than the previous example using the -r option because *zip* can take advantage of redundancy between files. The backup can be restored using the command
```
unzip –p backup | tar xf –
```
When no zip file name is given and stdout is not a terminal, *zip* acts as a filter, compressing standard input to standard output. For example,
```
tar cf – . | zip | dd of=/dev/nrst0 obs=16k
```
is equivalent to
```
tar cf – . | zip – – | dd of=/dev/nrst0 obs=16k
```
zip archives created in this manner can be extracted with the program *funzip* which is provided in the *unzip* package, or by *gunzip* which is provided in the *gzip* package (but some *gunzip* may not support this if *zip* used the Zip64 extensions). For example:
```
dd if=/dev/nrst0 ibs=16k | funzip | tar xvf -
```
The stream can also be saved to a file and *unzip* used.

If Zip64 support for large files and archives is enabled and *zip* is used as a filter, *zip* creates a Zip64 archive that requires a PKZIP 4.5 or later compatible unzip to read it. This is to avoid amgibuities in the zip file structure as defined in the current zip standard (PKWARE AppNote) where the decision to use Zip64 needs to be made before data is written for the entry, but for a stream the size of the data is not known at that point. If the data is known to be smaller than 4 GB, the option **-fz-** can be used to prevent use of Zip64, but *zip* will exit with an error if Zip64 was in fact needed. *zip 3* and *unzip 6* and later can read archives with Zip64 entries. Also, *zip* removes the Zip64 extensions if not needed when archive entries are copied (see the **-U** (**--copy**) option).

When directing the output to another file, note that all options should be before the redirection including **-x**. For example:

zip archive "*.h" "*.c" -x donotinclude.h orthis.h > tofile

Zip files. When changing an existing *zip* archive, *zip* will write a temporary file with the new contents, and only replace the old one when the process of creating the new version has been completed without error.

If the name of the *zip* archive does not contain an extension, the extension **.zip** is added. If the name already contains an extension other than **.zip**, the existing extension is kept unchanged. However, split archives (archives split over multiple files) require the **.zip** extension on the last split.

Scanning and reading files. When *zip* starts, it scans for files to process (if needed). If this scan takes longer than about 5 seconds, *zip* will display a "Scanning files" message and start displaying progress dots every 2 seconds or every so many entries processed, whichever takes longer. If there is more than 2 seconds between dots it could indicate that finding each file is taking time and could mean a slow network connection for example. (Actually the initial file scan is a two-step process where the directory scan is followed by a sort and these two steps are separated with a space in the dots. If updating an existing archive, a space also appears between the existing file scan and the new file scan.) The scanning files dots are not controlled by the **-ds** dot size option, but the dots are turned off by the **-q** quiet option. The **-sf** show files option can be used to scan for files and get the list of files scanned without actually processing them.

If *zip* is not able to read a file, it issues a warning but continues. See the **-MM** option below for more on how *zip* handles patterns that are not matched and files that are not readable. If some files were skipped, a warning is issued at the end of the zip operation noting how many files were read and how many skipped.

Command modes. *zip* now supports two distinct types of command modes, **external** and **internal**. The **external** modes (add, update, and freshen) read files from the file system (as well as from an existing archive) while the **internal** modes (delete and copy) operate exclusively on entries in an existing archive.

add

Update existing entries and add new files. If the archive does not exist create it. This is the default mode.

update (**-u**)
Update existing entries if newer on the file system and add new files. If the archive does not exist issue warning then create a new archive.

freshen (**-f**)
Update existing entries of an archive if newer on the file system. Does not add new files to the archive.

delete (**-d**)
Select entries in an existing archive and delete them.

copy (**-U**)
Select entries in an existing archive and copy them to a new archive. This new mode is similar to **update** but command line patterns select entries in the existing archive rather than files from the file system and it uses the **--out** option to write the resulting archive to a new file rather than update the existing archive, leaving the original archive unchanged.

The new File Sync option (**-FS**) is also considered a new mode, though it is similar to **update**. This mode synchronizes the archive with the files on the OS, only replacing files in the archive if the file time or size of the OS file is different, adding new files, and deleting entries from the archive where there is no matching file. As this mode can delete entries from the archive, consider making a backup copy of the archive.

Also see **-DF** for creating difference archives.

See each option description below for details and the **EXAMPLES** section below for examples.

Split archives. *zip* version 3.0 and later can create split archives. A **split archive** is a standard zip archive split over multiple files. (Note that split archives are not just archives split in to pieces, as the offsets of entries are now based on the start of each split. Concatenating the pieces together will invalidate these offsets, but *unzip* can usually deal with it. *zip* will usually refuse to process such a spliced archive unless the **-FF** fix option is used to fix the offsets.)

One use of split archives is storing a large archive on multiple removable media. For a split archive with 20 split files the files are typically named (replace ARCHIVE with the name of your archive) ARCHIVE.z01, ARCHIVE.z02, ..., ARCHIVE.z19, ARCHIVE.zip. Note that the last file is the **.zip** file. In contrast, **spanned archives** are the original multi-disk archive generally requiring floppy disks and using volume labels to store disk numbers. *zip* supports split archives but not spanned archives, though a procedure exists for converting split archives of the right size to spanned archives. The reverse is also true, where each file of a spanned archive can be copied in order to files with the above names to create a split archive.

Use **-s** to set the split size and create a split archive. The size is given as a number followed optionally by one of k (kB), m (MB), g (GB), or t (TB) (the default is m). The **-sp** option can be used to pause *zip* between splits to allow changing removable media, for example, but read the descriptions and warnings for both **-s** and **-sp** below.

Though *zip* does not update split archives, *zip* provides the new option **-O** (**--output-file** or **--out**) to allow split archives to be updated and saved in a new archive. For

example,

```
zip inarchive.zip foo.c bar.c --out outarchive.zip
```

reads archive **inarchive.zip**, even if split, adds the files **foo.c** and **bar.c**, and writes the resulting archive to **outarchive.zip**. If **inarchive.zip** is split then **outarchive.zip** defaults to the same split size. Be aware that if **outarchive.zip** and any split files that are created with it already exist, these are always overwritten as needed without warning. This may be changed in the future.

Unicode. Though the zip standard requires storing paths in an archive using a specific character set, in practice zips have stored paths in archives in whatever the local character set is. This creates problems when an archive is created or updated on a system using one character set and then extracted on another system using a different character set. When compiled with Unicode support enabled on platforms that support wide characters, *zip* now stores, in addition to the standard local path for backward compatibility, the UTF-8 translation of the path. This provides a common universal character set for storing paths that allows these paths to be fully extracted on other systems that support Unicode and to match as close as possible on systems that don't.

On Win32 systems where paths are internally stored as Unicode but represented in the local character set, it's possible that some paths will be skipped during a local character set directory scan. *zip* with Unicode support now can read and store these paths. Note that Win 9x systems and FAT file systems don't fully support Unicode.

Be aware that console windows on Win32 and Unix, for example, sometimes don't accurately show all characters due to how each operating system switches in character sets for display. However, directory navigation tools should show the correct paths if the needed fonts are loaded.

Command line format. This version of *zip* has updated command line processing and support for long options.

Short options take the form

```
-s[-][s[-]...][value][=value][ value]
```

where s is a one or two character short option. A short option that takes a value is last in an argument and anything after it is taken as the value. If the option can be negated and "-" immediately follows the option, the option is negated. Short options can also be given as separate arguments

```
-s[-][value][=value][ value] -s[-][value][=value][ value] ...
```

Short options in general take values either as part of the same argument or as the following argument. An optional = is also supported. So

```
-ttmmddyyyy
```

and

```
-tt=mmddyyyy
```

and

```
-tt mmddyyyy
```

all work. The **-x** and **-i** options accept lists of values and use a slightly different format described below. See the **-x** and **-i** options.

Long options take the form

```
--longoption[-][=value][ value]
```

where the option starts with --, has a multicharacter name, can include a trailing dash to negate the option (if the option supports it), and can have a value (option argument) specified by preceeding it with = (no spaces). Values can also follow the argument. So

```
--before-date=mmddyyyy
```

and

```
--before-date mmddyyyy
```

both work.

Long option names can be shortened to the shortest unique abbreviation. See the option descriptions below for which support long options. To avoid confusion, avoid abbreviating a negatable option with an embedded dash ("-") at the dash if you plan to negate it (the parser would consider a trailing dash, such as for the option **--some-option** using **--some-** as the option, as part of the name rather than a negating dash). This may be changed to force the last dash in **--some-** to be negating in the future.

Options

-a
--ascii
[Systems using EBCDIC] Translate file to ASCII format.

-A
--adjust-sfx
Adjust self-extracting executable archive. A self-extracting executable archive is created by prepending the SFX stub to an existing archive. The **-A** option tells *zip* to adjust the entry offsets stored in the archive to take into account this "preamble" data.

Note: self-extracting archives for the Amiga are a special case. At present, only the Amiga port of *zip* is capable of adjusting or updating these without corrupting them. -J can be used to remove the SFX stub if other updates need to be made.

-AC
--archive-clear
[WIN32] Once archive is created (and tested if **-T** is used, which is recommended), clear the archive bits of files processed. WARNING: Once the bits are cleared they are cleared. You may want to use the **-sf** show files option to store the list of files processed in case the archive operation must be repeated. Also consider using the **-MM** must match option. Be sure to check out **-DF** as a possibly better way to do incremental backups.

-AS
--archive-set
[WIN32] Only include files that have the archive bit set. Directories are not stored when **-AS** is used, though by default the paths of entries, including directories, are stored as usual and can be used by most unzips to recreate directories.

The archive bit is set by the operating system when a file is modified and, if used with **-AC**, **-AS** can provide an incremental backup capability. However, other

applications can modify the archive bit and it may not be a reliable indicator of which files have changed since the last archive operation. Alternative ways to create incremental backups are using **-t** to use file dates, though this won't catch old files copied to directories being archived, and **-DF** to create a differential archive.

-B
--binary
[VM/CMS and MVS] force file to be read binary (default is text).
-Bn

[TANDEM] set Edit/Enscribe formatting options with n defined as

bit 0: Don't add delimiter (Edit/Enscribe)
bit 1: Use LF rather than CR/LF as delimiter (Edit/Enscribe)
bit 2: Space fill record to maximum record length (Enscribe)
bit 3: Trim trailing space (Enscribe)
bit 8: Force 30K (Expand) large read for unstructured files
-b path
--temp-path path
Use the specified *path* for the temporary *zip* archive. For example:

```
zip -b /tmp stuff *
```

will put the temporary *zip* archive in the directory */tmp*, copying over *stuff.zip* to the current directory when done. This option is useful when updating an existing archive and the file system containing this old archive does not have enough space to hold both old and new archives at the same time. It may also be useful when streaming in some cases to avoid the need for data descriptors. Note that using this option may require *zip* take additional time to copy the archive file when done to the destination file system.
-c
--entry-comments
Add one-line comments for each file. File operations (adding, updating) are done first, and the user is then prompted for a one-line comment for each file. Enter the comment followed by return, or just return for no comment.
-C
--preserve-case
[VMS] Preserve case all on VMS. Negating this option (**-C-**) downcases.
-C2
--preserve-case-2
[VMS] Preserve case ODS2 on VMS. Negating this option (**-C2-**) downcases.
-C5
--preserve-case-5
[VMS] Preserve case ODS5 on VMS. Negating this option (**-C5-**) downcases.
-d
--delete
Remove (delete) entries from a *zip* archive. For example:

```
zip -d foo foo/tom/junk foo/harry/\* \*.o
```

will remove the entry *foo/tom/junk*, all of the files that start with *foo/harry/*, and all of the files that end with **.o** (in any path). Note that shell pathname expansion has been inhibited with backslashes, so that *zip* can see the asterisks, enabling *zip* to match on the contents of the *zip* archive instead of the contents of the current directory. (The backslashes are not used on MSDOS-based platforms.) Can also use quotes to escape the asterisks as in

```
zip -d foo foo/tom/junk "foo/harry/*" "*.o"
```

Not escaping the asterisks on a system where the shell expands wildcards could result in the asterisks being converted to a list of files in the current directory and that list used to delete entries from the archive.

Under MSDOS, **-d** is case sensitive when it matches names in the *zip* archive. This requires that file names be entered in upper case if they were zipped by PKZIP on an MSDOS system. (We considered making this case insensitive on systems where paths were case insensitive, but it is possible the archive came from a system where case does matter and the archive could include both **Bar** and **bar** as separate files in the archive.) But see the new option **-ic** to ignore case in the archive.

-db
--display-bytes
Display running byte counts showing the bytes zipped and the bytes to go.

-dc
--display-counts
Display running count of entries zipped and entries to go.

-dd
--display-dots
Display dots while each entry is zipped (except on ports that have their own progress indicator). See **-ds** below for setting dot size. The default is a dot every 10 MB of input file processed. The **-v** option also displays dots (previously at a much higher rate than this but now **-v** also defaults to 10 MB) and this rate is also controlled by **-ds**.

-df
--datafork
[MacOS] Include only data-fork of files zipped into the archive. Good for exporting files to foreign operating-systems. Resource-forks will be ignored at all.

-dg
--display-globaldots
Display progress dots for the archive instead of for each file. The command

```
zip -qdgds 10m
```

will turn off most output except dots every 10 MB.

-ds size
--dot-size size

Set amount of input file processed for each dot displayed. See **-dd** to enable displaying dots. Setting this option implies **-dd**. Size is in the format nm where n is a number and m is a multiplier. Currently m can be k (KB), m (MB), g (GB), or t (TB), so if n is 100 and m is k, size would be 100k which is 100 KB. The default is 10 MB.
The **-v** option also displays dots and now defaults to 10 MB also. This rate is also controlled by this option. A size of 0 turns dots off.

This option does not control the dots from the "Scanning files" message as *zip* scans for input files. The dot size for that is fixed at 2 seconds or a fixed number of entries, whichever is longer.

-du
--display-usize
Display the uncompressed size of each entry.

-dv
--display-volume
Display the volume (disk) number each entry is being read from, if reading an existing archive, and being written to.

-D
--no-dir-entries
Do not create entries in the *zip* archive for directories. Directory entries are created by default so that their attributes can be saved in the zip archive. The environment variable ZIPOPT can be used to change the default options. For example under Unix with sh:

ZIPOPT="-D"; export ZIPOPT

(The variable ZIPOPT can be used for any option, including **-i** and **-x** using a new option format detailed below, and can include several options.) The option **-D** is a shorthand for **-x** "*/" but the latter previously could not be set as default in the ZIPOPT environment variable as the contents of ZIPOPT gets inserted near the beginning of the command line and the file list had to end at the end of the line.

This version of *zip* does allow **-x** and **-i** options in ZIPOPT if the form

-x file file ... **@**

is used, where the @ (an argument that is just @) terminates the list.

-DF
--difference-archive
Create an archive that contains all new and changed files since the original archive was created. For this to work, the input file list and current directory must be the same as during the original *zip* operation.
For example, if the existing archive was created using

```
zip -r foofull .
```

from the *bar* directory, then the command

```
zip -r foofull . -DF --out foonew
```

also from the *bar* directory creates the archive *foonew* with just the files not in *foofull* and the files where the size or file time of the files do not match those in *foofull*.

Note that the timezone environment variable TZ should be set according to the local timezone in order for this option to work correctly. A change in timezone since the original archive was created could result in no times matching and all files being included.

A possible approach to backing up a directory might be to create a normal archive of the contents of the directory as a full backup, then use this option to create incremental backups.

-e
--encrypt

Encrypt the contents of the *zip* archive using a password which is entered on the terminal in response to a prompt (this will not be echoed; if standard error is not a tty, *zip* will exit with an error). The password prompt is repeated to save the user from typing errors.

-E
--longnames

[OS/2] Use the .LONGNAME Extended Attribute (if found) as filename.

-f
--freshen

Replace (freshen) an existing entry in the *zip* archive only if it has been modified more recently than the version already in the *zip* archive; unlike the update option (**-u**) this will not add files that are not already in the *zip* archive. For example:

```
zip -f foo
```

This command should be run from the same directory from which the original *zip* command was run, since paths stored in *zip* archives are always relative.

Note that the timezone environment variable TZ should be set according to the local timezone in order for the **-f**, **-u** and **-o** options to work correctly.

The reasons behind this are somewhat subtle but have to do with the differences between the Unix-format file times (always in GMT) and most of the other operating systems (always local time) and the necessity to compare the two. A typical TZ value is ''MET-1MEST'' (Middle European time with automatic adjustment for ''summertime'' or Daylight Savings Time).

The format is TTThhDDD, where TTT is the time zone such as MET, hh is the difference between GMT and local time such as -1 above, and DDD is the time zone when daylight savings time is in effect. Leave off the DDD if there is no daylight savings time. For the US Eastern time zone EST5EDT.

-F

--fix
-FF
--fixfix

Fix the *zip* archive. The **-F** option can be used if some portions of the archive are missing, but requires a reasonably intact central directory. The input archive is scanned as usual, but *zip* will ignore some problems. The resulting archive should be valid, but any inconsistent entries will be left out.
When doubled as in **-FF**, the archive is scanned from the beginning and *zip* scans for special signatures to identify the limits between the archive members. The single **-F** is more reliable if the archive is not too much damaged, so try this option first.

If the archive is too damaged or the end has been truncated, you must use **-FF**. This is a change from *zip 2.32*, where the **-F** option is able to read a truncated archive. The **-F** option now more reliably fixes archives with minor damage and the **-FF** option is needed to fix archives where **-F** might have been sufficient before.

Neither option will recover archives that have been incorrectly transferred in ascii mode instead of binary. After the repair, the **-t** option of *unzip* may show that some files have a bad CRC. Such files cannot be recovered; you can remove them from the archive using the **-d** option of *zip*.

Note that **-FF** may have trouble fixing archives that include an embedded zip archive that was stored (without compression) in the archive and, depending on the damage, it may find the entries in the embedded archive rather than the archive itself. Try **-F** first as it does not have this problem.

The format of the fix commands have changed. For example, to fix the damaged archive *foo.zip*,

```
zip -F foo --out foofix
```

tries to read the entries normally, copying good entries to the new archive *foofix.zip*. If this doesn't work, as when the archive is truncated, or if some entries you know are in the archive are missed, then try

```
zip -FF foo --out foofixfix
```

and compare the resulting archive to the archive created by **-F**. The **-FF** option may create an inconsistent archive. Depending on what is damaged, you can then use the **-F** option to fix that archive.

A split archive with missing split files can be fixed using **-F** if you have the last split of the archive (the **.zip** file). If this file is missing, you must use **-FF** to fix the archive, which will prompt you for the splits you have.

Currently the fix options can't recover entries that have a bad checksum or are otherwise damaged.

-FI

--fifo

[Unix] Normally *zip* skips reading any FIFOs (named pipes) encountered, as *zip* can hang if the FIFO is not being fed. This option tells *zip* to read the contents of any FIFO it finds.

-FS

--filesync

Synchronize the contents of an archive with the files on the OS. Normally when an archive is updated, new files are added and changed files are updated but files that no longer exist on the OS are not deleted from the archive. This option enables a new mode that checks entries in the archive against the file system. If the file time and file size of the entry matches that of the OS file, the entry is copied from the old archive instead of being read from the file system and compressed. If the OS file has changed, the entry is read and compressed as usual. If the entry in the archive does not match a file on the OS, the entry is deleted. Enabling this option should create archives that are the same as new archives, but since existing entries are copied instead of compressed, updating an existing archive with **-FS** can be much faster than creating a new archive. Also consider using **-u** for updating an archive.
For this option to work, the archive should be updated from the same directory it was created in so the relative paths match. If few files are being copied from the old archive, it may be faster to create a new archive instead.

Note that the timezone environment variable TZ should be set according to the local timezone in order for this option to work correctly. A change in timezone since the original archive was created could result in no times matching and recompression of all files.

This option deletes files from the archive. If you need to preserve the original archive, make a copy of the archive first or use the **--out** option to output the updated archive to a new file. Even though it may be slower, creating a new archive with a new archive name is safer, avoids mismatches between archive and OS paths, and is preferred.

-g
--grow

Grow (append to) the specified *zip* archive, instead of creating a new one. If this operation fails, *zip* attempts to restore the archive to its original state. If the restoration fails, the archive might become corrupted. This option is ignored when there's no existing archive or when at least one archive member must be updated or deleted.

-h

-?

--help

Display the *zip* help information (this also appears if *zip* is run with no arguments).

-h2

--more-help

Display extended help including more on command line format, pattern matching, and more obscure options.

-i files

--include files

Include only the specified files, as in:

```
zip –r foo . –i \*.c
```

which will include only the files that end in .c in the current directory and its subdirectories. (Note for PKZIP users: the equivalent command is

```
pkzip –rP foo *.c
```

PKZIP does not allow recursion in directories other than the current one.) The backslash avoids the shell filename substitution, so that the name matching is performed by *zip* at all directory levels. [This is for Unix and other systems where \ escapes the next character. For other systems where the shell does not process * do not use \ and the above is

```
zip –r foo . –i *.c
```

Examples are for Unix unless otherwise specified.] So to include dir, a directory directly under the current directory, use

```
zip –r foo . –i dir/\*
```

or

```
zip –r foo . –i "dir/*"
```

to match paths such as dir/a and dir/b/file.c [on ports without wildcard expansion in the shell such as MSDOS and Windows

```
zip –r foo . –i dir/*
```

is used.] Note that currently the trailing / is needed for directories (as in

```
zip –r foo . –i dir/
```

to include directory dir).

The long option form of the first example is

```
zip –r foo . ––include \*.c
```

and does the same thing as the short option form.

Though the command syntax used to require **-i** at the end of the command line, this version actually allows **-i** (or **--include**) anywhere. The list of files terminates

at the next argument starting with **-**, the end of the command line, or the list terminator **@** (an argument that is just @). So the above can be given as

zip -i *.c @ -r foo .

for example. There must be a space between the option and the first file of a list. For just one file you can use the single value form

```
zip –i\*.c –r foo .
```

(no space between option and value) or

```
zip ––include=\*.c –r foo .
```

as additional examples. The single value forms are not recommended because they can be confusing and, in particular, the **-ifile** format can cause problems if the first letter of **file** combines with **i** to form a two-letter option starting with **i**. Use **-sc** to see how your command line will be parsed.

Also possible:

```
zip –r foo . –i@include.lst
```

which will only include the files in the current directory and its subdirectories that match the patterns in the file include.lst.

Files to **-i** and **-x** are patterns matching internal archive paths. See **-R** for more on patterns.

-I
--no-image

[Acorn RISC OS] Don't scan through Image files. When used, *zip* will not consider Image files (eg. DOS partitions or Spark archives when SparkFS is loaded) as directories but will store them as single files.

For example, if you have SparkFS loaded, zipping a Spark archive will result in a zipfile containing a directory (and its content) while using the 'I' option will result in a zipfile containing a Spark archive. Obviously this second case will also be obtained (without the 'I' option) if SparkFS isn't loaded.

-ic
--ignore-case

[VMS, WIN32] Ignore case when matching archive entries. This option is only available on systems where the case of files is ignored. On systems with case-insensitive file systems, case is normally ignored when matching files on the file system but is not ignored for -f (freshen), -d (delete), -U (copy), and similar modes when matching against archive entries (currently -f ignores case on VMS) because archive entries can be from systems where case does matter and names that are the same except for case can exist in an archive. The **-ic** option makes all matching

case insensitive. This can result in multiple archive entries matching a command line pattern.

-j
--junk-paths
Store just the name of a saved file (junk the path), and do not store directory names. By default, *zip* will store the full path (relative to the current directory).

-jj
--absolute-path
[MacOS] record Fullpath (+ Volname). The complete path including volume will be stored. By default the relative path will be stored.

-J
--junk-sfx
Strip any prepended data (e.g. a SFX stub) from the archive.

-k
--DOS-names
Attempt to convert the names and paths to conform to MSDOS, store only the MSDOS attribute (just the user write attribute from Unix), and mark the entry as made under MSDOS (even though it was not); for compatibility with PKUNZIP under MSDOS which cannot handle certain names such as those with two dots.

-l
--to-crlf
Translate the Unix end-of-line character LF into the MSDOS convention CR LF. This option should not be used on binary files. This option can be used on Unix if the zip file is intended for PKUNZIP under MSDOS. If the input files already contain CR LF, this option adds an extra CR. This is to ensure that **unzip -a** on Unix will get back an exact copy of the original file, to undo the effect of **zip -l**. See **-ll** for how binary files are handled.

-la
--log-append
Append to existing logfile. Default is to overwrite.

-lf logfilepath
--logfile-path logfilepath
Open a logfile at the given path. By default any existing file at that location is overwritten, but the **-la** option will result in an existing file being opened and the new log information appended to any existing information. Only warnings and errors are written to the log unless the **-li** option is also given, then all information messages are also written to the log.

-li
--log-info
Include information messages, such as file names being zipped, in the log. The default is to only include the command line, any warnings and errors, and the final status.

-ll
--from-crlf
Translate the MSDOS end-of-line CR LF into Unix LF. This option should not be used on binary files. This option can be used on MSDOS if the zip file is intended for unzip under Unix. If the file is converted and the file is later determined to be

binary a warning is issued and the file is probably corrupted. In this release if **-ll** detects binary in the first buffer read from a file, *zip* now issues a warning and skips line end conversion on the file. This check seems to catch all binary files tested, but the original check remains and if a converted file is later determined to be binary that warning is still issued. A new algorithm is now being used for binary detection that should allow line end conversion of text files in **UTF-8** and similar encodings.

-L
--license

Display the *zip* license.

-m
--move

Move the specified files into the *zip* archive; actually, this deletes the target directories/files after making the specified *zip* archive. If a directory becomes empty after removal of the files, the directory is also removed. No deletions are done until *zip* has created the archive without error. This is useful for conserving disk space, but is potentially dangerous so it is recommended to use it in combination with **-T** to test the archive before removing all input files.

-MM
--must-match

All input patterns must match at least one file and all input files found must be readable. Normally when an input pattern does not match a file the "name not matched" warning is issued and when an input file has been found but later is missing or not readable a missing or not readable warning is issued. In either case *zip* continues creating the archive, with missing or unreadable new files being skipped and files already in the archive remaining unchanged. After the archive is created, if any files were not readable *zip* returns the OPEN error code (18 on most systems) instead of the normal success return (0 on most systems). With **-MM** set, *zip* exits as soon as an input pattern is not matched (whenever the "name not matched" warning would be issued) or when an input file is not readable. In either case *zip* exits with an OPEN error and no archive is created.

This option is useful when a known list of files is to be zipped so any missing or unreadable files will result in an error. It is less useful when used with wildcards, but *zip* will still exit with an error if any input pattern doesn't match at least one file and if any matched files are unreadable. If you want to create the archive anyway and only need to know if files were skipped, don't use **-MM** and just check the return code. Also **-lf** could be useful.

-n suffixes
--suffixes suffixes

Do not attempt to compress files named with the given **suffixes**. Such files are simply stored (0% compression) in the output zip file, so that *zip* doesn't waste its time trying to compress them. The suffixes are separated by either colons or semicolons. For example:

```
zip –rn .Z:.zip:.tiff:.gif:.snd foo foo
```

will copy everything from *foo* into *foo.zip*, but will store any files that end in *.Z*, *.zip*, *.tiff*, *.gif*, or *.snd* without trying to compress them (image and sound files often

have their own specialized compression methods). By default, *zip* does not compress files with extensions in the list *.Z:.zip:.zoo:.arc:.lzh:.arj.* Such files are stored directly in the output archive. The environment variable ZIPOPT can be used to change the default options. For example under Unix with csh:

setenv ZIPOPT "-n .gif:.zip"

To attempt compression on all files, use:

zip -n : foo

The maximum compression option **-9** also attempts compression on all files regardless of extension.

On Acorn RISC OS systems the suffixes are actually filetypes (3 hex digit format). By default, *zip* does not compress files with filetypes in the list DDC:D96:68E (i.e. Archives, CFS files and PackDir files).

-nw
--no-wild
Do not perform internal wildcard processing (shell processing of wildcards is still done by the shell unless the arguments are escaped). Useful if a list of paths is being read and no wildcard substitution is desired.

-N
--notes
[Amiga, MacOS] Save Amiga or MacOS filenotes as zipfile comments. They can be restored by using the -N option of *unzip*. If -c is used also, you are prompted for comments only for those files that do not have filenotes.

-o
--latest-time
Set the "last modified" time of the *zip* archive to the latest (oldest) "last modified" time found among the entries in the *zip* archive. This can be used without any other operations, if desired. For example:

```
zip –o foo
```

will change the last modified time of **foo.zip** to the latest time of the entries in **foo.zip**.

-O output-file
--output-file output-file
Process the archive changes as usual, but instead of updating the existing archive, output the new archive to output-file. Useful for updating an archive without changing the existing archive and the input archive must be a different file than the output archive.

This option can be used to create updated split archives. It can also be used with **-U** to copy entries from an existing archive to a new archive. See the **EXAMPLES** section below.

Another use is converting *zip* files from one split size to another. For instance, to convert an archive with 700 MB CD splits to one with 2 GB DVD splits, can use:

zip -s 2g cd-split.zip --out dvd-split.zip

which uses copy mode. See **-U** below. Also:

zip -s 0 split.zip --out unsplit.zip

will convert a split archive to a single-file archive.

Copy mode will convert stream entries (using data descriptors and which should be compatible with most unzips) to normal entries (which should be compatible with all unzips), except if standard encryption was used. For archives with encrypted entries, *zipcloak* will decrypt the entries and convert them to normal entries.

-p
--paths
Include relative file paths as part of the names of files stored in the archive. This is the default. The **-j** option junks the paths and just stores the names of the files.

-P password
--password password
Use *password* to encrypt zipfile entries (if any). **THIS IS INSECURE!** Many multi-user operating systems provide ways for any user to see the current command line of any other user; even on stand-alone systems there is always the threat of over-the-shoulder peeking. Storing the plaintext password as part of a command line in an automated script is even worse. Whenever possible, use the non-echoing, interactive prompt to enter passwords. (And where security is truly important, use strong encryption such as Pretty Good Privacy instead of the relatively weak standard encryption provided by zipfile utilities.)

-q
--quiet
Quiet mode; eliminate informational messages and comment prompts. (Useful, for example, in shell scripts and background tasks).

-Qn
--Q-flag n
[QDOS] store information about the file in the file header with n defined as
bit 0: Don't add headers for any file
bit 1: Add headers for all files
bit 2: Don't wait for interactive key press on exit

-r
--recurse-paths
Travel the directory structure recursively; for example:

zip -r foo.zip foo

or more concisely

zip -r foo foo

In this case, all the files and directories in **foo** are saved in a *zip* archive named **foo.zip**, including files with names starting with **"."**, since the recursion does not use the shell's file-name substitution mechanism. If you wish to include only a specific subset of the files in directory **foo** and its subdirectories, use the **-i** option to specify the pattern of files to be included. You should not use **-r** with the name **".*"**, since that matches **".."** which will attempt to zip up the parent directory (probably not what was intended).

Multiple source directories are allowed as in

```
zip –r foo foo1 foo2
```

which first zips up **foo1** and then **foo2**, going down each directory.

Note that while wildcards to **-r** are typically resolved while recursing down directories in the file system, any **-R, -x**, and **-i** wildcards are applied to internal archive pathnames once the directories are scanned. To have wildcards apply to files in subdirectories when recursing on Unix and similar systems where the shell does wildcard substitution, either escape all wildcards or put all arguments with wildcards in quotes. This lets *zip* see the wildcards and match files in subdirectories using them as it recurses.

-R
--recurse-patterns

Travel the directory structure recursively starting at the current directory; for example:

```
zip –R foo "*.c"
```

In this case, all the files matching ***.c** in the tree starting at the current directory are stored into a *zip* archive named **foo.zip**. Note that ***.c** will match **file.c**, **a/file.c** and **a/b/.c**. More than one pattern can be listed as separate arguments. Note for PKZIP users: the equivalent command is

```
pkzip –rP foo *.c
```

Patterns are relative file paths as they appear in the archive, or will after zipping, and can have optional wildcards in them. For example, given the current directory is **foo** and under it are directories **foo1** and **foo2** and in **foo1** is the file **bar.c**,

```
zip –R foo/*
```

will zip up **foo**, **foo/foo1**, **foo/foo1/bar.c**, and **foo/foo2**.

```
zip –R */bar.c
```

will zip up **foo/foo1/bar.c**. See the note for **-r** on escaping wildcards.

-RE
--regex

[WIN32] Before *zip 3.0*, regular expression list matching was enabled by default on Windows platforms. Because of confusion resulting from the need to escape "[" and "]" in names, it is now off by default for Windows so "[" and "]" are just normal characters in names. This option enables [] matching again.

-s splitsize
--split-size splitsize

Enable creating a split archive and set the split size. A split archive is an archive that could be split over many files. As the archive is created, if the size of the archive reaches the specified split size, that split is closed and the next split opened. In general all splits but the last will be the split size and the last will be whatever is left. If the entire archive is smaller than the split size a single-file archive is created.

Split archives are stored in numbered files. For example, if the output archive is named **archive** and three splits are required, the resulting archive will be in the three files **archive.z01**, **archive.z02**, and **archive.zip**. Do not change the numbering of these files or the archive will not be readable as these are used to determine the order the splits are read.

Split size is a number optionally followed by a multiplier. Currently the number must be an integer. The multiplier can currently be one of **k** (kilobytes), **m** (megabytes), **g** (gigabytes), or **t** (terabytes). As 64k is the minimum split size, numbers without multipliers default to megabytes. For example, to create a split archive called **foo** with the contents of the **bar** directory with splits of 670 MB that might be useful for burning on CDs, the command:

zip -s 670m -r foo bar

could be used.

Currently the old splits of a split archive are not excluded from a new archive, but they can be specifically excluded. If possible, keep the input and output archives out of the path being zipped when creating split archives.

Using **-s** without **-sp** as above creates all the splits where **foo** is being written, in this case the current directory. This split mode updates the splits as the archive is being created, requiring all splits to remain writable, but creates split archives that are readable by any unzip that supports split archives. See **-sp** below for enabling split pause mode which allows splits to be written directly to removable media.

The option **-sv** can be used to enable verbose splitting and provide details of how the splitting is being done. The **-sb** option can be used to ring the bell when *zip* pauses for the next split destination.

Split archives cannot be updated, but see the **-O** (**--out**) option for how a split archive can be updated as it is copied to a new archive. A split archive can also be converted into a single-file archive using a split size of 0 or negating the **-s** option:

zip -s 0 split.zip --out single.zip

Also see **-U** (**--copy**) for more on using copy mode.

-sb

--split-bell

If splitting and using split pause mode, ring the bell when *zip* pauses for each split destination.

-sc

--show-command

Show the command line starting *zip* as processed and exit. The new command parser permutes the arguments, putting all options and any values associated with them before any non-option arguments. This allows an option to appear anywhere in the command line as long as any values that go with the option go with it. This option displays the command line as *zip* sees it, including any arguments from the environment such as from the **ZIPOPT** variable. Where allowed, options later in the command line can override options earlier in the command line.

-sf

--show-files

Show the files that would be operated on, then exit. For instance, if creating a new archive, this will list the files that would be added. If the option is negated, **-sf-**, output only to an open log file. Screen display is not recommended for large lists.

-so

--show-options

Show all available options supported by *zip* as compiled on the current system. As this command reads the option table, it should include all options. Each line includes the short option (if defined), the long option (if defined), the format of any value that goes with the option, if the option can be negated, and a small description. The value format can be no value, required value, optional value, single character value, number value, or a list of values. The output of this option is not intended to show how to use any option but only show what options are available.

-sp

--split-pause

If splitting is enabled with **-s**, enable split pause mode. This creates split archives as **-s** does, but stream writing is used so each split can be closed as soon as it is written and *zip* will pause between each split to allow changing split destination or media.

Though this split mode allows writing splits directly to removable media, it uses stream archive format that may not be readable by some unzips. Before relying on splits created with **-sp**, test a split archive with the unzip you will be using.

To convert a stream split archive (created with **-sp**) to a standard archive see the **--out** option.

-su

--show-unicode

As **-sf**, but also show Unicode version of the path if exists.

-sU

--show-just-unicode

As **-sf**, but only show Unicode version of the path if exists, otherwise show the standard version of the path.

-sv
--split-verbose
Enable various verbose messages while splitting, showing how the splitting is being done.

-S
--system-hidden
[MSDOS, OS/2, WIN32 and ATARI] Include system and hidden files.
[MacOS] Includes finder invisible files, which are ignored otherwise.

-t mmddyyyy
--from-date mmddyyyy
Do not operate on files modified prior to the specified date, where **mm** is the month (00-12), **dd** is the day of the month (01-31), and **yyyy** is the year. The *ISO 8601* date format **yyyy-mm-dd** is also accepted. For example:

```
zip -rt 12071991 infamy foo

zip -rt 1991-12-07 infamy foo
```

will add all the files in **foo** and its subdirectories that were last modified on or after 7 December 1991, to the *zip* archive **infamy.zip**.

-tt mmddyyyy
--before-date mmddyyyy
Do not operate on files modified after or at the specified date, where **mm** is the month (00-12), **dd** is the day of the month (01-31), and **yyyy** is the year. The *ISO 8601* date format **yyyy-mm-dd** is also accepted. For example:

```
zip -rtt 11301995 infamy foo

zip -rtt 1995-11-30 infamy foo
```

will add all the files in **foo** and its subdirectories that were last modified before 30 November 1995, to the *zip* archive **infamy.zip**.

-T
--test
Test the integrity of the new zip file. If the check fails, the old zip file is unchanged and (with the **-m** option) no input files are removed.

-TT cmd
--unzip-command cmd
Use command cmd instead of 'unzip -tqq' to test an archive when the **-T** option is used. On Unix, to use a copy of unzip in the current directory instead of the standard system unzip, could use:

```
zip archive file1 file2 -T -TT "./unzip -tqq"
```

In cmd, {} is replaced by the name of the temporary archive, otherwise the name of the archive is appended to the end of the command. The return code is checked for success (0 on Unix).

-u
--update

Replace (update) an existing entry in the *zip* archive only if it has been modified more recently than the version already in the *zip* archive. For example:

```
zip –u stuff *
```

will add any new files in the current directory, and update any files which have been modified since the *zip* archive *stuff.zip* was last created/modified (note that *zip* will not try to pack *stuff.zip* into itself when you do this).

Note that the **-u** option with no input file arguments acts like the **-f** (freshen) option.

-U
--copy-entries

Copy entries from one archive to another. Requires the **--out** option to specify a different output file than the input archive. Copy mode is the reverse of **-d** delete. When delete is being used with **--out**, the selected entries are deleted from the archive and all other entries are copied to the new archive, while copy mode selects the files to include in the new archive. Unlike **-u** update, input patterns on the command line are matched against archive entries only and not the file system files. For instance,

```
zip inarchive "*.c" --copy --out outarchive
```

copies entries with names ending in **.c** from **inarchive** to **outarchive**. The wildcard must be escaped on some systems to prevent the shell from substituting names of files from the file system which may have no relevance to the entries in the archive.

If no input files appear on the command line and **--out** is used, copy mode is assumed:

```
zip inarchive --out outarchive
```

This is useful for changing split size for instance. Encrypting and decrypting entries is not yet supported using copy mode. Use *zipcloak* for that.

-UN v
--unicode v

Determine what *zip* should do with Unicode file names. *zip 3.0*, in addition to the standard file path, now includes the UTF-8 translation of the path if the entry path is not entirely 7-bit ASCII. When an entry is missing the Unicode path, *zip* reverts back to the standard file path. The problem with using the standard path is this path is in the local character set of the zip that created the entry, which may contain characters that are not valid in the character set being used by the unzip. When *zip* is reading an archive, if an entry also has a Unicode path, *zip* now defaults to using the Unicode path to recreate the standard path using the current local character set.

This option can be used to determine what *zip* should do with this path if there is a mismatch between the stored standard path and the stored UTF-8 path (which can happen if the standard path was updated). In all cases, if there is a mismatch it is assumed that the standard path is more current and *zip* uses that. Values for **v** are

q - quit if paths do not match

w - warn, continue with standard path

i - ignore, continue with standard path

n - no Unicode, do not use Unicode paths

The default is to warn and continue.

Characters that are not valid in the current character set are escaped as **#Uxxxx** and **#Lxxxxxx**, where x is an ASCII character for a hex digit. The first is used if a 16-bit character number is sufficient to represent the Unicode character and the second if the character needs more than 16 bits to represent it's Unicode character code. Setting **-UN** to

e - escape

as in

```
zip archive -sU -UN=e
```

forces *zip* to escape all characters that are not printable 7-bit ASCII.

Normally *zip* stores UTF-8 directly in the standard path field on systems where UTF-8 is the current character set and stores the UTF-8 in the new extra fields otherwise. The option

u - UTF-8

as in

```
zip archive dir -r -UN=UTF8
```

forces *zip* to store UTF-8 as native in the archive. Note that storing UTF-8 directly is the default on Unix systems that support it. This option could be useful on Windows systems where the escaped path is too large to be a valid path and the UTF-8 version of the path is smaller, but native UTF-8 is not backward compatible on Windows systems.

-v
--verbose

Verbose mode or print diagnostic version info.
Normally, when applied to real operations, this option enables the display of a progress indicator during compression (see **-dd** for more on dots) and requests verbose diagnostic info about zipfile structure oddities.

However, when **-v** is the only command line argument a diagnostic screen is printed instead. This should now work even if stdout is redirected to a file, allowing easy saving of the information for sending with bug reports to Info-ZIP. The version screen provides the help screen header with program name, version, and release date, some pointers to the Info-ZIP home and distribution sites, and shows information about the target environment (compiler type and version, OS version, compilation date and the enabled optional features used to create the *zip* executable).

-V
--VMS-portable
[VMS] Save VMS file attributes. (Files are truncated at EOF.) When a -V archive is unpacked on a non-VMS system, some file types (notably Stream_LF text files and pure binary files like fixed-512) should be extracted intact. Indexed files and file types with embedded record sizes (notably variable-length record types) will probably be seen as corrupt elsewhere.

-VV
--VMS-specific
[VMS] Save VMS file attributes, and all allocated blocks in a file, including any data beyond EOF. Useful for moving ill-formed files among VMS systems. When a -VV archive is unpacked on a non-VMS system, almost all files will appear corrupt.

-w
--VMS-versions
[VMS] Append the version number of the files to the name, including multiple versions of files. Default is to use only the most recent version of a specified file.

-ww
--VMS-dot-versions
[VMS] Append the version number of the files to the name, including multiple versions of files, using the .nnn format. Default is to use only the most recent version of a specified file.

-ws
--wild-stop-dirs
Wildcards match only at a directory level. Normally *zip* handles paths as strings and given the paths

/foo/bar/dir/file1.c

/foo/bar/file2.c

an input pattern such as

/foo/bar/*

normally would match both paths, the * matching **dir/file1.c** and **file2.c**. Note that in the first case a directory boundary (/) was crossed in the match. With **-ws** no directory bounds will be included in the match, making wildcards local to a specific directory level. So, with **-ws** enabled, only the second path would be matched.

When using **-ws**, use ** to match across directory boundaries as * does normally.

-x files
--exclude files

Explicitly exclude the specified files, as in:

```
zip –r foo foo –x \*.o
```

which will include the contents of **foo** in **foo.zip** while excluding all the files that end in **.o**. The backslash avoids the shell filename substitution, so that the name matching is performed by *zip* at all directory levels.

Also possible:

```
zip –r foo foo –x@exclude.lst
```

which will include the contents of **foo** in **foo.zip** while excluding all the files that match the patterns in the file **exclude.lst**.

The long option forms of the above are

```
zip –r foo foo ––exclude \*.o
```

and

```
zip –r foo foo ––exclude @exclude.lst
```

Multiple patterns can be specified, as in:

```
zip –r foo foo –x \*.o \*.c
```

If there is no space between **-x** and the pattern, just one value is assumed (no list):

```
zip –r foo foo –x\*.o
```

See **-i** for more on include and exclude.

-X
--no-extra

Do not save extra file attributes (Extended Attributes on OS/2, uid/gid and file times on Unix). The zip format uses extra fields to include additional information for each entry. Some extra fields are specific to particular systems while others are applicable to all systems. Normally when *zip* reads entries from an existing archive, it reads the extra fields it knows, strips the rest, and adds the extra fields applicable to that system. With **-X**, *zip* strips all old fields and only includes the Unicode and Zip64 extra fields (currently these two extra fields cannot be disabled).

Negating this option, **-X-**, includes all the default extra fields, but also copies over any unrecognized extra fields.

-y
--symlinks
For UNIX and VMS (V8.3 and later), store symbolic links as such in the *zip* archive, instead of compressing and storing the file referred to by the link. This can avoid multiple copies of files being included in the archive as *zip* recurses the directory trees and accesses files directly and by links.

-z
--archive-comment
Prompt for a multi-line comment for the entire *zip* archive. The comment is ended by a line containing just a period, or an end of file condition (^D on Unix, ^Z on MSDOS, OS/2, and VMS). The comment can be taken from a file:

```
zip –z foo < foowhat
```

-Z cm
--compression-method cm
Set the default compression method. Currently the main methods supported by *zip* are **store** and **deflate**. Compression method can be set to:

store - Setting the compression method to **store** forces *zip* to store entries with no compression. This is generally faster than compressing entries, but results in no space savings. This is the same as using **-0** (compression level zero).

deflate - This is the default method for *zip*. If *zip* determines that storing is better than deflation, the entry will be stored instead.

bzip2 - If **bzip2** support is compiled in, this compression method also becomes available. Only some modern unzips currently support the **bzip2** compression method, so test the unzip you will be using before relying on archives using this method (compression method 12).

For example, to add **bar.c** to archive **foo** using **bzip2** compression:

zip -Z bzip2 foo bar.c

The compression method can be abbreviated:

zip -Zb foo bar.c

-#
(-0, -1, -2, -3, -4, -5, -6, -7, -8, -9)
Regulate the speed of compression using the specified digit **#**, where **-0** indicates no compression (store all files), **-1** indicates the fastest compression speed (less compression) and **-9** indicates the slowest compression speed (optimal compression, ignores the suffix list). The default compression level is **-6.**

Though still being worked, the intention is this setting will control compression speed for all compression methods. Currently only deflation is controlled.

-!

--use-privileges
[WIN32] Use priviliges (if granted) to obtain all aspects of WinNT security.

-@
--names-stdin
Take the list of input files from standard input. Only one filename per line.

-$
--volume-label
[MSDOS, OS/2, WIN32] Include the volume label for the drive holding the first file to be compressed. If you want to include only the volume label or to force a specific drive, use the drive name as first file name, as in:

```
zip –$ foo a: c:bar
```

Examples

The simplest example:

```
zip stuff *
```

creates the archive *stuff.zip* (assuming it does not exist) and puts all the files in the current directory in it, in compressed form (the **.zip** suffix is added automatically, unless the archive name contains a dot already; this allows the explicit specification of other suffixes).

Because of the way the shell on Unix does filename substitution, files starting with "." are not included; to include these as well:

```
zip stuff .* *
```

Even this will not include any subdirectories from the current directory.

To zip up an entire directory, the command:

```
zip –r foo foo
```

creates the archive *foo.zip*, containing all the files and directories in the directory *foo* that is contained within the current directory.

You may want to make a *zip* archive that contains the files in *foo*, without recording the directory name, *foo*. You can use the **-j** option to leave off the paths, as in:

```
zip –j foo foo/*
```

If you are short on disk space, you might not have enough room to hold both the original directory and the corresponding compressed *zip* archive. In this case, you can create the archive in steps using the **-m** option. If *foo* contains the subdirectories *tom*, *dick*, and *harry*, you can:

```
zip –rm foo foo/tom
zip –rm foo foo/dick
zip –rm foo foo/harry
```

where the first command creates *foo.zip*, and the next two add to it. At the completion of each *zip* command, the last created archive is deleted, making room for the next *zip* command to function.

Use **-s** to set the split size and create a split archive. The size is given as a number followed optionally by one of k (kB), m (MB), g (GB), or t (TB). The command

```
zip –s 2g –r split.zip foo
```

creates a split archive of the directory foo with splits no bigger than 2 GB each. If foo contained 5 GB of contents and the contents were stored in the split archive without compression (to make this example simple), this would create three splits, split.z01 at 2 GB, split.z02 at 2 GB, and split.zip at a little over 1 GB.

The **-sp** option can be used to pause *zip* between splits to allow changing removable media, for example, but read the descriptions and warnings for both **-s** and **-sp** below.

Though *zip* does not update split archives, *zip* provides the new option **-O** (**--output-file**) to allow split archives to be updated and saved in a new archive. For example,

```
zip inarchive.zip foo.c bar.c ––out outarchive.zip
```

reads archive **inarchive.zip**, even if split, adds the files **foo.c** and **bar.c**, and writes the resulting archive to **outarchive.zip**. If **inarchive.zip** is split then **outarchive.zip** defaults to the same split size. Be aware that **outarchive.zip** and any split files that are created with it are always overwritten without warning. This may be changed in the future.

Pattern Matching

This section applies only to Unix. Watch this space for details on MSDOS and VMS operation. However, the special wildcard characters ***** and **[]** below apply to at least MSDOS also.

The Unix shells (*sh*, *csh*, *bash*, and others) normally do filename substitution (also called "globbing") on command arguments. Generally the special characters are:

?

match any single character

*****

match any number of characters (including none)

[]

match any character in the range indicated within the brackets (example: [a-f], [0-9]). This form of wildcard matching allows a user to specify a list of characters between square brackets and if any of the characters match the expression matches. For example:

```
zip archive "*.[hc]"
```

would archive all files in the current directory that end in **.h** or **.c**.

Ranges of characters are supported:

```
zip archive "[a-f]*"
```

would add to the archive all files starting with "a" through "f".

Negation is also supported, where any character in that position not in the list matches. Negation is supported by adding **!** or **^** to the beginning of the list:

```
zip archive "*.[!o]"
```

matches files that don't end in ".o".

On WIN32, [] matching needs to be turned on with the -RE option to avoid the confusion that names with [or] have caused.

When these characters are encountered (without being escaped with a backslash or quotes), the shell will look for files relative to the current path that match the pattern, and replace the argument with a list of the names that matched.

The *zip* program can do the same matching on names that are in the *zip* archive being modified or, in the case of the **-x** (exclude) or **-i** (include) options, on the list of files to be operated on, by using backslashes or quotes to tell the shell not to do the name expansion. In general, when *zip* encounters a name in the list of files to do, it first looks for the name in the file system. If it finds it, it then adds it to the list of files to do. If it does not find it, it looks for the name in the *zip* archive being modified (if it exists), using the pattern matching characters described above, if present. For each match, it will add that name to the list of files to be processed, unless this name matches one given with the **-x** option, or does not match any name given with the **-i** option.

The pattern matching includes the path, and so patterns like *.o match names that end in ".o", no matter what the path prefix is. Note that the backslash must precede every special character (i.e. ?*[]), or the entire argument must be enclosed in double quotes ("").

In general, use backslashes or double quotes for paths that have wildcards to make *zip* do the pattern matching for file paths, and always for paths and strings that have spaces or wildcards for **-i**, **-x**, **-R**, **-d**, and **-U** and anywhere *zip* needs to process the wildcards.

Environment

The following environment variables are read and used by *zip* as described.

ZIPOPT
contains default options that will be used when running *zip*. The contents of this environment variable will get added to the command line just after the **zip** command.

ZIP
[Not on RISC OS and VMS] see ZIPOPT

Zip$Options
[RISC OS] see ZIPOPT

Zip$Exts
[RISC OS] contains extensions separated by a : that will cause native filenames with one of the specified extensions to be added to the zip file with basename and extension swapped.

ZIP_OPTS
[VMS] see ZIPOPT

See Also

compress(1), **shar**(1L), **tar**(1), **unzip**(1L), **gzip**(1L)

Diagnostics

The exit status (or error level) approximates the exit codes defined by PKWARE and takes on the following values, except under VMS:

0

normal; no errors or warnings detected.

2

unexpected end of zip file.

3

a generic error in the zipfile format was detected. Processing may have completed successfully anyway; some broken zipfiles created by other archivers have simple work-arounds.

4

zip was unable to allocate memory for one or more buffers during program initialization.

5

a severe error in the zipfile format was detected. Processing probably failed immediately.

6

entry too large to be processed (such as input files larger than 2 GB when not using Zip64 or trying to read an existing archive that is too large) or entry too large to be split with *zipsplit*

7

invalid comment format

8

zip -T failed or out of memory

9

the user aborted *zip* prematurely with control-C (or similar)

10

zip encountered an error while using a temp file

11

read or seek error

12

zip has nothing to do

13

missing or empty zip file

14

error writing to a file

15

zip was unable to create a file to write to

16

bad command line parameters

18

zip could not open a specified file to read

19

zip was compiled with options not supported on this system

VMS interprets standard Unix (or PC) return values as other, scarier-looking things, so *zip* instead maps them into VMS-style status codes. In general, *zip* sets VMS Facility = 1955 (0x07A3), Code = 2* Unix_status, and an appropriate Severity (as specified in ziperr.h). More details are included in the VMS-specific documentation. See [.vms]NOTES.TXT and [.vms]vms_msg_gen.c.

Bugs

zip 3.0 is not compatible with PKUNZIP 1.10. Use *zip* 1.1 to produce *zip* files which can be extracted by PKUNZIP 1.10.

zip files produced by *zip* 3.0 must not be *updated* by *zip* 1.1 or PKZIP 1.10, if they contain encrypted members or if they have been produced in a pipe or on a non-seekable device. The old versions of *zip* or PKZIP would create an archive with an incorrect format. The old versions can list the contents of the zip file but cannot extract it anyway (because of the new compression algorithm). If you do not use encryption and use regular disk files, you do not have to care about this problem.

Under VMS, not all of the odd file formats are treated properly. Only stream-LF format *zip* files are expected to work with *zip*. Others can be converted using Rahul Dhesi's BILF program. This version of *zip* handles some of the conversion internally. When using Kermit to transfer zip files from VMS to MSDOS, type "set file type block" on VMS. When transfering from MSDOS to VMS, type "set file type fixed" on VMS. In both cases, type "set file type binary" on MSDOS.

Under some older VMS versions, *zip* may hang for file specifications that use DECnet syntax *foo::*.*.*

On OS/2, zip cannot match some names, such as those including an exclamation mark or a hash sign. This is a bug in OS/2 itself: the 32-bit DosFindFirst/Next don't find such names. Other programs such as GNU tar are also affected by this bug.

Under OS/2, the amount of Extended Attributes displayed by DIR is (for compatibility) the amount returned by the 16-bit version of DosQueryPathInfo(). Otherwise OS/2 1.3 and 2.0 would report different EA sizes when DIRing a file. However, the structure layout returned by the 32-bit DosQueryPathInfo() is a bit different, it uses extra padding bytes and link pointers (it's a linked list) to have all fields on 4-byte boundaries for portability to future RISC OS/2 versions. Therefore the value reported by *zip* (which uses this 32-bit-mode size) differs from that reported by DIR. *zip* stores the 32-bit format for portability, even the 16-bit MS-C-compiled version running on OS/2 1.3, so even this one shows the 32-bit-mode size.

Authors

Please send bug reports and comments using the web page at: *www.info-zip.org*. For bug reports, please include the version of *zip* (see *zip -h*), the make options used to compile it (see *zip -v*), the machine and operating system in use, and as much additional information as possible.

Acknowledgements

Thanks to R. P. Byrne for his *Shrink.Pas* program, which inspired this project, and from which the shrink algorithm was stolen; to Phil Katz for placing in the public domain the *zip* file format, compression format, and .ZIP filename extension, and for accepting minor changes to the file format; to Steve Burg for clarifications on the deflate format; to Haruhiko Okumura and Leonid Broukhis for providing some useful ideas for the compression algorithm; to Keith Petersen, Rich Wales, Hunter Goatley and Mark Adler for providing a mailing list and *ftp* site for the Info-ZIP group to use; and most importantly, to the Info-ZIP group itself (listed in the file *infozip.who*) without whose tireless testing and bug-fixing efforts a portable *zip* would not have been possible. Finally we should thank (blame) the first Info-ZIP moderator, David Kirschbaum, for getting us into this mess in the first place. The manual page was rewritten for Unix by R. P. C. Rodgers and updated by E. Gordon for *zip* 3.0.

Referenced By

7z(1), **7za**(1), **archivemount**(1), **ark**(1), **cboard**(6), **gzip**(1), **libarchive-formats**(5), **lrunzip**(1), **lrzcat**(1), **lrzip**(1), **lrztar**(1), **lrzuntar**(1), **pbzip2**(1), **stone**(3), **unzipsfx**(1), **zipcloak**(1), **zipnote**(1), **zipsplit**(1)