Adding Files to an Archive in UNIX

This chapter contains detailed information on the features and options available when you add files to an archive.

Default Values for Commands and Options

For each operation in this chapter, the command or option that represents that operation has a default value. The default value determines the way that the command or option is done when the command or option is used on the command line by itself, with no sub-option explicitly specified.

For example, the initial default value for the ***add*** command is ***all***, which causes the command to add all files. See this page for information on how to change default settings.

Creating and Updating Archives

add

The ***add*** command adds files to an archive.

To add files to a new or existing archive, specify the name of the archive on the command line, then list one or more files to add. If the archive does not already exist, PKZIP creates it.

The command line below adds all .txt files in the current directory to `myarchive.zip`.

> ***pkzipc -add myarchive.zip *.txt***

Adding All Files in a Directory

You can choose to compress all files in a particular directory with a single command. To do this, you do not have to specify each file. Simply type ***pkzipc -add,*** and the name of your ZIP file, as shown below:

> ***pkzipc -add test.zip***

In the example above, all files in the current directory are compressed into the test.zip file. (To learn how to compress files that appear in subdirectories, see "Compressing Files in Subdirectories" later in this chapter.)

You can also specify files from a different directory if you wish. For example, if you were in a parent directory to a directory called `temp` and you wanted to compress all the files in the `temp` directory, you could type the following:

> ***pkzipc -add test.zip temp/****

The resulting `test.zip` file is stored in the current directory (the parent directory to the `temp` directory in our example).

Note: The ***add*** command uses the ***all*** sub-option to add all files in a specified directory to your archive file by default. You can change the default with the ***configuration*** command to modify the default setting for ***add***.

For information on how to change default values for commands and options, see this page.

Adding New and Modified Files

add=update

The ***update*** sub-option can save time when you repeatedly archive the same files. PKZIP allows you to specify that only new or modified files are added to an archive. When the ***update*** sub-option is used, dates on the files specified for archiving are compared against dates of files having the same name already present in the archive. A file is added only if no file with the same name is already in the archive or if the file to be added is newer.

The sub-option differs from the ***freshen*** sub-option in that it adds files which are not in the archive already.

To compress only updated files or files not already archived in a specific .ZIP file, use the ***update*** sub-option with the ***add*** option, as shown below:

> ***pkzipc -add=update test.zip *.doc***

In this example, a .ZIP file called `test.zip` is created in the current directory. All files in the current directory matching the file specification (*.doc) will be added or updated into the `test.zip` archive.

Adding Only Files That Have Changed

add=freshen

The ***freshen*** value allows you to compress only changed files that exist in the .ZIP file. No new files will be added to the .ZIP file. To update only files that have changed, use the ***freshen*** value with the ***add*** command, as shown below:

pkzipc -add=freshen test.zip

The following command line abbreviates the value but has the same effect:

pkzipc -add=fre test.zip

If you only want to re-compress specific files, simply include those files in your command. For example, if you wanted to re-compress a file called `resume.doc`, you would type something like this:

pkzipc -add=freshen test.zip resume.doc

In the above example, only `resume.doc` will be re-compressed into the `test.zip` file. This assumes that the version of `resume.doc` being added is newer than the version of `resume.doc` that already exists in the .ZIP file.

Encrypting Files That You Add to an Archive

You can encrypt files when you add them to an archive. When you encrypt files, only people that you designate or who know a passphrase that you assign can decrypt and extract the files.

Depending on your platform and whether you have PKZIP or SecureZIP, you can encrypt using either traditional ZIP encryption or strong encryption. Strong encryption is far more secure than the older, traditional ZIP encryption, but people who want to decrypt your files are likely to need access to a recent version of a ZIP utility.

The ***passphrase*** and ***recipient*** options control encryption when you add files to an archive.

- With the ***passphrase*** option, you specify a passphrase to use to decrypt the files. The ***passphrase*** option is available in both PKZIP and SecureZIP. It is used to do both strong and traditional ZIP passphrase-based encryption.
- A passphrase is just a password. It is called a passphrase in the program to emphasize that PKZIP and SecureZIP support passwords that can contain spaces and other non-alphanumeric symbols.
- With the ***recipient*** option, you specify a recipient list. A recipient list is a list of digital certificates that belong to people whom you want to allow to decrypt. PKZIP automatically decrypts the files for the owners of the certificates when the owners extract the files. You will learn more about digital certificates on this page.

Note: The ***recipient*** option is used only to do strong encryption and is available only in SecureZIP. Both PKZIP and SecureZIP can decrypt files encrypted with either kind of strong encryption (passphrase or recipient list).

When you use strong encryption, you also have the option to encrypt not only the contents but the names of files and folders that you add to an archive. When you encrypt file names, you essentially encrypt the archive itself: the archive cannot even be opened except by someone who can decrypt its contents.

Encrypting Files with a Passphrase

passphrase

Use the ***passphrase*** option with the ***add*** command to encrypt files so that users can use a passphrase to decrypt them. You can do either strong or traditional ZIP encryption with the ***passphrase*** option.

To include a passphrase on the command line, use the ***passphrase*** option and enter a passphrase of at least eight characters (preceded by an equal sign). For example (where the passphrase is *mypassphrase*):

pkzipc -add -passphrase=mypassphrase test.zip

Note: Passphrases are case sensitive.

For more security, you can enter your passphrase separately from the command line, at a prompt. This method prevents other users from learning your passphrase by reviewing previously entered PKZIP command lines.

To have PKZIP prompt for a passphrase, include the ***passphrase*** option in the command line but do not specify a passphrase. For example:

pkzipc -add -passphrase test.zip

When you press ENTER, a prompt like the following appears:

```
Passphrase?
```

Type your passphrase. The characters appear on your screen as asterisks. Press ENTER. PKZIP asks you to confirm the passphrase:

```
Re-enter passphrase for verification.
Passphrase?
```

Re-enter the passphrase and press ENTER. If your entry matches the original one, PKZIP proceeds and compresses the files. If the passphrases do not match, PKZIP prompts you again:

```
Passphrases don't match! Please try again.
Passphrase?
```

Another way to enter a passphrase is to point PKZIP to a text file that contains one. For example:

pkzipc -add -passphrase=@secret.txt test.zip

The file (`secret.txt` in the example) should contain just the passphrase, on a line by itself.

For best security, choose a passphrase that is not easy for someone to guess. Ideally, a passphrase should be at least eight characters long, should contain a mix of numbers and upper- and lower-case letters, and should not be a word in the dictionary.

Specify an Encryption Method

listcryptalgorithms, cryptalgorithm

When you use strong encryption (available only with SecureZIP for UNIX), you have a choice of encryption algorithms to use. To list the available algorithms, use the ***listcryptalgorithms*** command.

pkzipc -listcryptalgorithms

The following output from ***listcryptalgorithms*** lists all supported algorithms:

```
AES,256 AES (256-bit)
AES,192 AES (192-bit)
AES,128 AES (128-bit)
3DES,168 3DES (168-bit)
```

Use the ***cryptalgorithm*** option to specify an algorithm.

pkzipc -add -passphrase -cryptalgorithm=aes,128 test.zip

By default, ***cryptalgorithm*** specifies AES,256. If you do not use ***cryptalgorithm*** when encrypting with a passphrase, SecureZIP applies traditional PKWARE encryption.

Encrypting Files with a Recipient List

With SecureZIP, you can use public keys to encrypt files in an archive, or to an archive itself. This practice, called a *recipient list*, allows you to send files securely to only specific people.

You can include anyone with a public/private key pair based on either the X.509 (version 3) standard or the OpenPGP (RFC 4880) standard. PKZIP will authenticate signatures in OpenPGP files and validated by PGP keyrings on your system.

Note: The ***recipient*** option is available only with SecureZIP. You will learn more about digital certificates and recipient lists in Chapter 6.

recipient

Use the ***recipient*** option with the ***add*** command to strongly encrypt files and specify one or more digital certificates representing the people whom you want to allow to decrypt, also known as a recipient list.

To encrypt using a recipient list, you must have a digital certificate, containing a public key, for each intended recipient. Any recipient on the list—that is, any person whose system has access to the private key for that certificate—can decrypt and extract the files simply by using the ***extract*** command. No one else can decrypt (unless a passphrase was also specified).

If you use the ***recipient*** option together with the ***passphrase*** option, PKZIP decrypts automatically for listed recipients when they extract the files, and other people can decrypt if, and only if, they have the passphrase.

Note: Ordinarily, PKZIP decrypts automatically for anyone on a recipient list. However, if necessary, a recipient can tell PKZIP where to find a private key that is not in one of the usual places. See the ***keyfile*** and ***keypassphrase*** options.

Specifying Recipients

You can specify a list of recipients either by specifying each recipient individually on the command line, or by specifying a file that contains a recipient list.

Be sure to specify yourself as a recipient if you want to be able to use your own certificate to decrypt.

By default, SecureZIP searches for certificates for listed recipients only in the system's local certificate stores. Use the ***ldap*** option to cause SecureZIP to search a specified LDAP directory.

Use any of the following criteria to specify recipients:

Criterion	*To use*	*For example*

Common name	Specify, in quotes, the common name of the subject of the certificate (that is, the *cn* field in a string representation of a certificate); optionally, precede with: `cn=` By default, SecureZIP searches for recipients by common name unless another sub-option is used or the value appears to be an email address.	***-recipient=cn="John Public"*** ***-recipient="John Public"***
Email address	Specify the email address of the certificate (that is, the *e* field in a string representation of a certificate); optionally, precede with: `e=`	***-recipient=e=john.public@xyz.com*** ***-recipient=john.public@xyz.com***

LDAP filter	Specify the LDAP filter that you want to use to filter a search for certificates on an LDAP server that you are accessing with the ***ldap*** option; precede with:
	
	`f=`
	
	Use quotes if the filter string contains a space. Place the quotes around the entire filter string, including "f=".
	Include the following LDAP presence filter, as shown in the examples at right, to limit the search to LDAP entries that are certificates:
	
	`(&(userCertificate=*)(…))`
	
	Use standard LDAP filter syntax after the "f=" prefix.
	
	This sub-option is for use only when the ***ldap*** option is used.
		recipient=f=(&(userCertificate=)* (ou=Sales))
		recipient="f=(&(userCertificate=)* (ou=Regional Sales))"

For example, if the common name of the subject is *John Q. Public*, you can specify that certificate as a recipient as follows:

pkzipc -add -recipient="John Q. Public" test.zip

You can specify multiple recipients by using the ***recipient*** option multiple times:

pkzipc -add -recipient="John Q. Public" -recipient="Mary Samplename" test.zip

You can also reference a recipient by email address:

pkzipc -add -recipient=john.public@nowhere.com test.zip
pkzipc -add -recipient=e=john.public@nowhere.com test.zip

The prefix ***e=*** when using an email address is optional. SecureZIP automatically looks for an email address if the string contains an @ and a dot and looks like an email address.

Note that a certificate must contain an email address in order to be found by this method. Not all certificates embed an email address.

Specifying a File That Contains a Recipient List

PKZIP can extract a recipient list from these kinds of files:

- An ordinary text file that lists the common name of each recipient's certificate on a line by itself

  To use the ***recipient*** option with a text file list of recipients as a sub-option, prefix the file name with the listfile character (@, by default):

  pkzipc -add -recipient=@recipient_list_file.txt test.zip

- Key container files: These kinds of files contain one or more certificates and conform to one of two standards. PKCS#7 files have the file name extensions .p7b and .p7c and do not contain private keys, only public ones. PKCS#12 files have the file name extensions .pfx and .p12 and may contain private keys as well as public keys.

  To use ***recipient*** to specify a key container file to define a recipient list, prefix the file name with a hash (#) character:

  pkzipc -add -recipient=#recipient_list_file.p7b test.zip

  The recipient list will contain the owners of all certificates included in the key container file.

Specifying an Encryption Method with a Recipient List

With the ***passphrase*** option, you can select either strong encryption or weaker, traditional ZIP encryption. The ***recipient*** option, however, always causes SecureZIP to use strong encryption. If you do not use the ***cryptalgorithm*** option to explicitly specify a strong encryption method with a recipient list, and no encryption method is configured for use by default, SecureZIP uses the first method listed in the output from the ***listcryptalgorithms*** command.

Note: The ***listcryptalgorithms*** command and the ***recipient*** and ***cryptalgorithm*** options are available only in SecureZIP.

Encrypting with OpenPGP

You can also specify OpenPGP keys to define a recipient list. You must first configure SecureZIP to enable OpenPGP on your system. See "Setting Up OpenPGP Keyrings."

When defining a recipient list, you can search on the name and email address as with X.509 certificates. You can also search on the OpenPGP KeyID (short or long) with this command:

pkzipc -add -recipient=kid=<KeyID_characters> test.zip

Encrypting File Names

cd

Note: The ***cd*** option uses strong encryption and is available only with SecureZIP.

Someone who cannot decrypt the contents of an archive may still be able to infer sensitive information just from the unencrypted names of files and folders. To prevent this, you can encrypt the names of files (and folders) in addition to their contents. Encrypted file names can be viewed in the clear—that is, unencrypted—only when the archive is opened by an intended recipient if the archive was encrypted using a recipient list, or by someone who has the passphrase, if the archive was encrypted using a passphrase.

Use the ***cd*** option (stands for "archive *central directory*") with the ***add*** command to encrypt file names. The ***cd*** option applies strong encryption to an archive's central directory, where file names and virtually all other metadata about the archive are stored.

An archive that contains encrypted file names requires PKZIP or SecureZIP version 8.0 or later to open it.

The ***cd*** option has two sub-options:

Sub-Option	*Effect*	*Example*
encrypt	Encrypts file names and the archive's central directory. This is the default sub-option, used if you enter ***-cd*** and do not explicitly specify a sub-option.	***-cd=encrypt***
normal	Does not encrypt file names; produces a normal ZIP file. Use to override a configured default setting that would otherwise encrypt file names.	***-cd=normal***

You must use strong encryption when you use the ***cd*** option. You can use either strong passphrase encryption or a recipient list (or both), but you must use one of the strong encryption methods. You cannot encrypt file names using traditional, passphrase encryption.

The following sample command line encrypts file names using a recipient list:

pkzipc -add -recipient="John Q. Public" -cd test.zip

The sample command line below encrypts file names using a passphrase. When you use the ***cd*** option with a passphrase, SecureZIP uses the default strong encryption algorithm (ordinarily AES 256) if you do not explicitly specify an algorithm.

pkzipc -add -passphrase=mysecret -cryptalgorithm=aes,256 -cd test.zip

Encrypting File Names in an Existing Archive

You can encrypt file names in either a new or an existing archive.

- If you add files to an archive that already contains files with unencrypted file names and specify ***cd*** to encrypt file names, SecureZIP encrypts the names of all files in the archive, not just names of newly added files.

  If the archive contains files whose contents are already encrypted, SecureZIP decrypts these files and then re-encrypts them, and their names, using the currently specified encryption method (passphrase/recipient list) and algorithm.

  If SecureZIP cannot decrypt the files, SecureZIP does not update the archive: no files are added, and file names are not encrypted.

- If you update an archive in which file names are encrypted, SecureZIP encrypts the newly added files and their names using the same passphrase or recipient list originally used to encrypt file names in the archive.

Encrypting Using Only FIPS-Approved Algorithms

fipsmode

"FIPS" is an abbreviation for "Federal Information Processing Standards," a set of standards for information processing in federal agencies. The standards are published by NIST (National Institute of Standards and Technology), a branch of the US government. The FIPS 140-2 standard defines security requirements for cryptographic modules and specifies the algorithms that federal agencies may use for cryptographic operations—encrypting, decrypting, signing, and authenticating digital signatures.

The ***fipsmode*** option restricts SecureZIP to using only algorithms that comply with the FIPS 140 standard to perform cryptographic operations.

With ***fipsmode*** on, SecureZIP exclusively uses FIPS-validated algorithms not only to encrypt but also to decrypt. If you try to decrypt a file that is encrypted using an algorithm that is not FIPS-validated, SecureZIP responds with an error or warning and does not decrypt it.

When applying or authenticating signatures, SecureZIP again uses only FIPS-validated hashing algorithms when the ***fipsmode*** option is on. If a signature was created using a hashing algorithm that is not FIPS-validated, SecureZIP shows a warning even if the signature is otherwise valid.

The ***fipsmode*** option is not compatible with the ***204*** option (which cannot create archives with strong encryption).

For the ***fipsmode*** option to work—that is, to actually result in FIPS-mode processing—a FIPS-validated cryptographic module must be installed on your system. SecureZIP supplies UNIX module itself.

Note: In response to NIST Special Publication 800-131A from the National Institute of Standards and Technology, the SHA-1 hashing algorithm is not supported in FIPS 140 mode.

In response to NIST Special Publication 800-131A from the National Institute of Standards and Technology, the 3DES-112 (also known as "two key" 3DES) algorithm is not supported in FIPS 140 mode.

When used with the ***fipsmode*** option, the commands ***listcryptalgorithms*** and ***listhashalgorithms*** list only available FIPS-validated algorithms. For example:

pkzipc -fipsmode -listcryptalgorithms
pkzipc -fipsmode -listhashalgorithms

The ***fipsmode*** option has two sub-options, ***Enabled*** and ***Disabled***, used to configure the default state of the option or, on the command line, to override the configured default.

On UNIX, the option is disabled by default.

Note: The fastest version of the Advanced Encryption Standard (AES) is not FIPS-compatible. If your system is FIPS-enabled, you will not be able to use the FastAES sub-option with the ***-cryptoptions*** command.

The following example turns on ***fipsmode*** for the current command line:

pkzipc -add -recipient="John Public" -fipsmode save.zip *.doc

The next example turns on ***fipsmode*** and uses the ***sfx*** option to create a graphical Windows self-extracting archive mysfx.exe. A self-extracting (SFX) archive created with ***fipsmode*** on extracts in FIPS mode, by default, too.

pkzipc -add -recipient="John Public" -fipsmode -sfx=win32_x86 mysfx *.doc

For more information on self-extracting archives, see "Working with Self-Extracting (PKSFX) Archives" later in this chapter.

The example below overrides a configured default setting of ***fipsmode=enabled*** to turn off ***fipsmode*** for the current command line:

pkzipc -extract -fipsmode=disabled wedding_plans.zip *.*

The following command line prefixes the ***fipsmode*** option with two hyphens (--) to turn off FIPS mode when extracting an SFX archive that was created with the ***fipsmode*** option on. Ordinarily, an SFX archive that was created with the ***fipsmode*** option on extracts in FIPS mode too. This example shows how to override the FIPS flag set internally in the SFX archive to allow files in the archive to be decrypted and authenticated without using only FIPS-validated algorithms:

mysfx.exe --fipsmode

Conversely, the ***fipsmode*** option can also be used with a single hyphen to apply FIPS-mode constraints on extraction to an SFX archive that was not created with the ***fipsmode*** option on.

mysfx.exe -fipsmode

Accessing Recipients in an LDAP Directory

ldap

The ***ldap*** option enables you to access X.509 digital certificates in a Lightweight Directory Access Protocol (LDAP) directory.

Note: To access certificate stores in directories requires SecureZIP Enterprise. SecureZIP accesses certificates in directory stores by making LDAP-based queries to the target directories .

Ordinarily, when you use the ***recipient*** option to do certificate-based encryption, SecureZIP looks for certificates only in your system's local certificate stores. The ***ldap*** option enables you to point SecureZIP to an LDAP directory instead. With the ***ldap*** option, SecureZIP searches the specified LDAP directory first and only looks in local stores if it does not find the certificate it is seeking on an LDAP server.

Note: You cannot use the ***ldap*** option to locate OpenPGP keys.

Use the ***keyserver*** option to locate OpenPGP keys. See the Command Reference page for details on this option.

You can use the ***ldap*** option multiple times to specify multiple LDAP directories to search. Directories are searched in the order listed. If SecureZIP is unable to connect to a directory, SecureZIP issues a warning and tries the next directory.

Here is what SecureZIP does if multiple certificates are found that match a recipient:

- If multiple matching certificates are found in the same LDAP entry, SecureZIP picks the (valid) certificate whose expiration date is farthest in the future. No warning is generated.
- If multiple LDAP entries are found, each containing a matching certificate, SecureZIP uses a certificate from each entry to encrypt the archive and issues warning 59 (*Multiple certificates found*). The certificates may belong to different people, in which case the owner of any of them can decrypt.

The ***ldap*** option has several components, or fields. Only the last one, ***ldap_base***, is always required. The other fields are required only if needed to access a particular LDAP server.

The ***ldap*** option has the following syntax (optional fields are bracketed):

-ldap=[[userid:passphrase@] server[:port][,ssl]/]ldap_base

where:

- `userid` (optional) is the user account with which to log in if the LDAP server requires a login.
- `passphrase` (optional) is the passphrase associated with the user account.
- `server` (optional) is the LDAP server name or TCP/IP address.
- `port` (optional) is the TCP/IP port to use. The default is 389 if no port is specified.
- `ldap_base` (required) is the name of the entry that SecureZIP should use as the base or root of the LDAP search for certificates, analogous to a root folder or directory in a file system.

  The query string format for ldap_base can vary between LDAP implementations. For example, a server may expect query strings in the Internet domain-style format used by default by Microsoft Active Directory (for example, `cn=users,dc=xyz,dc=com`), or it may expect them in X.500 naming format (for example, `o=xyz,c=US`). Check with your LDAP or network administrator for the format to use.

Examples:

pkzipc -add -ldap=john_p:mysecret@192.172.0.1:389/cn=users,dc=xyz,dc=com -recipient="Mary Samplename" save.zip *.doc

pkzipc -add -ldap=jon_p:mysecret@192.172.0.1/cn=users,dc=xyz,dc=com -recipient="Mary Samplename" save.zip *.doc

pkzipc -add -ldap=192.172.0.1/cn=users,dc=xyz,dc=com -recipient=e=mary.samplename@xyz.com save.zip *.doc

pkzipc -add -ldap=/cn=users,dc=xyz,dc=com -recipient=e=mary.samplename@xyz.com save.zip *.doc

The ***ldap*** option must appear *before* the ***recipient*** option, as shown in the examples above, when the two options are used together in a command line.

To avoid having to type a frequently used ***ldap*** option setting, use the ***configure*** command to enable the option setting by default. For example:

pkzipc -config -ldap=192.172.0.1/cn=users,dc=xyz,dc=com

SecureZIP tests an LDAP connection immediately when you configure it. If the connection is bad, SecureZIP returns a warning to inform you of the problem before you try to use the connection to do encryption.
If you configure a default ***ldap*** option setting, it is applied implicitly whenever you use the ***recipient*** option to encrypt.

To remove configured settings for LDAP servers, use the ***--ldap*** option (two hyphens):

- Use the ***--ldap*** option with the ***add*** command (and the ***recipient*** option) to ignore configured ***ldap*** settings just in the current command.
- Use the ***--ldap*** option with the ***configuration*** command to remove any configured default ***ldap*** settings.

  The ***default*** command, which globally restores initial defaults, also removes configured ***ldap*** settings.

Note: The ***ldap*** option can only be used to point SecureZIP to an LDAP server to search for X.509 certificates to use for encryption, not for digitally signing files. Certificate-based encryption uses *public keys*; attaching a digital signature requires access to a *private key*. SecureZIP can only access public keys in certificates in an LDAP directory.

Contingency Keys

Note: To configure contingency keys, you must have a license for SecureZIP command line Enterprise Edition. A separate Policy Manager tool, which runs on Windows, is used to configure contingency keys. The Policy Manager is not provided for PKZIP command line.

Contingency keys, if configured, are used whenever SecureZIP users encrypt.

Contingency keys are recipient keys that an administrator can have automatically included in the recipient list whenever SecureZIP does strong encryption.

Contingency keys enable an organization to decrypt files encrypted by anyone in the organization, whether the files are passphrase encrypted or encrypted for specific recipients. Contingency keys are a safeguard to be sure that important information belonging to the organization does not become inaccessible because no one in the organization can decrypt it.

A contingency key is an ordinary cryptographic key from a digital certificate. The special thing about it is that, once the key is designated as a contingency key, it is automatically included as a recipient whenever SecureZIP encrypts files. This enables the owner of the key to decrypt the files.

If defined, contingency keys are used whenever SecureZIP encrypts. They are used even when the user chooses (strong) passphrase-based encryption and does not pick any recipients.

When defining a contingency key, the administrator can set these options:

- ***Show Contingency Key Settings*** allows the administrator to display a list of contingency keys in use.
- ***Show Contingency Keys During Encryption*** causes SecureZIP to display a line that states the number of contingency keys in use when encrypting. For example:

```
Using 2 contingency keys
```

Creating OpenPGP Files

Some organizations use encryption tools based on the OpenPGP standard, rather than X.509. OpenPGP uses the same Public Key Infrastructure principles for exchanging encrypted files, but uses a decentralized "Web of Trust" method of authenticating signatures. See "Working with OpenPGP Files" for more information.

Before creating any OpenPGP files with SecureZIP, be sure to set up at least one keyring. See "Setting Up OpenPGP Keyrings" for details.

When you create an OpenPGP file containing more than one source file using ***-archivetype=pgp***, as in the following example, SecureZIP creates a GNU TAR archive, copies the selected files to the archive, and then encrypts the TAR archive using OpenPGP. This command-line takes all text files in the current directory, creates a PGP archive called `myfile.pgp`, encrypts it with 128-bit AES and makes it available to a recipient, Test:

pkzipc -add -archivetype=pgp -cryptalg=AES,128 -recipient="Test" -cert="Test" myfile.pgp *.txt

Note: Always use the ***-archivetype*** command when working with OpenPGP files.

If there is only one source file to be encrypted with OpenPGP, SecureZIP skips creating the GNU TAR archive and simply creates the **.pgp* file. You can also use the ***–archiveeach*** command to create separate OpenPGP files for each matching file. Be sure to add the command before specifying the file(s) you want to compress.

pkzipc -add -archivetype=pgp -cryptalg=AES,128 -recipient="Test" -cert="Test" -archiveeach *.txt

Note: When adding text files to an OpenPGP archive, be aware that the format automatically converts line endings in the text file to the Windows CR/LF format. By default, when you extract the file on a Macintosh or UNIX system using PKZIP or SecureZIP, line endings will convert to the native format.

To ensure proper handling of text files across platforms, add the ***–binary*** option to your command.

Attaching Digital Signatures

With SecureZIP, you can attach a digital signature to files in an archive, or to an archive itself. A digital signature assures people who receive the signed file that it is really from the person who signed it and has not been changed.

Note: PKZIP authenticates digital signatures on files signed by others, but you must have SecureZIP to attach digital signatures of your own .

SecureZIP allows you to digitally sign either individual files in an archive, the central directory of the archive, or both. The central directory contains a list of files in the archive. Signing the central directory enables a recipient to confirm that the archive as a whole has not changed. Both PKZIP and SecureZIP authenticate digital signatures on extraction.

Find more information on using digital certificates on this page.

Commands and Options for Signing Archives

certificate

Use the ***certificate*** option to specify a certificate or OpenPGP key to use to sign files. To specify a certificate or key, use one of the sub-options described in the following table.

Note: The ***certificate***, ***hash***, and ***sign*** options described below and the ability to use certificates to attach digital signatures are available only with SecureZIP.

Sub-Option	***To use***	***For example***
<common name>	Specify, in quotes, the common name of the subject of the certificate (that is, the *cn* field in a string representation of a certificate); optionally, precede with: **cn=** SecureZIP searches for certificates by common name by default.	***-certificate=cn="John Public"*** ***-certificate="John Public"***
<email address>	Specify the email address of the certificate (that is, the *e* field in a string representation of a certificate); optionally, precede with: **e=**	***-certificate= e=john.public@xyz.com*** ***-certificate=john.public@xyz.com***

#<file name>	Specify the name and location of a file containing the certificate to use. If the certificate's private key is not in the file with the certificate, use the **keyfile** option to point to the separate file that contains the private key. If necessary, use the **keypassphrase** option to specify a passphrase to read the private key.	***pkzipc -add -certificate=#mycert.pem -keyfile=mykey.key save.zip *.doc*** ***pkzipc -add -certificate=#mycert.p12 -keypassphrase="my passphrase" save.zip *.doc***

For example, if the common name of the subject is *John Q. Public*, you can specify that certificate as follows:

pkzipc -add -certificate="John Q. Public" test.zip

The command uses the *John Q. Public* certificate to sign files. By default, both the files in the archive and the archive itself are signed. Use the ***sign*** option to change what is signed. Use the ***hash*** option to change the hash method used for signing.

The following examples reference a certificate by email address:

pkzipc -add -certificate=john.public@nowhere.com test.zip

pkzipc -add -certificate=e=john.public@nowhere.com test.zip

The prefix "e=" when using an email address is optional. SecureZIP automatically looks for an email address if the string contains an "@" and a dot and looks like an email address.

Note that a certificate must contain an email address in order to be found by this method. Not all certificates embed an email address.

keyfile

You can reference a file that contains a certificate to use for signing with the ***#<filename>*** sub-option of ***certificate***. If the private key is not included in the file with the certificate, use the ***keyfile*** option to specify the file that contains the private key. For example:

pkzipc -add -certificate=#mycert.pem -keyfile=mykey.key save.zip *.doc

The ***keyfile*** option specifies a file containing the private key for the certificate specified by the ***certificate*** option. The option is most useful with SSL server certificates, which often have the private key and certificate in separate files.

keypassphrase

A private key in a file by itself or in a file that contains a certificate may be encrypted and require a passphrase for PKZIP to decrypt it to use. Use the ***keypassphrase*** option to supply the passphrase. For example:

pkzipc -add -certificate=#mycert.p12 -keypassphrase="my passphrase" save.zip *.doc

pkzipc -add -certificate=#mycert.pem -keyfile=mykey.key -keypassphrase="my passphrase" save.zip *.doc

The ***keypassphrase*** option specifies the passphrase used to decrypt private key information. This can be the passphrase used for your certificate store for a PKCS#12 file (specified with the ***certificate*** option), an OpenPGP private key, or a key file specified with the ***keyfile*** option.

hash

You can use the ***hash*** option with the ***certificate*** option to specify the hash method/algorithm to use for signing. The option has the sub-options shown in the following table.

Sub-option	***Description***
sha1	Uses the SHA-1 hashing algorithm (default) (not FIPS-compatible; cannot be used with the ***fipsmode*** option)
sha256	Uses the SHA-256 hashing algorithm (***fipsmode*** default)
sha384	Uses the SHA-384 hashing algorithm
sha512	Uses the SHA-512 hashing algorithm
md5	Uses the MD5 hashing algorithm (not FIPS-compatible; cannot be used with the ***fipsmode*** option)

The SHA algorithms are all stronger than the MD5 algorithm. Among the SHA algorithms, the higher-numbered ones are stronger than the lower-numbered ones. See the ***fipsmode*** option for information on which algorithms are supported for FIPS processing on different versions of Windows.

Use the ***listhashalgorithms*** command to list hashing algorithms available on your system. If ***fipsmode*** is on, the ***listhashalgorithms*** list shows only FIPS-validated algorithms.

The ***hash*** option's default is configurable.

The following example specifies the SHA-256 algorithm and the "My Cert" certificate to use to sign files:

pkzipc -add -certificate="My Cert" -hash=sha256 test.zip *.*

sign

You can use the ***sign*** option with the ***certificate*** option to specify whether to sign the central directory of the archive itself, the archived files, or both.

Signing the files enables a user to verify that the files are the same files you signed; signing the archive itself enables a user to verify that the contents of the archive have not changed—that, for example, no files have been added or removed. By default, SecureZIP signs both.

The sub-options are listed in the following table.

Sub-option	*Description*	*Example*
cd	Sign only the central directory of the archive, not the files in the archive	***-sign=cd***
files	Sign only the files in the archive, not the archive itself	***-sign=files***
all **(Default)**	Sign both the archived files and the archive itself	***-sign=all***
none	Do not sign files. This sub-option is used to turn signing off if it has been configured.	***-sign=none***

For example:

pkzipc -add -certificate="My Cert" -sign=cd test.zip *.*

You can also use ***sign*** to add a digital signature to an existing archive. See "Using X.509 Digital Signatures" for more information.

listcertificates

Use the ***listcertificates*** command to list the certificates that are in a specified store on your system. Information for each certificate tells whether the certificate is *Valid*, *Expired*, *Not Trusted*, or *Revoked* (if known). If OpenPGP keys are enabled and available on the system, these will be displayed, including the Key ID value.

Note: If an OpenPGP key contains more than one userid, multiple certificates will display.

Specify the store using one of the sub-options in the following table. Personal certificates in the MY store are listed by default if no sub-option is used.

Sub-option	*Description*	*Example*
my	Lists certificates in the MY store. This store contains your personal certificates with private keys. If OpenPGP keys are enabled and available on the system, these will be displayed, including the Key ID value.	***pkzipc -listcertificates*** ***or*** ***pkzipc -listcert=my***
address book	Lists certificates in the AddressBook store. This store contains public certificates and public keys belonging to other people. If OpenPGP keys are enabled and available on the system, these will be displayed, including the Key ID value.	***pkzipc -listcert=addressbook***
ca	Lists certificates in the CA store. These are intermediate certificates in a trust chain, created by a certificate authority to validate other certificates.	***pkzipc -listcert=ca***
root	Lists certificates in the Root store. These are certificates at the beginning of a trust chain, which are trusted by the system.	***pkzipc -listcert=root***

For example, the following command line lists certificates in the MY store:

pkzipc -listcertificates

The command line produces output like the following. In this case, the MY store contains four certificates, three of which have the same name, *John Doe*.

```
      John Doe: Valid
      John Doe: Expired
      John Doe: Expired
users,John Doe: Valid
```

Setting a Default Certificate

If you only use one digital certificate to sign your archives, you can skip the ***certificate*** command in your scripts. Do this by defining that certificate as your default.

To define the "John Q. Public" certificate as your default, use the following command:

pkzipc -config -certificate="John Q. Public"

Once you have a certificate configured, each time you use ***sign***, whether as a command or option, SecureZIP will attach this certificate to your archive.

You can still use the ***certificate*** option to attach a different certificate than your default.

See Chapter 8 for more information on setting and changing defaults in PKZIP command line.

Time Stamping Your Signed ZIP Archive

When you need to establish not only *who* is responsible for a file or set of files, but also *when* it was created, digital time stamping is a critical service. As you know, dates are critical for establishing original intellectual property rights, including copyright and patents. While all files carry a creation date as part of its default metadata, it is not very hard to manipulate this date before you create and sign the archive in question. The goal is to create a timestamp that cannot be changed, even by the owner of the file. Using a Time Stamp Authority outside of your computing environment takes the guesswork out of confirming the validity of a document.

The Internet Engineering Task Force governs digital time stamps through two standards: RFC 3161 establishes the method by which a client can connect to a secure computer that will stamp the document with its current date and time. This secure computer is called the Time Stamp Authority (TSA) or Time Stamp Server (TSS). RFC 4998, among many other things, defines what happens when a time stamp authority's certificates expire, or are otherwise compromised.

With PKZIP command line's support for digital time-stamping, you can add a timestamp to any signed archive.

Note: PKZIP command line only supports digital time-stamping for ZIP archives.

Before beginning the process, you need to know the URL of your Time Stamp Authority. The TSA server may be on your network or on a public server.

To sign a new archive containing all documents in the current directory, and add a digital time stamp, type:

pkzipc -add -sign=timestamp -ts=<TSA_URL> test.zip *.doc

where <TSA_URL> is the location of your Time Stamp Authority's service.

PKZIP will calculate a hash based on the archive's data and send that to the TSA. The TSA adds a timestamp to the hash and calculates the hash of this combination of the original hash with the timestamp. This second hash is then digitally signed with the TSA's private key. All this information is then sent back to you. PKZIP then adds the timestamp to the archive's central directory signatures.

Updating and Renewing Time-Stamped Archives

The IETF standards permit multiple timestamps on a file, allowing for time-stamped archives to be updated and refreshed. In this way, you can establish a record of creation and updates. PKZIP command line automatically handles updating the timestamp when you update the archive. Because the archive has changed, a new timestamp will be generated, but the original file signatures will be preserved by nesting the two signatures.

You must renew your time-stamped archives before the TSA's certificate expires. Use the ***sign*** command:

pkzipc -sign=timestamp -ts=http://<TSA URL> test.zip

When renewing the timestamp, the original items and their order in the archive will be preserved normally.

Note: Renewing time-stamped archives spanned across different media is not supported.

Writing an Archive to STDOUT and Special Files

Ordinarily, when you use the ***add*** command to archive files, you write the resulting archive to a physical file that you specify in the command line. For example, the following command line archives text files to the archive myfiles.zip:

pkzipc -add myfiles.zip *.txt

In this section, you will see how to create an archive as a data stream to send to destinations besides a physical file, notably, to STDOUT, a named pipe, a UNIX domain socket, or a device file.

Note: When PKZIP compresses and encrypts data to write an archive to a data stream, the data goes to the stream without ever appearing on disk in unencrypted form. PKZIP does create a temporary file to get the size of the data to put in local headers, which must be written before file data. But the data is already compressed and encrypted when it's placed in the temporary file. No security vulnerability is created.

Writing an Archive to STDOUT

You can write an archive to *standard output*, or STDOUT, instead of to a physical file. Data written to STDOUT appears on your computer screen but is not saved to disk (unless you do something extra to save it). It can also be piped to another program or be redirected to (for instance) a file.

To have PKZIP write the output of the ***add*** command to STDOUT, use a hyphen "-" in place of the name of an archive file. You must also use the ***noarchiveextension*** option to prevent PKZIP from outputting to a file named .zip instead of to STDOUT. And finally, you should include the ***silent*** option to suppress the informational messages that PKZIP normally outputs so that these are not inserted in the archive data stream. For example:

pkzipc -add -noarchiveextension -silent=normal - *.txt

PKZIP creates ZIP-format archives by default. To write a different type of archive to STDOUT, use the ***archivetype*** option to specify the type. For example, the following command line tells PKZIP to write a TAR-format archive to STDOUT:

pkzipc -add -archivetype=tar -noarchiveextension -silent=normal - *.txt

The command line below sends output to STDOUT and then redirects that output to archive myfile.zip.

pkzipc -add -noarchiveextension -silent=normal - *.txt > myfile.zip

When redirecting STDOUT to a file, you can use the ***exclude*** option to make sure that PKZIP does not include the file to receive the output in the set of files to be zipped. Unlike when writing directly to a specified archive file, PKZIP cannot infer from the command line that it should skip a file to which you redirect output. The ***exclude*** option explicitly tells PKZIP to skip specified files.

For example, the following command line archives all files in a directory and redirects output to a file in the same directory. The ***exclude*** option tells PKZIP not to add that file.

pkzipc -add -noarchiveextension -silent=normal -exclude=myfile.zip - *.* > myfile.zip

You can use a hyphen "-" in place of the name of an archive file when you extract, as well. Used in a command line with the ***extract*** command, the hyphen tells PKZIP to extract files from STDIN (*standard input*).

For example, the following command line extracts files from STDIN instead of from a named archive.

pkzipc -extract -noarchiveextension -silent=input -

When extracting from STDIN, set ***silent*** to the ***input*** sub-option, as in the command line above, to suppress any PKZIP requests for input (a passphrase, for example). If input is needed, the extraction fails with an error.

The ***noarchiveextension*** option is needed so that PKZIP does not try to extract from a file named .zip. If the archive is not a ZIP archive, use the ***archivetype*** option to specify its type. For example, the following command line tells PKZIP that the file is a BZIP2 archive:

pkzipc -extract -archivetype=bzip2 -noarchiveextension -silent=input -

You can combine writing to STDOUT and extracting from STDIN to securely transfer files between two systems. For example, the following command line compresses and encrypts the files to be transferred and adds them to a ZIP archive. The archive is written to STDOUT instead of to a file. The command line pipes the output to the *rsh* (*remote shell*) system command, which runs PKZIP on the remote system to extract the files from STDIN.

pkzipc -add -noarchiveextension -cryptalgorithm=aes,256 -recipient=Jon -silent - | (rsh user@remote_system pkzipc -extract -noarchiveextension -silent=input -)

Writing an Archive to a Named Pipe, UNIX Domain Socket, or Device File

An archive can be written to a named pipe (Windows or UNIX), UNIX socket or device file instead of to a physical file.

The named pipe, socket, or device must already exist. You can then write an archive to it with a command line like the following. Use the name of the pipe or socket in the command line in place of the name of an archive file.

pkzipc -add -noarchiveextension <name of pipe or socket> <files to zip>

As when writing to STDOUT, you must use the ***noarchiveextension*** option to prevent PKZIP from outputting to a .zip file—in this case, one named for the pipe or socket.
PKZIP creates ZIP-format archives by default. To write a different type of archive, use the ***archivetype*** option to specify the type. For example, the following command line tells PKZIP to write a TAR-format archive:

pkzipc -add -archivetype=tar -noarchiveextension <name of pipe or socket> <files to zip>

You must use the full UNC path when referring to a named pipe on Windows. For example:

pkzipc -add -noarchiveextension \\.\pipe\mypipe *.doc

In the preceding example, the dot in the path

\\.\pipe\mypipe

references the current machine. To reference a pipe on a different machine—named *boulder*—specify the machine.

\\boulder\pipe\mypipe

You can use either a name or an IP address to specify a machine.

Setting a Timeout for PKZIP to Wait

timeout

The ***timeout*** option sets a specified number of seconds for PKZIP to wait for another process to send or be ready to receive (more) data on a socket or device file. The timeout gives the other process time to handle data from PKZIP or to produce data for PKZIP to act on. By default, PKZIP waits 30 seconds. If the timeout period elapses without a response from the other process, PKZIP returns an error and halts processing.

For example, the following command line extracts files from an archive on a socket using a timeout of 60 seconds:

> ***pkzipc -extract -noarchiveextension -timeout=60 mysocket***

The timeout option has a default value of 30 seconds if no value is specified. The option is configurable.

Adding Data from STDIN or Special Files

Besides regular files, you can add to an archive data streamed from STDIN, a named pipe, a UNIX domain socket, or a device file. You must specify the ***stream*** option.

Adding Streamed Data

stream

On UNIX, PKZIP ordinarily ignores pipe and socket files. The ***filetype*** option can be used to cause PKZIP to include definitions (name, permissions, times, and so on) to recreate pipes, sockets, or devices but does not capture their data. The ***stream*** option, used with ***filetype***, gets the data from these sorts of files.

Sub-options of the ***filetype*** option enable you to specify types of files to include or exclude. The command line below uses ***filetype*** with the ***pipe*** sub-option to include pipe files. (There are also ***block***, ***char***, and ***socket*** sub-options). The command line uses the ***stream*** option to get the pipe data. Data is saved and referenced in the archive by the name of the pipe—mystream in the example.

> ***pkzipc -add -filetype=pipe -stream data.zip mystream***

You can use the ***rename*** option to change the name by which the data is stored in the archive. For example, the following command line renames it `strea m_data.txt`:

> ***pkzipc -add -filetype=pipe -stream -rename=/mystream/stream_data.txt/ data.zip mystream***

The ***stream*** option is also used when extracting streamed data, to write the data to a pipe of the same name as the file to be extracted:

> ***pkzipc -extract -stream data.zip mystream***

Without the ***stream*** option, the command line writes the data to an ordinary file named mystream. If a pipe of that name exists, the pipe is overwritten (See "Extracting Data to STDOUT or Special Files").

Adding Streamed Data from STDIN

To specify standard input (STDIN) as a data source, use a hyphen.

In the following command line, a program pipes data to PKZIP, which PKZIP reads from STDIN and archives. The data is stored and referenced in the archive as - (hyphen).

> ***<command> | pkzipc -add data.zip -***

Use the ***rename*** option to give the archived copy of the data a friendlier name. For example:

> ***pkzipc -add -rename=/-/output.txt/ data.zip -***

The command line below combines reading streamed data and writing it to a streamed archive. From STDIN, PKZIP reads the data output by <command>, archives the data, and writes the anonymous archive to STDOUT. (See "Writing an Archive to STDOUT.")

> ***<command> | pkzipc -add -noArchiveExtension -silent=all -stream --***

Like STDIN, STDOUT is indicated by a hyphen. At the end of the command line, the first hyphen identifies where to write the archive (STDOUT), and the second hyphen specifies the data source (STDIN).

Compressing Files in Subdirectories

recurse

PKZIP does not automatically compress files that appear in subdirectories, unless you specify those directories, or use the ***recurse*** option with the ***add*** command. With the ***recurse*** option, all specified files in a directory structure, including files located in subdirectories will be compressed.

If you have a directory called tut with a nested subdirectory called test, to compress all of the files in the tut directory and all files in the tut/test directory, you would type the following in the tut directory:

pkzipc -add -recurse test.zip *

All files in the tut directory as well as those files in subdirectories of the tut directory are compressed. However, directory path information is not stored within the .ZIP file. If you want to store directory information within your .ZIP file (in addition to compressing all the files in those directories), use the ***path*** option with the ***recurse*** option or simply use the ***directories*** option.

Note: Use the ***include*** option or place quotation marks around wildcard designations to avoid automatic wildcard expansion by the shell, which may interfere with your pattern search. See "Using Wildcards with PKZIP on UNIX."

Storing Directory Path Information

path

Normally, when PKZIP compresses files, only the files are stored within the .ZIP file, not the paths of those files. However, you can instruct PKZIP to store the directory path information of a file within the .ZIP file.This enables you to restore the directory structure when you extract the files.

For example, if a file you are compressing appears in the `doc/temp` directory, you can store the file within the .ZIP file as:

doc/temp/<file name>

To do this, use the ***path*** option with the ***add*** command. For example, the following command line adds all .TXT files in the specified directories and saves the specified path information:

pkzipc -add -path test.zip doc/temp/*.txt

If path information is saved, you can use the ***directories*** option with the ***extract*** command to extract files to the saved paths. PKZIP creates the directories on the saved path if they do not already exist.
Note that the ***path*** option gets files only from the specified directory. To get files in subdirectories of that directory as well, use the ***directories*** option instead of ***path***. Or use ***path*** together with ***recurse***.

Additional Methods for Storing Directory Path Information

The ***path*** option has sub-options that enable you to specify the path information stored. These sub-options are listed in the table below. By default, using the ***path*** option without a sub-option stores relative path information for all files added.

Sub-option	*To*	*For example*
current	Store the directory path relative to the current location.	***pkzipc -add -path=current docs.zip docs/**** In this example, only directory information under the docs directory will be stored. Parent directory information will not be stored.
root, full	Store the full path, starting from the root directory down.	***pkzipc -add -path=root docs.zip docs/**** In this example, the entire directory path, starting from `root` directory will be stored.
specify	Stores path information for subdirectories under the specified directories	***pkzipc -add -path=specify docs.zip temp/docs/**** Stores path information for subdirectories under `temp\docs`.
none	Turn off the path option. (Used to override configuration file).	***pkzipc -add -path=none docs.zip /temp/docs/**** In this example, only the file names are stored.

Storing and Recreating Directory Path Information

directories

The ***directories*** option works with both ***add*** and ***extract*** commands.

- With the ***add*** command, the ***directories*** option is equivalent to using the ***recurse*** and ***path*** options together. It instructs PKZIP to search subdirectories for files and to save the files and their directory path information in the .ZIP file.
- With the ***extract*** command, the ***directories*** option extracts any directory tree structure saved with files.

The following example uses the ***directories*** option with the ***add*** command to add any files called `whatsnew.htm` in the current directory or in any subdirectory of the current directory:

pkzipc -add -directories testdir.zip whatsnew.htm

Or abbreviated:

pkzipc -add -dir testdir.zip whatsnew.htm

Screen output lists any matching files found in subdirectories:

```
Creating .ZIP: testdir.zip

 Adding File: Win/PK/Whatsnew.htm Deflating (67.0%), done.
 Adding File: Win/SZ/Whatsnew.htm Deflating (66.7%), done.
```

The following example gets all .htm files in the current directory or its subdirectories:

pkzipc -add -dir testdir.zip *.htm

To tell PKZIP to start looking for matches from a subdirectory of the current directory, specify the path to the subdirectory. The following example gets all `wh atsnew.htm` files in *mysub/* or any of its subdirectories:

pkzipc -add -directories testdir.zip mysub/whatsnew.htm

The example below gets all .htm files in *mysub/* or any of its subdirectories:

pkzipc -add -directories testdir.zip mysub/*.htm

If you have multiple *mysub/* subdirectories under the current directory, you can get files from just those subdirectories by using a wildcard for the subdirectory from which to start the search:

pkzipc -add -directories testdir.zip */mysub/whatsnew.htm

The command line below is similar, but it limits the search for *mysub/* subdirectories to just those under the *nextsub/* subdirectory:

pkzipc -add -directories testdir.zip nextsub/*/mysub\whatsnew.htm

Even if the command line includes the ***directories*** option, you can turn off the searching of subdirectories for matching files by specifying a full path beginning with a backslash (for the root directory) in the pattern. The pattern must also not include any wildcard characters (* or ?).

For example, the following command line adds only the specified file; it does not add matching files from subdirectories of MyFiles:

pkzipc -add -directories testdir.zip /MyFiles/whatsnew.htm

For information on extracting files saved with directory information, see the section "Retaining Directory Structure while Extracting."

Note: Use the ***include*** option or place quotation marks around wildcard designations to avoid automatic wildcard expansion by the shell, which may interfere with your pattern search. See "Using Wildcards with PKZIP on UNIX."

As with the ***path*** option, PKZIP provides several choices for saving directory path information. The following table lists the sub-options you can use with ***dir ectories*** option:

Sub-option	***To***	***For example***
current	Store the directory path relative to the current location.	***pkzipc -add -directories=current docs.zip docs/**** In this example, only directory information under the `docs` directory will be stored. Parent directory information will not be stored.
root or full	Store the full path, starting from the root directory down.	***pkzipc -add -directories=root docs.zip docs/**** In this example, the entire directory path, starting from `root` directory will be stored.
specify	Store path information for subdirectories under the specified directories	***pkzipc -add -directories=specify docs.zip temp/docs/**** Stores path information for subdirectories under `temp/docs`.
none	Turn off the path option. (Used to override configuration file).	***pkzipc -add -directories=none docs.zip /temp/docs/**** In this example, only the file names are stored.

Setting the Compression Level

Native ZIP compression (which uses the Deflate compression algorithm) and the ***bzip2*** and ***deflate64*** compression options each support a range of compression levels from 0 (no compression) to 9 (maximum). By default, each of these options uses level 5, or *normal*, compression. Normal compression strikes a middle balance between compression and performance. In general, greater compression takes more time.

You can use the ***level*** option to specify a compression level from 0 to 9 when you create or update a ZIP file using one of the compression methods named above.

Alternatively, you can use the options ***normal***, ***store***, ***speed***, ***fast***, and ***maximum*** to specify a desired balance between speed and degree of compression. See "Specifying a Compression Level by Name."

With the ***dclimplode*** option, you set the compression level in a different way, namely, by specifying the dictionary type and size as sub-options.

Specifying a Compression Level from 0-9

level

The ***level*** option enables you to specify a level or degree of compression to use when creating or updating a ZIP archive with the Deflate64, BZIP2, or default Deflate compression methods. (See the ***deflate64*** and ***bzip2*** options to learn about using these compression methods.)

To set a compression level with the ***level*** option, specify a numeric value for the option from 0 to 9. A value of 0 specifies zero compression.

The following command line specifies a compression level of 2 and uses the native Deflate compression method:

pkzipc -add -level=2 test.zip *.doc

The following command line specifies level 2 compression and the BZIP2 compression method to create or update a ZIP archive:

pkzipc -add -bzip2 -level=2 test.zip myfile.doc

Level 5 is the default compression level for ***level***. You can use the ***configuration*** command to set a different default. For example, the following command line sets the default value for ***level*** to 9:

pkzipc -config -level=9

For information on changing default settings, see this page.

Specifying a Compression Level by Name

store, speed, fast, normal, maximum

As an alternative to setting numeric compression levels with ***level***, you can use the options ***normal***, ***store***, ***speed***, ***fast***, and ***maximum***.

These options enable you to use non-numeric names to specify a desired balance between speed and degree of compression. For example, the following command line specifies the ***fast*** compression option:

pkzipc -add -fast test.zip *.doc

The non-numeric compression level options are described in the following table:

Option	*Description*	*Example*
speed	Provides the fastest performance and the least compression: some files are compressed with the Deflate method, using level 1 compression; others* are stored (level 0) uncompressed.	***pkzipc -add -speed test.zip *.doc*** ***pkzipc -add -bzip2 -speed test.zip *.doc***
fast	Provides the second fastest compression: some files are compressed with the Deflate method, using level 2 compression; others* are stored (level 0) uncompressed	***pkzipc -add -fast test.zip *.doc***
maximum	Provides the highest level of compression (level 9)	***pkzipc -add -max test.zip *.doc***
store	Provides zero compression: just stores files inside the archive (level 0)	***pkzipc -add -store test.zip *.doc***
normal (Default)	Provides a middle balance of compression and speed (level 5)	***pkzipc -add -norm test.zip *.doc*** You would only need to use this option if you changed the default compression level. See this page for information on setting defaults.

* Types of files that the ***speed*** and ***fast*** options store uncompressed are listed below. The other named options (except ***store***) compress files of these types. You can also use the ***level*** option to compress files of these types.

*.bz2	*.jpeg

*.bzip2	*.jpg
*.cab	*.mp3
*.gz	*.mpeg
*.gzip	*.mpg
*.rar	*.sxw
*.gif	

Compressing Files with a List File

Instead of specifying a specific file or file pattern in your command line, you can point PKZIP to a list file that lists all the files or file patterns that you want to operate on. A list file is an ASCII text file that contains file names or file patterns and path information. A list file can be an ideal solution for users who archive specific file sets on a regular basis. Using a list file saves time in that you do not need to type file names and paths each time you wish to compress these files with PKZIP. A list file may contain wildcard specifications (*,?) as well as exact file names and paths.

A list file in a UNIX-based environment might look like this:

```
/usr/local/pkware/pkzipc/*.doc
/usr/local/pkware/pkzipc/pkzip.html
/usr/local/pkware/pkzipc/?????.exe
/*
```

You reference a list file in the command line by prefixing its name with the list character—"@" by default. See the ***listchar*** option if you want to use a different character.

The following example adds the files listed in `lst.txt` to the archive `test.zip`:

> ***pkzipc -add test.zip @lst.txt***

You can also use a list file to specify files to exclude from an archive, based on some criteria, using the ***exclude*** option. The ***exclude*** option is discussed in The Basics. For more information on the ***listchar*** option, see "Changing the List Character for List Files."

Note: The way you list files to extract is slightly different from the way you list files to add to an archive. See "Extracting Files with a List File" for more information.

Getting a List of Files from Standard Input

Use a hyphen prefixed with the list character ("@" by default) to identify a set of files in standard input as a list. For example, in the following command line, PKZIP treats a list of files output from *some program* as a list file and compresses the files into *test.zip*:

> ***<some program> | pkzipc -add test.zip @-***

The special, dynamically constructed list can also be used with the ***include*** and ***exclude*** options. For example:

> ***<some program> | pkzipc -add test.zip -include=@-***
> ***<some program> | pkzipc -add test.zip -exclude=@- *.doc***

Compressing Files with the Deflate64 Method

deflate64

The ***deflate64*** option enables you to use the Deflate64 compression method to compress files and create ZIP archives. The Deflate64 method can produce greater compression than the Deflate method that PKZIP uses by default because Deflate64 uses a larger dictionary window (64K compared to 32K).

Note: Not all ZIP-compatible programs from other vendors can extract files compressed with the Deflate64 method.

You can use the ***level*** option with ***deflate64*** to specify a level of compression from 0 to 9 (0 is zero compression).

The following command line uses the Deflate64 method with the ***level*** option set for maximum compression:

> ***pkzipc -add -deflate64 -level=9 mydocs.zip *.doc***

Compressing Files with the BZIP2 Method

bzip2

BZIP2 is an open-source compression algorithm that requires more memory and processing power than standard ZIP compression but provides greater compression. PKZIP can use BZIP2 compression to create either ZIP or BZIP2-format archives (.bz2 files). A BZIP2 archive, unlike a ZIP archive, can contain only a single file.

Files compressed with the BZIP2 method can be extracted with most versions of PKZIP, 4.6 and later, but other ZIP-compatible programs may not be able to extract files compressed with BZIP2.

You can use the ***level*** option with ***bzip2*** to specify a level of compression from 0 to 9 (0 is zero compression).

The following command line uses the BZIP2 method to create a ZIP file. The ***level*** option specifies maximum compression:

pkzipc -add -bzip2 -level=9 mydocs.zip *.doc

Compressing Files with the LZMA Method

lzma

The LZMA compression algorithm often produces a higher compression ratio than BZIP2 but uses a lot of memory—as much as 16 MB—and takes more time than Deflate.

Files compressed with the LZMA method can be extracted with PKZIP versions 12.3 and later, but other ZIP-compatible programs may not be able to extract such files.

Compressing Files Compatible with the Data Compression Library

dclimplode

The ***dclimplode*** option enables you to use the same compression algorithms used by the PKWARE Data Compression Library. Files compressed with this method can be extracted by most versions of PKZIP 2.5x and later, though not by other .ZIP-compatible programs.

When using the Implode compression method, you must specify dictionary type (ASCII or BINARY) and dictionary size (1024, 2048, or 4096). In general, the larger the dictionary, the greater the compression. Use the BINARY dictionary when compressing binary files (for example, executable programs) or when the type of the file is unknown. Use the ASCII dictionary with ASCII (text) files.

For example, to use the DCL Implode method to compress all text files in a directory, type the following:

pkzipc -add -dclimplode=ascii,4096 text.zip *.txt

Compressing Files with the PPMd Method

ppmd

The ***ppmd*** option achieves especially good compression for natural language text but can use a lot of memory (~16 MB) and takes more time than Deflate.

Files compressed with the PPMd method can be extracted with PKZIP versions 12.3 and later, but other ZIP-compatible programs may not be able to extract such files.

Compressing Files to a Specified Type of Archive

archivetype

The ***archivetype*** option explicitly tells PKZIP the type of archive to create or extract. Use the option when PKZIP cannot figure out the correct archive type from the archive's file name. For some examples, see "Writing an Archive to STDOUT."

PKZIP creates ZIP archives by default: When you use the ***add*** command to create a new archive, PKZIP creates a ZIP archive if you do not specify a file name extension that PKZIP recognizes as associated with a particular archive type.

For example, the following command creates a ZIP archive called *myfile.foo.zip*:

pkzipc -add myfile.foo

Similarly, if the command line does not tell PKZIP the type of archive to extract from, PKZIP tries to extract files from a ZIP-format file.

With the ***archivetype*** option, you can explicitly tell PKZIP the type of archive to work with.

For example, the following command line creates an archive *myfile.foo.bz2* of the BZIP2 archive type. The file name extension *bz2* associated with the BZIP2 archive type is added to the file name:

pkzipc -add -archivetype=bzip2 myfile.foo

A simpler way to create a BZIP2 archive called *myfile.foo.bz2* is to specify the file name extension as part of the file name In this case, you do not need the ***archivetype*** option:

pkzipc -add myfile.foo.bz2

Note: You cannot create an OpenPGP-based archive by only using the .pgp extension. Always use ***archivetype=pgp*** when working with OpenPGP files.

When you specify the archive type with ***archivetype***, you can include the ***noarchiveextension*** option to tell PKZIP not to add an extension to the file name. For example, the following command suppresses the *bz2* extension that would normally be appended and creates a BZIP2 archive named *myfile.foo*

pkzipc -add -archivetype=bzip2 -noarchiveextension myfile.foo

Compressing Files to Removable Media

span

With PKZIP, you can save your .ZIP or self-extracting archive to removable media when you create it (instead of saving it on your hard disk drive). You can also create a *split* archive that is saved as multiple files on your hard disk. You can also have PKZIP format or wipe your removable media before writing to it.

Creating a Split Archive

The ***span*** option is also used to create a *split* archive. A split archive is an archive created in segments, all of which are written to your hard disk as separate files.

To create a split archive on your computer disk, specify a size in bytes, or use a predefined size from the following table:

Predefined size	*Comment*
360	360KB floppy disk (362496 bytes)
720	720KB floppy disk (730112 bytes)
1.2	1.2MB floppy disk (1213952 bytes)
1.44	1.44MB floppy disk (1457664 bytes)
2.88	2.88MB floppy disk (2915328 bytes)
95.7	100MB ZIP disk (100431872 bytes)
650	650MB CD-ROM (681574400 bytes)
700	700MB CD-ROM (734003200 bytes)

For example, to create a split archive of size 1.44 Mb to your local system, type the following command:

pkzipc -add -span=1.44 /home/<user>/test.zip *.doc

You can also use ***k***, ***m***, ***g***, or ***t*** to specify split sizes in kilobytes (1024 bytes), megabytes (1024 Kb), gigabytes (1024 Mb), and terabytes (1024 Gb), respectively.

The following command line creates a split archive of 75 Mb:

pkzipc -add -span=75m /proc/bus/usb/test.zip *.doc

To have PKZIP format or wipe removable media before writing to it, use the ***span*** command with ***wipe***. For example, the following command line formats the media prior to creating a ZIP archive:

pkzipc -add -span=format /proc/bus/usb/test.zip *.doc

Preserving International Characters in File Names

utf8

The ***utf8*** option enables UTF-8 characters in file names and file comments to be correctly displayed when an archive's contents are viewed or extracted in compatible non-UTF-8 locales.

For example, with the ***utf8*** option, you can archive files in a Japanese locale using the EUC character set (and the ***utf8*** option) and then correctly view or extract the files in a Japanese locale using the Shift-JIS character set.

The option can be used with these commands/options (***comment*** can be either a command or an option):

- ***Add***
- ***Comment***

If a command line containing the ***utf8*** option modifies an archive in any way, UTF-8 characters are used in the names of all files in the archive.

Comments will always follow the format of the file name it is attached to. Applying ***--utf8*** to a comment on a file with UTF-8 character formatting will not remove UTF-8 characters from the comment.

In general, use the ***utf8*** option when you add to an archive files that contain international (that is, non-English) characters in file names and file comments. For example:

pkzipc -add test.zip -utf8 *.*

PKZIP displays the following message to highlight that the option is used:

```
Using UTF-8 file names and comments
```

PKZIP uses the ***utf8*** option automatically when run in a UTF-8 locale (such as *ja_JP.UTF-8*); you do not need to use it explicitly.

The ***utf8*** option is incompatible with the ***204*** option: an error results if the two options are used together. (PKZIP does not turn on the ***utf8*** option automatically on UNIX if the ***204*** option is used.)

PKZIP/SecureZIP Server version 8.6 or SecureZIP for Windows version 11 is required to extract files added with the ***utf8*** option, so use the option only with archives that you expect to be extracted with these (or later) versions of these programs.

Creating Multiple, Respective Archives

archiveeach

With the ***archiveeach*** option, you can create a separate archive for each of multiple files specified in a single command line.

pkzipc -add -archiveeach *.*

With ***archiveeach***, you do not specify names for new archives. PKZIP names each new archive after the file it contains, with an archive-type file name extension (*ZIP* by default) appended to the end. For example, a ZIP archive created for file `mydata.xls` is named `mydata.xls.zip`. An archive created for file `mydata.zip` is named `mydata.zip.zip`.

If an archive with the same name already exists in the target location, PKZIP appends a number to the archived file name before appending the .zip (or other file name extension). For example: `mydata.xls2.zip`. Use ***archiveeach*** with ***overwrite*** to specify different behavior, such as

pkzipc –add –archiveeach -overwrite=never *.xls

To specify an particular archive type, use the ***archivetype*** option with the ***archiveeach*** option. The ***archiveeach*** option can also be used with the ***encode*** option, to convert the archive initially created to a different type. By using ***archivetype*** and ***encode*** together with ***archiveeach***, you can, for example, create multiple .tar.gz files:

pkzipc -add -archiveeach -archivetype=tar -encode=gz /home/<user>/data*.*

You can specify a destination for the new archives in a sub-option to ***archiveeach***:

pkzipc -add -archiveeach=C:\newzips /home/<user>myfiles*.*

You can use the ***substitution*** option to have PKZIP add a timestamp to the name of a new destination directory created for the archives. See "Inserting a Timestamp in the Archive File Name."

Storing File Information

PKZIP allows you to store specific file attribute/information within your .ZIP file. You can:

- Store file attributes, including hidden, system, archive, and read-only.
- Store extended file attribute information.
- Remove (mask) file attributes.

Refer to the sections that follow for more information.

Compressing Files Based on File Type

filetype

Include or exclude files by type when adding or extracting files with the ***filetype*** option . Specify the type of file in the sub-option. Precede the sub-option with a hyphen to exclude files of that type, or use the sub-option without a hyphen to include such files. For example,

... -filetype=-hidden ...

on the command line excludes hidden files regardless of the default configuration setting. To specify multiple sub-options, separate them with commas.

The following table lists sub-options for file types you can specify with ***filetype***.

Sub-Option	*To*	*For example*
block	Include/exclude block special files. These are files with a mode that begins with a "b" (`brw-------`).	***pkzipc -add -filetype =block test.zip /dev/fd****
char	Include/exclude character special files. These are files with a mode that begins with a "c" (`crw-------`).	***pkzipc -add -filetype=char test.zip /dev/tty****
directory	Include/exclude directory information.	***pkzipc -add -filetype=dir test.zip***
hidden	Include/exclude hidden files. These are files that have a dot (.) in the first position of the file name (`.profile`).	***pkzipc -add -filetype=hid test.zip***
hlink	Include/exclude hard linked files. Hard linked files have a link count greater than one.	***pkzipc -add -filetype =hlink test.zip***
pipe	Include/exclude pipe files. These are files with a mode that begins with a "p" (prwxr-xr-x). Adds the pipe specification or definition (name, permissions, times, and so on), not pipe data.	***pkzipc -add -filetype=pipe test.zip***
regular	Include/exclude regular files. These are included by default if no file type is specified.	***pkzipc -add -filetype =regular test.zip***
slink	Include/exclude symbolically linked files. These are files with a mode that begins with a "l" (`lrwxr-xr-x`)	***pkzipc -add -filetype =slink test.zip***
socket	Include/exclude sockets. These are the items that the command ls -l lists with an "s" at the beginning of the permissions.	***pkzipc -add -filetype =socket test.zip***
none	Exclude all file types except for those specified on the command line. List file types to include after the *none* sub-option. For example, to include only pipe files: *-filetype=none,pipe* **Note**: If you use the *none* sub-option without listing a filetype to include, all files will be included.	***pkzipc -config -filetype =none,slink***
all	Include/exclude all file types.	***pkzipc -add -filetype=all test.zip***

Following Links

links

PKZIP allows you to follow the UNIX links of a file when compressing files by using the ***links*** option.

Note: When following links using the ***links*** option, the resulting .ZIP archive will be larger since two copies of the file data are compressed as though each link is a separate file. You must also use the ***filetype*** option with the ***links*** command.

Use the ***links*** option with the following sub-options to process specific file types:

Sub-Option	*To*	*For example*
slink	Symbolic links will be stored (followed) rather than preserved.	***pkzipc -add -links=slink save.zip***
hlink	Hard links will be stored (followed) rather than preserved.	***pkzipc -add -links=hlink save.zip***
none	Symbolic and hard links will be preserved (rather than stored).	***pkzipc -add -filetype =hlink -links=none save.zip***
all	Symbolic and hard links will be stored (followed).	***pkzipc -add -links=all save.zip***

Extended Attribute Storage

noextended

When PKZIP adds files to an archive, PKZIP stores the standard FAT file system attributes (Read-Only, Archive, System, Hidden, Directory). By default, various extended attributes are stored as well. These include NTFS times on Windows and userid, groupid, and UNIX times on UNIX. The extended attribute timestamps are more accurate than the DOS modification time, but you can slightly reduce the size of an archive by omitting this extended attribute information.

To exclude extended attribute information, use the ***noextended*** option, as in the following example:

> ***pkzipc -add -noextended test.zip readme.doc***

Note: The ***noextended*** option does not affect storage of the offline, temporary, and system attributes on DOS systems, or storage of filetype attributes on UNIX systems.

Extended Attributes and the OS

Extended attributes are automatically added to .ZIP archives when they are created. PKZIP does not display a message indicating that it is saving extended attributes.

PKZIP running on a UNIX system stores different extended attributes than PKZIP running on a Win32 system. The following table lists the extended attributes that PKZIP stores relative to the UNIX and Win32 operating systems:

UNIX	*Win32*
user ID	create time
group ID	last modification time.
last modification time	last access time.
last access time	.
link information	

Whether PKZIP overwrites existing files, directories and extended attributes with those stored in the archive when extracting depends on your file system privileges and the options and sub-options you use.

Extended Attributes and 204g Compatibility

204

By default, PKZIP does not enable PKZIP for DOS 2.04g compatibility. When 204g compatibility is enabled, extended attribute data is stored in both the Local header and Central header records. This will result in a slightly larger .ZIP file size, but improves the chance that extended attribute information can be recovered if the .ZIP file should become damaged. It also ensures the extended attribute information is always retained if the file is generated with a version of PKZIP other than 2.04g. This option is ignored when extracting. The ***204*** option also limits the number of files that can be added to a .ZIP archive to 16,383. To enable 204g compatibility, use the ***204*** option as in the following example:

> ***pkzipc -add -204 test.zip ****

Including Additional Information in a ZIP File

Text comments are useful for recording notes about the files in an archive. You can attach different comments to different files, or you can select several files and attach the same comment to all of them. You can also save comments to text files and load previously saved comments to use again.

You can include a:

- Text comment
- Header comment
- Date for the .ZIP file (other than the creation date)

Refer to the sections that follow for more information.

Including a Text Comment

comment

With PKZIP, you can include a comment for the individual files within a .ZIP file. There are several options for adding comments to your .ZIP files. To include a comment, use the ***comment*** option alone or with the ***add*** command. When you run the command, PKZIP prompts you to enter the comment.

The table below lists the available sub-options for adding comments to your .ZIP archives:

Sub-Option	*To*	*For example*

all	Comment all of the files and any new files added.	***pkzipc -add -comment=all test.zip **
unchanged	Comment only files existing in the ZIP file that are not either updated or being added.	***pkzipc -add -comment=unchanged test.zip **
add	Comment all files added.	***pkzipc -add -comment=add test.zip **
none	Disable the comment option.	***pkzipc -add -comment=none test.zip **
freshen	Comment all of the files updated in the ZIP file.	***pkzipc -add -comment=freshen test.zip **
update	Comment all files added and updated in the zip file.	***pkzipc -add -comment=update test.zip **

Note: Comment length is limited to 59 characters.

Including a Header Comment

header

With PKZIP, you can include a general comment for a .ZIP file. This is called a "header" comment because it appears in the header portion of a .ZIP file. This differs from the ***comment*** option in that the "header" comment applies to the entire .ZIP file, not to individual files within the .ZIP file.

Headers for .ZIP files are limited to 16K in size. PKZIP truncates headers larger than 16K.

To include a header comment, use the ***header*** option with the ***add*** command. PKZIP provides several ways to specify the comment. You can enter the comment with the ***header*** option, or you can specify a file that contains the comment.

To include the comment in the command line, specify the comment as a value for the ***header*** option. Enclose the comment text in quotes if the text includes spaces. For example:

***pkzipc -add -header="This is the comment" test.zip ***

If you include the ***header*** option alone, without a value, PKZIP prompts you for text to use, as follows:

```
Zip Header ?
```

Type your header comment and press ENTER.

To use header text from a file, specify the file name (and path, if necessary) as a value for the ***header*** option. Prefix the file name with the list character (@). Put the file name in quotes if it contains spaces. For example:

With this method, you type the ***header=@filename.ext*** option. If there are no spaces in the file name, it is not necessary to use quotation marks. For example:

***pkzipc -add -header=@header.txt test.zip ***
***pkzipc -add -header=@"my header.txt" test.zip ***

Note: You can also use ***header*** to add a comment to OpenPGP files wrapped in ASCII Armor (see "Encoding an Archive to another Type"). The comment is displayed when viewing, testing or extracting the archive.

Specifying the Date of a .ZIP File

archivedate

When you create an archive file, PKZIP gives it the current date by default. You can specify a different date for the file by using the ***archivedate*** option with the ***add*** command.

PKZIP provides several methods for applying a date to an archive file. The table below lists the available sub-options for applying date information to your archives:

Sub-Option	*To use*	*For example*
retain	The date on which the archive file was created.	***pkzipc -add=update -archivedate=retain test.zip **
none (Default)	The current date.	***pkzipc -add -archivedate=none test.zip **
oldest	The date of the oldest file within the archive file.	***pkzipc -add -archivedate=oldest test.zip **
newest	The date of the newest file within the archive file.	***pkzipc -add -archivedate=newest test.zip **

Removing File Attributes

mask

The ***mask*** option specifies a permissions mask for files to be added or extracted. The mask specifies permissions which should *not* be archived or restored on extraction.

On extraction, the ***mask*** option can be used with the ***permission*** option (configured or given on the command line) to explicitly strip permissions specified by that option. (The *setuid*, *setgid*, and *sticky* bits are set on extracted files only if the ***permission*** option is used.)

Use an octal value to specify a permissions mask for the ***mask*** option. For example, the following command line masks write permission for group:

pkzipc -add -mask=20 myfiles.zip

Sorting Files Within a .ZIP File

sort

With PKZIP, you can sort the files in an archive in several ways. If you do not change the sort order, the files are automatically sorted in the order in which they were compressed into the archive. This is called the "natural" order.

The ***sort*** option works with ***add***, ***extract***, ***test***, and ***view***. The value you include with ***sort*** depends on the command you select.

Sub-Option	*To sort by*	*For example*
date	File date.	***pkzipc -add -sort=date temp.zip***
size	Original uncompressed size of the file ("length" in display).	***pkzipc -add -sort=size temp.zip***
extension	File extension.	***pkzipc -add -sort=ext temp.zip***
name	Sorts files and folders by name in a single series. (Contrast with -sort=none.)	***pkzipc -add -sort=name temp.zip***
none	Groups folders first, sorted by name, and then groups files, sorted by name. (The default.)	***pkzipc -view -sort=none temp.zip***
natural	Preserves the order in which files were added to an archive.	***pkzipc -view -sort=natural temp.zip***
ratio	Ratio of uncompressed size to compressed size.	***pkzipc -view -sort=ratio temp.zip*** **Note**: The ratio sub-option will not work with the add command.
crc	CRC (Cyclic Redundancy Check) number.	***pkzipc -view -sort=crc temp.zip*** **Note**: The crc sub-option will not work with the add command.
comment	File comment.	***pkzipc -view -sort=comment temp.zip*** **Note**: The comment sub-option will not work with the add command.

The ***name*** sub-option sorts entire path names; it does not sort file names directly if folder information is present.

For example, the ***name*** sub-option sorts the two files *abacus.txt* and *zebra.txt* as follows if they are added to an archive without including any path or folder information:

```
abacus.txt
zebra.txt
```

However, if the files are added with folder information, the name of the outermost folder in the path determines their order of appearance. This is because ***name*** sorts the entire path name whether or not it includes folder names. For example:

```
all\junk\zebra.txt
everything\important\abacus.txt
```

By contrast, the ***none*** sub-option groups path names that contain folder names and sorts this group in a separate series from file names that do not include folder information. The names below are sorted by ***none***:

```
all\junk\zebra.txt
everything\important\abacus.txt
anotherfile.txt
lonefile.doc
somepix.gif
```

If no ***sort*** option is specified, files are sorted as if ***sort=none*** was specified (unless you have changed configuration defaults).

If you specify the ***sort*** option on your command line but do not specify a sub-option value, the ***name*** sub-option is applied.

Note: Using the ***sort*** option with the ***add*** command only works on new archive files. It does not work with an archive that is being updated.

Moving Files to a .ZIP File

move

Normally, when you compress files, you end up with two copies of each file: the original file and the compressed file. With PKZIP, you can choose to remove the original file "after" you compress it into the .ZIP file.

If you want to move only specific files, you must compress them separately since you can only move all or none of the files that you are compressing.

To move files, use the ***move*** option with the ***add*** command, as shown below:

> ***pkzipc -add -move test.zip *.doc***

This sample command line tells PKZIP to compress and add to archive test.zip all files that end in .doc and then to delete the original files.

CAUTION: Like any operation that deletes files, the ***move*** option should be used with care.

Shredding Deleted Files

shred
A deleted file still remains on your disk and can often be fully or partly recovered. So can the temporary files that PKZIP creates when updating an archive. To erase these files to prevent information from being retrieved from them, use the ***shred*** option with the ***add*** command. Shredding a file overwrites the file's data so that it cannot be read.

Shredding overwrites these files:

- Deleted originals that have been moved into an archive with the ***move*** option
- Temporary files that contain the previous version of an archive that has just been updated

Note that overwriting files with the ***shred*** option takes some additional time.

Shredding can overwrite files only if the file system applies the overwriting to the same physical disk sectors that the file to be overwritten used. Most UNIX and Linux file systems do not do this. For this reason, shredding works most reliably on Windows.

Shredding has a couple of other constraints:

- Files on the Windows NTFS file system that have been encrypted or compressed by NTFS itself have a special NTFS attribute. PKZIP cannot shred these files.
- The system temporary folder must be local; it cannot be on a removable or network drive for shredding to work. PKZIP can delete files that are on a removable or network drive but cannot shred them.

The ***shred*** option has these sub-options:

Sub-Option	*Description*
None	Turns shredding off if it is configured on
Random	Overwrites files once with random data (the default)
Dod5220	Overwrites files three times, to the DOD 5220.22-M specification
NSA	Overwrites files seven times, to the NSA standard. (Takes much longer.)

For example:

> ***pkzipc -add -move -cryptalgorithm -passphrase -shred=NSA secret.zip *.****

Working with Self-Extracting (PKSFX) Archives

sfx

If you have the PKZIP Self-Extractor add-on, you can use PKZIP to create PKSFX archives. A PKSFX archive is self-extracting: it has an .exe file name extension (instead of .zip, for instance), and it can be extracted just by executing it, even by someone who does not have PKZIP or another ZIP utility. (PKSFX archives are also called self-extractors or SFX files, for short.)

Note: You must have PKZIP Enterprise or SecureZIP to create a PKSFX archive.

You can create self-extractors of two general types:

- A native command line self-extractor for use in the command line environment of the operating system on which PKZIP is running. The native command line self-extractor extracts without using any graphical user-interface features such as dialog boxes.
- A graphical 32-bit Windows self-extractor for use in the graphical Windows environment. When run, a graphical Windows self-extractor opens a dialog that contains controls to view progress or set options for extracting files.

To create a self-extracting archive, use the ***sfx*** option with the ***add*** command. For example, the following line creates a native command line self-extractor `mysfx.exe`:

pkzipc -add -sfx mysfx *.doc

When used without a sub-option, the ***sfx*** option creates a native command line self-extractor by default.

Use the ***listsfxtypes*** command to list ***sfx*** sub-options for the types of self-extractors available to you. The exact types vary with your system and license. For example, the following command

pkzipc -listsfxtypes

may produce a display like this on a Windows system:

The SFX sub-option choices are:

```
AIX5X_PPC_C1230 - V12.30 Command Line SFX for AIX on PPC
DOSJR_X86_C250 - 2.04g compatible SFX Junior for DOS
DOS_X86_C250 - 2.04g compatible SFX for DOS
HPUX_ITA_C1230 - V12.30 Command Line SFX for HP-UX on Itanium
LNX2X_X86_C1230 - V12.30 Command Line SFX for Linux on X86
SOL2X_SPC_C1230 - V12.30 Command Line SFX for Solaris on SPARC
WIN32_X86_C1230 - V12.30 Command Line SFX for Windows on X86
WIN32_X86_G1230 - V12.30 Windows SFX for Windows on X86
```

In the list above, ***win32_x86_c...*** designates the native Windows command line self-extractor, and ***win32_x86_g...*** designates the graphical Windows self-extractor. The digits at the end give the version number.

To create a graphical Windows self-extractor, use the ***sfx*** option with the ***win32_x86_g1230*** sub-option. For example:

pkzipc -add -sfx=win32_x86_g1230 mysfx *.doc

You only need to enter enough of the name of an SFX type to uniquely identify it; you can leave off the version number at the end:

pkzipc -add -sfx=win32_x86_g mysfx *.doc

You can also use ***sfx*** as a command to convert an existing, ordinary ZIP file to a self-extractor. To do so, use the ***sfx*** command by itself on the command line, without the ***add*** command, and specify the ZIP file to convert. For example:

pkzipc -sfx=win32_x86_g1230 myfiles.zip

Notes:

- You cannot use the ***sfx*** option with the ***cd*** option to create or convert an archive with encrypted file names
- The ***sfx*** command can only convert ZIP archives that are physical files. It cannot convert ZIP archives that are special files (named pipes, sockets) or are presented from STDIN.

Setting the PKSFXSDATA Environment Variable

PKZIP requires the file pksfxs.dat to create PKSFX files. Ordinarily, PKZIP searches for this file where it is installed by default, namely, in the directory with the PKZIPC executable.

If you want to keep pksfxs.dat in a different location, you can set the environment variable PKSFXSDATA to tell PKZIP where to find the file. PKZIP searches for the file first on the path set in the environment variable, second on the current path, and last on a path specified on the command line.

To set the PKSFXSDATA environment variable, do the following:

1. Using a text editor such as vi, Pico, Kate, or emacs, open your start-up file.
2. What you do next depends on the shell you are using:

- If you are using the Korn Shell (***ksh***) or the Bourne Shell (***sh***), add the following lines to your .profile file:

  PKSFXSDATA=<path to the pksfxs.dat file>
  export PKSFXSDATA

- If you are using the C Shell (***csh***), add the following line to your .login file:

  setenv PKSFXSDATA <path to the pksfxs.dat file>

Save and exit the file.

To reset your current environment settings, log off your account. The PKSFXSDATA variable will be set the next time you log on to your account.

Converting a Standard Archive to a Self-Extractor

To convert a standard ZIP file to a self-extracting archive, use the ***sfx*** *command*, without the ***add*** command.

For example, the following command line converts standard archive test.zip to self-extractor test.exe. PKZIP replaces zip in the file name with exe.

pkzipc -sfx test.zip

Converting to a Self-Extractor with a Different Name

Ordinarily, when you use the ***sfx*** command to convert a standard archive to a self-extracting archive, the archive keeps its original name except for the extension, which PKZIP changes from zip to exe. To give an archive a different name, use the ***namesfx*** option to specify a new name when you convert the archive:

pkzipc -sfx -namesfx=test123.exe test.zip

If you omit the .exe in the new name, PKZIP supplies it.

Note: You cannot use the ***sfx*** option with the ***cd*** option to create or convert an archive with encrypted file names.

Options for Creating Self-Extractors

You can use the following options together with the ***sfx*** command/option to customize a self-extractor in various ways when you create it. The options are described in the following sections. Default values for all the options can be configured with the ***configuration*** command.

As indicated in the table below, some of the options require a GUI self-extractor and do not work with command line self-extractors.

Option	*Works only with GUI Self-Extractors*
SFXDestination	X
SFXDirectories	X
SFXLogfile	
SFXOverwrite	X
SFXUIType	X
RunAfter	

SFXDestination

The ***SFXDestination*** option specifies a default target folder for extracted files. For example:

pkzipc -add -sfx=win32_x86_g -sfxdestination="My Documents\newstuff" mysfx *.doc

If no drive letter is listed in the path, the self-extractor chooses the drive that contains the temporary folder and appends the path to the temporary folder.

If the specified destination folder or path does not exist, the self-extractor prompts the user whether to create it.

The ***SFXDestination*** option works only with a GUI self-extractor.

SFXDirectories

The ***SFXDirectories*** option causes the self-extractor to restore saved directory paths on extraction. To recurse subdirectories and save path information (relative to the current directory) when you add files to a self-extractor, use the ***directories*** option.

For example, the following command line archives the docs folder and all its files and subfolders. The docs folder and the saved subfolders are restored on extraction.

pkzipc -add -sfx=win32_x86_g -sfxdirectories -directories mysfx "docs*.*"

The ***SFXDirectories*** option works only with a GUI self-extractor.

SFXLogfile

The ***SFXLogfile*** option creates an ASCII text SFX error log named pkerrlog.txt in the destination directory on extraction.

pkzipc -add -sfx -sfxlogfile test.exe *.doc

SFXOverwrite

The ***SFXOverwrite*** option specifies when the self-extractor overwrites files that have the same name as a file being extracted. The option has the sub-options listed in the table below.

Sub-option	*Description*
prompt	(Default) The user is asked whether to overwrite files
always	Files that have the same name in the destination folders are overwritten without prompting
update	Only files that do not already exist or are newer than same-named files
freshen	Only newer versions of files that already exist in the destination folders are extracted; the older files are overwritten without prompting
never	Files are never overwritten

For example:

pkzipc -add -sfx=win32_x86_g -sfxoverwrite=freshen mysfx *.doc

The ***SFXOverwrite*** option works only with a GUI self-extractor.

SFXUIType

The ***SFXUIType*** option specifies the type of graphical interface that the self-extractor presents to the user. This option only affects GUI self-extractors. (Command line self-extractors do not present a GUI.) The option has the sub-options listed in the table below.

Note: Some interfaces do not support the full range of self-extracting options. You may want to experiment with sub-options before distributing your self-extracting archive.

Sub-option	*Description*
AutoSFX	Presents a dialog that displays a bar to show progress extracting, and a Cancel button. Supports adding a title and creating subfolders.
EasySFX	(Default) Presents a dialog that enables the user to select a destination folder and to turn off any **runafter** option set. (See "Run Programs with the Self-Extractor," below.) Supports adding a title, creating subfolders, and setting overwrite options.
Regular SFX	Presents a dialog that enables the user to change the destination folder and other options before the archive is extracted. Supports all PKZFX options.

For example:

pkzipc -add -sfx=win32_x86_g -sfxuitype=regularsfx mysfx *.doc

Run Programs with the Self-Extractor

Use the ***runafter*** option with the ***sfx*** option to create a self-extracting archive that runs a program after the self-extractor is run. This option enables you to create a self-extractor that runs a script or opens a file after the contents of the self-extractor are extracted.

The ***runafter*** option does not work with the following types of self-extractors:

- DOSJR_X86_C250 - 2.04g compatible SFX Junior for DOS
- DOS_X86_C250 - 2.04g compatible SFX for DOS

Use the ***listsfxtypes*** command to list the types of self-extractors available to you:

pkzipc -listsfxtypes

Here are examples showing uses of the ***runafter*** option.

Create a self-extractor to open a readme.txt file after extraction:

pkzipc -add -sfx -runafter="notepad.exe readme.txt" test.exe *

Create a self-extractor to open a file by means of its associated application:

pkzipc -add -sfx -runafter ="${}readme.txt" test.exe *

Create a self-extractor to run an install script:

pkzipc -add -sfx -runafter ="${install}install.inf" test.exe *

Create a self-extractor to run an install script, with the full path prepended (%0):

pkzipc -add -sfx -runafter ="${install}%0install.inf" test.exe *

Extraction Options for the Native Self-Extractor

To extract files from a self-extracting archive, you run the archive. For example, to extract files from self-extractor test.exe, use the following command line:

test.exe

Note: When extracting encrypted files on UNIX systems from a self-extracting archive, you may encounter a "Recipient not found" error message. This results from a change in the certificates database file in SecureZIP Server version 14.0. You should be able to extract the file as an ordinary ZIP archive using the ***noarchiveextension*** command:

pkzipc -extract -noarchiveextension test.exe

When you run a native command line self-extractor, you can use the command line options listed below. The options can be used only with a native self-extractor; they cannot be used with a Windows graphical self-extractor:

after	keypassphrase	silent
before	larger	smaller
console	license	sort
directories	lowercase	test
exclude	mask	times
extract	more	translate
filetype	newer	version
fipsmode	older	warning
help	overwrite	
id	passphrase	
include	permission	

For example, the following command line excludes all text (.txt) files from the set of files to be extracted:

test.exe -exclude="*.txt"