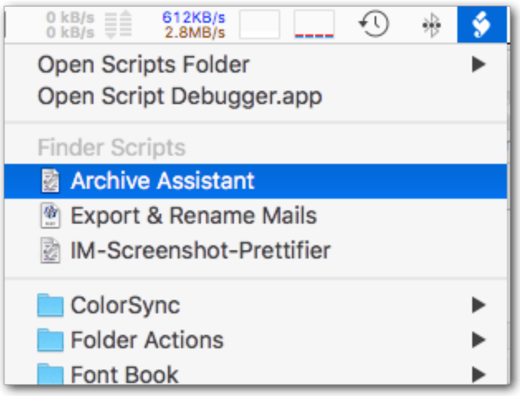
Archive Assistant

(AppleScript)

[Manual updated: 2017-04-23]

Installation

The usual way to install AppleScripts is to drop them into the User Scripts Folder (~/Library/scripts). The script will sit unobtrusively in the AppleScript menu:



(If not already done you might have to activate the AppleScript menu in the preferences of the Script Editor application.)

You may also launch the script with any launcher of your choice (LaunchBar, FastScripts, Keyboard Maestro, Alfred, etc.). These launchers allow you to assign a shortcut or an abbreviation to the script.

What the Script does and how it works

This AppleScript provides an easy and fast way to create a wide variety of compressed and uncompressed archive formats on your Mac. It is not an unarchiver.

Although easy to use this script is not intended for the novice user who is not familiar with different archive formats to a certain degree. For users who have no need for advanced archiving/compression I highly recommend to use the Finder's *Compress* in the contextual menu. This delivers metadata-friendly and universally compatible archives, although at a low compression ratio.

The script was written to comply with the following functional specifications:

- Efficient archiving while preserving all Mac metadata (all extended attributes)
- Secure encryption solutions (only AES, no weak zip encryption)
- At least one efficient solution for metadata-free archiving (transfer to Windows users)
- Careful selection of the most sensical format options
- Easy access to useful but little-known formats or features (e.g. *xar,* shadowed *dmg*)

- For compatibility: Access to all traditional / widely used formats, non-regarding their efficiency (e.g. *zip, gz, bz2*)
- Assistive functions (e.g. the script will only offer archive types that make sense for the selected items) and easy usage

The script puts an emphasis on the preservation of Mac metadata in the archives, i.e. Spotlight comments, openMeta tags, Finder labels etc. Methods that will not preserve the entirety of metadata are explicitly marked with *No Metadata*.

Usage of this AppleScript is simple:

1. Select the file(s)/folder(s) to archive in the Finder.
2. Launch the script.
3. Set archive type and options.
4. Wait until the archive is completed.

Just to be clear: This script does nothing that you couldn't also do on the command line. It's a mere "comfort" script: It puts the most useful formats and switches at your fingertips and provides a GUI-like access to file, options and destination selection.

Nevertheless it offers more flexibility and *sensical* archive options than most full-fledged GUI apps I know.

Some of the supported archive types, compression methods and features:

7z, lzma2, cpio, tar, cpio.gz, cpio.bz2, cpio.xz, zip, xar, compressed *dmg, sparse bundle; dmg* shadowing, resizing, conversion, verification, *sparse bundle* compacting, …

Step by step

1. Select one or more files or folders in a Finder window.

2. Launch the Script from the AppleScript menu or from any other script launcher.

3. The first window "Archive Type" will pop up.

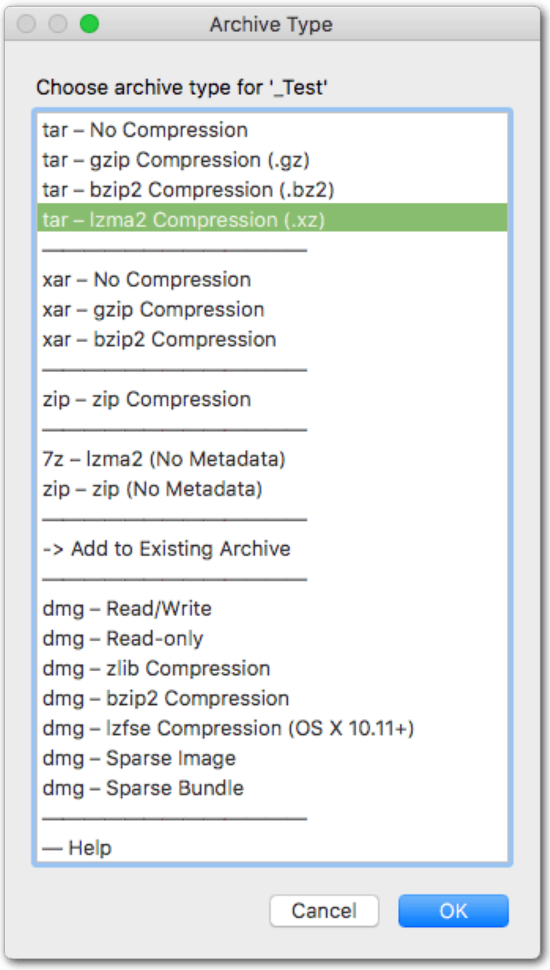
Make your choice. In the upper part of the list you have all kinds of "conventional" types of archives, in the lower part you find all sorts of Disk Images *(dmg)*. In-between of the two groups you'll see a special entry "Add to Existing Archive". More on this later.

You can only select one archive type. Clicking *Help* will open this document. Click *OK* to proceed.

4. The next window is the Options Window.

Its contents depend on the archive type you have chosen before. The main options are the various compression levels for compressed archives. The other option – if available for the archive type – is encryption.

If you don't select a compression level the scripts will default to *Normal* (*5*).


Archive Type

Choose archive type for '_Test'

tar – No Compression
tar – gzip Compression (.gz)
tar – bzip2 Compression (.bz2)
[✓ tar – lzma2 Compression (.xz)]

xar – No Compression
xar – gzip Compression
xar – bzip2 Compression

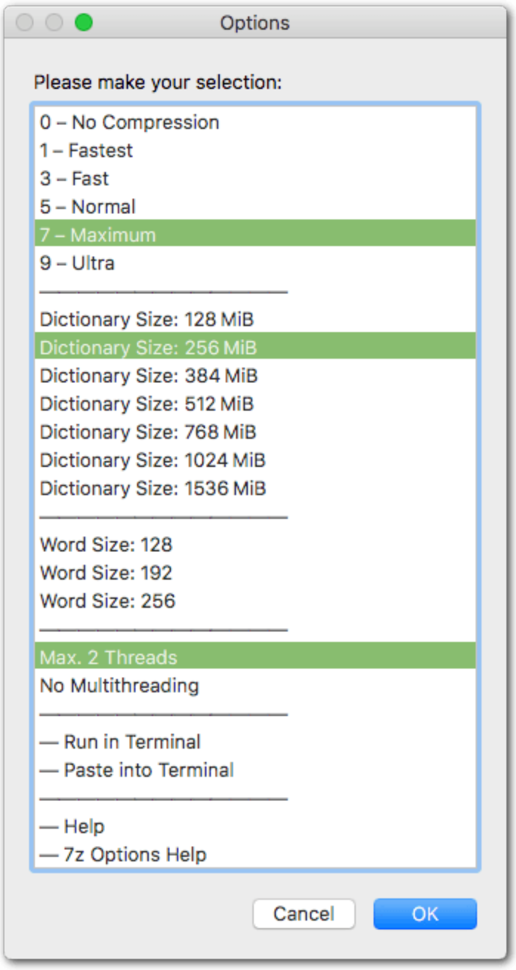
zip – zip Compression

7z – lzma2 (No Metadata)
zip – zip (No Metadata)

-> Add to Existing Archive

dmg – Read/Write
dmg – Read-only
dmg – zlib Compression
dmg – bzip2 Compression
dmg – lzfse Compression (OS X 10.11+)
dmg – Sparse Image
dmg – Sparse Bundle

— Help


Options

Please make your selection:

0 – No Compression
1 – Fastest
3 – Fast
5 – Normal
[✓ 7 – Maximum]
9 – Ultra

Dictionary Size: 128 MiB
[✓ Dictionary Size: 256 MiB]
Dictionary Size: 384 MiB
Dictionary Size: 512 MiB
Dictionary Size: 768 MiB
Dictionary Size: 1024 MiB
Dictionary Size: 1536 MiB

Word Size: 128
Word Size: 192
Word Size: 256

[✓ Max. 2 Threads]
No Multithreading

— Run in Terminal
— Paste into Terminal

— Help
— 7z Options Help

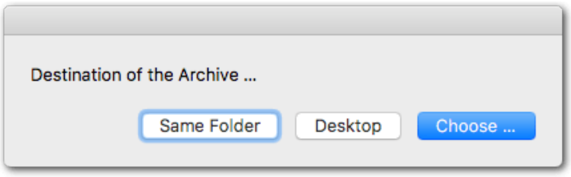
In the Options window you can make multiple selections with the Command (⌘) key pressed, that is, you can select a compression level *and* the encryption option. Of course, it doesn't make sense to select two compression level settings at the same time. At the bottom you have the run options, e.g. *Run in Terminal*. More on this later.

Help opens this document. The second Help options opens an archive-specific help document.

Click *OK* after you have selected your options.

5. The third and final window will ask you for the destination of the archive.

You can select between *Desktop*, *Same Folder* (i.e. the same location as the source file/folder) and *Choose …*. The latter opens the OS's folder selection dialog.


Destination of the Archive ...

Some archive types will pop-up an additional window, for example *dmg* formats will ask you for the size of the image.

You can cancel the script in any dialog window by clicking *Cancel.* To cancel the destination dialog click *Choose…* and then *Cancel.*

Note: If an archive with the same path & name already exists, and the archive type supports it (for example *tar, 7z, zip*), then the archive will automatically be updated. That is, the selected files/folders will be appended, or files of the same name inside the existing archive will be replaced. (Similar as with "Add to Existing Archive".)
So, if you want to create a new archive, make sure that an archive with the same name does not exist at the chosen path!

Explained

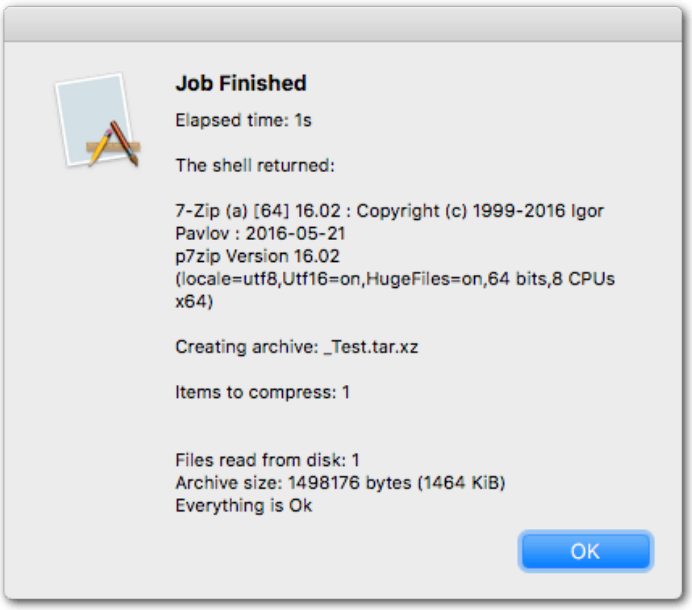
Add to Existing Archive

This special entry in the first window (Archive Type List) lets you append items to an already existing archive. In a dialog you will be asked to select the existing archive. You can only append to *uncompressed* unix-type archives *(tar, cpio, pax), zip* archives and *7z* archives.

Adding files to writable *dmg* formats can be easily done in the Finder, w/o any third-party tool, by simply dragging and dropping to the mounted image.

Run in Terminal vs normal mode

In **normal mode** (none of the terminal options is selected) the archiving scripts are run in the background. You won't get disturbed (and you won't get any progress report) until the



Job Finished

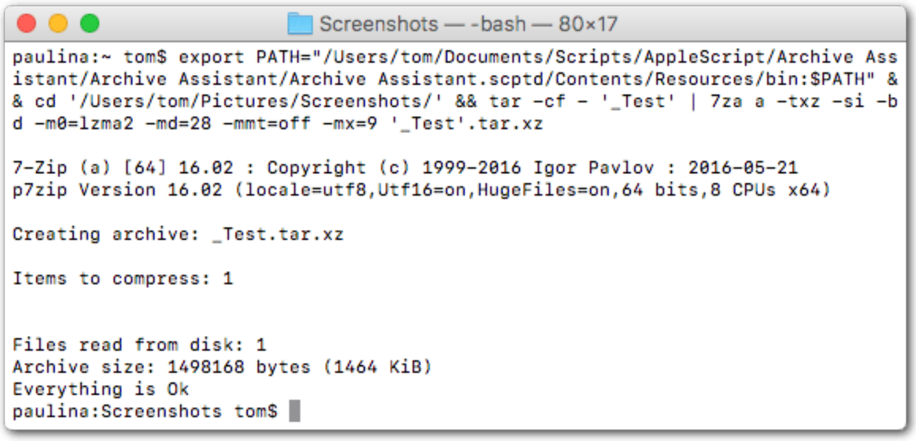
Elapsed time: 1s

The shell returned:

```
7-Zip (a) [64] 16.02 : Copyright (c) 1999-2016 Igor
Pavlov : 2016-05-21
p7zip Version 16.02
(locale=utf8,Utf16=on,HugeFiles=on,64 bits,8 CPUs
x64)

Creating archive: _Test.tar.xz

Items to compress: 1


Files read from disk: 1
Archive size: 1498176 bytes (1464 KiB)
Everything is Ok
```

completion window pops up when the job is finished. The completion window will show you the shell output if available.

If you select **Run in Terminal** the archiving scripts will run in a terminal window. This is less "elegant" than normal mode, but it provides you with live feedback on the archiving progress. You may choose this option when archiving larger amounts of data that will take a while. This way you can see the progress and you'll also see immediately if something goes wrong.



```
paulina:~ tom$ export PATH="/Users/tom/Documents/Scripts/AppleScript/Archive Ass
istant/Archive Assistant/Archive Assistant.scptd/Contents/Resources/bin:$PATH" &
& cd '/Users/tom/Pictures/Screenshots/' && tar -cf - '_Test' | 7za a -txz -si -b
d -m0=lzma2 -md=28 -mmt=off -mx=9 '_Test'.tar.xz

7-Zip (a) [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=utf8,Utf16=on,HugeFiles=on,64 bits,8 CPUs x64)

Creating archive: _Test.tar.xz

Items to compress: 1


Files read from disk: 1
Archive size: 1498168 bytes (1464 KiB)
Everything is Ok
paulina:Screenshots tom$ 
```

Another reason for choosing *Run in Terminal* could be the passphrase entry for encrypted archives. As the passphrase is directly entered in the terminal (w/o "passing" through the AppleScript) there is less risk of leaving traces of the passphrase on shared computers.

Paste into Terminal is like *Run in Terminal* but it doesn't start the archive script. It just pastes the complete precomposed command line to the terminal. This is useful if you want to do a one-time modification to the command line, e.g. adding a '-mmf=hc4' switch to do a lzma compression with the slower but less memory hungry hc4 match finder. To actually run the script just press Return.

You may also use this option if you just want to see what the command looks like.

Metadata

All archive types that are not annotated with *No Metadata* will fully preserve Mac metadata (spotlight comments, openmeta tags,"resource fork", any other extended attribute, ACLs). Restoring of ACLs depends of the unarchiver application.

Of course, if your files are already "metadata-safely" archived inside of a *tar, cpio, dmg* etc. you can safely further compress the archive/image with *7z – No Metadata* without loosing the metadata of your files. What you will loose in this case is only the metadata of the archive/image file itself. *However this is not recommended because with a tar.7z archive you loose the possibility to browse the archive without expanding it completely*.

Important: Xar archives must be extracted with 'xar -x' in order to restore metadata! 3rd-party extractors (e.g. 7z, The Unarchiver, Pacifist, BetterZip) will discard metadata in xar archives (they don't handle the TOC.xml).

No-metadata archives

Archive types annotated with *No Metadata* strip off all Mac metadata and are best suited for one-way transfer to Windows users. Since Windows users often are computer-illiterate, the most common archive type *zip (No Metadata)* is the most safe and thus preferred type for this purpose. If higher compression ratio is absolutely required try *7z (No Metadata).*

Avoid using *deflate64-*, *bzip2-* etc. compressed *zip* archives for this purpose. Also do not use any of the unix-like archive types (*tar, cpio, xz* etc.) or *dmg* images. Most Windows users won't be able to open them.

POSIX archive types (cpio, tar, pax)

Cpio and *tar* are great ways on OS X to secure all file metadata before further compression. Not every *cpio* or *tar* implementation is hfs metadata aware. This script uses OS X's *tar (bsdtar)* program to create *tar* archives.

Important: Although the metadata is correctly stored inside the *cpio* or *tar* archive, it is still possible that it will be discarded at extraction time by a metadata-unaware unpacking program. Most unpackers on OS X should be safe; I've tested it successfully with *pax, ditto, tar, Archive Utility,The Unarchiver, The Archive Browser*.

cpio or tar?

The traditional advantage of *tar (ustar)* is that hard-linked files are detected and not redundantly copied into the archive. However us*tar* has a severe file-name-length limitation (255 chars for the fullpath or 100 chars for the name part). *Cpio* doesn't have that limitation, but it is unable to handle hard-linked files properly.

The *tar (bsdtar)* that ships with macOS does accept long file names *and* can handle hard links. That's why it is proposed as default archiving tool in this script.

In theory the newer *pax* format should also combine the advantages of *cpio* and *tar*. Strangely the *pax* format isn't supported by OS X's *pax* (you can set *pax* as format with '-x pax' but this generates just a *ustar* file (with the file name length limitation) with a *pax* extension). The *cpio* utility on OS X supports the *pax* format, but it isn't metadata aware.

For *gzip*, *bzip2* and *lzma2 (xz)* compression of Posix archives the script uses *7z* instead of the usual unix tools because of its multithreading capabilities *(bz2, xz)* and better compression ratios *(gz)*. Multithreading actually makes a huge difference here.

As *gzip*, *bzip2* and *xz* don't offer encryption I've added the possibility to create encryptable *tar.7z* archives.[1] However if you don't need encryption you should stay with *tar.gz*, *tar.bz2* or *tar.xz*, because these formats will allow you to browse the archived files without unwrapping the archive.

[1] Currently this format is disabled because the presence of two *lzma2* formats *(.xz, .7z)* caused confusion among some users. To re-enable it just copy the commented line in the script back into the list section.

xar

Xar is one of the younger formats. It is metadata safe by default and seems to be used also by Apple's installers. For restoring metadata from *xar* archives please see the section *Metadata*.

Xar is not a bad choice especially for the backup of complex structures because it copies hard-linked files the same way as *tar* (non-redundantly) and additionally detects duplicate files; that is, files with the same data but different names, modification dates or paths will be archived just once, while their multiple instances will still be indexed in the archive header. Always expand *xar* archives with 'xar -xf' on the command line. Some GUI unarchivers are not capable of correctly extracting hard-linked files!

Note: Some time ago I noticed that when unpacking a *xar* archive that contains hard-linked files *xar* (the command line tool) reports a checksum error and doesn't extract all files. **[Update 2017:]** The checksum error is still reported but all files are extracted correctly with 'xar -xf'. However: When working with hard-linked files, double-check if *xar* behaves correctly on your system! >>> *This note refers to extracting on the command line; GUI unarchivers still fail to extract hard-linked files!* <<<

zip/gzip

The *zip* option (via ditto) is almost identical to the Finder's context menu *Compress*. The difference is that you can choose the compression level and that you can choose the output directory. On the other hand, Finder's *Compress* allows you to gather multiple selections into the archive.

The option *zip – No Metadata* is performed by *7z*. *7z's zip* implementation achieves better compression ratios and offers different compression level settings as *zlib* or *ditto*. This is also true for *7z's gzip*. The available compression level settings for *zip – No Metadata* (e.g. *7 – Maximum*) correspond to the predefined settings of the *7z* program.

The script doesn't offer the option to make *deflate64*-compressed *zip* archives because this would defeat the main purpose of the presence of *zip* in this script (maximum platform and user compatibility). If you aim for better compression just choose *bzip2* or *lzma2*.

bzip2

This is a traditional compression method for unix archives that is slower but compresses better than *gzip*, although far from reaching the compression ratio of *lzma2 (7z, xz)*. *bzip2* compression of *cpio* archives is done by *7z* as it has the best *bzip2* implementation IMO. The compression level settings (e.g. *7 – Maximum*) correspond to the predefined settings of the *7z* program.

The more commonly known method to create *bzip2* archives on the mac is 'ditto -cj'. However this is considerably slower (no multi-threading) and the compression ratio is not as good.

lzma2

Lzma2 compression (used in *7z, xz* archives) is the most efficient compression method that is easily available on the Mac.

As with the other compression methods *lzma2* can be configured in various ways. The compression level settings (e.g. *7 – Maximum*) correspond to the predefined settings of the *7z* program. Each of these settings represents a mix of various parameters put together in a meaningful way by the author of *7z*.

The most important parameters of the compression level settings are solid block size and dictionary size. Choose *7z Help* in the Options Dialog to see the parameter tables or click here.

Solid block size

Solid archives allow for a better compression ratio because the data of the individual files is merged together. By default solid blocks are enabled, with the block size depending on the compression level, e.g. 4GiB for level 7 and 9. You have two options to force a different behavior:

Non-solid: Each file is compressed individually. Compression ratio will be considerably lower, but in case of data corruption chances to recover files from the corrupted archive will be higher. The second advantage is that it takes less time to extract individual files from non-solid archives or to append files to a non-solid archive. With *Non-solid* set *7z* behaves like other programs that always compress files individually (e.g. *zip, xar*)

Solid by Filetype: Files with the same name extension are put together in a solid block. Let's see this as an intermediate solution between solid and non-solid. Interestingly sometimes *Solid by Filetype* yields a slightly better compression as solid (at lower dictionary sizes).

If you are making *lzma2*-compressed POSIX archives *(cpio.xz, tar.xz)* the solid settings are irrelevant as there is only one file to compress (the POSIX archive).

You can choose the solid-block behavior independently of the compression level.

Dictionary size

For most use cases the predefined compression level settings offer a good range of compromises between speed and compression ratio to choose from.

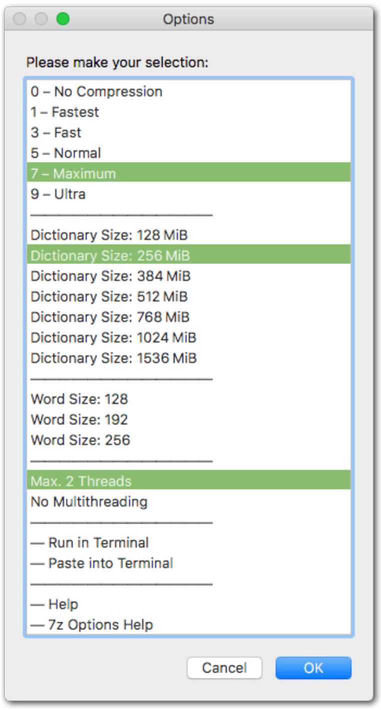
One exception is the dictionary size: The dictionary size maxes out at 64MiB at compression level 9 (Ultra). This size was obviously chosen by the author of *7z* in order to keep the amount of RAM needed for compression and decompression low. (With larger dictionaries compression slows down and RAM consumption increases dramatically. See *7z Help* for a table of dictionary sizes at predefined compression levels.)

If the data to compress exceeds 64MiB *and if* there is enough RAM for compression *and decompression* you will gain significantly better compression if you adapt the dictionary size to the amount of data. You can do this by forcing a higher dictionary size in the Options Dialog with the maximum being 1GiB.

For example, if you have 490MiB of data to compress set the dictionary size to 512MiB to achieve maximum compression. (Setting it to 1GiB doesn't yield any benefit for 490MiB of data.)

If you force large dictionaries keep in mind that the consumed RAM for compression will be about 11 times the dictionary size and the RAM needed for decompression will be equal to the dictionary size.

If you choose a forced dictionary size the compression level will be automatically set to 9, no matter which level is selected in the list.



Options

Please make your selection:

0 – No Compression
1 – Fastest
3 – Fast
5 – Normal
[✓ 7 – Maximum]
9 – Ultra

Dictionary Size: 128 MiB
[✓ Dictionary Size: 256 MiB]
Dictionary Size: 384 MiB
Dictionary Size: 512 MiB
Dictionary Size: 768 MiB
Dictionary Size: 1024 MiB
Dictionary Size: 1536 MiB

Word Size: 128
Word Size: 192
Word Size: 256

[✓ Max. 2 Threads]
No Multithreading

— Run in Terminal
— Paste into Terminal

— Help
— 7z Options Help

Cancel OK

As shown in the screenshot you can select one option from every group.

Word size

The third parameter you can explicitly set is the word size (fast bytes). If this has any effect on the compression ratio will heavily depend on the data structure. A bigger word size will always slow down the compression. In most cases you should be fine with not touching it. See the *7z Help* for more info.

If you choose a forced word length the compression level will be automatically set to 9.

This is an experimental feature. I still have to find a real-world usecase where a word size > 64MiB has a significant effect on the compression ratio.

Conclusion for best compression ratio

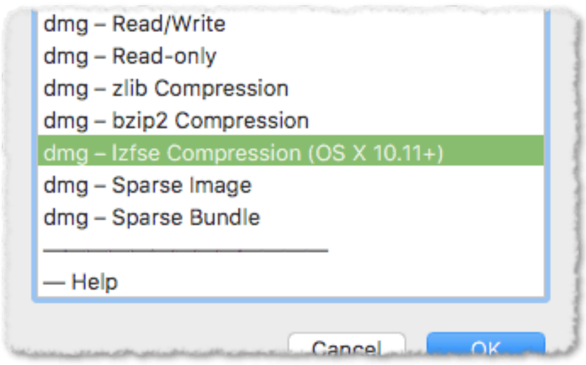
Choose *lzma2* compression (that is, *7z* or *xz*), compression level *9*, and set the dictionary size to at least the size of the data (but don't set it higher as necessary).

If you can't set the dictionary size to the size of your data (limited RAM or too much data) set it to the largest possible size and try *Solid by Filetype*, if you are compressing multiple files.

Play around with the word length setting, if you like. Keep in mind that at large dictionary sizes the compression will not run in multiple threads.

Disk Images

In a strict sense disk images are not archives. However disk images on OS X are an extremely versatile tool and can perfectly be used as archives. That's why I've added them to the script. Because of their versatility one can fill a whole book explaining all the details. So here only some aspects concerning the use of disk images as archive format.



Metadata

Disk images "come" with a complete HFS+ file system, so metadata will be absolutely safe.

Compression

Unfortunately disk images don't offer compression ratios comparable to *lzma2*-compressed *xz* or *7z* archives.

zlib-compressed images *(UDZO)* achieve a compression ratio comparable to zip archives. The best compression ratio you get with *bzip2*-compressed images *(UDBZ)*. *lzfse* is a new format with about the same compression ratio as *zlib* at level 5, but both compression and decompression are significantly faster. *lzfse* is only compatible with Mac OS X 10.11 (El Capitan) or higher.

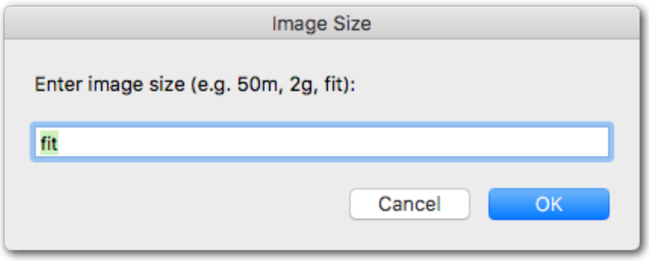
Uncompressed image formats are: *Read-only, Read/Write, Sparse image, Sparse bundle*.

Of course you can always *lzma*-compress a finished disk image with 7z. (If you plan to do this it is best to choose an uncompressed format for the disk image itself.)

Expandability

This is a huge advantage of disk images. You can always add stuff later, *as long as the image is large enough to hold the stuff.* If you plan to add stuff later to your image it's always a good idea to make the image large enough when you create it.

You can set the size in the Image Size dialog box:

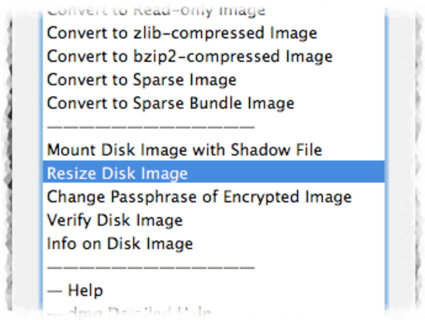


"fit" makes the image just large enough to hold the source items (the items or the folder you have selected in the Finder). This is fine if you don't plan to add anything later. Otherwise make the image large enough for future addition of items.

It is important to know that the image size will not necessarily affect the actual file size of the disk image (size on disk). For example: You create a *Sparse image* of a folder of 5MiB and choose as size 1GiB (1g). This means that the image will offer space up to 1GiB but the initial file size on disk will still be only a few megabyte. The file size grows as you add more stuff to the image later. Similar with compressed image formats: The unused space will effectively be compressed to 0 bytes.

Resize

You can always resize an existing image by selecting it and launching the script. Choose the *Resize Disk Image* option to expand an existing image:


Convert to Read-only Image
Convert to zlib-compressed Image
Convert to bzip2-compressed Image
Convert to Sparse Image
Convert to Sparse Bundle Image
Mount Disk Image with Shadow File
Resize Disk Image
Change Passphrase of Encrypted Image
Verify Disk Image
Info on Disk Image
— Help

You can only resize *Read/Write images, Sparse images and Sparse bundles!* Trying to resize other image types will result in an error. If your image is not in one of these formats choose *Convert…* to convert the image to a resizable format.

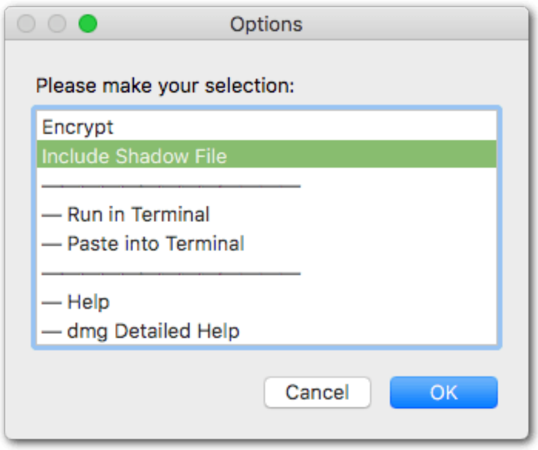
How to add files to a disk image

The way how to add files to an image depends on the image format. If the format is a writable format *(Read/Write, Sparse image, Sparse bundle)* you can add files by simply dragging them to the *mounted* image.

But you can also add files to the non-writable formats *zlib-compressed, bzip2-compressed* and *Read-only*. To do this you have to mount the image with the *Shadow file* option (see figure above). Once the image is mounted you will see a file with the same name as the image file with a *.shadow* extension in the same folder as the disk image file.

Now you can add files by dragging them to the mounted image, just as if it was a "really" writable disk image. The files you add will be stored in the shadow file. You can unmount the image and the added files will be safely preserved in the shadow file.

To incorporate the files from the shadow file into the disk image file you just have to convert the image to a format of your choice with the *Include Shadow File* option activated:



At conversion your original image and shadow files are renamed to *<imagename>.old.<extension>*. So you can check the integrity of the converted file before deleting the original.

Please note that independently if you add files directly to a *Sparse image* or if you add files to a non-writable image via the shadow file you can only add files as long as there is enough space on the disk image (see previous section).

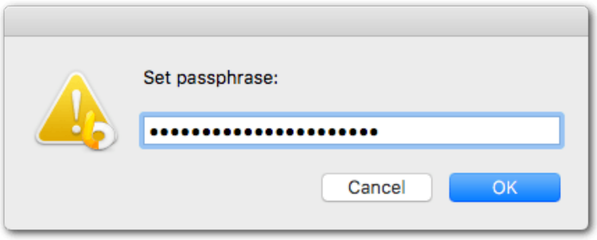
Removing files

If you remove files from a *Sparse image* or a *Sparse bundle* the disk image will not shrink automatically. To reclaim the free space (and thus reduce the occupied disk space) you need to use *Compact Image*. This is available from the primary window if a Sparse image/bundle is selected.

To remove files from a non-writable image you have to mount it with *Shadowed file* and re-convert it with the *Include Shadow File* option.

Encryption

Encryption is available for *7z's zip* and *7z (lzma2)* formats and for all *dmg* types. Encryption is always *AES-256* (no *zipcrypto*).



There are two concepts/types of encryption:

File encryption

The files inside the archive are encrypted. You can still see and browse the content of the archive. You must provide the password when you try to extract an encrypted file. This encryption type is offered for the *7z* format; it is the only available type for the *zip* format.

Attention:

It is possible to have a mixed archive consisting of encrypted an unencrypted files, or of files encrypted with different passphrases. This may easily lead to confusion later. If you use this type of encrypted archive be especially careful when you *add* files to the archive: If you provide a passphrase for the new files *7z* will not check the passphrase against the passphrase of existing files in the archive. It will simply encrypt the new files with the actually entered passphrase; so chances are good that you will end up with an archive consisting of files encrypted with different passphrases, maybe without knowing.

It's up to you if you consider this a feature or rather a threat to your data. I recommend to use this type of encryption only if you really need to be able to browse the content and if you are sure that you won't add files to this archive in the future. Otherwise use full (incl. header) encryption.

File and header encryption (full encryption)

Archive content and archive header are encrypted. You won't be able to see the contents of an archive without providing the passphrase. You can only add files to the archive if you provide the correct archive passphrase. This encryption type is offered by *7z* and is the only available type for *dmg*.

Multithreading

Multithreading is supported by *7z's zip*, *bzip2* and *lzma2* compression (not *gzip*). Multi-threaded *lzma2* compression delivers slightly larger output than single-threaded *lzma2* compression. But the difference is negligible. Please note that *lzma2* multithreading will only come into play if the total amount of data is *n*-times the dictionary size. So don't expect multithreading if you compress a 50MiB file at compression level 9 (dictionary size 64MiB). See the 7z info for details.

With the multithreading capability of *lzma2* we can say that *lzma2 compressed Posix archives (cpio.xz, tar.xz) are actually the most time-effective way to get highly compressed output while preserving all metadata.*

All archiving methods that support multithreading have an option to turn multithreading off. Without multithreading archiving will take more time of course, but it also allows you to do other CPU-intensive tasks parallel to the archiving process.

Why uncompressed archive formats?

The script offers quite some archive formats that are not compressed *(tar, R/W dmg, Read-only dmg, Sparse image, Sparse bundle)*. You may have asked yourself *what's the point of uncompressed archives?*

High compression ratios are only achievable with *lzma*.

You achieve this either in one step, for example with the *tar-lzma2* option *(=> tar.xz)* or you can prepare some *tar* (or *cpio*) archives of metadata-sensible files and compress them later, possibly together with other files, in a solid-block *7z*. In this case we prefer uncompressed *cpio/tar* archives, because at the final *7z-lzma2* compression they will compress *much* better than pre-compressed archives (e.g. *cpio.gz*) would do.

The same goes for disk images: If you want to archive for example several distribution *dmg* in a highly compressed *7z* archive, the final archive will be smaller if the included *dmg* are uncompressed.

Note that you can still add files to *cpio/tar* archives as long as they aren't compressed. You can also add files to the disk image formats *R/W, Sparse Image, Sparse Bundle*. Using the *shadow file* mechanism you can add files to all types of *dmg*.

Common errors

Archive/image creation fails because of existing file

There is already an archive / a disk image with the same name in the same location.

Solution: Remove or rename that item and run the script again.

No new archive is written

If the archive is of the type *tar, cpio, pax, 7z or zip* and there is an existing archive with the same name and at the same path, then the selected file/folder will be appended to the existing archive. See also this note.

Solution: Remove or rename the existing archive or save the new archive to a different path.

Password error / Operation not permitted

This happens if you try to add a file to a header-encrypted (fully encrypted) *7z* archive without having entered the passphrase while the script is set to run in the background.

Solution: In order to add files to header-encrypted archives you must provide the correct passphrase. So you either have to choose the *Encrypted* option from the Options List or run the archive script in Terminal Mode (then you will be prompted automatically for a password if an encrypted destination archive is detected). See also section Encryption.

Filename too long for tar archives

This happens when you try to *tar* files whose name (last path component) exceeds 100 chars or whose full path exceeds 255 chars.

Solution: Use *cpio* or shorten the offending filenames.

Verify of disk images fails

Some disk image types, e.g. R/W, sparse images/bundles, don't have a checksum, so they can't be verified.

Solution: None. This is not an error.

Programs used by this AppleScript

Pre-installed on OS X and macOS in /usr/bin/:

- *ditto*
- *tar (bsdtar)*, optionally *pax*
- *xar*
- *dirname, basename*
- *hdiutil*

Bundled with the AppleScript:

- *7za* (from the *p7zip* project, http://p7zip.sourceforge.net, under GNU Library or Lesser General Public License (LGPL))

The *7za* executable is located in the *bin* folder inside the script bundle's *Resources* folder.

If you rather want to use the *7z* installed on your system (e.g. from Homebrew), just comment/uncomment the corresponding lines in the script (at the beginning) to enforce the */usr/local/bin* path.