NAME

funzip − filter for extracting from a ZIP archive in a pipe

SYNOPSIS

funzip [**−password**] [*input[.zip*|*.gz]*]

ARGUMENTS

[*−password*]

Optional password to be used if ZIP archive is encrypted. Decryption may not be supported at some sites. See DESCRIPTION for more details.

[*input[.zip*|*.gz]*]

Optional input archive file specification. See DESCRIPTION for details.

DESCRIPTION

funzip without a file argument acts as a filter; that is, it assumes that a ZIP archive (or a *gzip*'d(1) file) is being piped into standard input, and it extracts the first member from the archive to stdout. When stdin comes from a tty device, *funzip* assumes that this cannot be a stream of (binary) compressed data and shows a short help text, instead. If there is a file argument, then input is read from the specified file instead of from stdin.

A password for encrypted zip files can be specified on the command line (preceding the file name, if any) by prefixing the password with a dash. Note that this constitutes a security risk on many systems; currently running processes are often visible via simple commands (e.g., *ps*(1) under Unix), and command-line histories can be read. If the first entry of the zip file is encrypted and no password is specified on the command line, then the user is prompted for a password and the password is not echoed on the console.

Given the limitation on single-member extraction, *funzip* is most useful in conjunction with a secondary archiver program such as *tar*(1). The following section includes an example illustrating this usage in the case of disk backups to tape.

EXAMPLES

To use *funzip* to extract the first member file of the archive test.zip and to pipe it into *more*(1):

```
funzip test.zip | more
```

To use *funzip* to test the first member file of test.zip (any errors will be reported on standard error):

```
funzip test.zip > /dev/null
```

To use *zip* and *funzip* in place of *compress*(1) and *zcat*(1) (or *gzip*(1L) and *gzcat*(1L)) for tape backups:

```
tar cf − . | zip −7 | dd of=/dev/nrst0 obs=8k
dd if=/dev/nrst0 ibs=8k | funzip | tar xf −
```

(where, for example, nrst0 is a SCSI tape drive).

BUGS

When piping an encrypted file into *more* and allowing *funzip* to prompt for password, the terminal may sometimes be reset to a non-echo mode. This is apparently due to a race condition between the two programs; *funzip* changes the terminal mode to non-echo before *more* reads its state, and *more* then ‘‘restores’’ the terminal to this mode before exiting. To recover, run *funzip* on the same file but redirect to /dev/null rather than piping into more; after prompting again for the password, *funzip* will reset the terminal properly.

There is presently no way to extract any member but the first from a ZIP archive. This would be useful in the case where a ZIP archive is included within another archive. In the case where the first member is a directory, *funzip* simply creates the directory and exits.

The functionality of *funzip* should be incorporated into *unzip* itself (future release).

SEE ALSO

gzip(1L), *unzip*(1L), *unzipsfx*(1L), *zip*(1L), *zipcloak*(1L), *zipinfo*(1L), *zipnote*(1L), *zipsplit*(1L)

URL

The Info-ZIP home page is currently at

```
http://www.info-zip.org/pub/infozip/
```

or

`ftp://ftp.info-zip.org/pub/infozip/` .

AUTHOR

Mark Adler (Info-ZIP)

NAME

unzip − list, test and extract compressed files in a ZIP archive

SYNOPSIS

unzip [**−Z**] [**−cflptuvz**[**abjnoqsCLMVX$/:**]] *file*[*.zip*] [*file(s)* . . .] [**−x** *xfile(s)* . . .] [**−d** *exdir*]

DESCRIPTION

unzip will list, test, or extract files from a ZIP archive, commonly found on MS-DOS systems. The default behavior (with no options) is to extract into the current directory (and subdirectories below it) all files from the specified ZIP archive. A companion program, *zip*(1L), creates ZIP archives; both programs are compatible with archives created by PKWARE's *PKZIP* and *PKUNZIP* for MS-DOS, but in many cases the program options or default behaviors differ.

ARGUMENTS

file[*.zip*]

Path of the ZIP archive(s). If the file specification is a wildcard, each matching file is processed in an order determined by the operating system (or file system). Only the filename can be a wildcard; the path itself cannot. Wildcard expressions are similar to those supported in commonly used Unix shells (*sh*, *ksh*, *csh*) and may contain:

*	matches a sequence of 0 or more characters
?	matches exactly 1 character
[. . .]	matches any single character found inside the brackets; ranges are specified by a beginning character, a hyphen, and an ending character. If an exclamation point or a caret ('!' or '^') follows the left bracket, then the range of characters within the brackets is complemented (that is, anything *except* the characters inside the brackets is considered a match).

(Be sure to quote any character that might otherwise be interpreted or modified by the operating system, particularly under Unix and VMS.) If no matches are found, the specification is assumed to be a literal filename; and if that also fails, the suffix `.zip` is appended. Note that self-extracting ZIP files are supported, as with any other ZIP archive; just specify the `.exe` suffix (if any) explicitly.

[*file(s)*] An optional list of archive members to be processed, separated by spaces. (VMS versions compiled with VMSCLI defined must delimit files with commas instead. See **−v** in **OPTIONS** below.) Regular expressions (wildcards) may be used to match multiple members; see above. Again, be sure to quote expressions that would otherwise be expanded or modified by the operating system.

[**−x** *xfile(s)*]

An optional list of archive members to be excluded from processing. Since wildcard characters match directory separators ('/'), this option may be used to exclude any files that are in subdirectories. For example, "`unzip foo *.[ch] -x */*`" would extract all C source files in the main directory, but none in any subdirectories. Without the **−x** option, all C source files in all directories within the zipfile would be extracted.

[**−d** *exdir*]

An optional directory to which to extract files. By default, all files and subdirectories are recreated in the current directory; the **−d** option allows extraction in an arbitrary directory (always assuming one has permission to write to the directory). This option need not appear at the end of the command line; it is also accepted before the zipfile specification (with the normal options), immediately after the zipfile specification, or between the *file(s)* and the **−x** option. The option and directory may be concatenated without any white space between them, but note that this may cause normal shell behavior to be suppressed. In particular, "`−d ˜`" (tilde) is expanded by Unix C shells into the name of the user's home directory, but "`−d˜`" is treated as a literal subdirectory "`˜`" of the current directory.

OPTIONS

Note that, in order to support obsolescent hardware, *unzip*'s usage screen is limited to 22 or 23 lines and should therefore be considered only a reminder of the basic *unzip* syntax rather than an exhaustive list of all possible flags. The exhaustive list follows:

−Z *zipinfo*(1L) mode. If the first option on the command line is **−Z**, the remaining options are taken to be *zipinfo*(1L) options. See the appropriate manual page for a description of these options.

−A [OS/2, Unix DLL] print extended help for the DLL's programming interface (API).

−c extract files to stdout/screen (''CRT''). This option is similar to the **−p** option except that the name of each file is printed as it is extracted, the **−a** option is allowed, and ASCII-EBCDIC conversion is automatically performed if appropriate. This option is not listed in the *unzip* usage screen.

−f freshen existing files, i.e., extract only those files that already exist on disk and that are newer than the disk copies. By default *unzip* queries before overwriting, but the **−o** option may be used to suppress the queries. Note that under many operating systems, the TZ (timezone) environment variable must be set correctly in order for **−f** and **−u** to work properly (under Unix the variable is usually set automatically). The reasons for this are somewhat subtle but have to do with the differences between DOS-format file times (always local time) and Unix-format times (always in GMT/UTC) and the necessity to compare the two. A typical TZ value is ''PST8PDT'' (US Pacific time with automatic adjustment for Daylight Savings Time or ''summer time'').

−l list archive files (short format). The names, uncompressed file sizes and modification dates and times of the specified files are printed, along with totals for all files specified. If UnZip was compiled with OS2_EAS defined, the **−l** option also lists columns for the sizes of stored OS/2 extended attributes (EAs) and OS/2 access control lists (ACLs). In addition, the zipfile comment and individual file comments (if any) are displayed. If a file was archived from a single-case file system (for example, the old MS-DOS FAT file system) and the **−L** option was given, the filename is converted to lowercase and is prefixed with a caret (^).

−p extract files to pipe (stdout). Nothing but the file data is sent to stdout, and the files are always extracted in binary format, just as they are stored (no conversions).

−t test archive files. This option extracts each specified file in memory and compares the CRC (cyclic redundancy check, an enhanced checksum) of the expanded file with the original file's stored CRC value.

−T [most OSes] set the timestamp on the archive(s) to that of the newest file in each one. This corresponds to *zip*'s **−go** option except that it can be used on wildcard zipfiles (e.g., ''`unzip -T \*.zip`'') and is much faster.

−u update existing files and create new ones if needed. This option performs the same function as the **−f** option, extracting (with query) files that are newer than those with the same name on disk, and in addition it extracts those files that do not already exist on disk. See **−f** above for information on setting the timezone properly.

−v be verbose or print diagnostic version info. This option has evolved and now behaves as both an option and a modifier. As an option it has two purposes: when a zipfile is specified with no other options, **−v** lists archive files verbosely, adding to the basic **−l** info the compression method, compressed size, compression ratio and 32-bit CRC. When no zipfile is specified (that is, the complete command is simply ''`unzip -v`''), a diagnostic screen is printed. In addition to the normal header with release date and version, *unzip* lists the home Info-ZIP ftp site and where to find a list of other ftp and non-ftp sites; the target operating system for which it was compiled, as well as (possibly) the hardware on which it was compiled, the compiler and version used, and the compilation date; any special compilation options that might affect the program's operation (see also **DECRYPTION** below); and any options stored in environment variables that might do the same (see **ENVIRONMENT OPTIONS** below). As a modifier it works in conjunction with other options (e.g., **−t**) to produce more verbose or debugging output; this is not yet fully implemented but will be in future releases.

−z display only the archive comment.

MODIFIERS

−a convert text files. Ordinarily all files are extracted exactly as they are stored (as ''binary'' files). The **−a** option causes files identified by *zip* as text files (those with the 't' label in *zipinfo* listings, rather than 'b') to be automatically extracted as such, converting line endings, end-of-file characters and the character set itself as necessary. (For example, Unix files use line feeds (LFs) for end-of-line (EOL) and have no end-of-file (EOF) marker; Macintoshes use carriage returns (CRs) for EOLs; and most PC operating systems use CR+LF for EOLs and control-Z for EOF. In addition, IBM mainframes and the Michigan Terminal System use EBCDIC rather than the more common ASCII character set, and NT supports Unicode.) Note that *zip*'s identification of text files is by no means perfect; some ''text'' files may actually be binary and vice versa. *unzip* therefore prints ''`[text]`'' or ''`[binary]`'' as a visual check for each file it extracts when using the **−a** option. The **−aa** option forces all files to be extracted as text, regardless of the supposed file type.

−b [general] treat all files as binary (no text conversions). This is a shortcut for `---`**a**.

−b [Tandem] force the creation files with filecode type 180 ('C') when extracting Zip entries marked as "text". (On Tandem, **−a** is enabled by default, see above).

−b [VMS] auto-convert binary files (see **−a** above) to fixed-length, 512-byte record format. Doubling the option (**−bb**) forces all files to be extracted in this format. When extracting to standard output (**−c** or **−p** option in effect), the default conversion of text record delimiters is disabled for binary (**−b**) resp. all (**−bb**) files.

−B [Unix only, and only if compiled with UNIXBACKUP defined] save a backup copy of each overwritten file with a tilde appended (e.g., the old copy of ''`foo`'' is renamed to ''`foo~`''). This is similar to the default behavior of *emacs*(1) in many locations.

−C match filenames case-insensitively. *unzip*'s philosophy is ''you get what you ask for'' (this is also responsible for the **−L**/**−U** change; see the relevant options below). Because some file systems are fully case-sensitive (notably those under the Unix operating system) and because both ZIP archives and *unzip* itself are portable across platforms, *unzip*'s default behavior is to match both wildcard and literal filenames case-sensitively. That is, specifying ''`makefile`'' on the command line will *only* match ''makefile'' in the archive, not ''Makefile'' or ''MAKEFILE'' (and similarly for wildcard specifications). Since this does not correspond to the behavior of many other operating/file systems (for example, OS/2 HPFS, which preserves mixed case but is not sensitive to it), the **−C** option may be used to force all filename matches to be case-insensitive. In the example above, all three files would then match ''`makefile`'' (or ''`make*`'', or similar). The **−C** option affects files in both the normal file list and the excluded-file list (xlist).

−E [MacOS only] display contents of MacOS extra field during restore operation.

−F [Acorn only] suppress removal of NFS filetype extension from stored filenames.

−F [non-Acorn systems supporting long filenames with embedded commas, and only if compiled with ACORN_FTYPE_NFS defined] translate filetype information from ACORN RISC OS extra field blocks into a NFS filetype extension and append it to the names of the extracted files. (When the stored filename appears to already have an appended NFS filetype extension, it is replaced by the info from the extra field.)

−i [MacOS only] ignore filenames stored in MacOS extra fields. Instead, the most compatible filename stored in the generic part of the entry's header is used.

−j junk paths. The archive's directory structure is not recreated; all files are deposited in the extraction directory (by default, the current one).

−J [BeOS only] junk file attributes. The file's BeOS file attributes are not restored, just the file's data.

−J [MacOS only] ignore MacOS extra fields. All Macintosh specific info is skipped. Data-fork and resource-fork are restored as separate files.

−L convert to lowercase any filename originating on an uppercase-only operating system or file system. (This was *unzip*'s default behavior in releases prior to 5.11; the new default behavior is identical to the old behavior with the **−U** option, which is now obsolete and will be removed in a future release.) Depending on the archiver, files archived under single-case file systems (VMS, old MS-DOS FAT, etc.) may be stored as all-uppercase names; this can be ugly or inconvenient when extracting to a case-preserving file system such as OS/2 HPFS or a case-sensitive one such as under Unix. By default *unzip* lists and extracts such filenames exactly as they're stored (excepting truncation, conversion of unsupported characters, etc.); this option causes the names of all files from certain systems to be converted to lowercase. The **−LL** option forces conversion of every filename to lowercase, regardless of the originating file system.

−M pipe all output through an internal pager similar to the Unix *more*(1) command. At the end of a screenful of output, *unzip* pauses with a "−−More−−" prompt; the next screenful may be viewed by pressing the Enter (Return) key or the space bar. *unzip* can be terminated by pressing the "q" key and, on some systems, the Enter/Return key. Unlike Unix *more*(1), there is no forward-searching or editing capability. Also, *unzip* doesn't notice if long lines wrap at the edge of the screen, effectively resulting in the printing of two or more lines and the likelihood that some text will scroll off the top of the screen before being viewed. On some systems the number of available lines on the screen is not detected, in which case *unzip* assumes the height is 24 lines.

−n never overwrite existing files. If a file already exists, skip the extraction of that file without prompting. By default *unzip* queries before extracting any file that already exists; the user may choose to overwrite only the current file, overwrite all files, skip extraction of the current file, skip extraction of all existing files, or rename the current file.

−N [Amiga] extract file comments as Amiga filenotes. File comments are created with the −c option of *zip*(1L), or with the −N option of the Amiga port of *zip*(1L), which stores filenotes as comments.

−o overwrite existing files without prompting. This is a dangerous option, so use it with care. (It is often used with **−f**, however, and is the only way to overwrite directory EAs under OS/2.)

−P *password*
use *password* to decrypt encrypted 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 encryption provided by standard zipfile utilities.)

−q perform operations quietly (**−qq** = even quieter). Ordinarily *unzip* prints the names of the files it's extracting or testing, the extraction methods, any file or zipfile comments that may be stored in the archive, and possibly a summary when finished with each archive. The **−q**[**q**] options suppress the printing of some or all of these messages.

−s [OS/2, NT, MS-DOS] convert spaces in filenames to underscores. Since all PC operating systems allow spaces in filenames, *unzip* by default extracts filenames with spaces intact (e.g., "`EA DATA. SF`"). This can be awkward, however, since MS-DOS in particular does not gracefully support spaces in filenames. Conversion of spaces to underscores can eliminate the awkwardness in some cases.

−U (obsolete; to be removed in a future release) leave filenames uppercase if created under MS-DOS, VMS, etc. See **−L** above.

−V retain (VMS) file version numbers. VMS files can be stored with a version number, in the format `file.ext;##`. By default the "`;##`" version numbers are stripped, but this option allows them to be retained. (On file systems that limit filenames to particularly short lengths, the version numbers may be truncated or stripped regardless of this option.)

−X [VMS, Unix, OS/2, NT] restore owner/protection info (UICs) under VMS, or user and group info (UID/GID) under Unix, or access control lists (ACLs) under certain network-enabled versions of OS/2 (Warp Server with IBM LAN Server/Requester 3.0 to 5.0; Warp Connect with IBM Peer 1.0), or security ACLs under Windows NT. In most cases this will require special system privileges, and doubling the option (**−XX**) under NT instructs *unzip* to use privileges for extraction; but under Unix, for example, a user who belongs to several groups can restore files owned by any of those groups, as long as the user IDs match his or her own. Note that ordinary file attributes are always restored--this option applies only to optional, extra ownership info available on some operating systems. [NT's access control lists do not appear to be especially compatible with OS/2's, so no attempt is made at cross-platform portability of access privileges. It is not clear under what conditions this would ever be useful anyway.]

−$** [MS-DOS, OS/2, NT] restore the volume label if the extraction medium is removable (e.g., a diskette). Doubling the option (**−$$) allows fixed media (hard disks) to be labelled as well. By default, volume labels are ignored.

−/ *extensions*
[Acorn only] overrides the extension list supplied by Unzip$Ext environment variable. During extraction, filename extensions that match one of the items in this extension list are swapped in front of the base name of the extracted file.

−: [all but Acorn, VM/CMS, MVS, Tandem] allows to extract archive members into locations outside of the current '' extraction root folder''. For security reasons, *unzip* normally removes ''parent dir'' path components (''../'') from the names of extracted file. This safety feature (new for version 5.50) prevents *unzip* from accidentally writing files to ''sensitive'' areas outside the active extraction folder tree head. The **−:** option lets *unzip* switch back to its previous, more liberal behaviour, to allow exact extraction of (older) archives that used ''../'' components to create multiple directory trees at the level of the current extraction folder. This option does not enable writing explicitly to the root directory (''/''). To achieve this, it is necessary to set the extraction target folder to root (e.g. **−d /**). However, when the **−:** option is specified, it is still possible to implicitly write to the root directory by specifiying enough ''../'' path components within the zip file. Use this option with extreme caution.

ENVIRONMENT OPTIONS

unzip's default behavior may be modified via options placed in an environment variable. This can be done with any option, but it is probably most useful with the **−a**, **−L**, **−C**, **−q**, **−o**, or **−n** modifiers: make *unzip* auto-convert text files by default, make it convert filenames from uppercase systems to lowercase, make it match names case-insensitively, make it quieter, or make it always overwrite or never overwrite files as it extracts them. For example, to make *unzip* act as quietly as possible, only reporting errors, one would use one of the following commands:

Unix Bourne shell:
```
UNZIP=−qq; export UNZIP
```

Unix C shell:
```
setenv UNZIP −qq
```

OS/2 or MS-DOS:
```
set UNZIP=−qq
```

VMS (quotes for *lowercase*):
```
define UNZIP_OPTS ""−qq""
```

Environment options are, in effect, considered to be just like any other command-line options, except that they are effectively the first options on the command line. To override an environment option, one may use the ''minus operator'' to remove it. For instance, to override one of the quiet-flags in the example above, use the command

```
unzip −−q[other options] zipfile
```

The first hyphen is the normal switch character, and the second is a minus sign, acting on the q option. Thus the effect here is to cancel one quantum of quietness. To cancel both quiet flags, two (or more) minuses may be used:

```
unzip −t−−q zipfile
unzip −−−qt zipfile
```

(the two are equivalent). This may seem awkward or confusing, but it is reasonably intuitive: just ignore the first hyphen and go from there. It is also consistent with the behavior of Unix *nice*(1).

As suggested by the examples above, the default variable names are UNZIP_OPTS for VMS (where the symbol used to install *unzip* as a foreign command would otherwise be confused with the environment variable), and UNZIP for all other operating systems. For compatibility with *zip*(1L), UNZIPOPT is also accepted (don't ask). If both UNZIP and UNZIPOPT are defined, however, UNZIP takes precedence. *unzip*'s diagnostic option (**−v** with no zipfile name) can be used to check the values of all four possible *unzip* and *zipinfo* environment variables.

The timezone variable (TZ) should be set according to the local timezone in order for the **−f** and **−u** to operate correctly. See the description of **−f** above for details. This variable may also be necessary in order for timestamps on extracted files to be set correctly. Under Windows 95/NT *unzip* should know the correct timezone even if TZ is unset, assuming the timezone is correctly set in the Control Panel.

DECRYPTION

Encrypted archives are fully supported by Info-ZIP software, but due to United States export restrictions, de-/encryption support might be disabled in your compiled binary. However, since spring 2000, US export restrictions have been liberated, and our source archives do now include full crypt code. In case you need binary distributions with crypt support enabled, see the file ''WHERE'' in any Info-ZIP source or binary distribution for locations both inside and outside the US.

Some compiled versions of *unzip* may not support decryption. To check a version for crypt support, either attempt to test or extract an encrypted archive, or else check *unzip*'s diagnostic screen (see the **−v** option above) for ''`[decryption]`'' as one of the special compilation options.

As noted above, the **−P** option may be used to supply a password on the command line, but at a cost in security. The preferred decryption method is simply to extract normally; if a zipfile member is encrypted, *unzip* will prompt for the password without echoing what is typed. *unzip* continues to use the same password as long as it appears to be valid, by testing a 12-byte header on each file. The correct password will always check out against the header, but there is a 1-in-256 chance that an incorrect password will as well. (This is a security feature of the PKWARE zipfile format; it helps prevent brute-force attacks that might otherwise gain a large speed advantage by testing only the header.) In the case that an incorrect password is given but it passes the header test anyway, either an incorrect CRC will be generated for the extracted data or else *unzip* will fail during the extraction because the ''decrypted'' bytes do not constitute a valid compressed data stream.

If the first password fails the header check on some file, *unzip* will prompt for another password, and so on until all files are extracted. If a password is not known, entering a null password (that is, just a carriage return or ''Enter'') is taken as a signal to skip all further prompting. Only unencrypted files in the archive(s) will thereafter be extracted. (In fact, that's not quite true; older versions of *zip*(1L) and *zipcloak*(1L) allowed null passwords, so *unzip* checks each encrypted file to see if the null password works. This may result in ''false positives'' and extraction errors, as noted above.)

Archives encrypted with 8-bit passwords (for example, passwords with accented European characters) may not be portable across systems and/or other archivers. This problem stems from the use of multiple encoding methods for such characters, including Latin-1 (ISO 8859-1) and OEM code page 850. DOS *PKZIP* 2.04g uses the OEM code page; Windows *PKZIP* 2.50 uses Latin-1 (and is therefore incompatible with DOS *PKZIP*); Info-ZIP uses the OEM code page on DOS, OS/2 and Win3.x ports but Latin-1 everywhere else; and Nico Mak's *WinZip* 6.x does not allow 8-bit passwords at all. *UnZip* 5.3 (or newer) attempts to use the default character set first (e.g., Latin-1), followed by the alternate one (e.g., OEM code page) to test passwords. On EBCDIC systems, if both of these fail, EBCDIC encoding will be tested as a last resort. (EBCDIC is not tested on non-EBCDIC systems, because there are no known archivers that encrypt using EBCDIC encoding.) ISO character encodings other than Latin-1 are not supported.

EXAMPLES

To use *unzip* to extract all members of the archive *letters.zip* into the current directory and subdirectories below it, creating any subdirectories as necessary:

```
unzip letters
```

To extract all members of *letters.zip* into the current directory only:

```
unzip -j letters
```

To test *letters.zip*, printing only a summary message indicating whether the archive is OK or not:

```
unzip -tq letters
```

To test *all* zipfiles in the current directory, printing only the summaries:

```
unzip -tq \*.zip
```

(The backslash before the asterisk is only required if the shell expands wildcards, as in Unix; double quotes could have been used instead, as in the source examples below.) To extract to standard output all members of *letters.zip* whose names end in *.tex*, auto-converting to the local end-of-line convention and piping the output into *more*(1):

```
unzip −ca letters \*.tex | more
```

To extract the binary file *paper1.dvi* to standard output and pipe it to a printing program:

```
unzip −p articles paper1.dvi | dvips
```

To extract all FORTRAN and C source files--*.f, *.c, *.h, and Makefile--into the /tmp directory:

```
unzip source.zip "*.[fch]" Makefile -d /tmp
```

(the double quotes are necessary only in Unix and only if globbing is turned on). To extract all FORTRAN and C source files, regardless of case (e.g., both *.c and *.C, and any makefile, Makefile, MAKEFILE or similar):

```
unzip −C source.zip "*.[fch]" makefile -d /tmp
```

To extract any such files but convert any uppercase MS-DOS or VMS names to lowercase and convert the line-endings of all of the files to the local standard (without respect to any files that might be marked ‘‘binary’’):

```
unzip −aaCL source.zip "*.[fch]" makefile -d /tmp
```

To extract only newer versions of the files already in the current directory, without querying (NOTE: be careful of unzipping in one timezone a zipfile created in another--ZIP archives other than those created by Zip 2.1 or later contain no timezone information, and a ‘‘newer’’ file from an eastern timezone may, in fact, be older):

```
unzip −fo sources
```

To extract newer versions of the files already in the current directory and to create any files not already there (same caveat as previous example):

```
unzip −uo sources
```

To display a diagnostic screen showing which *unzip* and *zipinfo* options are stored in environment variables, whether decryption support was compiled in, the compiler with which *unzip* was compiled, etc.:

```
unzip −v
```

In the last five examples, assume that UNZIP or UNZIP_OPTS is set to -q. To do a singly quiet listing:

```
unzip −l file.zip
```

To do a doubly quiet listing:

```
unzip −ql file.zip
```

(Note that the ‘‘`.zip`’’ is generally not necessary.) To do a standard listing:

```
unzip −−ql file.zip
```

or

```
unzip −l−q file.zip
```

or

```
unzip −l−−q file.zip
```

(Extra minuses in options don’t hurt.)

TIPS

The current maintainer, being a lazy sort, finds it very useful to define a pair of aliases: `tt` for ‘‘`unzip −tq`’’ and `ii` for ‘‘`unzip −Z`’’ (or ‘‘`zipinfo`’’). One may then simply type ‘‘`tt zipfile`’’ to test an archive, something that is worth making a habit of doing. With luck *unzip* will report ‘‘`No errors detected in compressed data of zipfile.zip`,’’ after which one may breathe a sigh of relief.

The maintainer also finds it useful to set the UNZIP environment variable to ‘‘`−aL`’’ and is tempted to add ‘‘`−C`’’ as well. His ZIPINFO variable is set to ‘‘`−z`’’.

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.

1	one or more warning errors were encountered, but processing completed successfully anyway. This includes zipfiles where one or more files was skipped due to unsupported compression method or encryption with an unknown password.
2	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.
3	a severe error in the zipfile format was detected. Processing probably failed immediately.
4	*unzip* was unable to allocate memory for one or more buffers during program initialization.
5	*unzip* was unable to allocate memory or unable to obtain a tty to read the decryption password(s).
6	*unzip* was unable to allocate memory during decompression to disk.
7	*unzip* was unable to allocate memory during in-memory decompression.
8	[currently not used]
9	the specified zipfiles were not found.
10	invalid options were specified on the command line.
11	no matching files were found.
50	the disk is (or was) full during extraction.
51	the end of the ZIP archive was encountered prematurely.
80	the user aborted *unzip* prematurely with control-C (or similar)
81	testing or extraction of one or more files failed due to unsupported compression methods or unsupported decryption.
82	no files were found due to bad decryption password(s). (If even one file is successfully processed, however, the exit status is 1.)

VMS interprets standard Unix (or PC) return values as other, scarier-looking things, so *unzip* instead maps them into VMS-style status codes. The current mapping is as follows: 1 (success) for normal exit, 0x7fff0001 for warning errors, and (0x7fff000? + 16*normal_unzip_exit_status) for all other errors, where the '?' is 2 (error) for *unzip* values 2, 9-11 and 80-82, and 4 (fatal error) for the remaining ones (3-8, 50, 51). In addition, there is a compilation option to expand upon this behavior: defining RETURN_CODES results in a human-readable explanation of what the error status means.

BUGS

Multi-part archives are not yet supported, except in conjunction with *zip*. (All parts must be concatenated together in order, and then ``zip -F'' must be performed on the concatenated archive in order to ``fix'' it.) This will definitely be corrected in the next major release.

Archives read from standard input are not yet supported, except with *funzip* (and then only the first member of the archive can be extracted).

Archives encrypted with 8-bit passwords (e.g., passwords with accented European characters) may not be portable across systems and/or other archivers. See the discussion in **DECRYPTION** above.

unzip's **−M** (``more'') option tries to take into account automatic wrapping of long lines. However, the code may fail to detect the correct wrapping locations. First, TAB characters (and similar control sequences) are not taken into account, they are handled as ordinary printable characters. Second, depending on the actual system / OS port, *unzip* may not detect the true screen geometry but rather rely on "commonly used" default dimensions. The correct handling of tabs would require the implementation of a query for the actual tabulator setup on the output console.

Dates, times and permissions of stored directories are not restored except under Unix. (On Windows NT and successors, timestamps are now restored.)

[MS-DOS] When extracting or testing files from an archive on a defective floppy diskette, if the ``Fail'' option is chosen from DOS's ``Abort, Retry, Fail?'' message, older versions of *unzip* may hang the system, requiring a reboot. This problem appears to be fixed, but control-C (or control-Break) can still be used to terminate *unzip*.

Under DEC Ultrix, *unzip* would sometimes fail on long zipfiles (bad CRC, not always reproducible). This was apparently due either to a hardware bug (cache memory) or an operating system bug (improper handling of page faults?). Since Ultrix has been abandoned in favor of Digital Unix (OSF/1), this may not be an issue anymore.

[Unix] Unix special files such as FIFO buffers (named pipes), block devices and character devices are not restored even if they are somehow represented in the zipfile, nor are hard-linked files relinked. Basically the only file types restored by *unzip* are regular files, directories and symbolic (soft) links.

[OS/2] Extended attributes for existing directories are only updated if the **−o** (‘‘overwrite all’’) option is given. This is a limitation of the operating system; because directories only have a creation time associated with them, *unzip* has no way to determine whether the stored attributes are newer or older than those on disk. In practice this may mean a two-pass approach is required: first unpack the archive normally (with or without freshening/updating existing files), then overwrite just the directory entries (e.g., ‘‘`unzip -o foo */`’’).

[VMS] When extracting to another directory, only the *[.foo]* syntax is accepted for the **−d** option; the simple Unix *foo* syntax is silently ignored (as is the less common VMS *foo.dir* syntax).

[VMS] When the file being extracted already exists, *unzip*’s query only allows skipping, overwriting or renaming; there should additionally be a choice for creating a new version of the file. In fact, the ‘‘overwrite’’ choice does create a new version; the old version is not overwritten or deleted.

SEE ALSO

funzip(1L), *zip*(1L), *zipcloak*(1L), *zipgrep*(1L), *zipinfo*(1L), *zipnote*(1L), *zipsplit*(1L)

URL

The Info-ZIP home page is currently at

`http://www.info-zip.org/pub/infozip/`

or

`ftp://ftp.info-zip.org/pub/infozip/` .

AUTHORS

The primary Info-ZIP authors (current semi-active members of the Zip-Bugs workgroup) are: Onno van der Linden (Zip); Christian Spieler (UnZip maintenance coordination, VMS, MS-DOS, Win32, shared code, general Zip and UnZip integration and optimization); Mike White (Windows GUI, Windows DLLs); Kai Uwe Rommel (OS/2); Paul Kienitz (Amiga, Win32); Chris Herborth (BeOS, QNX, Atari); Jonathan Hudson (SMS/QDOS); Sergio Monesi (Acorn RISC OS); Harald Denker (Atari, MVS); John Bush (Solaris, Amiga); Hunter Goatley (VMS); Steve Salisbury (Win32); Steve Miller (Windows CE GUI), Johnny Lee (MS-DOS, Win32); and Dave Smith (Tandem NSK).

The following people were former members of the Info-ZIP development group and provided major contributions to key parts of the current code: Greg ‘‘Cave Newt’’ Roelofs (UnZip, unshrink decompression); Jean-loup Gailly (deflate compression); Mark Adler (inflate decompression, fUnZip).

The author of the original unzip code upon which Info-ZIP’s was based is Samuel H. Smith; Carl Mascott did the first Unix port; and David P. Kirschbaum organized and led Info-ZIP in its early days with Keith Petersen hosting the original mailing list at WSMR-SimTel20. The full list of contributors to UnZip has grown quite large; please refer to the CONTRIBS file in the UnZip source distribution for a relatively complete version.

VERSIONS

v1.2	15 Mar 89	Samuel H. Smith
v2.0	9 Sep 89	Samuel H. Smith
v2.x	fall 1989	many Usenet contributors
v3.0	1 May 90	Info-ZIP (DPK, consolidator)
v3.1	15 Aug 90	Info-ZIP (DPK, consolidator)
v4.0	1 Dec 90	Info-ZIP (GRR, maintainer)
v4.1	12 May 91	Info-ZIP
v4.2	20 Mar 92	Info-ZIP (Zip-Bugs subgroup, GRR)
v5.0	21 Aug 92	Info-ZIP (Zip-Bugs subgroup, GRR)
v5.01	15 Jan 93	Info-ZIP (Zip-Bugs subgroup, GRR)

v5.1	7 Feb 94	Info-ZIP (Zip-Bugs subgroup, GRR)
v5.11	2 Aug 94	Info-ZIP (Zip-Bugs subgroup, GRR)
v5.12	28 Aug 94	Info-ZIP (Zip-Bugs subgroup, GRR)
v5.2	30 Apr 96	Info-ZIP (Zip-Bugs subgroup, GRR)
v5.3	22 Apr 97	Info-ZIP (Zip-Bugs subgroup, GRR)
v5.31	31 May 97	Info-ZIP (Zip-Bugs subgroup, GRR)
v5.32	3 Nov 97	Info-ZIP (Zip-Bugs subgroup, GRR)
v5.4	28 Nov 98	Info-ZIP (Zip-Bugs subgroup, SPC)
v5.41	16 Apr 00	Info-ZIP (Zip-Bugs subgroup, SPC)
v5.42	14 Jan 01	Info-ZIP (Zip-Bugs subgroup, SPC)
v5.5	17 Feb 02	Info-ZIP (Zip-Bugs subgroup, SPC)
v5.51	22 May 04	Info-ZIP (Zip-Bugs subgroup, SPC)

NAME

unzipsfx − self-extracting stub for prepending to ZIP archives

SYNOPSIS

<name of unzipsfx+archive combo> [**−cfptuz**[**ajnoqsCLV$**]] [*file(s)* . . . [**−x** *xfile(s)* . . .]]

DESCRIPTION

unzipsfx is a modified version of *unzip*(1L) designed to be prepended to existing ZIP archives in order to form self-extracting archives. Instead of taking its first non-flag argument to be the zipfile(s) to be extracted, *unzipsfx* seeks itself under the name by which it was invoked and tests or extracts the contents of the appended archive. Because the executable stub adds bulk to the archive (the whole purpose of which is to be as small as possible), a number of the less-vital capabilities in regular *unzip* have been removed. Among these are the usage (or help) screen, the listing and diagnostic functions (**−l** and **−v**), the ability to decompress older compression formats (the ''reduce,'' ''shrink'' and ''implode'' methods). The ability to extract to a directory other than the current one can be selected as a compile-time option, which is now enabled by default since UnZipSFX version 5.5. Similary, decryption is supported as a compile-time option but should be avoided unless the attached archive contains encrypted files. Starting with release 5.5, another compile-time option adds a simple ''run command after extraction'' feature. This feature is currently incompatible with the ''extract to different directory'' feature and remains disabled by default.

Note that self-extracting archives made with *unzipsfx* **are no more (or less) portable across different operating systems than is the** *unzip* **executable itself.** In general a self-extracting archive made on a particular Unix system, for example, will only self-extract under the same flavor of Unix. Regular *unzip* may still be used to extract the embedded archive as with any normal zipfile, although it will generate a harmless warning about extra bytes at the beginning of the zipfile. *Despite this*, however, the self-extracting archive is technically *not* a valid ZIP archive, and PKUNZIP may be unable to test or extract it. This limitation is due to the simplistic manner in which the archive is created; the internal directory structure is not updated to reflect the extra bytes prepended to the original zipfile.

ARGUMENTS

[*file(s)*] An optional list of archive members to be processed. Regular expressions (wildcards) similar to those in Unix *egrep*(1) may be used to match multiple members. These wildcards may contain:

*	matches a sequence of 0 or more characters
?	matches exactly 1 character
[. . .]	matches any single character found inside the brackets; ranges are specified by a beginning character, a hyphen, and an ending character. If an exclamation point or a caret ('!' or '^') follows the left bracket, then the range of characters within the brackets is complemented (that is, anything *except* the characters inside the brackets is considered a match).

(Be sure to quote any character that might otherwise be interpreted or modified by the operating system, particularly under Unix and VMS.)

[**−x** *xfile(s)*]

An optional list of archive members to be excluded from processing. Since wildcard characters match directory separators ('/'), this option may be used to exclude any files that are in subdirectories. For example, ''`foosfx *.[ch] -x */*`'' would extract all C source files in the main directory, but none in any subdirectories. Without the **−x** option, all C source files in all directories within the zipfile would be extracted.

If *unzipsfx* is compiled with SFX_EXDIR defined, the following option is also enabled:

[**−d** *exdir*]

An optional directory to which to extract files. By default, all files and subdirectories are recreated in the current directory; the **−d** option allows extraction in an arbitrary directory (always assuming one has permission to write to the directory). The option and directory may be concatenated without any white space between them, but note that this may cause normal shell behavior to be suppressed. In particular, ''`−d ~`'' (tilde) is expanded by Unix C shells into the name of the user's home directory, but ''`−d~`'' is treated as a literal subdirectory ''~''

of the current directory.

OPTIONS

unzipsfx supports the following *unzip*(1L) options: **−c** and **−p** (extract to standard output/screen), **−f** and **−u** (freshen and update existing files upon extraction), **−t** (test archive) and **−z** (print archive comment). All normal listing options (**−l**, **−v** and **−Z**) have been removed, but the testing option (**−t**) may be used as a ''poor man's'' listing. Alternatively, those creating self-extracting archives may wish to include a short listing in the zipfile comment.

See *unzip*(1L) for a more complete description of these options.

MODIFIERS

unzipsfx currently supports all *unzip*(1L) modifiers: **−a** (convert text files), **−n** (never overwrite), **−o** (overwrite without prompting), **−q** (operate quietly), **−C** (match names case-insensitively), **−L** (convert uppercase-OS names to lowercase), **−j** (junk paths) and **−V** (retain version numbers); plus the following operating-system specific options: **−X** (restore VMS owner/protection info), **−s** (convert spaces in filenames to underscores [DOS, OS/2, NT]) and **−$** (restore volume label [DOS, OS/2, NT, Amiga]).

(Support for regular ASCII text-conversion may be removed in future versions, since it is simple enough for the archive's creator to ensure that text files have the appropriate format for the local OS. EBCDIC conversion will of course continue to be supported since the zipfile format implies ASCII storage of text files.)

See *unzip*(1L) for a more complete description of these modifiers.

ENVIRONMENT OPTIONS

unzipsfx uses the same environment variables as *unzip*(1L) does, although this is likely to be an issue only for the person creating and testing the self-extracting archive. See *unzip*(1L) for details.

DECRYPTION

Decryption is supported exactly as in *unzip*(1L); that is, interactively with a non-echoing prompt for the password(s). See *unzip*(1L) for details. Once again, note that if the archive has no encrypted files there is no reason to use a version of *unzipsfx* with decryption support; that only adds to the size of the archive.

AUTORUN COMMAND

When *unzipsfx* was compiled with CHEAP_SFX_AUTORUN defined, a simple ''command autorun'' feature is supported. You may enter a command into the Zip archive comment, using the following format:

```
$AUTORUN$>[command line string]
```

When *unzipsfxP recognizes the "$AUTORUN$>" token at the beginning of the Zip archive comment, the remainder of the first line of the comment (until the first newline character) is passed as a shell command to the operating system using the C rtl "system" function. Before executing the command, unzipsfxP displays the command on the console and prompts the user for confirmation. When the user has switched off prompting by specifying the* **-q** *option, autorun commands are never executed.*

In case the archive comment contains additonal lines of text, the remainder of the archive comment following the first line is displayed normally, unless quiet operation was requested by supplying a **-q** option.

EXAMPLES

To create a self-extracting archive *letters* from a regular zipfile *letters.zip* and change the new archive's permissions to be world-executable under Unix:

```
cat unzipsfx letters.zip > letters
chmod 755 letters
zip -A letters
```

To create the same archive under MS-DOS, OS/2 or NT (note the use of the **/b** [binary] option to the *copy* command):

```
copy /b unzipsfx.exe+letters.zip letters.exe
zip -A letters.exe
```

Under VMS:

```
copy unzipsfx.exe,letters.zip letters.exe
```

```
letters == "$currentdisk:[currentdir]letters.exe"
zip -A letters.exe
```

(The VMS *append* command may also be used. The second command installs the new program as a ‘‘foreign command’’ capable of taking arguments. The third line assumes that Zip is already installed as a foreign command.) Under AmigaDOS:

```
MakeSFX letters letters.zip UnZipSFX
```

(MakeSFX is included with the UnZip source distribution and with Amiga binary distributions. ‘‘`zip -A`’’ doesn’t work on Amiga self-extracting archives.) To test (or list) the newly created self-extracting archive:

```
letters −t
```

To test *letters* quietly, printing only a summary message indicating whether the archive is OK or not:

```
letters −tqq
```

To extract the complete contents into the current directory, recreating all files and subdirectories as necessary:

```
letters
```

To extract all `*.txt` files (in Unix quote the ‘*’):

```
letters *.txt
```

To extract everything *except* the `*.txt` files:

```
letters -x *.txt
```

To extract only the README file to standard output (the screen):

```
letters -c README
```

To print only the zipfile comment:

```
letters −z
```

LIMITATIONS

The principle and fundamental limitation of *unzipsfx* is that it is not portable across architectures or operating systems, and therefore neither are the resulting archives. For some architectures there is limited portability, however (e.g., between some flavors of Intel-based Unix).

Another problem with the current implementation is that any archive with ‘‘junk’’ prepended to the beginning technically is no longer a zipfile (unless *zip*(1) is used to adjust the zipfile offsets appropriately, as noted above). *unzip*(1) takes note of the prepended bytes and ignores them since some file-transfer protocols, notably MacBinary, are also known to prepend junk. But PKWARE’s archiver suite may not be able to deal with the modified archive unless its offsets have been adjusted.

unzipsfx has no knowledge of the user’s PATH, so in general an archive must either be in the current directory when it is invoked, or else a full or relative path must be given. If a user attempts to extract the archive from a directory in the PATH other than the current one, *unzipsfx* will print a warning to the effect, ‘‘can’t find myself.’’ This is always true under Unix and may be true in some cases under MS-DOS, depending on the compiler used (Microsoft C fully qualifies the program name, but other compilers may not). Under OS/2 and NT there are operating-system calls available that provide the full path name, so the archive may be invoked from anywhere in the user’s path. The situation is not known for AmigaDOS, Atari TOS, MacOS, etc.

As noted above, a number of the normal *unzip*(1L) functions have been removed in order to make *unzipsfx* smaller: usage and diagnostic info, listing functions and extraction to other directories. Also, only stored and deflated files are supported. The latter limitation is mainly relevant to those who create SFX archives, however.

VMS users must know how to set up self-extracting archives as foreign commands in order to use any of *unzipsfx*’s options. This is not necessary for simple extraction, but the command to do so then becomes, e.g., ‘‘`run letters`’’ (to continue the examples given above).

unzipsfx on the Amiga requires the use of a special program, MakeSFX, in order to create working self-extracting archives; simple concatenation does not work. (For technically oriented users, the attached archive is defined as a ‘‘debug hunk.’’) There may be compatibility problems between the ROM levels of older Amigas and newer ones.

All current bugs in *unzip*(1L) exist in *unzipsfx* as well.

DIAGNOSTICS

unzipsfx's exit status (error level) is identical to that of *unzip*(1L); see the corresponding man page.

SEE ALSO

funzip(1L), *unzip*(1L), *zip*(1L), *zipcloak*(1L), *zipgrep*(1L), *zipinfo*(1L), *zipnote*(1L), *zipsplit*(1L)

URL

The Info-ZIP home page is currently at

`http://www.info-zip.org/pub/infozip/`

or

`ftp://ftp.info-zip.org/pub/infozip/` .

AUTHORS

Greg Roelofs was responsible for the basic modifications to UnZip necessary to create UnZipSFX. See *unzip*(1L) for the current list of Zip-Bugs authors, or the file CONTRIBS in the UnZip source distribution for the full list of Info-ZIP contributors.

NAME

zipgrep − search files in a ZIP archive for lines matching a pattern

SYNOPSIS

zipgrep [**egrep_options**] *pattern file*[*.zip*] [*file(s)* ...] [**−x** *xfile(s)* ...]

DESCRIPTION

zipgrep will search files within a ZIP archive for lines matching the given string or pattern. *zipgrep* is a shell script and requires *egrep*(1) and *unzip*(1L) to function. Its output is identical to that of *egrep*(1).

ARGUMENTS

pattern The pattern to be located within a ZIP archive. Any string or regular expression accepted by *egrep*(1) may be used. *file*[*.zip*] Path of the ZIP archive. (Wildcard expressions for the ZIP archive name are not supported.) If the literal filename is not found, the suffix `.zip` is appended. Note that self-extracting ZIP files are supported, as with any other ZIP archive; just specify the `.exe` suffix (if any) explicitly.

[*file(s)*] An optional list of archive members to be processed, separated by spaces. If no member files are specified, all members of the ZIP archive are searched. Regular expressions (wildcards) may be used to match multiple members:

*	matches a sequence of 0 or more characters
?	matches exactly 1 character
[...]	matches any single character found inside the brackets; ranges are specified by a beginning character, a hyphen, and an ending character. If an exclamation point or a caret (‘!’ or ‘^’) follows the left bracket, then the range of characters within the brackets is complemented (that is, anything *except* the characters inside the brackets is considered a match).

(Be sure to quote any character that might otherwise be interpreted or modified by the operating system.)

[**−x** *xfile(s)*]
An optional list of archive members to be excluded from processing. Since wildcard characters match directory separators (‘/’), this option may be used to exclude any files that are in subdirectories. For example, “`zipgrep grumpy foo *.[ch] -x */*`” would search for the string “grumpy” in all C source files in the main directory of the “foo” archive, but none in any subdirectories. Without the **−x** option, all C source files in all directories within the zipfile would be searched.

OPTIONS

All options prior to the ZIP archive filename are passed to *egrep*(1).

SEE ALSO

egrep(1), *unzip*(1L), *zip*(1L), *funzip*(1L), *zipcloak*(1L), *zipinfo*(1L), *zipnote*(1L), *zipsplit*(1L)

URL

The Info-ZIP home page is currently at
`http://www.info-zip.org/pub/infozip/`
or
`ftp://ftp.info-zip.org/pub/infozip/` .

AUTHORS

zipgrep was written by Jean-loup Gailly.

NAME

zipinfo − list detailed information about a ZIP archive

SYNOPSIS

zipinfo [**−12smlvhMtTz**] *file*[*.zip*] [*file(s)* . . .] [**−x** *xfile(s)* . . .]

unzip −Z [**−12smlvhMtTz**] *file*[*.zip*] [*file(s)* . . .] [**−x** *xfile(s)* . . .]

DESCRIPTION

zipinfo lists technical information about files in a ZIP archive, most commonly found on MS-DOS systems. Such information includes file access permissions, encryption status, type of compression, version and operating system or file system of compressing program, and the like. The default behavior (with no options) is to list single-line entries for each file in the archive, with header and trailer lines providing summary information for the entire archive. The format is a cross between Unix "`ls −l`" and "`unzip −v`" output. See **DETAILED DESCRIPTION** below. Note that *zipinfo* is the same program as *unzip* (under Unix, a link to it); on some systems, however, *zipinfo* support may have been omitted when *unzip* was compiled.

ARGUMENTS

file[*.zip*]

Path of the ZIP archive(s). If the file specification is a wildcard, each matching file is processed in an order determined by the operating system (or file system). Only the filename can be a wildcard; the path itself cannot. Wildcard expressions are similar to Unix *egrep*(1) (regular) expressions and may contain:

*	matches a sequence of 0 or more characters
?	matches exactly 1 character
[. . .]	matches any single character found inside the brackets; ranges are specified by a beginning character, a hyphen, and an ending character. If an exclamation point or a caret ('!' or '^') follows the left bracket, then the range of characters within the brackets is complemented (that is, anything *except* the characters inside the brackets is considered a match).

(Be sure to quote any character that might otherwise be interpreted or modified by the operating system, particularly under Unix and VMS.) If no matches are found, the specification is assumed to be a literal filename; and if that also fails, the suffix `.zip` is appended. Note that self-extracting ZIP files are supported; just specify the `.exe` suffix (if any) explicitly.

[*file(s)*] An optional list of archive members to be processed. Regular expressions (wildcards) may be used to match multiple members; see above. Again, be sure to quote expressions that would otherwise be expanded or modified by the operating system.

[**−x** *xfile(s)*]

An optional list of archive members to be excluded from processing.

OPTIONS

−1 list filenames only, one per line. This option excludes all others; headers, trailers and zipfile comments are never printed. It is intended for use in Unix shell scripts.

−2 list filenames only, one per line, but allow headers (**−h**), trailers (**−t**) and zipfile comments (**−z**), as well. This option may be useful in cases where the stored filenames are particularly long.

−s list zipfile info in short Unix "`ls −l`" format. This is the default behavior; see below.

−m list zipfile info in medium Unix "`ls −l`" format. Identical to the **−s** output, except that the compression factor, expressed as a percentage, is also listed.

−l list zipfile info in long Unix "`ls −l`" format. As with **−m** except that the compressed size (in bytes) is printed instead of the compression ratio.

−v list zipfile information in verbose, multi-page format.

−h list header line. The archive name, actual size (in bytes) and total number of files is printed.

−M pipe all output through an internal pager similar to the Unix *more*(1) command. At the end of a screenful of output, *zipinfo* pauses with a ‘‘−−More−−’’ prompt; the next screenful may be viewed by pressing the Enter (Return) key or the space bar. *zipinfo* can be terminated by pressing the ‘‘q’’ key and, on some systems, the Enter/Return key. Unlike Unix *more*(1), there is no forward-searching or editing capability. Also, *zipinfo* doesn’t notice if long lines wrap at the edge of the screen, effectively resulting in the printing of two or more lines and the likelihood that some text will scroll off the top of the screen before being viewed. On some systems the number of available lines on the screen is not detected, in which case *zipinfo* assumes the height is 24 lines.

−t list totals for files listed or for all files. The number of files listed, their uncompressed and compressed total sizes, and their overall compression factor is printed; or, if only the totals line is being printed, the values for the entire archive are given. Note that the total compressed (data) size will never match the actual zipfile size, since the latter includes all of the internal zipfile headers in addition to the compressed data.

−T print the file dates and times in a sortable decimal format (yymmdd.hhmmss). The default date format is a more standard, human-readable version with abbreviated month names (see examples below).

−z include the archive comment (if any) in the listing.

DETAILED DESCRIPTION

zipinfo has a number of modes, and its behavior can be rather difficult to fathom if one isn’t familiar with Unix *ls*(1) (or even if one is). The default behavior is to list files in the following format:

```
-rw-rws---  1.9 unx    2802 t- defX 11-Aug-91 13:48 perms.2660
```

The last three fields are the modification date and time of the file, and its name. The case of the filename is respected; thus files that come from MS-DOS PKZIP are always capitalized. If the file was zipped with a stored directory name, that is also displayed as part of the filename.

The second and third fields indicate that the file was zipped under Unix with version 1.9 of *zip*. Since it comes from Unix, the file permissions at the beginning of the line are printed in Unix format. The uncompressed file-size (2802 in this example) is the fourth field.

The fifth field consists of two characters, either of which may take on several values. The first character may be either ‘t’ or ‘b’, indicating that *zip* believes the file to be text or binary, respectively; but if the file is encrypted, *zipinfo* notes this fact by capitalizing the character (‘T’ or ‘B’). The second character may also take on four values, depending on whether there is an extended local header and/or an ‘‘extra field’’ associated with the file (fully explained in PKWare’s APPNOTE.TXT, but basically analogous to pragmas in ANSI C--i.e., they provide a standard way to include non-standard information in the archive). If neither exists, the character will be a hyphen (‘−’); if there is an extended local header but no extra field, ‘l’; if the reverse, ‘x’; and if both exist, ‘X’. Thus the file in this example is (probably) a text file, is not encrypted, and has neither an extra field nor an extended local header associated with it. The example below, on the other hand, is an encrypted binary file with an extra field:

```
RWD,R,R     0.9 vms     168 Bx shrk  9-Aug-91 19:15 perms.0644
```

Extra fields are used for various purposes (see discussion of the **−v** option below) including the storage of VMS file attributes, which is presumably the case here. Note that the file attributes are listed in VMS format. Some other possibilities for the host operating system (which is actually a misnomer--host file system is more correct) include OS/2 or NT with High Performance File System (HPFS), MS-DOS, OS/2 or NT with File Allocation Table (FAT) file system, and Macintosh. These are denoted as follows:

```
-rw-a--     1.0 hpf    5358 Tl i4:3  4-Dec-91 11:33 longfilename.hpfs
-r--ahs     1.1 fat    4096 b- i4:2 14-Jul-91 12:58 EA DATA. SF
--w-------  1.0 mac   17357 bx i8:2  4-May-92 04:02 unzip.macr
```

File attributes in the first two cases are indicated in a Unix-like format, where the seven subfields indicate whether the file: (1) is a directory, (2) is readable (always true), (3) is writable, (4) is executable (guessed on the basis of the extension--*.exe*, *.com*, *.bat*, *.cmd* and *.btm* files are assumed to be so), (5) has its archive bit set, (6) is hidden, and (7) is a system file. Interpretation of Macintosh file attributes is unreliable because some Macintosh archivers don’t store any attributes in the archive.

Finally, the sixth field indicates the compression method and possible sub-method used. There are six methods known at present: storing (no compression), reducing, shrinking, imploding, tokenizing (never publicly released), and deflating. In addition, there are four levels of reducing (1 through 4); four types of imploding (4K or 8K sliding dictionary, and 2 or 3 Shannon-Fano trees); and four levels of deflating (superfast, fast, normal, maximum compression). *zipinfo* represents these methods and their sub-methods as follows: *stor*; *re:1, re:2,* etc.; *shrk*; *i4:2, i8:3,* etc.; *tokn*; and *defS, defF, defN,* and *defX*.

The medium and long listings are almost identical to the short format except that they add information on the file's compression. The medium format lists the file's compression factor as a percentage indicating the amount of space that has been ``removed'':

```
-rw-rws---  1.5 unx    2802 t- 81% defX 11-Aug-91 13:48 perms.2660
```

In this example, the file has been compressed by more than a factor of five; the compressed data are only 19% of the original size. The long format gives the compressed file's size in bytes, instead:

```
-rw-rws---  1.5 unx    2802 t-     538 defX 11-Aug-91 13:48 perms.2660
```

Adding the **−T** option changes the file date and time to decimal format:

```
-rw-rws---  1.5 unx    2802 t-     538 defX 910811.134804 perms.2660
```

Note that because of limitations in the MS-DOS format used to store file times, the seconds field is always rounded to the nearest even second. For Unix files this is expected to change in the next major releases of *zip*(1L) and *unzip*.

In addition to individual file information, a default zipfile listing also includes header and trailer lines:

```
Archive:  OS2.zip   5453 bytes   5 files
,,rw,       1.0 hpf     730 b- i4:3 26-Jun-92 23:40 Contents
,,rw,       1.0 hpf    3710 b- i4:3 26-Jun-92 23:33 makefile.os2
,,rw,       1.0 hpf    8753 b- i8:3 26-Jun-92 15:29 os2unzip.c
,,rw,       1.0 hpf      98 b- stor 21-Aug-91 15:34 unzip.def
,,rw,       1.0 hpf      95 b- stor 21-Aug-91 17:51 zipinfo.def
5 files, 13386 bytes uncompressed, 4951 bytes compressed:  63.0%
```

The header line gives the name of the archive, its total size, and the total number of files; the trailer gives the number of files listed, their total uncompressed size, and their total compressed size (not including any of *zip*'s internal overhead). If, however, one or more *file(s)* are provided, the header and trailer lines are not listed. This behavior is also similar to that of Unix's ``ls −l''; it may be overridden by specifying the **−h** and **−t** options explicitly. In such a case the listing format must also be specified explicitly, since **−h** or **−t** (or both) in the absence of other options implies that ONLY the header or trailer line (or both) is listed. See the **EXAMPLES** section below for a semi-intelligible translation of this nonsense.

The verbose listing is mostly self-explanatory. It also lists file comments and the zipfile comment, if any, and the type and number of bytes in any stored extra fields. Currently known types of extra fields include PKWARE's authentication (``AV'') info; OS/2 extended attributes; VMS filesystem info, both PKWARE and Info-ZIP versions; Macintosh resource forks; Acorn/Archimedes SparkFS info; and so on. (Note that in the case of OS/2 extended attributes--perhaps the most common use of zipfile extra fields--the size of the stored EAs as reported by *zipinfo* may not match the number given by OS/2's *dir* command: OS/2 always reports the number of bytes required in 16-bit format, whereas *zipinfo* always reports the 32-bit storage.)

ENVIRONMENT OPTIONS

Modifying *zipinfo*'s default behavior via options placed in an environment variable can be a bit complicated to explain, due to *zipinfo*'s attempts to handle various defaults in an intuitive, yet Unix-like, manner. (Try not to laugh.) Nevertheless, there is some underlying logic. In brief, there are three ``priority levels'' of options: the default options; environment options, which can override or add to the defaults; and explicit options given by the user, which can override or add to either of the above.

The default listing format, as noted above, corresponds roughly to the "`zipinfo −hst`" command (except when individual zipfile members are specified). A user who prefers the long-listing format (**−l**) can make use of the *zipinfo*'s environment variable to change this default:

Unix Bourne shell:

```
ZIPINFO=−l; export ZIPINFO
```

Unix C shell:

```
setenv ZIPINFO −l
```

OS/2 or MS-DOS:

```
set ZIPINFO=−l
```

VMS (quotes for *lowercase*):

```
define ZIPINFO_OPTS "−l"
```

If, in addition, the user dislikes the trailer line, *zipinfo*'s concept of ''negative options'' may be used to override the default inclusion of the line. This is accomplished by preceding the undesired option with one or more minuses: e.g., ''`−l−t`'' or ''`−−tl`'', in this example. The first hyphen is the regular switch character, but the one before the 't' is a minus sign. The dual use of hyphens may seem a little awkward, but it's reasonably intuitive nonetheless: simply ignore the first hyphen and go from there. It is also consistent with the behavior of the Unix command *nice*(1).

As suggested above, the default variable names are ZIPINFO_OPTS for VMS (where the symbol used to install *zipinfo* as a foreign command would otherwise be confused with the environment variable), and ZIPINFO for all other operating systems. For compatibility with *zip*(1L), ZIPINFOOPT is also accepted (don't ask). If both ZIPINFO and ZIPINFOOPT are defined, however, ZIPINFO takes precedence. *unzip*'s diagnostic option (**−v** with no zipfile name) can be used to check the values of all four possible *unzip* and *zipinfo* environment variables.

EXAMPLES

To get a basic, short-format listing of the complete contents of a ZIP archive *storage.zip*, with both header and totals lines, use only the archive name as an argument to zipinfo:

```
zipinfo storage
```

To produce a basic, long-format listing (not verbose), including header and totals lines, use **−l**:

```
zipinfo −l storage
```

To list the complete contents of the archive without header and totals lines, either negate the **−h** and **−t** options or else specify the contents explicitly:

```
zipinfo −−h−t storage
zipinfo storage \*
```

(where the backslash is required only if the shell would otherwise expand the '*' wildcard, as in Unix when globbing is turned on--double quotes around the asterisk would have worked as well). To turn off the totals line by default, use the environment variable (C shell is assumed here):

```
setenv ZIPINFO −−t
zipinfo storage
```

To get the full, short-format listing of the first example again, given that the environment variable is set as in the previous example, it is necessary to specify the **−s** option explicitly, since the **−t** option by itself implies that ONLY the footer line is to be printed:

```
setenv ZIPINFO −−t
zipinfo −t storage              [only totals line]
zipinfo −st storage             [full listing]
```

The **−s** option, like **−m** and **−l**, includes headers and footers by default, unless otherwise specified. Since the environment variable specified no footers and that has a higher precedence than the default behavior of **−s**, an explicit **−t** option was necessary to produce the full listing. Nothing was indicated about the header, however, so the **−s** option was sufficient. Note that both the **−h** and **−t** options, when used by themselves or with each other, override any default listing of member files; only the header and/or footer are printed. This behavior is useful when *zipinfo* is used with a wildcard zipfile specification; the contents of all zipfiles are then summarized with a single command.

To list information on a single file within the archive, in medium format, specify the filename explicitly:

```
zipinfo −m storage unshrink.c
```

The specification of any member file, as in this example, will override the default header and totals lines; only the single line of information about the requested file will be printed. This is intuitively what one would expect when requesting information about a single file. For multiple files, it is often useful to know the total compressed and uncompressed size; in such cases **−t** may be specified explicitly:

```
zipinfo −mt storage "*.[ch]" Mak\*
```

To get maximal information about the ZIP archive, use the verbose option. It is usually wise to pipe the output into a filter such as Unix *more*(1) if the operating system allows it:

```
zipinfo −v storage | more
```

Finally, to see the most recently modified files in the archive, use the **−T** option in conjunction with an external sorting utility such as Unix *sort*(1) (and *sed*(1) as well, in this example):

```
zipinfo −T storage | sort -nr -k 7 | sed 15q
```

The **−nr** option to *sort*(1) tells it to sort numerically in reverse order rather than in textual order, and the **−k 7** option tells it to sort on the seventh field. This assumes the default short-listing format; if **−m** or **−l** is used, the proper *sort*(1) option would be **−k 8**. Older versions of *sort*(1) do not support the **−k** option, but you can use the traditional **+** option instead, e.g., **+6** instead of **−k 7**. The *sed*(1) command filters out all but the first 15 lines of the listing. Future releases of *zipinfo* may incorporate date/time and filename sorting as built-in options.

TIPS

The author finds it convenient to define an alias *ii* for *zipinfo* on systems that allow aliases (or, on other systems, copy/rename the executable, create a link or create a command file with the name *ii*). The *ii* usage parallels the common *ll* alias for long listings in Unix, and the similarity between the outputs of the two commands was intentional.

BUGS

As with *unzip*, *zipinfo*'s **−M** (“more”) option is overly simplistic in its handling of screen output; as noted above, it fails to detect the wrapping of long lines and may thereby cause lines at the top of the screen to be scrolled off before being read. *zipinfo* should detect and treat each occurrence of line-wrap as one additional line printed. This requires knowledge of the screen's width as well as its height. In addition, *zipinfo* should detect the true screen geometry on all systems.

zipinfo's listing-format behavior is unnecessarily complex and should be simplified. (This is not to say that it will be.)

SEE ALSO

ls(1), *funzip*(1L), *unzip*(1L), *unzipsfx*(1L), *zip*(1L), *zipcloak*(1L), *zipnote*(1L), *zipsplit*(1L)

URL

The Info-ZIP home page is currently at

```
http://www.info-zip.org/pub/infozip/
```

or

```
ftp://ftp.info-zip.org/pub/infozip/
```

.

AUTHOR

Greg “Cave Newt” Roelofs. ZipInfo contains pattern-matching code by Mark Adler and fixes/improvements by many others. Please refer to the CONTRIBS file in the UnZip source distribution for a more complete list.