GNU tar: an archiver tool (*DRAFT*)

For version 1.11.8, June 1995

Jay Fenlason, Michael Bushnell
Amy Gorin, Francois Pinard

1 Introduction

This chapter introduces of a few words which will recur all over this manual, like “archive”, “member”, “name”, “unpack”, etc. It then explains who wrote GNU `tar` and its documentation, and says where to send bug reports or comments.

1.1 What `tar` Does

The `tar` program is used to create and manipulate `tar` archives. An *archive* is a single file which contains within it the contents of many files. In addition, the archive identifies the names of the files, their owner, and so forth. (Archives record access permissions, user and group, size in bytes, and last modification time. Some archives also record the file names in each archived directory, as well as other file and directory information.)

The files inside an archive are called *members*. Within this manual, we use the term *file* to refer only to files accessible in the normal ways (by `ls`, `cat`, and so forth), and the term *members* to refer only to the members of an archive. Similarly, a *file name* is the name of a file, as it resides in the filesystem, and a *member name* is the name of an archive member within the archive.

Initially, `tar` archives were used to store files conveniently on magnetic tape. The name ‘`tar`’ comes from this use; it stands for: `t`ape `ar`chiver. Despite the utility’s name, `tar` can direct its output to any available device, as well as store it in a file or direct it to another program via a pipe. `tar` may even access, as archives, remote devices or files.

You can use `tar` archives in many ways. We want to stress a few of them: storage, backup or transportation.

Storage

Often, `tar` archives are used to store related files for convenient file transfer over a network. For example, the GNU Project distributes its software bundled into `tar` archives, so that all the files relating to a particular program (or set of related programs) can be transferred as a single unit.

A magnetic tape can store several files in sequence, but has no names for them, just relative position on the tape. A `tar` archive or something like it is one way to store several files on one tape and retain their names. Even when the basic transfer mechanism can keep track of names, as FTP can, the nuisance of handling multiple files, directories, and multiple links, makes `tar` archives an attractive method.

Archive files are also used for long-term storage, which you can think of as transportation from one time to another.

Backup

Because the archive created by `tar` is capable of preserving file information and directory structure, `tar` is commonly used for performing full and incremental backups of disks, putting all together bunch of files possibly pertaining to many users and different projects, to secure against accidental destruction of those disks.

The GNU version of `tar` has special features that allow it to be used to make incremental and full dumps of all the files in a filesystem.

Transportation

Archive files can be used for transporting a group of files from one system to another: put all relevant files into an archive on one computer system, transfer

> the archive to another, and extract the contents there. The basic transfer medium might be magnetic tape, Internet FTP, or even electronic mail (though you must encode the archive with `uuencode` or some functional equivalent in order to transport it properly by mail). Both machines do not have to use the same operating system, as long as they both support the `tar` program.
>
> Piping one `tar` to another is an easy way to copy a directory's contents from one disk to another, while preserving the dates, modes, owners and link structure of all the files therein. `tar` is also ideal for transferring directories over networks. We sometimes see a copy of `tar` packing many files into one archive on one machine, and sending the produced archive over a pipe over the network to another copy of `tar` on another machine, reading its archive from the pipe and unpacking all files there.

The `tar` program provides the ability to create `tar` archives, as well as for various other kinds of manipulation. For example, you can use `tar` on previously created archives to extract files, to store additional files, or to update or list files already stored. The term *extraction* is used to refer to the process of copying an archive member into a file in the filesystem. One might speak of extracting a single member. Extracting all the members of an archive is often called extracting the archive. Also, the term *unpack* is used to refer to the extraction of many or all the members of an archive.

Conventionally, `tar` archives are given names ending with '`.tar`'. This is not necessary for `tar` to operate properly, but this manual follows the convention in order to get the reader used to seeing it.

Occasionally, `tar` archives are referred to as `tar` files, archive members are referred to as files, or entries. For people familiar with the operation of `tar`, this causes no difficulty. However, this manual consistently uses the terminology above in referring to archives and archive members, to make it easier to learn how to use `tar`.

1.2 GNU `tar` Authors

GNU `tar` was originally written by John Gilmore, and modified by many people. The GNU enhancements were written by Jay Fenlason, and the whole package has been further maintained by Michael Bushnell, then Francois Pinard, with the help of numerous and kind users. I wish to stress that `tar` is somewhat a collective work, and owe much to all those people who reported problems, offerred solutions and other insights, or shared their thoughts and suggestions. Even if we lost track of many of those contributors, a partial list can be found in the '`THANKS`' file from the GNU `tar` distribution.

Jay Fenlason put together a draft of a GNU `tar` manual, also borrowing notes from the original man page from John Gilmore, this draft has been distributed in `tar` versions 1.04 (or even before?) through 1.10, then withdrawn in version 1.11. Michael Bushnell and Amy Gorin worked at a tutorial and manual for GNU `tar`, and left a few unpublished versions of each. For version 1.11.8, Francois Pinard put together a new manual by grabbing from all these sources and merging them in a single manual.

I heard that there is another manual in the works, by another team, which should say everything about archives and related utilities, and which will surely be nicer than this one. In the meantime, please consider this manual is a placeholder for `tar` option list and a few random notes the maintainer wants to save somewhere, so users can read them. I wish

GNU `tar` users will be happier with this imperfect manual than with no documentation at all.

1.3 Reporting bugs or suggestions

Please report problems or suggestions about this program to '`bug-gnu-utils@prep.ai.mit.edu`'. You may also write directly, and less officially, to '`pinard@iro.umontreal.ca`'. There is a lot of mail flowing about `tar`, and some accumulated in the past years. You might expect a quick acknowledgement of your invoices, but the proper handling of your reports may be delayed for a long while.

Many nodes of this document have not been revised much, these all start with a little comment telling so. I accept documentation bug reports, of course. But please do not torture yourself into systematically reporting all inadequacies for the unrevised nodes of this document, unless you really feel like revising them.

1.4 Support considerations

This informal appendix is for the maintainer to share a few words and thoughts, while considering GNU `tar` support.

1.4.1 Stability of GNU `tar`

User reports mainly fall in three categories: portability problems, execution bugs, and requests for enhancements. For 1.11.X, the emphasis has been on solving portability problems, then trying to make GNU `tar` more solid. Enhancements have fairly low priority, yet I sometime slip one in just for taking a kind of rest :-).

Many bugs have been corrected since 1.11.2. If you are curious, glance through ChangeLog. I had only very few reports for things that *might* be new bugs not present in 1.11.2. If you are really curious, and have access to the FSF machines, see '`/gd/gnu/tar/rmail/`' hierarchy for all reports. Subdirectories '`0`', '`1`', '`2`' and '`3`' represent decreasing levels in priority. Most problems in there were reported against 1.10, 1.11 or 1.11.2 and still exist. The only thing I have consciously broken between 1.11.2 and 1.11.5 is '`--record-number`' ('`-R`'), because I wanted some modification to be done to '`gnulib/error.c`', which is outside my control. This modification is now done, but I did not revisit this area yet.

Here is my candid opinion. GNU `tar` has many areas of unreliability. See '`BACKLOG`' for the horrorful picture of the situation. Yet, for most users and usages, GNU `tar` looks very dependable. For me as a mere user, GNU `tar` did not give problems in years. And I think it offers a lot of functionality. Many problems have been solved since 1.11.2, even if true that many more remain to be solved. I'm not discouraged myself and feel positive about maintaining it, simply because when I bite, that usually lasts for quite long. I might not have all the time I would want, but I surely have good will and am happily surrounded by many collaborating pretesters. So, I still think GNU `tar` is on the winning side in the long run.

1.4.2 Should we rewrite the thing?

Working in `tar` sources is not always pleasurable. The problem is that `tar` sources are very fragile. Just cleaning around breaks things. The current sequence of prereleases is for slowly trying to solidify it, so `tar` becomes more maintainable. I think that the ugliness of sources could be corrected to a certain extent, too.

A few efforts to replace GNU `tar` have been done already and it seems that all failed so far. A toy program, for me, is another kind of failure. I think people underestimate the number of portability problems such a program can raise. This is not only a matter of programming style, there is really a wide variability in systems out there. GNU `tar` has a long history, met a rich variety of porting problems, machine peculiarities, system idiosyncrasies, which are unrelated to programming style. My own opinion is that we cannot dismiss all the experience gleaned along the years, and saved (if not hidden) in GNU `tar` sources, pretending to start anew, from scratch.

Even if a new program replacing GNU `tar` would be marvelous, GNU `tar` stalled for a few years waiting for such a program, and we are now faced to nothing, with hundreds of user reports to catch on. We need a working archiver *now*, and cannot live on promises. Any new program will take hundreds of user reports, and many years, to stabilize enough to become a plausible `tar` replacement. I rather plan to clean up GNU `tar`. This alone is a big task for me, because GNU `tar` coding is not ideal, and I have to find ways to transform it slowly, while having it fully working at all times.

1.4.3 Why maintaining it?

I confess that I am a little afraid of `tar` maintainance. It is difficult for many reasons, the first tree being more evident than the others:

- the algorithmic design was initially oriented for machines having very small memory, it was later much adapted for new features without doing everything necessary for the whole to stay clean;
- the wide visibility of `tar` forces many stunts at portability;
- GNU `tar` has to be sensitive to file systems and device variance.
- GNU central has been seduced by some users promising to write wonderful `tar` replacements, which never came, so development has been put aside for years, while bug reports accumulated;
- the `tar` manual has been withdrawn, promising users a fine replacement for it, so raising their expectations;
- maintenance was once split between four maintainers (one for `tar`, one for `mt`, another for scripts, and a team for documentation), and also, '`rtapelib.[ch]`' from `tar` is used in `cpio`, and synchronisation has not always been easy.

However, even if difficult, I do feel like doing a careful cleanup, so `tar` would become less painful to maintain after a while (and less subject to criticism). And besides, I'm surrounded by a marvelous team of pretesters and by many other collaborating users, which I should learn to serve better. Getting more experience with maintainance in GNU, I hope being careful enough modifying `tar` so not hurting users too much, being aware that `tar` is a sensitive product in GNU. Once cleaned up, I might be happy to return `tar` maintainance to someone else...

`tar` requires more work alone that all my other things together, and I have to resist being swallowed whole in it. This resistance makes `tar` development somewhat slower. Sorry!

1.4.4 MSDOS and other systems?

GNU does not necessarily support non-UNIX systems, that is to say, MSDOS is not supported. It is very true that ports can sometimes be very intrusive in the sources, cluttering them significantly with conditionals and extra code, and distract GNU maintainers from the main development line.

However, a special argument might be made for `tar`. Both `tar` and `gzip` are the required tools for getting something out of the GNU archives, `tar` should be more opened to ports than the GNU rule states. Jean-loup did a tremendous job at porting `gzip` on smaller systems. It would be comfortable that a few other GNU tools be available on MSDOS and others, among which `tar`. These ports for `tar` have theoretically no priority at all. Nevertheless, a port is interesting, because `tar` is so central in GNU distributions, and `gzip` is already ported.

Some porting efforts have been done in the past. There are traces of a few exchanges on this subject in '`BACKLOG`'. GNU `tar` sources have been modified a lot recently at a cosmetic level, and I would certainly have a hard time integrating diffs provided by someone else. If people want porting `tar` to MSDOS or other non-UNIX systems, they should be committed in supporting their ports after the fact, as I cannot do it myself.

2 Tutorial Introduction to `tar`

This chapter guides you through some basic examples of `tar` operations. If you already know how to use some other version of `tar`, then you probably don't need to read this chapter. This chapter omits complicated details about many of the ways `tar` works. See later chapters for full information.

Before proceeding further with this tutorial chapter, be sure you understand already and clearly what is meant by "archive" and "archive member".

FIXME: xref What tar Does

.

This chapter guides you through some basic examples of `tar` operations. In the examples, the lines you should type are preceded by a '`%`', which is a typical shell prompt. We use mnemonic forms of operations and options in the examples, and in discussions in the text, but short forms produce the same result.

Most of the options to `tar` come in both long forms and short forms. The options described in this tutorial have the following abbreviations (except '`--delete`', which has no shorthand form):

'`--create`'	'`-c`'
'`--list`'	'`-t`'
'`--extract`'	'`-x`'
'`--append`'	'`-r`'
'`--verbose`'	'`-v`'
'`--file=`*`archive-name`*'	'`-f` *`archive-name`*'

These options make typing long `tar` commands easier. For example, instead of typing

```
tar --create --file=/tmp/afiles.tar --verbose apple angst asparagus
```

you can type

```
tar -c -f /tmp/afiles.tar -v apple angst asparagus
```

For more information on option syntax,

FIXME: ref Invoking tar

. In discussions in the text, when we present some mnemonic option, we also give the corresponding short option within parentheses.

2.1 How to Create Archives

(This message will disappear, once this node revised.)

To create a new archive, use the '`--create`' ('`-c`') option to `tar`. You can use options to specify the name and format of the archive (as well as other characteristics), and you can use file name arguments to specify which files and directories are to be put in the archive.

FIXME: xref Creating

, for more information about the '`--create`' ('`-c`') operation.

To create a new archive, use the '`--create`' ('`-c`') option to `tar`. You should generally use the '`--file=`*`archive-name`*' ('`-f` *`archive-name`*') option to specify the name the `tar` archive will have. Then specify the names of the files you wish to place in the new archive. For example, to place the files '`apple`', '`angst`', and '`asparagus`' into an archive named '`afiles.tar`', use the following command:

```
tar --create --file=afiles.tar apple angst asparagus
```

The order of the arguments is not important when using mnemonic option style. You could also say:

```
tar apple --create angst --file=afiles.tar asparagus
```

This order is harder to understand however. In this manual, we will list the arguments in a reasonable order to make the commands easier to understand, but you can type them in any order you wish.

If you don't specify the names of any files to put in the archive, then `tar` will create an empty archive. So, the following command will create an archive with nothing in it:

```
tar --create --file=empty-archive.tar
```

Whenever you use '`--create`' ('`-c`'), `tar` will erase the current contents of the file named by '`--file=`*`archive-name`*' ('`-f` *`archive-name`*') if it exists. To add files to an existing archive, you need to use a different option.

FIXME: xref Adding to Archives, for information on how to do this.

When an archive is created through '`--create`' ('`-c`'), the member names of the members of the archive are exactly the same as the file names as you typed them in the `tar` command. So, the member names of '`afiles`' (as created by the first example above) are '`apple`', '`angst`', and '`asparagus`'. However, suppose an archive were created with this command:

```
tar --create --file=bfiles.tar ./balloons baboon ./bodacious
```

Then, the three files '`balloons`', '`baboon`', and '`bodacious`' would get placed in the archive (because '`./`' is a synonym for the current directory), but their member names would be '`./balloons`', '`baboon`', and '`./bodacious`'.

If you want to see the progress of `tar` as it writes files into the archive, you can use the '`--verbose`' ('`-v`') option.

If one of the files named with '`--create`' ('`-c`') is a directory, then the operation of `tar` is more complicated.

FIXME: xref Tar and Directories,
FIXME: the last section of this tutorial, for more information.

If you don't specify the '`--file=`*`archive-name`*' ('`-f` *`archive-name`*') option, then `tar` will use a default. Usually this default is some physical tape drive attached to your machine.

If there is no tape drive attached, or the default is not meaningful, then `tar` will print an error message. This error message might look roughly like one of the following:

```
tar: can't open /dev/rmt8 : No such device or address
tar: can't open /dev/rsmt0 : I/O error
```

If you get an error like this, mentioning a file you didn't specify ('`/dev/rmt8`' or '`/dev/rsmt0`' in the examples above), then `tar` is using a default value for '`--file=`*`archive-name`*' ('`-f` *`archive-name`*'). You should generally specify a '`--file=`*`archive-name`*' ('`-f` *`archive-name`*') argument whenever you use `tar`, rather than relying on a default.

To create a new archive, use the '`--create`' ('`-c`') option to `tar`. You can use options to specify the name and format of the archive (as well as other characteristics), and you can use file name arguments to specify which files to put in the archive. If you don't use any options or file name arguments, `tar` will use default values.

FIXME: xref Creating Example

, for more information about the '`--create`' ('`-c`') option.

2.1.1 Creating Archives of Files

(This message will disappear, once this node revised.)

This example shows you how to create an archive file in your working directory containing other files in the same directory. The three files you archive in this example are called '`blues`', '`folk`', and '`jazz`'. The archive file is called '`records`'. While the archive in this example is written to the file system, it could also be written to tape. (If you want to follow along with this and future examples, create a practice subdirectory containing files with these names. To create the directory, type '`mkdir practice`' at the system prompt. You can create the files using a text editor, such as `emacs`).

While in the directory containing the files you want to archive, list the directory's contents.

Type:

```
% cd practice
% ls
```

The system responds:

```
blues folk jazz
%
```

This is to check that the files to be archived do in fact exist in the working directory, and to check that the archive name you have chosen isn't already in use. If it is, `tar` will overwrite the old archive and its contents will be lost.

Then,

- Create a new archive by giving '`--create`' ('`-c`') to `tar`.
- Explicitly name the archive file being created—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*'). If you don't use this option `tar` will write the archive to the default storage device, which varies from system to system.
- Specify which files to put into the archive. If you don't specify any *file name* arguments, `tar` will archive everything in the working directory.

Type:

```
% tar --create --file=records blues folk jazz
```

If you now list the contents of the working directory (‘`ls`’), you will find the archive file listed as well as the files you saw previously.

```
% ls
blues folk jazz records
%
```

This example shows you how to create an archive file in the working directory containing other files in the working directory. The three files you archive in this example are called ‘`blues`’, ‘`folk`’, and ‘`jazz`’. The archive file is called ‘`records`’. While the archive in this example is written to the file system, it could also be written to any other device.

(If you want to follow along with this and future examples, create a directory called ‘`practice`’ containing files called ‘`blues`’, ‘`folk`’ and ‘`jazz`’. To create the directory, type ‘`mkdir practice`’ at the system prompt. It will probably be easiest to create the files using a text editor, such as Emacs.)

First, change into the directory containing the files you want to archive:

```
% cd practice
```

‘`~/practice`’ is now your working directory.

Then, check that the files to be archived do in fact exist in the working directory, and make sure there isn’t already a file in the working directory with the archive name you intend to use. If you specify an archive file name that is already in use, `tar` will overwrite the old file and its contents will be lost.

To list the names of files in the working directory, type:

```
% ls
```

The system responds:

```
blues   folk    jazz
%
```

Then,

- Create a new archive by giving the ‘`--create`’ (‘`-c`’) option to `tar`.
- Explicitly name the archive file being created—‘`--file=`*`archive-name`*’ (‘`-f` *`archive-name`*’). If you don’t use this option `tar` will write the archive to the default storage device, which varies from system to system.

  FIXME: this syntax may change. OK now–check before printing

  `tar` interprets archive file names relative to the working directory. Make sure you have write access to the working directory before using `tar`.
- Specify which files to put into the archive (`tar` interprets file names relative to the working directory). If you don’t use any *name* arguments, `tar` will archive everything in the working directory.

Type:

```
% tar --create --file=records blues folk jazz
```

If you now list the contents of the working directory (‘`ls`’), you will find the archive file listed as well as the files you saw previously.

```
% ls
blues folk jazz records
%
```

2.1.2 Using `tar` in Verbose Mode

(This message will disappear, once this node revised.)

If you include the '`--verbose`' ('`-v`') option on the command line, `tar` will list the files it is acting on as it is working. The example above in verbose mode would be:

```
% tar --create --file=records --verbose blues folk jazz
blues
folk
jazz
```

The first line, which is preceeded by a '`%`', is the command line. The lines after the first line are generated by `tar` as it works. In the following examples we usually use verbose mode, though it is almost never required.

If you include the '`--verbose`' ('`-v`') option on the command line, `tar` will list the files it is acting on as it is working. In verbose mode, the creation example above would appear as:

```
% tar --create --file=records --verbose blues folk jazz
blues
folk
jazz
```

The first line is the command typed in by the user. The remaining lines are generated by `tar`. In the following examples we usually use verbose mode, though it is almost never required.

2.1.3 How to Archive Directories

(This message will disappear, once this node revised.)

When the names of files or members specify directories, the operation of `tar` is more complex. Generally, when a directory is named, `tar` also operates on all the contents of the directory, recursively. Thus, to `tar`, the file name '`/`' names the entire file system.

To archive the entire contents of a directory, use '`--create`' ('`-c`') or '`--append`' ('`-r`') as usual, and specify the name of the directory. For example, to archive all the contents of the current directory, use '`tar --create --file=`*`archive-name`*` .`'. Doing this will give the archive members names starting with '`./`'. To archive the contents of a directory named '`foodir`', use '`tar --create --file=`*`archive-name`*` foodir`'. In this case, the member names will all start with '`foodir/`'.

If you give `tar` a command such as '`tar --create --file=foo.tar .`', it will report '`tar: foo.tar is the archive; not dumped`'. This happens because the archive '`foo.tar`' is created before putting any files into it. Then, when `tar` attempts to add all the files in the directory '`.`' to the archive, it notices that the file '`foo.tar`' is the same as the archive, and skips it. (It makes no sense to put an archive into itself.) GNU `tar` will continue in this case, and create the archive as normal, except for the exclusion of that one file. Other versions of `tar`, however, are not so clever, and will enter an infinite loop when this happens, so you should not depend on this behavior. In general, make sure that the archive is not inside a directory being dumped.

When extracting files, you can also name directory archive members on the command line. In this case, `tar` extracts all the archive members whose names begin with the name of the directory. As usual, `tar` is not particularly clever about interpreting member names. The command '`tar --extract --file=`*`archive-name`* `.`' will not extract all the contents of the archive, but only those members whose member names begin with '`./`'.

2.1.4 Creating an Archive from the Superior Directory

(This message will disappear, once this node revised.)

You can archive a directory by specifying its directory name as a file name argument to `tar`. The files in the directory will be archived relative to the working directory, and the directory will be re-created along with its contents when the archive is extracted.

To archive a directory, first move to its superior directory. If you have been following the tutorial, you should type:

```
% cd ..
%
```

Once in the superior directory, you can specify the subdirectory as a file name argument. To store the directory '`practice`' in the archive file '`music`', type:

```
% tar --create --verbose --file=music practice
```

`tar` should output:

```
practice/
practice/blues
practice/folk
practice/jazz
practice/records
```

Note that the archive thus created is not in the subdirectory '`practice`', but rather in the working directory—the directory from which `tar` was invoked. Before trying to archive a directory from its superior directory, you should make sure you have write access to the superior directory itself, not only the directory you are trying archive with `tar`. Trying to store your home directory in an archive by invoking `tar` from the root directory will probably not work.

FIXME: xref absolute-names

(Note also that '`records`', the original archive file, has itself been archived. `tar` will accept any file as a file to be archived, regardless of its content. When '`music`' is extracted, the archive file '`records`' will be re-written into the file system).

You can store a directory in an archive by using the directory name as a file name argument to `tar`. When you specify a directory file, `tar` archives the directory file and all the files it contains. The names of the directory and the files it contains are stored in the archive relative to the current working directory–when the directory is extracted they will be written into the file system relative to the working directory at that time.

FIXME: add an xref to –absolute-names

To archive a directory, first move to its superior directory. If you have been following the tutorial, you should type:

```
% cd ..
%
```

Once in the superior directory, specify the subdirectory using a file name argument. To store the directory file '`~/practice`' in the archive file '`music`', type:

```
% tar --create --verbose --file=music practice
```

`tar` should respond:

```
practice/
practice/blues
practice/folk
practice/jazz
practice/records
```

Note that '`~/practice/records`', another archive file, has itself been archived. `tar` will accept any file as a file to be archived, even an archive file.

FIXME: symbolic links and changing directories are now in main body,
FIXME: not in tutorial.

2.1.5 Comparing Files in an Archive with Files in the File System

(This message will disappear, once this node revised.)

While the '`--list`' ('`-t`') operation with the '`--verbose`' ('`-v`') option specified is useful in keeping files in the archive current with files in the file system (by allowing the user to compare size and modification dates), it is simpler to have `tar` itself compare file attributes and report back on file differences. To do so, use the '`--compare`' ('`-d`') or '`--diff`' operation.

The '`--compare`' ('`-d`') operation, as its name implies, causes `tar` to compare files and directories in the archive with their counterparts (files of the same name) in the file system, and report back differences in file size, mode, owner and modification date. When performing the '`--compare`' ('`-d`') operation, `tar` acts only on files actually in the archive—it will ignore files in the active file system that do not exist in the archive. If `tar` with '`--compare`' ('`-d`') specified is given, as a file name argument, the name of a file that does not exist in the archive, it will return an error message.

To compare the files in the practice directory with their counterparts in the archive file '`records`', in the same directory, you would, while in the '`practice`' directory:

- Invoke `tar` and specify the operation to compare files in the archive with their counterparts in the file system—'`--compare`' ('`-d`') or '`--diff`'.
- Specify the name of the archive where the files to be compared are stored—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').
- Specify the names of the files or directories to be compared, as file name arguments (in this case, you are comparing all the files in the archive, so nothing need be specified).

```
% tar --compare --file=records
%
```

While it looks like nothing has happened, `tar` has, in fact, done the comparison—and found nothing to report. The same example with the '`--verbose`' ('`-v`') option specified would list the files in the archive as they are being compared with their counterparts of the same name:

```
% tar --compare --verbose --file=records
blues
folk
jazz
%
```

If `tar` had had anything to report, it would have done so as it was comparing each file. If you remove the file '`jazz`' from the file system ('`rm jazz`'), and modify the file '`blues`' (for instance, by adding text to it with a text editor), the above example would look like:

```
% tar --compare --verbose --file=records
blues
blues: mod time differs
blues: size differs
folk
jazz
jazz: does not exist
%
```

You should note again that while '`--compare`' ('`-d`') does cause `tar` to report back on files in the archive that do not exist in the file system, `tar` will ignore files in the active file system that do not exist in the archive. To demonstrate this, create a file in the '`practice`' directory called '`rock`' (using any text editor). If you generate a directory listing the new file will appear.

```
% ls
blues  folk   records  rock
```

If you run the '`--compare`' ('`-d`') example again you will obtain the following:

```
% tar --compare --verbose --file=records
blues
blues: mod time differs
blues: size differs
folk
jazz
jazz: does not exist
%
```

`tar` ignores the file '`rock`' because `tar` is comparing files in the archive to files in the file system, not vice versa. If '`rock`' had been passed to `tar` explicitly (as a file name argument), `tar` would have returned an error message, as follows:

```
% tar --compare --verbose --file=records rock
tar: rock not found in archive
%
```

To compare the attributes of archive members with the attributes of their counterparts in the file system, use the '`--compare`' ('`-d`') or '`--diff`'operation. While you could use '`--list --verbose`' ('`-tv`') to manually compare some file attributes, it is simpler to have `tar` itself compare file attributes and report back on file differences.

FIXME: "manually"? suggestions?

The '`--compare`' ('`-d`') operation, as its name implies, compares archive members with files of the same name in the file system, and reports back differences in file size, mode, owner and modification date. '`tar +compare`' acts only on archive members–it ignores files in the file system that are not stored in the archive. If you give with '`--compare`' ('`-d`') a *name* argument that does not correspond to the name of an archive member, `tar` responds with an error message.

To compare archive members in the archive file '`records`' with files in the '`~/practice`' directory, first change into the '`practice`' directory. Then:

- Invoke `tar` and specify the '`--compare`' ('`-d`') operation—'`--compare`' ('`-d`') or '`--diff`'.
- Specify the archive where the files to be compared are stored—'`--file=archive-name`' ('`-f archive-name`').
- Specify the archive members to be compared. (In this example you are comparing all the archive members in the archive. Since this is the default, you don't need to use any file name arguments).

  ```
  % tar --compare --file=records
  %
  ```

While it looks like nothing has happened, `tar` has, in fact, done the comparison—and found nothing to report.

Use the '`--verbose`' ('`-v`') option to list the names of archive members as they are being compared with their counterparts of the same name in the file system:

```
% tar --compare --verbose --file=records
blues
folk
jazz
%
```

If `tar` had had anything to report, it would have done so as it was comparing each file.

If you remove the file '`jazz`' from the file system ('`rm jazz`'), and modify the file '`blues`' (for instance, by adding text to it with an editor such as Emacs), the above example would look like:

```
% tar --compare --verbose --file=records
blues
blues: mod time differs
blues: size differs
folk
jazz
jazz: does not exist
%
```

Note again that while '`--compare`' ('`-d`') reports the names of archive members that do not have counterparts in the file system, '`--compare`' ('`-d`') ignores files in the file system that do not have counterparts in the archive. To demonstrate this, create a file in the '`practice`' directory called '`rock`' (using any text editor). The new file appears when you list the directory's contents:

FIXME: Given an example

2.1.6 Using Compare from the Superior Directory

(This message will disappear, once this node revised.)

In addition to using '`--compare`' ('`-d`') to compare individual files in an archive with their counterparts in the file system, you can use '`--compare`' ('`-d`') to compare archived directories with their counterparts in the active file system. You could re-create the examples above using your home directory as the working directory, and using the archive file '`music`' (in which is stored the '`practice`' directory) instead of the archive file '`records`'.

First, change into the home directory ('`cd ..`'). Then, try the above example using '`music`' as the specified archive file, and the '`practice`' subdirectory as a file name argument.

```
% tar --compare --verbose --file=music practice
practice
practice/blues
practice/blues: mod time differs
practice/blues: size differs
practice/folk
practice/jazz
practice/jazz: does not exist
practice/records
```

In addition to using '`--compare`' ('`-d`') to compare text files, you can use '`--compare`' ('`-d`') to compare directories. To illustrate this, re-create the examples above using your home directory as the working directory, and using the archive file '`~/music`' instead of the archive file '`~/practice/records`'.

First, change into your home directory ('`cd ~`'). Then, try the above example using '`music`' as the specified archive file, and '`practice`' as a file name argument.

```
% tar --compare --verbose --file=music practice
```

If you have been following along with the tutorial, `tar` will respond:

```
practice
practice/blues
practice/blues: mod time differs
practice/blues: size differs
practice/folk
practice/jazz
practice/jazz: does not exist
practice/records
```

2.2 How to List Archives

(This message will disappear, once this node revised.)

Use '`--list`' ('`-t`') to print the names of members stored in an archive. Use a '`--file=`*`archive-name`*' ('`-f` *`archive-name`*') option just as with '`--create`' ('`-c`') to specify the name of the archive. For example, the archive '`afiles.tar`' created in the last section could be examined with the command '`tar --list --file=afiles.tar`'. The output of `tar` would then be:

```
apple
angst
```

```
asparagus
```

The archive '`bfiles.tar`' would list as follows:

```
./baloons
baboon
./bodacious
```

(Of course, '`tar --list --file=empty-archive.tar`' would produce no output.)

If you use the '`--verbose`' ('`-v`') option with '`--list`' ('`-t`'), then `tar` will print out a listing reminiscent of '`ls -l`', showing owner, file size, and so forth.

You can also specify member names when using '`--list`' ('`-t`'). In this case, `tar` will only list the names of members you identify. For example, '`tar --list --file=afiles.tar apple`' would only print '`apple`'. It is essential when specifying member names to `tar` that you give the exact member names. For example, '`tar --list --file=bfiles baloons`' would produce no output, because there is no member named '`baloons`', only one named '`./baloons`'. While the file names '`baloons`' and '`./baloons`' name the same file, member names are compared using a simplistic name comparison, in which an exact match is necessary.

2.2.1 Listing the Contents of an Archive

(This message will disappear, once this node revised.)

You can list the contents of the archive you just created with another option of `tar`: '`--list`' ('`-t`'). To list the contents of an archive, type:

```
% tar --list --file=records
```

`tar` will respond:

```
blues folk jazz
```

FIXME: xref Listing Archive Contents

, for a more detailed tutorial of the '`--list`' ('`-t`') operation.

FIXME: xref Listing Contents

for more information about the '`--list`' ('`-t`') operation.

FIXME:

You can use '`--list`' ('`-t`') to output a list of the files in an archive. If you use file name arguments with this operation, `tar` will look in the archive for the files specified and display their names only if they are, in fact, stored. You can use '`--list`' ('`-t`') with the '`--verbose`' ('`-v`') option to find out the attributes (owner, size, etc.) of stored files.

You can list the contents of an archive with another operation of `tar`: '`--list`' ('`-t`'). To list the contents of the archive you just created, type:

```
% tar --list --file=records
```

`tar` will respond:

```
blues folk jazz
```

FIXME: xref Listing Archive Contents

, for a more detailed tutorial of the '`--list`' ('`-t`') operation.

FIXME: xref Listing Contents

, for more information about the '`--list`' ('`-t`') operation.

In a previous example, you created the archive '`music`' in the home directory. To list the contents of '`music`':

- List the contents of an archive by using '`--list`' ('`-t`') with `tar`.
- Specify the name of the archive to be listed—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').

Thus:

```
% tar --list --file=music
practice/
practice/blues
practice/folk
practice/jazz
practice/records
```

Use '`--list`' ('`-t`') to print the names of files stored in an archive. If you use file name arguments with this operation, `tar` prints the names of the specified files if they are stored in the archive. If you use a directory name as a file name argument, `tar` also prints the names of all underlying files, including sub-directories. If you use no file name arguments, `tar` prints the names of all the archive members.

You can use '`--list`' ('`-t`') with the '`--verbose`' ('`-v`') option to print archive members' attributes (owner, size, etc.).

To list the names of files stored in an archive, use the '`--list`' ('`-t`') operation of `tar`.

In a previous example, you created the archive '`~/music`'. To list the contents of '`music`', while in your home directory:

- List the contents of an archive by using [No value for "–list"] with `tar`.
- Specify the archive to be listed—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').

Thus:

```
% tar --list --file=music
practice/
practice/blues
practice/folk
practice/jazz
practice/records
```

2.2.2 Getting Additional File Information

(This message will disappear, once this node revised.)

When you specify the '`--verbose`' ('`-v`') option in conjunction with '`--list`' ('`-t`'), `tar` will print additional information about the files being listed (file protection, owner and group ID, size, and date and time of creation). The example above, in verbose mode, would be:

```
% tar --list --verbose --file=music
drwxrwxrwx myself/user 0 May 31 21:49 1990 practice/
```

```
-rw-rw-rw- myself/user 42 May 21 13:29 1990 practice/blues
-rw-rw-rw- myself/user 62 May 23 10:55 1990 practice/folk
-rw-rw-rw- myself/user 40 May 21 13:30 1990 practice/jazz
-rw-rw-rw- myself/user 10240 May 31 21:49 1990 practice/records
%
```

Note that using '`--verbose`' ('`-v`') with '`--list`' ('`-t`') does not cause `tar` to print the names of files as they are being acted on, though the '`--verbose`' ('`-v`') option will have this effect with all other operations.

To get more information when you list the names of files stored in an archive, specify the '`--verbose`' ('`-v`') option in conjunction with '`--list`' ('`-t`').

`tar` will print archive member's file protection, owner and group ID, size, and date and time of creation.

For example:

```
% tar --list --verbose --file=music
drwxrwxrwx myself/user 0 May 31 21:49 1990 practice/
-rw-rw-rw- myself/user 42 May 21 13:29 1990 practice/blues
-rw-rw-rw- myself/user 62 May 23 10:55 1990 practice/folk
-rw-rw-rw- myself/user 40 May 21 13:30 1990 practice/jazz
-rw-rw-rw- myself/user 10240 May 31 21:49 1990 practice/records
%
```

Note that when you use '`--verbose`' ('`-v`') with '`--list`' ('`-t`'), `tar` doesn't print the names of files as they are being acted on, though the '`--verbose`' ('`-v`') option will have this effect when used with all other operations.

2.2.3 List A Specific File in an Archive

(This message will disappear, once this node revised.)

FIXME:

To to see if a particular file is in an archive, specify the name of the file in question as a file name argument while specifying the '`--list`' ('`-t`') operation. For example, if you wanted to see if the file '`folk`' were in the archive file '`music`', you would:

- Invoke `tar`, and specify the operation to list the contents of an archive—'`--list`' ('`-t`').
- Specify the name of the archive file to be acted on—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').
- Specify the name of the file `tar` is to look for, as a file name argument. Because `tar` preserves paths, file names must be specified as they appear in the archive (ie.. as they are relative to the directory from which the archive was created).

  FIXME: xref -P

Type:

```
% tar --list --file=music practice/folk
```

`tar` responds:

```
practice/folk
```

If the file were not in the archive (for example, the file '`practice/rock`'), the example above would look like:

```
% tar --list --file=music practice/rock
tar: practice/rock not found in archive
```

The '`--verbose`' ('`-v`') option does not have any effect on execution of the '`--list`' ('`-t`') operation when you have specified file name arguments.

FIXME: this is a bug (?)

To to see if a particular file is in an archive, use the name of the file in question as a file name argument while specifying the '`--list`' ('`-t`') operation. For example, to see whether the file '`folk`' is in the archive file '`music`', do the following:

- Invoke `tar`, and specify the '`--list`' ('`-t`') operation.
- Specify the archive file to be acted on—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').
- Specify the files to look for, by typing their names as file name arguments. You have to type the file name as it appears in the archive (normally, as it is relative to the relative to the directory from which the archive was created).

  FIXME: xref absolute-names

Type:

```
% tar --list --file=music practice/folk
```

`tar` responds:

```
practice/folk
```

If the file were not stored in the archive (for example, the file '`practice/rock`'), the example above would look like:

```
% tar --list --file=music practice/rock
tar: practice/rock not found in archive
```

If you had used '`--verbose`' ('`-v`') mode, the example above would look like:

```
% tar --list --file=music practice/folk
-rw-rw-rw- myself/user 62 May 23 10:55 1990 practice/folk
```

2.2.4 Listing the Contents of a Stored Directory

(This message will disappear, once this node revised.)

To get information about the contents of an archived directory, use the directory name as a file name argument in conjunction with '`--list`' ('`-t`'). To find out file attributes, include the '`--verbose`' ('`-v`') option.

For example, to find out about files in the directory '`practice`', in the archive file '`music`', type:

```
% tar --list --file=music practice
```

`tar` responds:

```
drwxrwxrwx myself/user 0 May 31 21:49 1990 practice/
-rw-rw-rw- myself/user 42 May 21 13:29 1990 practice/blues
-rw-rw-rw- myself/user 62 May 23 10:55 1990 practice/folk
-rw-rw-rw- myself/user 40 May 21 13:30 1990 practice/jazz
-rw-rw-rw- myself/user 10240 May 31 21:49 1990 practice/records
```

When you use a directory name as a file name argument, `tar` acts on all the files (including sub-directories) in that directory.

2.3 How to Extract Members from an Archive

(This message will disappear, once this node revised.)

In order to extract members from an archive, use the '`--extract`' ('`-x`') option. Specify the name of the archive with '`--file=`*`archive-name`*' ('`-f` *`archive-name`*'). To extract specific archive members, give their member names as arguments. It essential to give their exact member name, as printed by '`--list`' ('`-t`'). This will create a copy of the archive member, with a file name the same as its name in the archive.

Keeping the example of the two archives created at the beginning of this tutorial, '`tar --extract --file=afiles.tar apple`' would create a file '`apple`' in the current directory with the contents of the archive member '`apple`'. It would remove any file named '`apple`' already present in the directory, but it would not change the archive in any way.

Remember that specifying the exact member name is important. '`tar --extract --file=bfiles.tar baloons`' will fail, because there is no member named '`baloons`'. To extract the member named '`./baloons`' you would need to specify '`tar --extract --file=bfiles.tar ./baloons`'. To find the exact member names of the members of an archive, use '`--list`' ('`-t`').

FIXME: xref Listing Archives.

If you do not list any archive member names, then '`--extract`' ('`-x`') will extract all the members of the archive.

If you give the '`--verbose`' ('`-v`') option, then '`--extract`' ('`-x`') will print the names of the archive members as it extracts them.

2.3.1 Extract Files from an Archive into Your Current Directory

(This message will disappear, once this node revised.)

Obviously, the ultimate goal of `tar` users is to eventually get their files back. To do this, use the '`--extract`' ('`-x`') or '`--get`' operation. '`--extract`' ('`-x`') can be used to retrieve individual files from an archive, or can be used to write all the files in the archive back into the file system.

In the previous example you concatenated two archives, '`music`', and '`practice/records`'. To now retrieve the complete contents of '`music`' (the target file in the concatenation process), you would, from the home directory:

- Invoke `tar` and specify the operation to extract files from an archive ('`--extract`' ('`-x`') or '`--get`'.
- Specify the name of the archive the files will be extracted from—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').
- Specify the names of the files you wish to extract, as file name arguments (in this case you want to extract the entire archive, so you don't need to specify anything).

```
% tar --extract --file=music
tar: Could not make directory practice : File exists
```

Because the files stored originally in '`music`' were stored as files in a subdirectory (not as files in the working directory), they are stored in the archive with a leading directory name—`tar`, in restoring them, has tried to recreate that directory and failed: the directory already exists. The extraction has not been aborted, however. If you now change into the 'practice' directory and generate a directory listing, you will find that 'jazz', which we removed in an earlier example, has been resurrected.

```
% cd practice
% ls
blues    classical  folk  jazz     records    rock
```

If you look more closely at the files in the directory, however, you will find that 'blues' and 'folk' are, in fact, the original versions of the file, which were stored in 'music' at the beginning of the tutorial. `tar`, in extracting the original files from 'music', has overwritten the existing files in the file system.

While the newer versions of the files were stored in 'records' above, they can no longer be extracted from it. 'records' too was archived by `tar` when the 'practice' directory was stored in the archive file 'music', and was restored to its older incarnation when the files in 'practice' were overwritten. However, the newer version of 'records' was concatenated with 'music'. The contents of the newer version of 'records', therefore, should have been extracted when all the contents of 'music' were extracted. They were. `tar` has restored them into the working directory using the names with which they were originally stored. Because they were originally stored as part of 'records', in the 'practice' directory, they had no preceeding directory stored as part of their file names. To find the latest versions of 'blues', 'folk', 'jazz', 'rock' and 'classical', look in your home directory.

You may wish to restore the files in your 'practice' directory to their last state before we extracted the files from 'music'. Rather than moving the files from your home directory to the 'practice' subdirectory, you can run the same extraction procedure as above using the 'practice' subdirectory as your working directory:

```
% cd practice
% tar --extract --verbose --file=~/music
practice/
practice/blues
practice/folk
practice/jazz
practice/records
blues
folk
jazz
blues
rock
blues
classical
%
```

If you now examine the files in the practice directory, you will find that the files have been restored to their previous, newer, states. The old versions of the files, which were stored in 'music' with a preceeding directory name, have been written into a newly created subdirectory under the working directory (which is your 'practice' subdirectory). The new subdirectory is also called 'practice'.

2.3.2 Extracting Files from an Archive

(This message will disappear, once this node revised.)

Creating an archive is only half the job—there would be no point in storing files in an archive if you couldn't retrieve them. To extract files from an archive, use the '`--extract`' ('`-x`') operation.

To extract specific files, use their names as file name arguments. If you use a directory name as a file name argument, `tar` extracts all the files (including subdirectories) in that directory. If you don't use any file name arguments, `tar` extracts all the files in the archive.

Note: `tar` will extract an archive member into the file system without checking to see if there is already a file with the archive member's file name. If there is a file with that name, `tar` will *overwrite* that file and its contents will be lost.

FIXME: xref keep-old

2.3.3 Extracting Specific Files

(This message will disappear, once this node revised.)

To extract specific files, specify them using file name arguments.

In an example above, you created the archive file '`~/practice/records`', which contained the files '`blues`', '`folk`' and '`jazz`' in the '`practice`' directory. If, for some reason, you were to lose one of those text files ('`rm ~/practice/blues`'), you could extract it from the archive file.

First, change into the '`practice`' directory. Then,

- Invoke `tar` and specify the '`--extract`' ('`-x`') or '`--get`' operation.
- Specify the archive that the files will be extracted from—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').
- Specify the files to extract, using file name arguments (if you don't specify any files, `tar` extracts all the archive members)

```
% tar --extract --file=records blues
```

If you list the contents of the directory, you will see that '`blues`' is back:

```
% ls
folk
jazz
records
blues
```

2.3.4 Extracting Directories

(This message will disappear, once this node revised.)

To extract a directory and all the files it contains, use the directory's name as a file name argument in conjunction with '`tar +extract`'. Remember–`tar` stores and extracts file names relative to the working directory.

In a previous example you stored the directory '`~/practice`' in the archive file '`~/music`'. If you delete the contents of '`practice`', you can restore them using `tar`.

First, change into the '`practice`' subdirectory ('`cd ~/practice`'). Then, remove all the files in '`~/practice`' ('`rm *`'). If you list the contents of the directory, you should now see that it is empty:

```
%ls
%
```

Let's try to restore the contents of '`practice`' by extracting them from the archive file '`~/music`':

```
tar --extract --file=~/music practice
```

Now, list the contents of '`practice`' again:

```
%ls
practice
```

What happened to the files? When you created '`~/music`', your working directory was your home directory. When you extracted '`~/music`', your working directory was '`~/practice`'. `tar` stored the files in '`practice`' relative to your home directory, and then extracted them relative to '`~/practice`'. The files are now in a new subdirectory, called '`~/practice/practice`'.

To restore your files to their old positions, delete the new directory and its contents, and then redo the example above with your home directory as the working directory:

```
% rm ~/practice/practice/*
% rmdir practice
% cd ..
% tar --extract --file=music practice
```

(`tar` will report that it is unable to create the directory '`~/practice`' because it already exists. This will not effect the extraction of the other archive members.)

2.4 How to Add Files to Existing Archives

(This message will disappear, once this node revised.)

If you want to add files to an existing archive, then don't use '`--create`' ('`-c`'). That will erase the archive and create a new one in its place. Instead, use '`--append`' ('`-r`'). The command '`tar --append --file=afiles.tar arbalest`' would add the file '`arbalest`' to the existing archive '`afiles.tar`'. The archive must already exist in order to use '`--append`' ('`-r`').

As with '`--create`' ('`-c`'), the member names of the newly added files will be the exact same as their names given on the command line. The '`--verbose`' ('`-v`') option will print out the names of the files as they are written into the archive.

If you add a file to an archive using '`--append`' ('`-r`') with the same name as an archive member already present in the archive, then the old member is not deleted. What does happen, however, is somewhat complex.

FIXME: xref Multiple Members with the Same Name.

If you want to replace an archive member, use '`--delete`' first, and then use '`--append`' ('`-r`').

FIXME: we want people to use the script for backups, so I an not going
FIXME: to use backups as an explanation in the tutorial. (people can still
FIXME: do it if they really want to)

While you can use `tar` to create a new archive every time you want to store a file, it is more sometimes efficient to add files to an existing archive.

To add new files to an existing archive, use the '`--append`' ('`-r`') operation. To add newer versions of archive members to an archive, use the '`--update`' ('`-u`') operation.

While you can use `tar` to create an archive of an entire directory or directory tree, it is more efficient when performing backups to only archive those files which have been newly created or changed since the last backup.

To add new files to an existing archive, or to add newer versions of old files, you can use the '`--append`' ('`-r`') operation, or the '`--update`' ('`-u`') operation.

2.4.1 Appending Files to an Archive

(This message will disappear, once this node revised.)

The simplest method of adding a file to an already existing archive is the '`--append`' ('`-r`') operation, which writes the files specified into the archive without regard to whether or not they are already among the archived files. When you use '`--append`' ('`-r`') you must specify file name arguments, there is no default. If you specify a file that already exists in the archive another copy of the file will be added to the end of the archive anyway.

In the previous examples you created a file called '`rock`' in the practice directory which did not exist in either the archive file '`records`', in the practice directory, or the archive file '`music`', in the home directory. To add '`rock`' to '`records`', you would, while in the practice directory:

- Invoke `tar` and specify the operation to add a file—'`--append`' ('`-r`').
- Specify the name of the archive to which the file will be added—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').
- Specify the name(s) of the file(s) to be added to the archive as the file name argument(s)

```
% tar --append --file=records rock
```

If you now use the '`--list`' ('`-t`') operation, you will see that '`rock`' has been added to the archive:

```
% tar --list --file=records
blues
folk
jazz
rock
```

While all newly created files have now been added to '`records`', it is still not current with respect to the contents of the practice directory. If you recall from the examples using '`--compare`' ('`-d`') above, '`blues`' was changed after the archive '`records`' was created. It is simple, however, to use '`--append`' ('`-r`') to correct the problem:

```
% tar --append --verbose --file=records blues
blues
```

Because you specified the '`--verbose`' ('`-v`') option, `tar` has printed the name of the file being appended as it was acted on. If you now use `tar` with the '`--list`' ('`-t`') option specified to get the contents of the archive, you will optain the following:

```
% tar --list -f records
blues
folk
jazz
```

```
rock
blues
```

The newest version of '`blues`' is now at the end of the archive. Because files are extracted from archives in the order in which they appear in the archive, and because extracted files are given the same names in the file system as they are stored under in the archive, when the files in '`records`' are extracted the newer version of '`blues`' (which has the same name as the older) will overwrite the version stored first.

FIXME: xref Keep Old Files

FIXME: –update wont take a directory argument if files that have been
FIXME: archived from that directory are now no longer in it. (I assume
FIXME: because it looks in the archive first for the directory listing.)
FIXME: this is a bug

The simplest method of adding a file to an existing archive is the '`--append`' ('`-r`') operation, which writes files into the archive without regard to whether or not they are already archive members. When you use '`--append`' ('`-r`') you must use file name arguments; there is no default. If you specify a file that is already stored in the archive, `tar` adds another copy of the file to the archive.

If you have been following the previous examples, you should have a text file called '`~/practice/rock`' which has not been stored in either the archive file '`~/practice/records`', or the archive file '`~/music`'. To add '`rock`' to '`records`', first make '`practice`' the working directory ('`cd practice`'). Then:

- Invoke `tar` and specify the '`--append`' ('`-r`') operation.
- Specify the archive to which the file will be added—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').
- Specify the files to be added to the archive, using file name arguments

For example:

```
% tar --append --file=records rock
```

If you list the archive members in '`records`', you will see that '`rock`' has been added to the archive:

```
% tar --list --file=records
blues
folk
jazz
rock
```

FIXME: this should be some kind of node.

You can use '`--append`' ('`-r`') to keep archive members current with active files. Because '`--append`' ('`-r`') stores a file whether or not there is already an archive member with the same file name, you can use '`--append`' ('`-r`') to add newer versions of archive members to an archive. When you extract the file, only the version stored last will wind up in the file system. Because '`--extract`' ('`-x`') extracts files from an archive in sequence, and overwrites files with the same name in the file system, if a file name appears more than once in an archive the last version of the file will overwrite the previous versions which have just been extracted.

If you recall from the examples using '`--compare`' ('`-d`') above, '`blues`' was changed after the archive '`records`' was created. It is simple, however, to use '`--append`' ('`-r`') to add the new version of '`blues`' to '`records`':

```
% tar --append --verbose --file=records blues
blues
```

If you now list the contents of the archive, you will obtain the following:

```
% tar --list -f records
blues
folk
jazz
rock
blues
```

The newest version of '`blues`' is at the end of the archive. When the files in '`records`' are extracted, the newer version of '`blues`' (which has the same name as the older) will overwrite the version stored first. When '`--extract`' ('`-x`') is finished, only the newer version of '`blues`' is in the file system.

FIXME: xref keep-old-files

2.4.2 Updating Files in an Archive

(This message will disappear, once this node revised.)

While the '`--append`' ('`-r`') option is useful for updating files in an archive, to keep an archive current with '`--append`' ('`-r`') you must first use the '`--compare`' ('`-d`') or '`--list`' ('`-t`') options to determine what files have been changed (or be willing to waste space by adding identical copies of archived files to the ends of archives). It is simpler to use the '`--update`' ('`-u`') operation, and let `tar` do the work for you.

The '`--update`' ('`-u`') option causes `tar` to add files to the end of an archive, just like the '`--append`' ('`-r`') option. When you invoke `tar` with the '`--update`' ('`-u`') option specified you must specify file name arguments. Unlike '`--append`' ('`-r`'), the '`--update`' ('`-u`') option causes `tar` to check the archive to be updated to see if the specified file is already stored. If the file (or one with the same name) is already in the archive, `tar` checks the modification date of the file in the archive and compares it to the file of the same name in the file system. The file is only appended to the archive if it is new or if its modification date has changed to a later one.

FIXME: xref After-Date

To see the '`--update`' ('`-u`') option at work, create a new file, '`classical`', in your practice directory, and add a line to the file '`blues`', using any text editor. Then invoke `tar` with the '`--update`' ('`-u`') operation and the '`--verbose`' ('`-v`') option specified, using the names of all the files in the practice directory as file name arguments:

```
% tar --update --verbose --file=records blues folk rock classical
blues
classical
%
```

Because we have specified verbose mode, `tar` prints out the names of the files it is working on, which in this case are the names of the files that needed to be updated. If you now invoke `tar` with the '`--list`' ('`-t`') operation specified, to generate a listing of the files in the archive, you will see that '`blues`' and '`classical`' have been added to its end.

[The reason `tar` does not overwrite the older file when updating it is because writing to the middle of a section of tape is a difficult process. Tapes are not designed to go backward. Even if they were, imagine what would happen if the newer version were longer than the older one.]

To keep archive members up to date with their counterparts of the same name in the file system, use the '`--update`' ('`-u`') option. This adds a specified file to an archive if no file of that name is already stored in the archive. If there is already an archive member with the same name, `tar` checks the modification date of the archive member, and adds the file only if its modification date is later. If a file is stored in the archive but no longer exists under the same name in the active file system, `tar` reports an error.

You could use the '`--append`' ('`-r`') option to keep an archive current, but do so you would either have to use the '`--compare`' ('`-d`') and '`--list`' ('`-t`') options to determine what files needed to be re-archived (which could waste a lot of time), or you would have to be willing to add identical copies of already archived files to the archive (which could waste a lot of space).

You must use file name arguments with the '`--update`' ('`-u`') operation–if you don't specify any files, `tar` won't act on any files.

To see the '`--update`' ('`-u`') option at work, create a new file, '`~/practice/classical`', and modify the file '`~/practice/blues`' (you can use a text editor, such as Emacs, to do both these things). Then, with '`practice`' as your working directory, invoke `tar` with the '`--update`' ('`-u`') option, using the names of all the files in the practice directory as file name arguments, and specifying the '`--verbose`' ('`-v`') option:

```
% tar --update --verbose --file=records blues folk rock classical
blues
classical
%
```

Because you specified verbose mode, `tar` printed out the names of the files it acted on. If you now list the archive members of the archive, ('`tar --list --file=records`'), you will see that the file '`classical`' and another version of the file '`blues`' have been added to '`records`'.

Note: When you update an archive, `tar` does not overwrite old archive members when it stores newer versions of a file. This is because archive members appear in an archive in the order in which they are stored, and some archive devices do not allow writing in the middle of an archive.

2.4.3 Concatenating Archives

(This message will disappear, once this node revised.)

Rather than adding individual files onto the end of an archive, it may be more convenient to add archives themselves onto the end of an archive. While it may seem intuitive to use `cat`, the utility for adding files together, for this purpose, archives created by `tar` incorporate an end of file marker which must be removed if the concatenated archives are to be read properly as one archive

FIXME: xref Ignore zeros.

To add archives to the end of another archive, therefore, you should use the '`--concatenate`' ('`-A`') operation.

In earlier examples you created an archive file, '`music`', in your home directory. You have, however, since changed the contents of the '`practice`' directory which was stored in that archive. '`records`', the archive file in the '`practice`' directory, has recently been updated, and contains a current version of the files in '`practice`'. Rather than update the contents of '`music`', let's add '`records`' to it.

- Change into the home directory ('`cd ..`')
- Invoke `tar`, and specify the operation to add archives to the end of another archive—'`--concatenate`' ('`-A`').
- Specify the name of the archive file to be added to—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').
- Specify the file name arguments, which are, unusually, the names of archive files. Remember to include the directory name in the file name, if the archive file is not in your working directory.

```
% cd ..
% tar --concatenate --file=music practice/records
```

Rather than list the new contents of '`music`', let's extract all the files and see what happens.

To concatenate archive files, use the '`--concatenate`' ('`-A`') option. This operation adds other archives to the end of an archive. While it may seem intuitive to concatenate archives using `cat`, the utility for adding files together, archive files which have been "catted" together cannot be read properly by `tar`. Archive files incorporate an end of file marker–if archives are concatenated using `cat`, this marker will appear before the end of the new archive. This will interfere with operations on that archive.

FIXME: xref ignore-zeros

In earlier examples, you stored the '`~/practice`' directory in an archive file, '`~/music`'. If you have been following the examples, you have since changed the contents of the '`~/practice`' directory. There is a current version of the files in the '`practice`' directory, however, stored in the archive file '`~/practice/records`'.

To store current versions of the files in '`practice`' in the archive file '`music`', you can use '`--concatenate`' ('`-A`') to add the archive file '`~/practice/records`' to '`music`'. First, make sure you are in your home directory ('`cd ~`'). Then:

- Invoke `tar`, and specify the '`--concatenate`' ('`-A`') operation.
- Specify the archive file to be added to—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').
- Specify the archives to be added, using file name arguments. In this case, the file name arguments are, unusually, the names of archive files. (Remember to include the path in the archive name, if the archive file is not in your working directory.)

```
% cd ~
% tar --concatenate --file=music practice/records
```

If you now list the contents of the '`music`', you see it now contains the archive members of '`practice/records`':

```
%tar --list --file=music
blues
folk
jazz
rock
blues
practice/blues
practice/folk
practice/jazz
practice/rock
practice/blues
practice/classical
```

2.5 How to Delete Members from Archives

(This message will disappear, once this node revised.)

You can delete members from an archive using '`--delete`'. Specify the name of the archive with '`--file=`*`archive-name`*' ('`-f` *`archive-name`*'). List the member names of the members to be deleted. (If you list no member names, then nothing will be deleted.) The '`--verbose`' ('`-v`') option will cause `tar` to print the names of the members as they are deleted. As with '`--extract`' ('`-x`'), it is important that you give the exact member names when using '`tar --delete`'. Use '`--list`' ('`-t`') to find out the exact member names in an archive.

FIXME: xref Listing Archives.

The '`--delete`' option only works with archives stored on disk. You cannot delete members from an archive stored on a tape.

In some instances, it may be advantageous to remove some files from an archive stored on disk (it is never advantageous to delete files from an archive stored on tape—the linear nature of tape storage makes this action likely to scramble the archive). You can use the '`--delete`' operation to remove files from an archive. The names of files to be removed must be specified to `tar` as file name arguments. All versions of the named file are removed from the archive. Execution of the '`--delete`' operation can be very slow.

To delete all versions of the file '`blues`' from the archive '`records`' in the '`practice`' directory, make sure you are in that directory, and then,

- List the contents of the archive file '`records`' (see above for the steps involved) to insure that the file(s) you wish to delete are stored in the archive. (This step is optional)
- Invoke `tar` and specify the operation to delete files from an archive ('`--delete`')
- Specify the name of the archive file that the file(s) will be deleted—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').
- Specify the name(s) of the file(s) to be deleted, as file name arguments
- List the contents of the archive file again—note that the files have been removed. (this step is also optional)

  ```
  % tar --list --file=records
  blues
  folk
  ```

```
jazz
% tar --delete --file=records blues
% tar --list --file=records
folk
jazz
%
```

In some instances, you may want to remove some files from an archive stored on disk

> *Caution:* you should never delete files from an archive stored on tape–because of the linear nature of tape storage, doing this is likely to scramble the archive.

To remove archive members from an archive, use the '`--delete`' operation. You must specify the names of files to be removed as file name arguments. All versions of the named file are removed from the archive.

Execution of the '`--delete`' operation can be very slow.

To delete all versions of the file '`blues`' from the archive '`records`' in the '`practice`' directory, make sure you are in that directory, and then:

- List the contents of the archive file '`records`' (see above for the steps involved) to insure that the file(s) you wish to delete are stored in the archive. (This step is optional)
- Invoke `tar` and specify the '`--delete`' operation ('`--delete`').
- Specify the name of the archive file that the file(s) will be deleted from—'`--file=`*`archive-name`*' ('`-f` *`archive-name`*').
- Specify the files to be deleted, using file name arguments.
- List the contents of the archive file again—note that the files have been removed. (this step is also optional)

  ```
  % tar --list --file=records
  blues
  folk
  jazz
  % tar --delete --file=records blues
  % tar --list --file=records
  folk
  jazz
  %
  ```

3 Invoking GNU `tar`

(This message will disappear, once this node revised.)

3.1 General Synopsis of `tar`

The usual way to invoke `tar` is:

```
tar option... [name]...
```

You can actually type in arguments in any order, but in this manual the options always precede the other arguments, to make examples easier to understand. Further, the option stating the main operation mode (the `tar` *main command*) is usually given first.

There are surely many options to `tar`, and three different style for writing them: mnemonic options, short options, and old options. These styles are discussed below. Some options make sense with any main command, while others are meaningful only with particular main commands. One option should state the main command, all others are truly optional.

Beware that `tar` options are case sensitive. For example, [No value for "List"] or [No value for "List"] options are not equivalent to '`--list`' ('`-t`'), in fact, they do not even exist. Options '`-T`' and '`-t`' are different options, the first requires an argument for stating the name of a file providing a list of *name*s, the second does not require an argument and is another way to write '`--list`' ('`-t`').

Each *name* in the synopsis above is interpreted as an archive member name when the main command is one of '`--compare`' ('`-d`'), '`--delete`', '`--extract`' ('`-x`'), '`--list`' ('`-t`') or '`--update`' ('`-u`'). For all other main commands, *name*s are interpreted as the names of files (including directories) in the file system. `tar` interprets relative file names as being relative to the working directory.

`tar` will make all file names relative (by removing leading '`/`'s when archiving or restoring files), unless you specify otherwise (using the '`--absolute-names`' ('`-P`') option).

FIXME: xref File Name
Interpretation

, for more information about '`--absolute-names`' ('`-P`').

FIXME: yet another node name that is probably wrong.

The distinction between file names and archive member names is especially important when shell globbing is used, and sometimes a source of confusion for newcomers. *Globbing* is the operation by which *wildcard* characters, '`*`' or '`?`' for example, are replaced and expanded into all existing files matching the given pattern. The problem is that shells may only glob using existing files in the file system. Only `tar` may glob on archive members, so when needed, you must ensure that wildcard characters reach `tar` without being interpreted by the shell first. Using a backslash before '`*`' or '`?`', or putting the whole argument between quotes, is usually sufficient for this.

Even if *name*s are often specified on the command line, they can also be read from a text file in the file system, using the '`--files-from=file-of-names`' ('`-T file-of-names`') option.

Each of the following subsection groups some options under a common functionality.

You can use `tar` to store files in an archive, to extract them from an archive, and to do other types of archive manipulation. The primary argument to `tar`, which is called the *operation*, specifies which action to take. The other arguments to `tar` are either *options*, which change the way `tar` performs an operation, or *file names*, which specify the files `tar` is to act on. The typical `tar` command line syntax is:

GNU `tar` returns only a few exit statuses. I'm really aiming simplicity in that area, for now. If you are not using the '`--compare`' ('`-d`') option, zero means that everything went well, besides maybe innocuous warnings. Nonzero means that something went wrong. Right now, as of today, "nonzero" is almost always 2, except for remote operations, where it may be 128.

3.2 Many Styles for Options

3.2.1 Mnemonic Option Style

Each *option* has at least one mnemonic option name starting with two dashes in a row, v.g. '`--list`' ('`-t`'). The mnemonic option names are more legible than the corresponding short or old option names, you may prefer them if you highly praise clarity. It sometimes happens that a single mnemonic option has many different different names, which are then synonymous. In addition, mnemonic option names can be given unique abbreviations. For example, '`--cre`' can be used in place of '`--create`' because there is no other mnemonic option which begins with '`cre`'.

Some options require an argument. This is the case of the '`--file=`*`archive-name`*' ('`-f` *`archive-name`*') option, which tells the name of the `tar` archive. The argument of a mnemonic option is usually given right after the option itself, and introduced by an equal sign. For example, the '`--file=`*`archive-name`*' ('`-f` *`archive-name`*') option is given the '`archive.tar`' file as argument by using the notation '`--file=archive.tar`' for the mnemonic option.

Mnemonic options are meant to be obvious and easy to remember, possibly more so than their corresponding short options, below. For example:

```
tar --create --verbose --block-size=20 --file=/dev/rmt0
```

gives a fairly good set of hints about what the command does, even for those not fully acquainted with `tar`.

3.2.2 Short Option Style

Most options, but not all of them, also have a short option name starting with a single dash, and followed by a single character, v.g. '`-t`'. In fact, '`-t`' is exactly the short option name for the mnemonic option '`--list`' ('`-t`'), both having exactly the same meaning. The forms are absolutely identical in function.

The short option names are faster to type than mnemonic option names. All along this manual, whenever a mnemonic option name is given, its equivalent short option name follows between parentheses, if such a short option name exists.

Short options which require arguments use the immediately following argument, so it may be thought as being merely introduced right after it, usually separate by white space. It is also possible to stick the argument right after the short option name, using no intervening space. So one might write '`-f archive.tar`' or '`-farchive.tar`' instead of using

'`--file=archive.tar`'. Both '`--file=`*`archive-name`*' and '`-f` *`archive-name`*' denote the option to give the archive a non-default name, which in the example is '`archive.tar`'. When the option is given separately, its argument follows it, as is usual for Unix programs. For example:

```
tar -c -v -b 20 -f /dev/rmt0
```

Short options letters may be lumped together, but contrary to old options, they do not necessarily have to. When short options are nevertheless coalesced, use a single dash for them all. Only the last one in such a set is allowed to have an argument. (Clustering many options, the last of which taking an argument, seems to be fairly opaque writing to me. I would even like that GNU `getopt` be helpful enough to make this illegal.)

If you move short options in the command, be sure to move their arguments along with them, if any.

3.2.3 Old Option Style

(This message will disappear, once this node revised.)

Old options are single letters not preceeded by any dash at all, and appearing *only* in the position immediately following the '`tar`' keyword in the command, after some white space. The letter of an old option is exactly the same letter as the corresponding short option. For example, the old option '`t`' is the same as the short option '`-t`', and consequently, the same as long option '`--list`' ('`-t`').

As far as we know, all `tar` programs, GNU and non-GNU, support old options. GNU `tar` supports them not only for historical reasons, but also because many people are used to them.

All old options should be written as a single argument, without separating spaces, by lumping together all letters specifying these options. This set of letters should be the first to appear on the command line, after the `tar` program name; old options cannot appear anywhere else. Then, for any old option required an argument, the argument should follow on the command line. Arguments to the options should appear in the same order as the letters to which they correspond. The `tar` command synopsis might be rewritten:

```
tar letter... [argument]... [option]... [name]...
```

when old options are being used.

This command syntax is useful because it lets you type the single letter forms of the operation and options as a single argument to `tar`, without writing preceding '`-`'s or inserting spaces between letters. '`tar cv`' or '`tar -cv`' are equivalent to '`tar -c -v`'.

For compatibility with Unix `tar`, the first argument can contain an option letter (or a cluster of option letters) *not* introduced by a dash; for example, '`tar cv`' specifies the option '`-v`' in addition to the command '`-c`'. When options that need arguments are given together with the command, all the associated arguments follow, in the same order as the options. Thus, the example above could also be written in the old style as follows:

```
tar cvbf 20 /dev/rmt0
```

Here '`20`' is the argument of '`-b`' and '`/dev/rmt0`' is the argument of '`-f`'.

On the other hand, this old style syntax makes it difficult to match option letters with their corresponding arguments, and is often confusing. In the command '`tar cvbf 20 /dev/rmt0`', for example, '`20`' is the argument for '`-b`', '`/dev/rmt0`' is the argument for

‘`-f`’, and ‘`-v`’ does not have a corresponding argument. Even using short options like in ‘`tar -c -v -b 20 -f /dev/rmt0`’ is clearer, putting all arguments next to the option they pertain to.

If you want to reorder the letters in the old option argument, be sure to appropriately reorder any corresponding argument.

This old way of writing `tar` options can surprise even experienced users. For example, the two commands:

```
tar cfz archive.tar.gz file
tar -cfz archive.tar.gz file
```

are quite different. The first example uses ‘`archive.tar.gz`’ as the value for option ‘`f`’ and recognizes the option ‘`z`’. The second example, however, uses ‘`z`’ as the value for option ‘`f`’—probably not what was intended. (I find it quite inelegant that `getopt` batches the remaining ‘`z`’ as the value for ‘`f`’. I think that clarity dictates that clustering of option letters, when some require arguments, should be diagnosed and disallowed. But compatibility with traditional systems dictates it.) This second example could be corrected in many ways, among which:

```
tar -czf archive.tar.gz file
tar -cf archive.tar.gz -z file
tar cf archive.tar.gz -z file
```

3.2.4 Mixing Option Styles

All three styles may be intermixed in a single `tar` command, as long as the rules for each style are fully respected.

In GNU `tar` up to 1.11.6, using old style options was cutting out the possibility of using many options not having short forms. Many users rightly expressed their frustration at fighting with the current `tar` option decoder, so I changed it. However, GNU `tar` needs to be compatible with other `tar`s, in the things that other `tar`s can do. Short options should provide upward compatibility. So, if the current option decoding raises incompatibilities, please let me know.

Old style options and modern options may be mixed on a single call to the `tar` program. However, old style options should be introduced in the first argument only; modern options may be given only after all arguments to old style options have been collected. If this rule is not respected, a modern option might be falsely interpreted as the value of the argument to one of the old style options.

For example, currently, all the following commands are wholly equivalent, and illustrate many combinations and orderings of option styles.

```
tar --create --file=archive.tar
tar --create -f archive.tar
tar --create -farchive.tar
tar --file=archive.tar --create
tar --file=archive.tar -c
tar -c --file=archive.tar
tar -c -f archive.tar
tar -c -farchive.tar
tar -cf archive.tar
tar -cfarchive.tar
tar -f archive.tar --create
tar -f archive.tar -c
tar -farchive.tar --create
tar -farchive.tar -c
tar c --file=archive.tar
tar c -f archive.tar
tar c -farchive.tar
tar cf archive.tar
tar f archive.tar --create
tar f archive.tar -c
tar fc archive.tar
```

On the other hand, the following commands are *not* equivalent to the previous set:

```
tar -f -c archive.tar
tar -fc archive.tar
tar -fcarchive.tar
tar -farchive.tarc
tar cfarchive.tar
```

These last examples mean something completely different of what the user might have intended. The first four specify that the `tar` archive would be a file named '`-c`', '`c`', '`carchive.tar`' or '`archive.tarc`', respectively. The first two examples also specify a single non-option, *name* argument having value '`archive.tar`'. The last example contains only old style option letters (repeating option '`c`' twice) and no argument value.

3.3 All Available Options

(This message will disappear, once this node revised.)

Options change the way `tar` performs an operation.

'`--absolute-names`'
'`--after-date=`*`date`*'

Limit the operation to files changed after the given date.

FIXME: xref File Exclusion

.

'`--block-size=`*`number`*'

Specify the blocking factor of an archive.

FIXME: xref Blocking Factor

.

'`--compress`'

Specify a compressed archive.

FIXME: xref Compressed Archives

.

'`--compress-block.`'

Create a whole block sized compressed archive.

FIXME: xref Compressed Archives

.

‘`--confirmation`’
Solicit confirmation for each file.

FIXME: xref Interactive Operation

FIXME: –selective should be a synonym.

‘`--dereference`’
Treat a symbolic link as an alternate name for the file the link points to.

FIXME: xref Symbolic Links

.

‘`--directory=‘directory’`’
Change the working directory.

FIXME: xref Changing Working Directory

.

‘`--exclude=`*pattern*’
Exclude files which match the regular expression *pattern*.

FIXME: xref File Exclusion

.

‘`--exclude-from=‘file’`’
Exclude files which match any of the regular expressions listed in the file ‘`file`’.

FIXME: xref File Exclusion

.

‘`--file=`*archive-name*’
Name the archive.

FIXME: xref Archive Name

).

‘`--files-from=‘file’`’
Read file name arguments from a file on the file system.

FIXME: xref File Name Lists

.

‘`--ignore-umask`’
Set modes of extracted files to those recorded in the archive.

FIXME: xref File Writing Options

.

‘`--ignore-zeros`’
Ignore end-of-archive entries.

FIXME: xref Archive Reading Options

.

FIXME: this should be changed to –ignore-end

‘`--tape-length=`*`n`*` (-L)`’
FIXME: alternate way of doing multi archive, will go to that length and
FIXME: prompts for new tape, automatically turns on multi-volume. this
FIXME: needs to be written into main body as well

‘`--info-script=`*`program-file`*’
Create a multi-volume archive via a script.

FIXME: xref Multi-Volume Archives

.

‘`--interactive`’
Ask for confirmation before performing any operation on a file or archive member.

‘`--keep-old-files`’
Prevent overwriting during extraction.

FIXME: xref File Writing Options

.

‘`--label=`*`archive-label`*’
Include an archive-label in the archive being created.

FIXME: xref Archive
Label

.

‘`--modification-time`’
Set the modification time of extracted files to the time they were extracted.

FIXME: xref File Writing Options

.

‘`--multi-volume`’
Specify a multi-volume archive.

FIXME: xref Multi-Volume Archives

.

‘`--newer=`*`date`*’
Limit the operation to files changed after the given date.

FIXME: xref File Exclusion

.

‘`--newer-mtime=`*`date`*’
Limit the operation to files modified after the given date.

FIXME: xref File
Exclusion

.

‘`--old`’ Create an old format archive.

FIXME: xref Old Style File Information

.

FIXME: did we agree this should go away as a synonym?

‘`--old-archive`’

Create an old format archive.

FIXME: xref Old Style File Information

.

‘`--one-file-system`’

Prevent `tar` from crossing file system boundaries when archiving.

FIXME: xref File Exclusion

.

‘`--portability`’

Create an old format archive.

FIXME: xref Old Style File Information

.

FIXME: was portability, may still need to be changed

‘`--preserve-order`’

Help process large lists of file names on machines with small amounts of memory.

FIXME: xref Archive Reading Options

.

‘`--preserve-permission`’

Set modes of extracted files to those recorded in the archive.

FIXME: xref File Writing Options

.

‘`--read-full-blocks`’

Read an archive with a smaller than specified block size or which contains incomplete blocks.

FIXME: xref Archive Reading Options

).

FIXME: should be –partial-blocks (!)

‘`--record-number`’

Print the record number where a message is generated.

FIXME: xref Additional Information

.

‘`--same-order`’

Help process large lists of file names on machines with small amounts of memory.

FIXME: xref Archive Reading Options

.

‘`--same-permission`’

Set the modes of extracted files to those recorded in the archive.

FIXME: xref File Writing Options

.

‘`--sparse`’
: Archive sparse files sparsely.

FIXME: xref Sparse Files

.

‘`--starting-file=`*`file name`*’
: Begin reading in the middle of an archive.

FIXME: xref Scarce Disk Space

.

‘`--to-stdout`’
: Write files to the standard output.

FIXME: xref File Writing Options

.

‘`--uncompress`’
: Specifdo a compressed archive.

FIXME: xref Compressed Archives

.

‘`-V` *`archive-label`*’
: Include an archive-label in the archive being created.

FIXME: xref Archive
Label

.

FIXME: was –volume

‘`--verbose`’
: Print the names of files or archive members as they are being operated on.

FIXME: xref Additional Information

.

‘`--verify`’
: Check for discrepancies in the archive immediately after it is written.

FIXME: xref Write Verification

.

‘`-B`’ Read an archive with a smaller than specified block size or which contains incomplete blocks.

FIXME: xref Archive Reading Options

).

‘`-K` *`file name`*’
: Begin reading in the middle of an archive.

FIXME: xref Scarce Disk Space

.

‘`-M`’
: Specify a multi-volume archive.

FIXME: xref Multi-Volume Archives

.

‘`-N` *`date`*’
: Limit operation to files changed after the given date.

FIXME: xref File Exclusion

.

‘`-O`’
: Write files to the standard output.

FIXME: xref File Writing Options

.

FIXME: - P is absolute names, add when resolved.

‘`-R`’
: Print the record number where a message is generated.

FIXME: xref Additional Information

.

‘`-S`’
: Archive sparse files sparsely.

FIXME: xref Sparse Files

.

‘`-T` *`file`*’
: Read file name arguments from a file on the file system.

FIXME: xref File Name Lists

.

‘`-W`’
: Check for discrepancies in the archive immediately after it is written.

FIXME: xref Write Verification

.

‘`-Z`’
: Specify a compressed archive.

FIXME: xref Compressed Archives

.

‘`-b` *`number`*’

Specify the blocking factor of an archive.

FIXME: xref Blocking Factor

.

‘`-f` *`archive-name`*’

Name the archive.

FIXME: xref Archive Name

).

‘`-h`’
: Treat a symbolic link as an alternate name for the file the link points to.

	FIXME: xref Symbolic Links .
‘`-i`’	Ignore end-of-archive entries. FIXME: xref Archive Reading Options .
‘`-k`’	Prevent overwriting during extraction. FIXME: xref File Writing Options .
‘`-l`’	Prevent `tar` from crossing file system boundaries when archiving. FIXME: xref File Exclusion .
‘`-m`’	Set the modification time of extracted files to the time they were extracted. FIXME: xref File Writing Options .
‘`-o`’	Create an old format archive. FIXME: xref Old Style File Information .
‘`-p`’	Set the modes of extracted files to those recorded in the archive. FIXME: xref File Writing Options .
‘`-s`’	Help process large lists of file names on machines with small amounts of memory. FIXME: xref Archive Reading Options .
‘`-v`’	Print the names of files or archive members they are being operated on. FIXME: xref Additional Information .
‘`-w`’	FIXME: see –interactive.
‘`-z`’	Specify a compressed archive. FIXME: xref Compressed Archives .
‘`-z -z`’	Create a whole block sized compressed archive. FIXME: xref Compressed Archives . FIXME: I would rather this were -Z. it is the only double letter short FIXME: form.

‘`-C ‘directory’`’

Change the working directory.

FIXME: xref Changing Working Directory

.

‘`-F` *`program-file`*’

Create a multi-volume archive via a script.

FIXME: xref Multi-Volume Archives

.

‘`-X ‘file’`’

Exclude files which match any of the regular expressions listed in the file ‘`file`’.

FIXME: xref File Exclusion

.

3.3.1 Device selection and switching

(This message will disappear, once this node revised.)

`-f [`*`hostname`*`:]`*`file`*
`--file=[`*`hostname`*`:]`*`file`*

Use archive file or device *file* on *hostname*.

FIXME: xref Device

.

`--force-local`

Archive file is local even if it contains a colon.

FIXME: xref Device

.

`--rsh-command=`*`command`*

Use remote *command* instead of `rsh`.

FIXME: xref Device

.

`-[0-7][lmh]`

Specify drive and density.

FIXME: xref Device

.

`-M`
`--multi-volume`

Create/list/extract multi-volume archive.

FIXME: xref Multi

.

`-L` *`num`*
`--tape-length=`*`num`*

Change tape after writing *num* x 1024 bytes.

FIXME: xref Multi

.

`-F` *`file`*
`--info-script=`*`file`*
`--new-volume-script=`*`file`*

Execute '`file`' at end of each tape. This implies '`--multi-volume`' ('`-M`')).

FIXME: xref Multi

.

3.3.2 Device blocking

(This message will disappear, once this node revised.)

`-b` *`blocks`*
`--block-size=`*`blocks`*

Set block size to $blocks * 512$ bytes.

FIXME: xref Blocking

.

`--block-compress`

Block the output of compression for tapes.

FIXME: xref Blocking

.

`-i`
`--ignore-zeros`

Ignore blocks of zeros in archive (means EOF).

FIXME: xref Blocking

.

`-B`
`--read-full-blocks`

Reblock as we read (for reading 4.2BSD pipes).

FIXME: xref Blocking

.

3.3.3 Old classification of options

(This message will disappear, once this node revised.)

The information here is to be revised and merged into the remainder of this document, possibly altering its structure.

Options could be regrouped in three categories:

General Options

Options that are always meaningful.

Creation Options

Options for creating or updating an archive.

Extraction Options
Options for listing or extracting files.

Here are the options that are always meaningful.

‘`-B` *`number`*’, ‘`--block-size` *`number`*’
‘`-f` *`filename`*’, ‘`--file` *`filename`*’
‘`-C` *`dir`*’, ‘`--directory` *`dir`*’
‘`-M`’, ‘`--multi-volume`’
‘`-N` *`date`*’, ‘`--after-date` *`date`*’
‘`-R`’, ‘`--record-number`’ (‘`-R`’)
‘`-T` *`filename`*’, ‘`--files-from` *`filename`*’
‘`-v`’, ‘`--verbose`’ (‘`-v`’)
‘`-w`’, ‘`--interactive`’
‘`-X` *`file`*’, ‘`--exclude` *`file`*’
‘`-z`’, ‘`-Z`’, ‘`--compress`’, ‘`--uncompress`’

Here are the options for creating or updating an archive. These options are used to control which files `tar` puts in an archive, or to control the format the archive is written in

FIXME: ref Format

. Except as noted elsewhere, these options are useful with the ‘`--create`’ (‘`-c`’), ‘`--append`’ (‘`-r`’), ‘`--update`’ (‘`-u`’), ‘`--concatenate`’ (‘`-A`’), and ‘`--delete`’ commands. Also note that the [No value for “read-full-block”] option

FIXME: (pxref Extraction Options),

is also useful with the ‘`--append`’ (‘`-r`’), ‘`--update`’ (‘`-u`’), ‘`--concatenate`’ (‘`-A`’), and ‘`--delete`’ commands.

‘`-G`’, ‘`--incremental`’
‘`-h`’, ‘`--dereference`’ (‘`-h`’)
‘`-l`’, ‘`--one-file-system`’
‘`-o`’, ‘`--old-archive`’ (‘`-o`’)
‘`--old`’, ‘`--portability`’
‘`-S`’, ‘`--sparse`’
‘`-V NAME`’, ‘`--volume NAME`’
‘`-W`’, ‘`--verify`’

Here are the options for listing or extracting files. The options in this section are meaningful with the ‘`--extract`’ (‘`-x`’) command. Unless otherwise stated, they are also meaningful with the ‘`--list`’ (‘`-t`’) command.

‘`-B`’, ‘`--read-full-blocks`’
‘`-G`’, ‘`--incremental`’
‘`-i`’, ‘`--ignore-zeros`’
‘`-k`’, ‘`--keep-old-files`’
‘`-K` *`filename`*’, ‘`--starting-file` *`filename`*’
‘`-m`’, ‘`--modification-time`’
‘`-O`’, ‘`--to-stdout`’
‘`-p`’, ‘`--same-permissions`’, ‘`--preserve-permissions`’
‘`-P`’, ‘`--absolute-names`’ (‘`-P`’)
‘`-s`’, ‘`--same-order`’, ‘`--preserve-order`’
‘`--preserve`’

4 Basic tar Operations

(This message will disappear, once this node revised.)

This chapter describes the basic operations supported by the `tar` program. A given invocation of `tar` will do exactly one of these operations.

An archive member in normally extracted into a file with the same name as the archive member. However, you can use the '`--to-stdout`' ('`-O`') to cause `tar` to write extracted archive members to standard output. If you extract multiple members, they appear on standard output concatenated, in the order they are found in the archive.

The '`--create`' ('`-c`') operation writes a new archive, and the '`--extract`' ('`-x`') operation reads files from an archive and writes them into the file system. You can use other `tar` operations to write new information into an existing archive (adding files to it, adding another archive to it, or deleting files from it), and you can read a list of the files in an archive without extracting it using the '`--list`' ('`-t`') operation.

The primary argument to `tar` is the *operation*, which specifies what `tar` does. `tar` can be used to:

- Add files to an existing archive—'`--append`' ('`-r`').
- Compare files in an archive with files in the file system—'`--compare`' ('`-d`') or '`--diff`'.
- Add archives to another archive—'`--concatenate`' ('`-A`').
- Create an archive—'`--create`' ('`-c`').
- Delete files from an archive—'`--delete`'.
- Extract files from an archive—'`--extract`' ('`-x`') or '`--get`'.
- List the files in an archive—'`--list`' ('`-t`').
- Update an archive by appending newer versions of already stored files—'`--update`' ('`-u`').

FIXME: xref Reading and Writing

, for more information about these operations.

Option arguments to `tar` change details of the operation, such as archive format, archive name, or level of user interaction. You can specify more than one option. All options are optional.

File Name arguments specify which files (including directory files) to archive, extract, delete or otherwise operate on.

If you don't use any file name arguments, '`--append`' ('`-r`'), '`--update`' ('`-u`') and '`--delete`' will do nothing. The other operations of `tar` will act on defaults.

When you use a file name argument to specify a directory file, `tar` acts on all the files in that directory, including sub-directories.

You must give exactly one option from the following list to `tar`. This option specifies the basic operation for `tar` to perform.

'`--create`'
'`-c`' Create a new archive

'`--catenate`'
'`--concatenate`'
'`-A`' Add the contents of one or more archives to another archive

‘`--append`’ ‘`-a`’	Add files to an existing archive
‘`--list`’ ‘`-t`’	List the members in an archive
‘`--delete`’	Delete members from an archive
‘`--extract`’ ‘`--get`’ ‘`-x`’	Extract members from an archive
‘`--compare`’ ‘`--diff`’ ‘`-d`’	Compare members in an archive with files in the file system
‘`--update`’ ‘`-u`’	Update an archive by appending newer versions of already stored files

The remaining options to `tar` change details of the operation, such as archive format, archive name, or level of user interaction. You can specify more than one option.

The remaining arguments are interpreted either as file names or as member names, depending on the basic operation `tar` is performing. For ‘`--append`’ (‘`-r`’) and ‘`--create`’ (‘`-c`’) these arguments specify the names of files (which must already exist) to place in the archive. For the remaining operation types, the additional arguments specify archive members to compare, delete, extract, list, or update. When naming archive members, you must give the exact name of the member in the archive, as it is printed by ‘`--list`’ (‘`-t`’). When naming files, the normal file name rules apply.

If you don’t use any additional arguments, ‘`--append`’ (‘`-r`’), ‘`--concatenate`’ (‘`-A`’), and ‘`--delete`’ will do nothing. Naturally, ‘`--create`’ (‘`-c`’) will make an empty archive if given no files to add. The other operations of `tar` (‘`--list`’ (‘`-t`’), ‘`--extract`’ (‘`-x`’), ‘`--compare`’ (‘`-d`’), and ‘`--update`’ (‘`-u`’)) will act on the entire contents of the archive.

If you give the name of a directory as either a file name or a member name, then `tar` acts recursively on all the files and directories beneath that directory. For example, the name ‘`/`’ identifies all the files in the filesystem to `tar`.

The operation argument to `tar` specifies which action you want to take.

‘`-A`’	Adds copies of an archive or archives to the end of another archive.
‘`-c`’	Creates a new archive.
‘`-d`’	Compares files in the archive with their counterparts in the file system, and reports differences in file size, mode, owner, modification date and contents.
‘`-r`’	Adds files to the end of the archive.
‘`-t`’	Prints a list of the contents of the archive.
‘`-x`’	Reads files from the archive and writes them into the active file system.

'`-u`'	Adds files to the end of the archive, but only if they are newer than their counterparts already in the archive, or if they do not already exist in the archive.
'`--catenate`'	Adds copies of an archive or archives to the end of another archive.
'`--append`'	Adds files to the end of the archive.
'`--append`'	Adds files to the end of the archive.
'`--catenate`'	Adds copies of an archive or archives to the end of another archive.
'`--compare`'	Compares files in the archive with their counterparts in the file system, and reports differences in file size, mode, owner, modification date and contents.
'`--concatenate`'	Adds copies of an archive or archives to the end of another archive.
'`--create`'	Creates a new archive.
'`--delete`'	Deletes files from the archive. All versions of the files are deleted.
'`--diff`'	Compares files in the archive with their counterparts in the file system, and reports differences in file size, mode, owner, modification date and contents.
'`--extract`'	Reads files from the archive and writes them into the active file system.
'`--get`'	Reads files from the archive and writes them into the active file system.
'`--list`'	Prints a list of the contents of the archive.
'`--update`'	Adds files to the end of the archive, but only if they are newer than their counterparts already in the archive, or if they do not already exist in the archive.
'`--version`'	Prints the version number of the `tar` program to the standard error.

The program `tar` can create an archive, extract files from an archive, modify an archive, or list an archive's contents. Each time you run `tar`, you must give a *command* to specify which one of these things you want to do.

The command must always be in the first argument to `tar`. This argument can also contain options (

FIXME: pxref Invoking tar

). For compatibility with Unix `tar`, the first argument is always treated as containing command and option letters even if it doesn't start with '`-`'. Thus, '`tar c`' is equivalent to '`tar -c`': both of them specify the '`--create`' ('`-c`') command to create an archive.

In addition, a set of long-named options are provided which can be used instead of or intermixed with the single-letter flags. The long-named options are meant to be easy to remember and logical, while the single letter flags may not always be. Long-named options begin with '`--`'.

Arguments after the first are either options, if they start with '`-`' or '`--`', or files to operate on.

The file names that you give as arguments are the files that `tar` will act on—for example, they are the files to put in the archive, or the files to extract from it. If you don't give any file name arguments, the default depends on which command you used. Some commands use all relevant files; some commands have no default and will report an error if you don't specify files.

If a file name argument actually names a directory, then that directory and all files and subdirectories (recursively) in it are used.

Here is a list of the `tar` commands:

`-c`
`--create` Create a new archive.

> This command tells `tar` to create a new archive that contains the file(s) specified on the command line. If you don't specify files, all the files in the current directory are used.
>
> If the archive file already exists, it is overwritten; the old contents are lost.

`-x`
`--extract`
`--get` Extract files from an archive.

> This command causes `tar` to extract the specified files from the archive. If no file names are given, all the files in the archive will be extracted.

`-t`
`--list` List the contents of an archive.

> This command causes `tar` to display a list of the files in the archive. If you specify file names, only the files that you specify will be mentioned (but each of them is mentioned only if it appears in the archive).

`-d`
`--diff`
`--compare`

> Find differences between an archive and the corresponding online files.
>
> This command causes `tar` to compare the archive with the files in the file system. It will report differences in file size, mode, owner, and contents. If a file exists in the archive, but not in the file system, `tar` will report this.
>
> If you specify file names, those files are compared with the tape and they must all exist in the archive. If you don't specify files, all the files in the archive are compared.

`-r`
`--append` Append files to the end of an archive.

This command causes `tar` to add the specified file(s) to the end of the archive. This assumes that the archive file already exists and is in the proper format (which probably means it was created previously with the `tar` program). If the archive is not in a format that `tar` understands, the results will be unpredictable.

You must specify the files to be used; there is no default.

`-u`
`--update` Only append files newer than the version in an archive.

This command causes `tar` to add the specified files to the end of the archive, like '`--append`' ('`-r`'), but only when a file doesn't already exist in the archive or is newer than the version in the archive (the last-modification time is compared). Adding files to the end of an archive can be very slow.

You must specify the files to be used; there is no default.

`-A`
`--catenate`
`--concatenate`

Append existing archives to another archive.

This command is used for concatenating several archive files into one big archive file. The files to operate on should all be archive files. They are all appended to the end of the archive file which `tar` works on. (The other files are not changed).

You might be tempted to use `cat` for this, but it won't ordinarily work. A `tar` archive contains data which indicates the end of the archive, so appended material is ignored. This command works because it removes the end-of-archive markers from the middle of the result.

`--delete` Delete from the archive (not on tapes!).

This command causes `tar` to delete the specified files from the archive. This command is extremely slow. Warning: Use of this command on archives stored on magnetic tape may result in a scrambled archive. There is no safe way (except for completely re-writing the archive) to delete files from a magnetic tape.

The program `tar` can create an archive, extract files from an archive, modify an archive, or list an archive's contents. Each time you run `tar`, you must give a *command* to specify which one of these things you want to do.

The command must always be in the first argument to `tar`. This argument can also contain options (

FIXME: pxref Invoking tar

). For compatibility with Unix `tar`, the first argument is always treated as containing command and option letters even if it doesn't start with '`-`'. Thus, '`tar c`' is equivalent to '`tar -c`': both of them specify the '`--create`' ('`-c`') command to create an archive.

In addition, a set of long-named options are provided which can be used instead of or intermixed with the single-letter flags. The long-named options are meant to be easy to remember and logical, while the single letter flags may not always be. Long-named options begin with '`--`'.

Arguments after the first are either options, if they start with '`-`' or '`--`', or files to operate on.

The file names that you give as arguments are the files that `tar` will act on—for example, they are the files to put in the archive, or the files to extract from it. If you don't give any file name arguments, the default depends on which command you used. Some commands use all relevant files; some commands have no default and will report an error if you don't specify files.

If a file name argument actually names a directory, then that directory and all files and subdirectories (recursively) in it are used.

Modifying Archives

Once an archive is created, you can add new archive members to it, add the contents of another archive, add newer versions of members already stored, or delete archive members already stored.

To find out what files are already stored in an archive, use '`tar --list --file=`*`archive-name`*'.

FIXME: xref Listing Contents

.

4.1 Creating a New Archive

(This message will disappear, once this node revised.)

The '`--create`' ('`-c`') option causes `tar` to create a new archive. The files to be archived are then named on the command line. Each file will be added to the archive with a member name exactly the same as the name given on the command line. (When you give an absolute file name `tar` actually modifies it slightly,

FIXME: ref Absolute
Names

.) If you list no files to be archived, then an empty archive is created.

If there are two many files to conveniently list on the command line, you can list the names in a file, and `tar` will read that file.

FIXME: xref Reading Names from a File

.

If you name a directory, then `tar` will archive not only the directory, but all its contents, recursively. For example, if you name '`/`', then `tar` will archive the entire filesystem.

Do not use the option to add files to an existing archive; it will delete the archive and write a new one. Use '`--append`' ('`-r`') instead. (

FIXME: xref Adding to an Existing Archive

.)

There are various ways of causing `tar` to skip over some files, and not archive them.

FIXME: xref Specifying Names to tar

.

FIXME: operations should probably have examples, not tables.

To create an archive, use '`--create`' ('`-c`'). To name the archive, use '`--file=`*`archive-name`*' in conjunction with the '`--create`' ('`-c`') operation (

FIXME: pxref Archive Name

). If you do not name the archive, `tar` uses the value of the environment variable `TAPE` as the file name for the archive, or, if that is not available, `tar` uses a default archive name, usually that for tape unit zero.

FIXME: xref Archive Name

, for more information about specifying an archive name.

The following example creates an archive named '`stooges`', containing the files '`larry`', '`moe`' and '`curley`':

```
tar --create --file=stooges larry moe curley
```

If you specify a directory name as a file name argument, `tar` will archive all the files in that directory. The following example creates an archive named '`hail/hail/fredonia`', containing the contents of the directory '`marx`':

```
tar --create --file=hail/hail/fredonia marx
```

If you don't specify files to put in the archive, `tar` archives all the files in the working directory. The following example creates an archive named '`home`' containing all the files in the working directory:

```
tar --create --file=home
```

FIXME: xref File Name Lists

, for other ways to specify files to archive.

Note: In the example above, an archive containing all the files in the working directory is being written to the working directory. GNU `tar` stores files in the working directory in an archive which is itself in the working directory without falling into an infinite loop. Other versions of `tar` may fall into this trap.

4.2 Adding to an Existing Archive

(This message will disappear, once this node revised.)

The '`--append`' ('`-r`') option will case `tar` to add new files to an existing archive. It interprets file names and member names in exactly the same manner as '`--create`' ('`-c`'). Nothing happens if you don't list any names.

This option never deletes members. If a new member is added under the same name as an existing member, then both will be in the archive, with the new member after the old one. For information on how this affects reading the archive,

FIXME: ref Multiple Members with the Same Name

.

This operation cannot be performed on some tape drives, unfortunately, due to deficiencies in the formats thoes tape drives use.

To add files to an archive, use '`--append`' ('`-r`'). The archive to be added to must already exist and be in proper archive format (which normally means it was created previously using `tar`). If the archive was created with a different block size than now specified, `tar` will report an error (

FIXME: pxref Blocking Factor

). If the archive is not a valid `tar` archive, the results will be unpredictable. You cannot add files to a compressed archive, however you can add files to the last volume of a multi-volume archive.

FIXME: xref Matching Format Parameters

.

The following example adds the file '`shemp`' to the archive '`stooges`' created above:

```
tar --append --file=stooges shemp
```

You must specify the files to be added; there is no default.

'`--update`' ('`-u`') acts like '`--append`' ('`-r`'), but does not add files to the archive if there is already a file entry with that name in the archive that has the same modification time.

Both '`--update`' ('`-u`') and '`--append`' ('`-r`') work by adding to the end of the archive. When you extract a file from the archive, only the version stored last will wind up in the file system. Because '`--extract`' ('`-x`') extracts files from an archive in sequence, and overwrites files with the same name in the file system, if a file name appears more than once in an archive the last version of the file will overwrite the previous versions which have just been extracted. You should avoid storing older versions of a file later in the archive.

Note: '`--update`' ('`-u`') is not suitable for performing backups, because it doesn't change directory content entries, and because it lengthens the archive every time it is used.

FIXME: xref to scripted backup, listed incremental, for info on backups.

4.3 Updating an Archive

(This message will disappear, once this node revised.)

The '`--update`' ('`-u`') option updates a `tar` archive by comparing the date of the specified archive members against the date of the file with the same name. If the file has been modified more recently than the archive member, then the archive member is deleted (as with '`--delete`') and then the file is added to the archive (as with '`--append`' ('`-r`')). On media where the '`--delete`' option cannot be performed (such as magnetic tapes), the '`--update`' ('`-u`') option similarly fails.

If no archive members are named (either on the command line or via '`--files-from=`*`file-of-names`*' ('`-T` *`file-of-names`*')), then the entire archive is processed in this manner.

4.4 Combining Archives

(This message will disappear, once this node revised.)

The '`--concatenate`' ('`-A`') or [No value for "catenate"] option causes `tar` to add the contents of several archives to an existing archive.

Name the archives to be catenated on the command line. (Nothing happens if you don't list any.) The members, and their member names, will be copied verbatim from those archives. If this causes multiple members to have the same name, it does not delete either; all the members with the same name coexist. For information on how this affects reading the archive, FIXME: ref Multiple Members with the Same Name

.

You must use this option to concatenate archives. If you just combine them with `cat`, the result will not be a valid `tar` format archive.

This operation cannot be performed on some tape drives, unfortunately, due to deficiencies in the formats thoes tape drives use.

To append copies of an archive or archives to the end of another archive, use '`--concatenate`' ('`-A`'). The source and target archives must already exist and have been created using compatable format parameters (

FIXME: pxref Matching Format Parameters

).

`tar` will stop reading an archive if it encounters an end-of-archive marker. The `cat` utility does not remove end-of-archive markers, and is therefore unsuitable for concatenating archives. '`--concatenate`' ('`-A`') removes the end-of-archive marker from the target archive before each new archive is appended.

FIXME: xref ignore-zeros

You must specify the source archives using '`--file=`*`archive-name`*' ('`-f` *`archive-name`*') (

FIXME: pxref Archive
Name

). If you do not specify the target archive , `tar` uses the value of the environment variable `TAPE`, or, if this has not been set, the default archive name.

The following example adds the contents of the archive '`hail/hail/fredonia`' to the archive '`stooges`' (both archives were created in examples above):

```
tar --catenate --file=stooges hail/hail/fredonia
```

If you need to retrieve files from an archive that was added to using the `cat` utility, use the '`--ignore-zeros`' ('`-i`') option (

FIXME: pxref Archive Reading Options

).

4.5 Removing Archive Members

(This message will disappear, once this node revised.)

You can use the '`--delete`' option to remove members from an archive. Name the members on the command line to be deleted. This option will rewrite the archive; because of this, it does not work on tape drives. If you list no members to be deleted, nothing happens.

To delete archive members from an archive, use '`--delete`'. You must specify the file names of the members to be deleted. All archive members with the specified file names will be removed from the archive.

The following example removes the file '`curley`' from the archive '`stooges`':

```
tar --delete --file=stooges curley
```

You can only use '`--delete`' on an archive if the archive device allows you to write to any point on the media.

> **Warning:** Don't try to delete an archive member from a magnetic tape, lest you scramble the archive. There is no safe way (except by completely re-writing the archive) to delete files from most kinds of magnetic tape.

FIXME: how about automatic detection of archive media? give error
FIXME: unless the archive device is either an ordinary file or different
FIXME: input and output (–file=-).

4.6 Listing Archive Members

(This message will disappear, once this node revised.)

The '`--list`' ('`-t`') option will list the names of members of the archive. Name the members to be listed on the command line (to modify the way these names are interpreted,

FIXME: pxref Specifying Names to
tar

). If you name no members, then '`--list`' ('`-t`') will list the names of all the members of the archive.

To see more than just the names of the members, use the '`--verbose`' ('`-v`') option to cause `tar` to print out a listing similar to that of '`ls -l`'.

Listing the Contents of an Archive

'`--list`' ('`-t`') prints a list of the file names of the archive members on the standard output. If you specify *file name* arguments on the command line (or using the '`--files-from=`*`file-of-names`*' ('`-T` *`file-of-names`*') option,

FIXME: pxref File Name Lists

), only the files you specify will be listed, and only if they exist in the archive. Files not specified will be ignored, unless they are under a specific directory.

If you include the '`--verbose`' ('`-v`') option, `tar` prints an '`ls -l`' type listing for the archive.

FIXME: pxref Additional
Information

, for a description of the '`--verbose`' ('`-v`') option.

If the blocking factor of the archive differs from the default, `tar` reports this.

FIXME: xref Blocking Factor

.

FIXME: xref Archive Reading Options

for a list of options which can be used to modify '`--list`' ('`-t`')'s operation.

This example prints a list of the archive members of the archive '`stooges`':

```
tar --list --file=stooges
```

`tar` responds:

```
larry
moe
shemp
marx/julius
marx/alexander
marx/karl
```

This example generates a verbose list of the archive members of the archive file '`dwarves`', which has a blocking factor of two:

```
tar --list -v --file=blocks
```

`tar` responds:

```
tar: Blocksize = 2 records
-rw------- ringo/user 42 May   1 13:29 1990 .bashful
-rw-rw-rw- ringo/user 42 Oct   4 13:29 1990 doc
-rw-rw-rw- ringo/user 42 Jul  20 18:01 1969 dopey
-rw-rw---- ringo/user 42 Nov  26 13:42 1963 grumpy
-rw-rw-rw- ringo/user 42 May   5 13:29 1990 happy
-rw-rw-rw- ringo/user 42 May   1 12:00 1868 sleepy
-rw-rw-rw- ringo/user 42 Jul   4 17:29 1776 sneezy
```

4.7 Extracting Archive Members

(This message will disappear, once this node revised.)

Use '`--extract`' ('`-x`') or '`--get`' to extract members from an archive. For each member named (or for the entire archive if no members are named) on the command line—or with '`--files-from=file-of-names`' ('`-T file-of-names`')—the a file is created with the contents of the archive member. The name of the file is the same as the member name.

Various options cause `tar` to extract more than just file contents, such as the owner, the permissions, the modification date, and so forth.

FIXME: begin

The '`--same-permissions`' ('`-p`') or '`--preserve-permissions`' options cause `tar` to cause the new file to have the same permissions as the original file did when it was placed in the archive. Without this option, the current `umask` is used to affect the permissions.

When extrating, `tar` normally sets the modification time of the file to the value recorded in the archive. The '`--modification-time`' ('`-m`') option causes `tar` to omit doing this.

FIXME: end

To read archive members from the archive and write them into the file system, use '`--extract`' ('`-x`'). The archive itself is left unchanged.

If you do not specify the files to extract, `tar` extracts all the files in the archive. If you specify the name of a directory as a file name argument, `tar` will extract all files which have been stored as part of that directory. If a file was stored with a directory name as part of its file name, and that directory does not exist under the working directory when the file is extracted, `tar` will create the directory.

FIXME: xref Selecting Archive
Members

, for information on specifying files to extract.

The following example shows the extraction of the archive '`stooges`' into an empty directory:

```
tar --extract --file=stooges
```

Generating a listing of the directory ('`ls`') produces:

```
larry
moe
shemp
marx
```

The subdirectory '`marx`' contains the files '`julius`', '`alexander`' and '`karl`'.

If you wanted to just extract the files in the subdirectory '`marx`', you could specify that directory as a file name argument in conjunction with the '`--extract`' ('`-x`') operation:

```
tar --extract --file=stooges marx
```

> **Warning:** Extraction can overwrite files in the file system. To avoid losing files in the file system when extracting files from the archive with the same name, use the '`--keep-old-files`' ('`-k`') option (
>
> FIXME: pxref File Writing Options
>
>).

If the archive was created using '`--block-size=`*`512-size`*' ('`-b` *`512-size`*'), '`--compress`' ('`-Z`') or '`--multi-volume`' ('`-M`'), you must specify those format options again when extracting files from the archive (

FIXME: pxref Format Variations

).

4.7.1 Options to Help Read Archives

(This message will disappear, once this node revised.)

FIXME: each option wants its own node. summary after menu

Normally, `tar` will request data in full block increments from an archive storage device. If the device cannot return a full block, `tar` will report an error. However, some devices do not always return full blocks, or do not require the last block of an archive to be padded out to the next block boundary. To keep reading until you obtain a full block, or to accept an incomplete block if it contains an end-of-archive marker, specify the '`--read-full-blocks`' ('`-B`') option in conjunction with the '`--extract`' ('`-x`') or '`--list`' ('`-t`') operations.

FIXME: xref Listing Contents

.

The '`--read-full-blocks`' ('`-B`') option is turned on by default when `tar` reads an archive from standard input, or from a remote machine. This is because on BSD Unix systems, attempting to read a pipe returns however much happens to be in the pipe, even if it is less than was requested. If this option were not enabled, `tar` would fail as soon as it read an incomplete block from the pipe.

If you're not sure of the blocking factor of an archive, you can read the archive by specifying '`--read-full-blocks`' ('`-B`') and '`--block-size=`*`512-size`*' ('`-b` *`512-size`*'), using a blocking factor larger than what the archive uses. This lets you avoid having to determine the blocking factor of an archive.

FIXME: xref Blocking Factor

.

‘`--read-full-blocks`’
‘`-B`’ Use in conjunction with ‘`--extract`’ (‘`-x`’) to read an archive which contains incomplete blocks, or one which has a blocking factor less than the one specified.

Normally `tar` stops reading when it encounters a block of zeros between file entries (which usually indicates the end of the archive). ‘`--ignore-zeros`’ (‘`-i`’) allows `tar` to completely read an archive which contains a block of zeros before the end (i.e. a damaged archive, or one which was created by `cat`-ing several archives together).

The ‘`--ignore-zeros`’ (‘`-i`’) option is turned off by default because many versions of `tar` write garbage after the end-of-archive entry, since that part of the media is never supposed to be read. GNU `tar` does not write after the end of an archive, but seeks to maintain compatablity among archiving utilities.

‘`--ignore-zeros`’
‘`-i`’ To ignore blocks of zeros (ie. end-of-archive entries) which may be encountered while reading an archive. Use in conjunction with ‘`--extract`’ (‘`-x`’) or ‘`--list`’ (‘`-t`’).

If you are using a machine with a small amount of memory, and you need to process large list of file names, you can reduce the amount of space `tar` needs to process the list. To do so, specify the ‘`--same-order`’ (‘`-s`’) option and provide an ordered list of file names. This option tells `tar` that the *name* arguments provided on the command line, or read from a file using the ‘`--files-from=`*`file-of-names`*’ (‘`-T` *`file-of-names`*’) option, are listed in the same order as the files in the archive.

You can create a file containing an ordered list of files in the archive by storing the output produced by ‘`tar --list --file=`*`archive-name`*’.

FIXME: xref Listing Contents

, for information on the ‘`--list`’ (‘`-t`’) operation.

This option is probably never needed on modern computer systems.

‘`--same-order`’
‘`--preserve-order`’
‘`-s`’ To process large lists of file names on machines with small amounts of memory. Use in conjunction with ‘`--compare`’ (‘`-d`’), ‘`--list`’ (‘`-t`’) or ‘`--extract`’ (‘`-x`’).

FIXME: we don’t need/want –preserve to exist any more

4.7.2 Changing How tar Writes Files

FIXME: find a better title

(This message will disappear, once this node revised.)

Normally, `tar` writes extracted files into the file system without regard to the files already on the system—files with the same name as archive members are overwritten. To prevent `tar` from extracting an archive member from an archive, if doing so will overwrite a file in the file system, use '`--keep-old-files`' ('`-k`') in conjunction with the '`--extract`' ('`-x`') operation. When this option is specified, `tar` reports an error stating the name of the files in conflict, instead of writing the file from the archive.

'`--keep-old files`'
'`-k`' Prevents `tar` from overwriting files in the file system during extraction.

Normally, `tar` sets the modification times of extracted files to the modification times recorded for the files in the archive, but limits the permissions of extracted files by the current `umask` setting.

To set the modification times of extracted files to the time when the files were extracted, use the '`--modification-time`' ('`-m`') option in conjunction with '`--extract`' ('`-x`').

'`--modification-time`'
'`-m`' Sets the modification time of extracted archive members to the time they were extracted, not the time recorded for them in the archive. Use in conjunction with '`--extract`' ('`-x`').

To set the modes (access permissions) of extracted files to those recorded for those files in the archive, use the [No value for "same-persmissions"] option in conjunction with the '`--extract`' ('`-x`') operation.

FIXME: mib — should be aliased to ignore-umask.

'`--preserve-permission`'
'`--same-permission`'
'`--ignore-umask`'
'`-p`' Set modes of extracted archive members to those recorded in the archive, instead of current umask settings. Use in conjunction with '`--extract`' ('`-x`').

FIXME: following paragraph needs to be rewritten: why doesnt' this cat
FIXME: files together, why is this useful. is it really useful with
FIXME: more than one file?

To write the files extracted to the standard output, instead of creating the files on the file system, use '`--to-stdout`' ('`-O`') in conjunction with '`--extract`' ('`-x`'). This option is useful if you are extracting files to send them through a pipe, and do not need to preserve them in the file system.

'`--to-stdout`'
'`-O`' Writes files to the standard output. Used in conjunction with '`--extract`' ('`-x`').

FIXME: why would you want to do such a thing, how are files separated on
FIXME: the standard output? is this useful with more that one file? are
FIXME: pipes the real reason?

4.7.3 Recovering From Scarce Disk Space

(This message will disappear, once this node revised.)

If a previous attempt to extract files failed due to lack of disk space, you can use '`--starting-file=`*`name`*' ('`-K` *`name`*') to start extracting only after file *name* when extracting files from the archive. This assumes, of course, that there is now free space, or that you are now extracting into a different file system.

'`--starting-file=`*`file name`*'
'`-K` *`file name`*'
Starts an operation in the middle of an archive. Use in conjunction with '`--extract`' ('`-x`') or '`--list`' ('`-t`').

If you notice you are running out of disk space during an extraction operation, you can also suspend `tar`, remove unnecessary files from the file system, and then restart the same `tar` operation. In this case, '`--starting-file=`*`name`*' ('`-K` *`name`*') is not necessary.

FIXME: xref –incremental, xref –interactive, xref –exclude

4.8 Comparing Archives Members with Files

(This message will disappear, once this node revised.)

The '`--compare`' ('`-d`') or '`--diff`' option compares the contents of the specified archive members against the files with the same names, and reports its findings. If no members are named on the command line, or through '`--files-from=`*`file-of-names`*' ('`-T` *`file-of-names`*'), then the entire archive is so compared.

4.9 Matching the Format Parameters

(This message will disappear, once this node revised.)

Some format parameters must be taken into consideration when modifying an archive:

Compressed archives cannot be modified.

You have to specify the block size of the archive when modifying an archive with a non-default block size.

Multi-volume archives can be modified like any other archive. To add files to a multi-volume archive, you need to only mount the last volume of the archive media (and new volumes, if needed). For all other operations, you need to use the entire archive.

If a multi-volume archive was labeled using '`--label=`*`archive-label`*' ('`-V` *`archive-label`*') (

FIXME: pxref Archive Label

) when it was created, `tar` will not automatically label volumes which are added later. To label subsequent volumes, specify '`--label=`*`archive-label`*' ('`-V` *`archive-label`*') again in conjunction with the '`--append`' ('`-r`'), '`--update`' ('`-u`') or '`--concatenate`' ('`-A`') operation.

FIXME: example

FIXME: xref somewhere, for more information about format parameters.

5 Specifying Names to `tar`

(This message will disappear, once this node revised.)

5.1 Changing the Archive Name

(This message will disappear, once this node revised.)

By default, `tar` uses an archive file name compiled in when `tar` was built. Usually this refers to some physical tape drive on the machine. Often, the installer of `tar` didn't set the default to anything meaningful at all.

As a result, most uses of `tar` need to tell `tar` where to find (or create) the archive. The '`--file=`*`archive-name`*' ('`-f` *`archive-name`*') option selects another file to use as the archive.

If the archive file name includes a colon ('`:`'), then it is assumed to be a file on another machine. If the archive file is '*`user@host:file`*', then *file* is used on the host *host*. The remote host is accessed using the `rsh` program, with a username of *user*. If the username is omitted (along with the '`@`' sign), then your user name will be used. (This is the normal `rsh` behavior.) It is necessary for the remote machine, in addition to permitting your `rsh` access, to have the '`/usr/ucb/rmt`' program installed. If you need to use a file whose name includes a colon, then the remote tape drive behavior can be inhibited by using the '`--force-local`' option.

If the file name you give to '`--file=`*`archive-name`*' ('`-f` *`archive-name`*') is a single dash ('`-`'), then `tar` will read the archive from (or write it to) standard input (or standard output).

The Name of an Archive

An archive can be saved as a file in the file system, sent through a pipe or over a network, or written to an I/O device such as a tape or disk drive. To specify the name of the archive, use the '`--file=`*`archive-name`*' ('`-f` *`archive-name`*') option.

An archive name can be the name of an ordinary file or the name of an I/O device. `tar` always needs an archive name—if you do not specify an archive name, the archive name comes from the environment variable `TAPE` or, if that variable is not specified, a default archive name, which is usually the name of tape unit zero (ie. /dev/tu00).

If you use '`-`' as an *archive-name*, `tar` reads the archive from standard input (when listing or extracting files), or writes it to standard output (when creating an archive). If you use '`-`' as an *archive-name* when modifying an archive, `tar` reads the original archive from its standard input and writes the entire new archive to its standard output.

FIXME: does standard input and output redirection work with all
FIXME: operations?
FIXME: need example for standard input and output (screen and keyboard?)

To specify an archive file on a device attached to a remote machine, use the following:

```
--file=hostname:/dev/file name
```

`tar` will complete the remote connection, if possible, and prompt you for a username and password. If you use '`--file=@`*`hostname`*`:/`*`dev/file name`*', `tar` will complete the remote connection, if possible, using your username as the username on the remote machine.

FIXME: is this clear?

'`--file=`*`archive-name`*'
'`-f` *`archive-name`*'
Names the archive to create or operate on. Use in conjunction with any operation.

Selecting Archive Members

File Name arguments specify which files in the file system `tar` operates on, when creating or adding to an archive, or which archive members `tar` operates on, when reading or deleting from an archive. (

FIXME: pxref Reading and Writing

.)

To specify file names, you can include them as the last arguments on the command line, as follows:

```
tar operation [option1 option2 ..] [file name-1 file name-2 ...]
```

If you specify a directory name as a file name argument, all the files in that directory are operated on by `tar`.

If you do not specify files when `tar` is invoked, `tar` operates on all the non-directory files in the working directory (if the operation is '`--create`' ('`-c`')), all the archive members in the archive (if a read operation is specified), or does nothing (if any other operation is specified).

When specifying the names of files or members to `tar`, it by default takes the names of the files from the command line. There are other ways, however, to specify file or member names, or to modify the manner in which `tar` selects the files or members upon which to operate. In general, these methods work both for specifying the names of files and archive members.

5.2 Selecting Files by Characteristic

To avoid crossing file system boundaries when archiving parts of a directory tree, use '`--one-file-system`' ('`-l`'). This option only affects files that are archived because they are in a directory that is being archived; files explicitly named on the command line are archived regardless of where they reside.

This option is useful for making full or incremental archival backups of a file system.

If this option is used in conjunction with '`--verbose`' ('`-v`'), files that are excluded are mentioned by name on the standard error.

'`--one-file-system`'
'`-l`' Prevents `tar` from crossing file system boundaries when archiving. Use in conjunction with any write operation.

To avoid operating on files whose names match a particular pattern, use the '`--exclude=`*`pattern`*' or '`--exclude-from=`*`file-of-patterns`*' ('`-X` *`file-of-patterns`*') options.

When you specify the '`--exclude=`*`pattern`*' option, `tar` ignores files which match the *pattern*, which can be a single file name or a more complex expression. Thus, if you invoke `tar` with ‘`tar --create --exclude=*.o`’, no files whose names end in ‘`.o`’ are included in the archive.

FIXME: what other things can you use besides "*"?

‘`--exclude-from=`*`file-of-patterns`*’ (‘`-X` *`file-of-patterns`*’) acts like ‘`--exclude=`*`pattern`*’, but specifies a file *file* containing a list of patterns. `tar` ignores files with names that fit any of these patterns.

You can use either option more than once in a single command.

‘`--exclude=`*`pattern`*’
  Causes `tar` to ignore files that match the *pattern*.

‘`--exclude-from=`*`file`*’
  Causes `tar` to ignore files that match the patterns listed in *file*.

FIXME: –exclude-from used to be "–exclude", –exclude didn’t used to
FIXME: exist.

To operate only on files with modification or status-change times after a particular date, use ‘`--after-date=`*`date`*’ (‘`-N` *`date`*’). You can use this option with ‘`--create`’ (‘`-c`’) or ‘`--append`’ (‘`-r`’) to insure only new files are archived, or with ‘`--extract`’ (‘`-x`’) to insure only recent files are resurrected.

FIXME: or –newer *date*

‘`--newer-mtime=`*`date`*’ acts like ‘`--after-date=`*`date`*’ (‘`-N` *`date`*’) but tests just the modification times of the files, ignoring status-change times.

FIXME: need example of –newer-mtime with quoted argument

Remember that the entire date argument should be quoted if it contains any spaces.

Please Note: ‘`--after-date=`*`date`*’ (‘`-N` *`date`*’) and ‘`--newer-mtime=`*`date`*’ should not be used for incremental backups. Some files (such as those in renamed directories) are not selected up properly by these options.

FIXME: xref to incremental backup chapter when node name is decided.

‘`--after-date=`*`date`*’
‘`--newer=`*`date`*’
‘`-N` *`date`*’ Acts on files only if their modification or inode-changed times are later than *date*. Use in conjunction with any operation.

‘`--newer-mtime=`*`date`*’
  Acts like ‘`--after-date=`*`date`*’ (‘`-N` *`date`*’), but only looks at modification times.

FIXME: following is the getdate date format — needs to be re-written,
FIXME: made a sub-node:

Time/Date Formats Accepted by getdate (omitting obscure constructions)

The input consists of one or more of: time zone day date year in any order.

Those in turn consist of (‘`|`’ and ‘/’ mean ‘or’, ‘[]’ means ‘optional’):

time: H am/pm `|` H:M [am/pm] `|` H:M:S [am/pm] zone: timezone-name `|` timezone-name dst day: day-name `|` day-name, `|` N day-name date: M/D `|` M/D/Y `|` month-name D `|` month-name D, Y `|` D month-name `|` D month-name Y year: Y am can also be a.m., pm can also be p.m. case and spaces around punctuation are not significant.

FIXME: month and day names can be abbreviated.

5.2.1 Reading Names from a File

(This message will disappear, once this node revised.)

Instead of giving the names of files or archive members on the command line, you can put the names into a file, and then use the '`--files-from=file-of-names`' ('`-T file-of-names`') option to `tar`. Give the name of the file which contains the list as the argument to '`--files-from=file-of-names`' ('`-T file-of-names`'). The file names should be separated by newlines in the list. If you give a single dash as a file name for '`--files-from=file-of-names`' ('`-T file-of-names`'), that is, you specify '`--files-from=-`' ('`-T -`'), then the file names are read from standard input.

If you want to specify names that might contain newlines, use the '`--null`' option. Then, the file names should be separated by NUL characters (ASCII 000) instead of newlines. In addition, the '`--null`' option turns off the '`--directory=directory`' ('`-C directory`') option (

FIXME: pxref Changing Directory

).

Reading a List of File Names from a File

(This message will disappear, once this node revised.)

To read file names from a file on the file system, instead of from the command line, use the '`--files-from=file-of-names`' ('`-T file-of-names`') option. If you specify '`-`' as *file*, the file names are read from standard input. Note that using both '`--files-from=-`' ('`-T -`') and '`--file=-`' ('`-f -`') in the same command will not work unless the operation is '`--create`' ('`-c`').

FIXME: xref Archive Name

, for an explanation of the '`--file=archive-name`' ('`-f archive-name`') option.

'`--files-from=file`'
'`-T file`' Reads file name arguments from a file on the file system, instead of from the command line. Use in conjunction with any operation.

5.2.2 Excluding Some Files

(This message will disappear, once this node revised.)

The '`--exclude=pattern`' option will prevent any file or member which matches the regular expression *pattern* from being operated on. For example, if you want to create an archive with all the contents of '`/tmp`' except the file '`/tmp/foo`', you can use the command '`tar --create --file=arch.tar --exclude=foo`'.

If there are many files you want to exclude, you can use the '`--exclude-from=file-of-patterns`' ('`-X file-of-patterns`') option. This works just like the '`--files-from=file-of-names`' ('`-T file-of-names`') option: specify the name of a file as *exclude-list* which contains the list of patterns you want to exclude.

FIXME: xref Regular Expressions

for more information on the syntax and meaning of regular expressions.

5.2.3 Operating Only on New Files

(This message will disappear, once this node revised.)

The '`--after-date=`*date*' ('`-N` *date*') or '`--newer=`*date*' limits `tar` to only operating on files which have been modified after the date specified. (For more information on how to specify a date,

FIXME: xref Date Formats

.) A file is considered to have changed if the contents have been modified, or if the owner, permissions, and so forth, have been changed.

If you only want `tar` make the date comparison on the basis of the actual contents of the file's modification, then use the '`--newer-mtime=`*date*' option.

You should never use this option for making incremental dumps. To learn how to use `tar` to make backups,

FIXME: ref Making Backups

.

5.2.4 Crossing Filesystem Boundaries

(This message will disappear, once this node revised.)

The '`--one-file-system`' ('`-l`') option causes `tar` to modify its normal behavior in archiving the contents of directories. If a file in a directory is not on the same filesystem as the directory itself (because it is a mounted filesystem in its own right), then `tar` will not archive that file, or (if it is a directory itself) anything beneath it.

This does not necessarily limit `tar` to only archiving the contents of a single filesystem, because all files named on the command line, or through the '`--files-from=`*file-of-names*' ('`-T` *file-of-names*') option, will always be archived.

5.3 Local file selection

(This message will disappear, once this node revised.)

Local file selection

`-C` *dir*
`--directory` *dir*

Change to directory *dir*.

This option causes `tar` to change into the directory *dir* before continuing. This option can be interspersed with the files `tar` is to work on. For example,

```
tar -c iggy ziggy -C baz melvin
```

will place the files '`iggy`' and '`ziggy`' from the current directory on the tape, followed by the file '`melvin`' from the directory '`baz`'. This option is especially useful when you have several widely separated files that you want to store in the same directory in the archive.

Here, the file '`melvin`' is recorded in the archive under the precise name '`melvin`', *not* '`baz/melvin`'. Thus, the archive will contain three files that all appear to have come from the same directory; if the archive is extracted with plain '`tar -x`', all three files will be created in the current directory.

Contrast this with the command:

```
tar -c iggy ziggy bar/melvin
```

which records the third file in the archive under the name '`bar/melvin`' so that, if plain '`tar -x`' is used, the third file will be created in a subdirectory named '`bar`'.

Suppose that, without changing your current directory, you want to call `tar` to dump files from '`/users/ctd/dipp`' say. Then '`--directory=`*`directory`*' ('`-C` *`directory`*') is for you. You could do things like:

```
tar cfC archive.tar /users/ctd/dipp .
```

(the '`.`' means the current directory, once the '`--directory=`*`directory`*' ('`-C` *`directory`*') obeyed).

Some people might want some option to extract everything from an archive in the current directory, ignore directory structure in the archive. This is so rarely proper that I doubt such an option would be really useful. It would only help getting around improper `tar` usage, it might even encourage improper usage. In general, '`--directory=`*`directory`*' ('`-C` *`directory`*') might be used to produce archives with a cleaner structure in the first place.

`-T` *`filename`*
`--files-from=`*`filename`*

Get names to extract or create from file *filename*.

Instead of taking the list of files to work on from the command line, the list of files to work on is read from the file *filename*. If *filename* is given as '`-`', the list is read from standard input. Note that using both '`-T -`' and '`-f -`' will not work unless you are using the '`--create`' ('`-c`') command.

This is typically useful when you have generated the list of files to archive with `find`.

`--null` This option causes '`--files-from=`*`file-of-names`*' ('`-T` *`file-of-names`*') to read file names terminated by a NUL instead of a newline, so files whose names contain newlines can be archived using '`--files-from=`*`file-of-names`*' ('`-T` *`file-of-names`*'). The '`--null`' option is just like the one in GNU `xargs` and `cpio`, and is useful with the '`-print0`' predicate of GNU `find`. In `tar`, '`--null`' also causes '`--directory=`*`directory`*' ('`-C` *`directory`*') options to be treated as file names to archive, in case there are any files out there called '`-C`'.

`--exclude=`*`file`*

Exclude file *file*.

`-X` *`file`*
`--exclude-from=`*`file`*

Exclude files listed in *file*.

This option causes `tar` to read a list of regular expressions (in shell wildcard syntax), one per line, from *file*; `tar` will ignore files matching those regular expressions. Thus if `tar` is called as '`tar -c -X foo .`' and the file '`foo`' contains a single line '`*.o`', no files whose names end in '`.o`' will be added to the archive. Multiple '`--exclude=`*`pattern`*' options may be given.

`-P`
`--absolute-names`

Do not strip leading `/`s from file names.

By default, GNU `tar` drops a leading '`/`' on input or output. This option turns off this behavior; it's equivalent to changing to the root directory before running `tar` (except it also turns off the usual warning message).

`-l`
`--one-file-system`

Stay in local filesystem when creating archive.

This option causes `tar` to not cross filesystem boundaries when archiving parts of a directory tree. This option only affects files that are archived because they are in a directory that is archived; files named on the command line are archived regardless, and they can be from various file systems.

This option is useful for making full or incremental archival backups of a filesystem, as with the Unix `dump` command.

Files skipped due to this option are mentioned on standard error.

`-K` *`name`*
`--starting-file=`*`name`*

Begin at file *name* in the archive.

The '`--starting-file=`*`name`*' ('`-K` *`name`*') option causes `tar` to begin extracting or listing the archive with the file *filename*, and to consider only the files starting at that point in the archive. This is useful if a previous attempt to extract files failed when it reached *filename* due to lack of free space. (Assuming, of course, that there is now free space, or that you are now extracting into a different file system.)

`-N` *`date`*
`--newer=`*`date`*
`--after-date=`*`date`*

Only store files newer than *date*.

This option causes `tar` to only work on files whose modification or inode-changed times are newer than the *date* given. The main use is for creating an archive; then only new files are written. If extracting, only newer files are extracted.

Remember that the entire date argument must be quoted if it contains any spaces.

The date is parsed using `getdate`.

Changing the Names of Members when Archiving

5.3.1 Changing Directory

(This message will disappear, once this node revised.)

The '`--directory=`*`directory`*' ('`-C` *`directory`*') option causes `tar` to change its current working directory to *directory*. Unlike most options, this one is processed at the point it occurs within the list of files to be processed. Consider the following command:

```
tar --create --file=foo.tar -C /etc passwd hosts -C /lib libc.a
```

This command will place the files '`/etc/passwd`', '`/etc/hosts`', and '`/lib/libc.a`' into the archive. However, the names of the archive members will be exactly what they were on the command line: '`passwd`', '`hosts`', and '`libc.a`'. The '`--directory=`*`directory`*' ('`-C` *`directory`*') option is frequently used to make the archive independent of the original name of the directory holding the files.

Note that '`--directory=`*`directory`*' ('`-C` *`directory`*') options are interpreted consecutively. If '`--directory=`*`directory`*' ('`-C` *`directory`*') option specifies a relative file name, it is interpreted relative to the then current directory, which might not be the same as the original current working directory of `tar`, due to a previous '`--directory=`*`directory`*' ('`-C` *`directory`*') option.

When using '`--files-from=`*`file-of-names`*' ('`-T` *`file-of-names`*') (

FIXME: pxref Reading Names from a File

), you can put '`-C`' options in the file list. Unfortunately, you cannot put '`--directory`' options in the file list. (This interpretation can be disabled by using the '`--null`' option.)

Changing the Working Directory Within a List of File Names

(This message will disappear, once this node revised.)

To change working directory in the middle of a list of file names, either on the command line or in a file specified using '`--files-from=`*`file-of-names`*' ('`-T` *`file-of-names`*'), use '`--directory=`*`directory`*' ('`-C` *`directory`*'). This will change the working directory to the directory *directory* after that point in the list. For example,

```
tar --create iggy ziggy --directory=baz melvin
```

will place the files '`iggy`' and '`ziggy`' from the current directory into the archive, followed by the file '`melvin`' from the directory '`baz`'. This option is especially useful when you have several widely separated files that you want to store in the same directory in the archive.

Note that the file '`melvin`' is recorded in the archive under the precise name '`melvin`', *not* '`baz/melvin`'. Thus, the archive will contain three files that all appear to have come from the same directory; if the archive is extracted with plain '`tar --extract`', all three files will be written in the current directory.

Contrast this with the command

```
tar -c iggy ziggy bar/melvin
```

which records the third file in the archive under the name '`bar/melvin`' so that, if the archive is extracted using '`tar --extract`', the third file will be written in a subdirectory named '`bar`'.

'`--directory='directory''`'
'`-C 'directory''`'

Changes the working directory.

FIXME: need to test how extract deals with this, and add an example

5.3.2 Absolute File Names

(This message will disappear, once this node revised.)

When `tar` extracts archive members from an archive, it strips any leading slashes ('`/`') from the member name. This causes absolute member names in the archive to be treated as relative file names. This allows you to have such members extracted wherever you want, instead of being restricted to extracting the member in the exact directory named in the archive. For example, if the archive member has the name '`/etc/passwd`', `tar` will extract it as if the name were really '`etc/passwd`'.

Other `tar` programs do not do this. As a result, if you create an archive whose member names start with a slash, they will be difficult for other people with an inferior `tar` program to use. Therefore, GNU `tar` also strips leading slashes from member names when putting members into the archive. For example, if you ask `tar` to add the file '`/bin/ls`' to an archive, it will do so, but the member name will be '`bin/ls`'.

If you use the '`--absolute-names`' ('`-P`') option, `tar` will do neither of these transformations.

FIXME: is this what this does, or does it just preserve the slash?

To archive or extract files relative to the root directory, specify the '`--absolute-names`' ('`-P`') option.

Normally, `tar` acts on files relative to the working directory—ignoring superior directory names when archiving, and ignoring leading slashes when extracting.

When you specify '`--absolute-names`' ('`-P`'), `tar` stores file names including all superior directory names, and preserves leading slashes. If you only invoked `tar` from the root directory you would never need the '`--absolute-names`' ('`-P`') option, but using this option may be more convenient than switching to root.

FIXME: should be an example in the tutorial/wizardry section using this
FIXME: to transfer files between systems.

FIXME: is write access an issue?

'`--absolute-names`'

Preserves full file names (inclusing superior dirctory names) when archiving files. Preserves leading slash when extracting files.

6 Being Even More Careful

6.1 GNU `tar` documentation

Being careful, the first thing is really checking that you are using GNU `tar`, indeed. The '`--version`' option will generate a message giving confirmation that you are using GNU `tar`, with the precise version of GNU `tar` you are using. `tar` identifies itself and prints the version number to the standard output, then immediately exits successfully, without doing anything else, ignoring all other options. For example, '`tar --version`' might return:

```
GNU tar version 1.11.8
```

Another thing you might want to do is checking the spelling or meaning of some particular `tar` option, without resorting to this manual, for once you have carefully read it. GNU `tar` has a short help feature, triggerable through the '`--help`' option. By using this option, `tar` will print a usage message listing all available options on standard output, then exit successfully, without doing anything else and ignoring all other options. Even if this is only a brief summary, it may be several screens long. So, if you are not using some kind of scrollable window, you might prefer to use something like:

```
tar --help | less
```

presuming, here, that you like using `less` for a pager. Other popular pagers are `more` and `pg`.

The perceptive reader would have noticed some contradiction in the previous paragraphs. It is written that both '`--version`' and '`--help`' print something, and have all other options ignored. In fact, they cannot ignore each other, and one of them has to win. We do not specify which is stronger, here; experiment if you really wonder!

The short help output is quite succint, and you might have to get back to the full documentation for precise points. If you are reading this paragraph, you already have the `tar` manual in some form. This manual is available in printed form, as a kind of small book. It may printed out of the GNU `tar` distribution, provided you have TEX already installed somewhere, and a laser printer around. Just configure the distribution, execute the command '`make dvi`', then print '`doc/tar.dvi`' the usual way (contact your local guru to know how). If GNU `tar` has been conveniently installed at your place, this manual is also available in interactive, hypertextual form as an Info file. Just call '`info tar`' or, if you do not have the `info` program handy, use the Info reader provided within GNU Emacs, calling '`tar`' from the main Info menu.

6.2 Checking `tar` progress

Typically, `tar` performs most operations without reporting any information to the user except error messages. When using `tar` with many options, particularly ones with complicated or difficult-to-predict behavior, it is possible to make serious mistakes. `tar` provides several options that make observing `tar` easier. These options cause `tar` to print information as it progresses in its job, and you might want to use them just for being more careful about what is going on, or merely for entertaining yourself. If you have encountered a problem when operating on an archive, however, you may need more information than just an error message in order to solve the problem. The following options can be helpful diagnostic tools.

Normally, the '`--list`' ('`-t`') command to list an archive prints just the file names (one per line) and the other commands are silent. When used with most operations, the '`--verbose`' ('`-v`') option causes `tar` to print the name of each file or archive member as it is processed. This and the other options which make `tar` print status information can be useful in monitoring `tar`.

With '`--create`' ('`-c`') or '`--extract`' ('`-x`'), '`--verbose`' ('`-v`') used once just prints the names of the files or members as they are processed. Using it twice causes `tar` to print a longer listing (reminiscent of '`ls -l`') for each member. Since '`--list`' ('`-t`') already prints the names of the members, '`--verbose`' ('`-v`') used once with '`--list`' ('`-t`') causes `tar` to print an '`ls -l`' type listing of the files in the archive. The following examples both extract members with long list output:

```
tar --extract --file=archive.tar --verbose --verbose
tar xvv archive.tar
```

Verbose output appears on the standard output except when an archive is being written to the standard output, as with '`tar --create --file=- --verbose`' ('`tar cfv -`', or even '`tar cv`'—if the installer let standard output be the default archive). In that case `tar` writes verbose output to the standard error stream.

The '`--totals`' option—which is only meaningful when used with '`--create`' ('`-c`')—causes `tar` to print the total amount written to the archive, after it has been fully created.

The '`--checkpoint`' option prints an occasional message as `tar` reads or writes the archive. In fact, it print directory names while reading the archive. It is designed for those who don't need the more detailed (and voluminous) output of '`--record-number`' ('`-R`'), but do want visual confirmation that `tar` is actually making forward progress.

FIXME: There is some confusion here. It seems that -R once wrote a
FIXME: message at '`every`' block read or written.

The '`--show-omitted-dirs`' option, when reading an archive—with '`--list`' ('`-t`') or '`--extract`' ('`-x`'), for example—causes a message to be printed for each directory in the archive which is skipped. This happens regardless of the reason for skipping: the directory might not have been named on the command line (implicitly or explicitly), it might be excluded by the use of the '`--exclude=`*`pattern`*' option, or some other reason.

If '`--record-number`' ('`-R`') is used, `tar` prints, along with every message it would normally produce, the record number within the archive where the message was triggered. This option is especially useful when reading damaged archives, since it helps pinpoint the damaged sections. It can also be used with '`--list`' ('`-t`') when listing a file-system backup tape, allowing you to choose among several backup tapes when retrieving a file later, in favor of the tape where the file appears earliest (closest to the front of the tape).

FIXME: xref when the node name is set and the backup section written

6.3 Asking for Confirmation During Operations

Typically, `tar` carries out a command without stopping for further instructions. In some situations however, you may want to exclude some files and archive members from the operation (for instance if disk or storage space is tight). You can do this by excluding certain files automatically (

FIXME: pxref File Exclusion

), or by performing an operation interactively, using the '`--interactive`' ('`-w`') option. `tar` also accepts '`--confirmation`' for this option.

When the '`--interactive`' ('`-w`') option is specified, before reading, writing, or deleting files, `tar` first prints a message for each such file, telling what operation it intends to take, then asks for confirmation on the terminal. The actions which require confirmation include adding a file to the archive, extracting a file from the archive, deleting a file from the archive, and deleting a file from disk. To confirm the action, you must type a line of input beginning with '`y`'. If your input line begins with anything other than '`y`', `tar` skips that file.

If `tar` is reading the archive from the standard input, `tar` opens the file '`/dev/tty`' to support the interactive communications.

6.4 Verifying Data as It is Stored

You can insure the accuracy of an archive by comparing files in the system with archive members. `tar` can compare an archive to the file system as the archive is being written, to verify a write operation, or can compare a previously written archive, to insure that it is up to date.

To check for discrepancies in an archive immediately after it is written, use the '`--verify`' ('`-W`') option in conjunction with the '`--create`' ('`-c`') operation. When this option is specified, `tar` checks archive members against their counterparts in the file system, and reports discrepancies on the standard error. In multi-volume archives, each volume is verified after it is written, before the next volume is written.

To verify an archive, you must be able to read it from before the end of the last written entry. This option is useful for detecting data errors on some tapes. Archives written to pipes, some cartridge tape drives, and some other devices cannot be verified.

6.5 Comparing an Archive with the File System

(This message will disappear, once this node revised.)

'`--compare`' ('`-d`') compares archive members in an existing archive with their counterparts in the file system, and reports differences in file size, mode, owner, modification date and contents. If a file is represented in the archive but does not exist in the file system, `tar` reports a difference.

If you use *file name* arguments in conjunction with '`tar --compare`', `tar` compares the archived versions of the files specified with their counterparts in the file system. If you specify a file that is not in the archive, `tar` will report an error. If you don't specify any files, `tar` compares all the files in the archive.

Because `tar` only checks files in the archive against files in the file system, and not vice versa, it ignores files in the file system that do not exist in the archive.

The following example compares the archive members '`larry`', '`moe`' and '`curly`' in the archive '`stooges`' with files of the same name in the file system.

```
tar --compare --file=stooges larry moe curly
```

If a file, for example '`curly`', did not exist in the archive, `tar` would report an error, as follows:

```
curly: does not exist
```

6.6 Making `tar` Archives More Portable

Creating a `tar` archive on a particular system, meant to be later useful on many other machines and with other versions of `tar`, is more a challenge than you might think. `tar` archive formats evolved since the first versions of Unix, many such formats are flying around, not always comptabile between them. This section wants to discuss a few problems, and give some advice, for making `tar` archives more portable.

One golden rule is simplicity. For example, limit your `tar` archives to contain only regular files and directories, avoiding other kind of special files. Do not attempt to save sparse files or contiguous files as such. Let's discuss a few more problems, in turn.

6.6.1 Portable Names

Use *straight* file and directory names, made up of printable ASCII characters, avoiding colons, slashes, backslashes, and other *dangerous* characters. Avoid deep directory nesting. Accounting for oldish System V machines, limit your file and directory names to 14 characters or less.

If you intend to have your `tar` archives to be read under MSDOS, you should not rely on case distinction for file names, and you might use the GNU `doschk` program for helping you further diagnosing illegal MSDOS names, which are even more limited than System V's.

6.6.2 Symbolic Links

Normally, when `tar` archives a symbolic link, it writes a record to the archive naming the target of the link. In that way, the `tar` archive is a faithful record of the filesystem contents. '`--dereference`' ('`-h`') is used with '`--create`' ('`-c`'), and causes `tar` to archive the files symbolic links point to, instead of the links themselves. When this option is used, when `tar` encounters a symbolic link, it will archive the linked-to file, instead of simply recording the presence of a symbolic link.

The name under which the file is stored in the file system is not recorded in the archive. To record both the symbolic link name and the file name in the system, archive the file under both names. If all links were recorded automatically by `tar`, an extracted file might be linked to a file name that no longer exists in the file system.

If a linked-to file is encountered again by `tar` while creating the same archive, an entire second copy of it will be stored. (This *might* be considered a bug.)

So, for portable archives, do not archive symbolic links as such, and use '`--dereference`' ('`-h`'): many systems do not support symbolic links, and moreover, your distribution might be unusable if it contains unresolved symbolic links.

6.6.3 Old V7 and POSIX Archives

GNU `tar` implements an early draft of the POSIX 1003.1 `ustar` standard which is different from the final standard. Adding support for the new changes in a backward-compatible fashion is not trivial.

Certain old versions of `tar` cannot handle additional information recorded by newer `tar` programs. To create an archive in V7 format (not ANSI), which can be read by these old versions, specify the '`--old-archive`' ('`-o`') option in conjunction with the '`--create`' ('`-c`'). `tar` also accepts '`--portability`' for this option. When you specify it, `tar` leaves out information about directories, pipes, fifos, contiguous files, and device files, and specifies file ownership by group and user IDs instead of group and user names.

When updating an archive, do not use '`--old-archive`' ('`-o`') unless the archive was created with using this option.

In most cases, a *new* format archive can be read by an *old* `tar` program without serious trouble, so this option should seldom be needed. On the other hand, most modern `tar`s are able to read old format archives, so it might be safer for you to always use '`--old-archive`' ('`-o`') for your distributions.

6.6.4 Checksumming Problems

SunOS and HP-UX `tar` fail to accept archives created using GNU `tar` and containing non-ASCII file names, because they use signed checksums, while GNU `tar` uses unsigned checksums while creating archives, as per POSIX standards. On reading, GNU `tar` computes both checksums and accept any. It is somewhat worrying that a lot of people may go around doing backup of their files using faulty (or at least non-standard) software, not learning about it until it's time to restore their missing files with an incompatible file extractor, or vice versa.

GNU `tar` is supposed to compute both checksums, signed and unsigned, and accept any. However, 1.11.2 has a bug by which signed checksums are incorrectly initialized, so they do not work. This is corrected in the subsequent GNU `tar` versions. However, GNU `tar` has not been modified to *produce* incorrect archives to be read by buggy `tar`'s.

I've been told that when Sun first imported `tar` on their system, they recompiled it without realizing that the checksums were computed differently, because of a change in the default signing of `char`'s in their compiler. So they started computing checksums wrongly, and stayed compatible with themselves afterwards. It now falls on the shoulders of SunOS and HP-UX users to get a `tar` able to read the good archives they receive.

6.7 Write Protection

All tapes and disks can be *write protected*, to protect data on them from being changed. Once an archive is written, you should write protect the media to prevent the archive from being accidently overwritten or deleted. (This will protect the archive from being changed with a tape or floppy drive—it will not protect it from magnet fields or other physical hazards).

The write protection device itself is usually an integral part of the physical media, and can be a two position (write enabled/write disabled) switch, a notch which can be popped out or covered, a ring which can be removed from the center of a tape reel, or some other changeable feature.

7 Controlling the Archive Format

7.1 Handling of file attributes

(This message will disappear, once this node revised.)

Handling of file attributes

`--atime-preserve`
: Do not change access times on dumped files.

`-m`
`--modification-time`
: Do not extract file modified time.

When this option is used, `tar` leaves the modification times of the files it extracts as the time when the files were extracted, instead of setting it to the time recorded in the archive.

This option is meaningless with '`--list`' ('`-t`').

`--same-owner`
: Create extracted files with the same ownership.

`-p`
`--same-permissions`
`--preserve-permissions`
: Extract all protection information.

This option causes `tar` to set the modes (access permissions) of extracted files exactly as recorded in the archive. If this option is not used, the current `umask` setting limits the permissions on extracted files.

This option is meaningless with '`--list`' ('`-t`').

`-s`
`--same-order`
`--preserve-order`
: Sort names to extract to match archive.

This option tells `tar` that the list of file names to be listed or extracted is sorted in the same order as the files in the archive. This allows a large list of names to be used, even on a small machine that would not otherwise be able to hold all the names in memory at the same time. Such a sorted list can easily be created by running '`tar -t`' on the archive and editing its output.

This option is probably never needed on modern computer systems.

`--preserve`
: Same as both '`--same-permissions`' ('`-p`') and '`--same-order`' ('`-s`').

The '`--preserve`' option has no equivalent short option name. It is equivalent to '`--same-permissions`' ('`-p`') plus '`--same-order`' ('`-s`').

7.2 Archive format selection

(This message will disappear, once this node revised.)

Archive format selection

`-V name`
`--label=name`
: Create archive with volume name *name*.

This option causes `tar` to write out a *volume header* at the beginning of the archive. If '`--multi-volume`' ('`-M`') is used, each volume of the archive will have a volume header of '`name Volume n`', where *n* is 1 for the first volume, 2 for the next, and so on.

`-z`
`--gzip`
`--ungzip`
: Filter the archive through `gzip`.

This option works on physical devices (tape drives, etc.) and remote files as well as on normal files; data to or from such devices or remote files is reblocked by another copy of the `tar` program to enforce the specified (or default) block size. The default compression parameters are used; if you need to override them, avoid the '`--gzip`' ('`-z`') option and run `gzip` explicitly. (Or set the '`GZIP`' environment variable.)

If the '`--gzip`' ('`-z`') option is given twice, or the '`--compress-blocks`' option is used, `tar` will pad the archive out to the next block boundary (

FIXME: pxref Blocking

). This may be useful with some devices that require that all write operations be a multiple of a certain size.

The '`--gzip`' ('`-z`') option does not work with the '`--multi-volume`' ('`-M`') option, or with the '`--update`' ('`-u`'), '`--append`' ('`-r`'), '`--concatenate`' ('`-A`'), or '`--delete`' commands.

It is not exact to say that GNU `tar` is to work in concert with `gzip` in a way similar to `zip`, say. Surely, it is possible that `tar` and `gzip` be done with a single call, like in:

```
tar cfz archive.tar.gz subdir
```

to save all of '`subdir`' into a `gzip`'ed archive. Later you can do:

```
tar xfz archive.tar.gz
```

to explode and unpack.

The difference is that the whole archive is compressed. With `zip`, archive members are archived individually. `tar`'s method yields better compression. On the other hand, one can view the contents of a `zip` archive without having to decompress it. As for the `tar` and `gzip` tandem, you need to decompress the archive to see its contents. However, this may be done without needing disk space, by using pipes internally:

```
tar tfz archive.tar.gz
```

About corrupted compressed archives: `gzip`'ed files have no redundancy, for maximum compression. The adaptive nature of the compression scheme means that the compression tables are implicitly spread all over the archive. If you lose a few blocks, the dynamic construction of the compression tables becomes unsychronized, and there is little chance that you could recover later in the archive.

There are pending suggestions for having a per-volume or per-file compression in GNU `tar`. This would allow for viewing the contents without decompression, and for resynchronizing decompression at every volume or file, in case of corrupted archives. Doing so, we might loose some compressibility. But this would have make recovering easier. So, there are pros and cons. We'll see!

`-Z`
`--compress`
`--uncompress`

Filter the archive through `compress`. Otherwise like '`--gzip`' ('`-z`').

`--use-compress-program=`*`prog`*

Filter through *prog* (must accept '`-d`').

7.3 Using Less Space through Compression

7.3.1 Creating and Reading Compressed Archives

(This message will disappear, once this node revised.)

'`--compress`' ('`-Z`') indicates an archive stored in compressed format. The '`--compress`' ('`-Z`') option is useful in saving time over networks and space in pipes, and when storage space is at a premium. '`--compress`' ('`-Z`') causes `tar` to compress when writing the archive, or to uncompress when reading the archive.

To perform compression and uncompression on the archive, `tar` runs the `compress` utility. `tar` uses the default compression parameters; if you need to override them, avoid the '`--compress`' ('`-Z`') option and run the `compress` utility explicitly. It is useful to be able to call the `compress` utility from within `tar` because the `compress` utility by itself cannot access remote tape drives.

The '`--compress`' ('`-Z`') option will not work in conjunction with the '`--multi-volume`' ('`-M`') option or the '`--append`' ('`-r`'), '`--update`' ('`-u`'), '`--append`' ('`-r`') and '`--delete`' operations.

FIXME: xref Modifying

, for more information on these operations.

If there is no compress utility available, `tar` will report an error.

'`--compress-blocks`' is like '`--compress`' ('`-Z`'), but when used in conjunction with '`--create`' ('`-c`') also causes `tar` to pad the last block of the archive out to the next block boundary as it is written. This is useful with certain devices which require all write operations be a multiple of a specific size.

> **Please Note:** The `compress` program may be covered by a patent, and therefore we recommend you stop using it. We hope to have a different compress program in the future. We may change the name of this option at that time.

‘`--compress`’
‘`--uncompress`’
‘`-z`’
‘`-Z`’ When this option is specified, `tar` will compress (when writing an archive), or uncompress (when reading an archive). Used in conjunction with the ‘`--create`’ (‘`-c`’), ‘`--extract`’ (‘`-x`’), ‘`--list`’ (‘`-t`’) and ‘`--compare`’ (‘`-d`’) operations.

‘`--compress-block`’
‘`-z -z`’ Acts like ‘`--compress`’ (‘`-Z`’), but pads the archive out to the next block boundary as it is written when used in conjunction with the ‘`--create`’ (‘`-c`’) operation.

FIXME: why not use -Z instead of -z -z ?

7.3.2 Dealing with Compressed Archives

(This message will disappear, once this node revised.)

You can have archives be compressed by using the ‘`--gzip`’ (‘`-z`’) option. This will arrange for `tar` to use the `gzip` program to be used to compress or uncompress the archive wren writing or reading it.

To use the older, obsolete, `compress` program, use the ‘`--compress`’ (‘`-Z`’) option. The GNU Project recommends you not use `compress`, because there is a patent covering the algorithm it uses. Merely by running `compress` you could be sued for patent infringment.

When using either ‘`--gzip`’ (‘`-z`’) or ‘`--compress`’ (‘`-Z`’), `tar` does not do blocking (

FIXME: pxref Blocking

) correctly. Use ‘`--gzip-block`’ or ‘`--compress-blocks`’ instead when using real tape drives.

7.3.3 Archiving Sparse Files

(This message will disappear, once this node revised.)

A file is sparse if it contains blocks of zeros whose existance is recorded, but that have no space allocated on disk. When you specify the ‘`--sparse`’ (‘`-S`’) option in conjunction with the ‘`--create`’ (‘`-c`’) operation, `tar` tests all files for sparseness while archiving. If `tar` finds a file to be sparse, it uses a sparse representation of the file in the archive.

FIXME: xref Creating Archives

, for more information about creating archives.

‘`--sparse`’ (‘`-S`’) is useful when archiving files, such as dbm files, likely to contain many nulls. This option dramatically decreases the amount of space needed to store such an archive.

> **Please Note:** Always use ‘`--sparse`’ (‘`-S`’) when performing file system backups, to avoid archiving the expanded forms of files stored sparsely in the system.
>
> Even if your system has no no sparse files currently, some may be created in the future. If you use ‘`--sparse`’ (‘`-S`’) while making file system backups as a matter of course, you can be assured the archive will always take no more space on the media than the files take on disk (otherwise, archiving a disk filled with sparse files might take hundreds of tapes).

FIXME: xref incremental when node name is set.

`tar` ignores the '`--sparse`' ('`-S`') option when reading an archive.

'`--sparse`'
'`-S`' Files stored sparsely in the file system are represented sparsely in the archive. Use in conjunction with write operations.

7.4 Special Options for Archiving

(This message will disappear, once this node revised.)

To give the archive a name which will be recorded in it, use the '`--label=`*`archive-label`*' ('`-V` *`archive-label`*') option. This will write a special record identifying *volume-label* as the name of the archive to the front of the archive which will be displayed when the archive is listed with '`--list`' ('`-t`'). If you are creating a multi-volume archive with '`--multi-volume`' ('`-M`') (

FIXME: pxref Using Multiple Tapes

), then the volume label will have '`Volume` *`nnn`*' appended to the name you give, where *nnn* is the number of the volume of the archive. (If you use the '`--label=`*`archive-label`*' ('`-V` *`archive-label`*') option when reading an archive, it checks to make sure the label on the tape matches the one you give.

FIXME: xref Special Options for Archiving

.)

Files in the filesystem occasionally have "holes." A hole in a file is a section of the file's contents which was never written. The contents of a hole read as all zeros. On many operating systems, actualdisk storage is not allocated for holes, but they are counted in the length of the file. If you archive such a file, `tar` could create an archive longer than the original. To have `tar` attempt to recognize the holes in a file, use '`--sparse`' ('`-S`'). When you use the '`--sparse`' ('`-S`') option, then, for any file using less disk space than would be expected from its length, `tar` searches the file for consecutive stretches of zeros. It then records in the archive for the file where the consecutive stretches of zeros are, and only archives the "real contents" of the file. On extraction (using '`--sparse`' ('`-S`') is not needed on extraction) any such files have hols created wherever the continuous stretches of zeros were found. Thus, if you use '`--sparse`' ('`-S`'), `tar` archives won't take more space than the original.

When `tar` reads files, this causes them to have the access times updated. To have `tar` attempt to set the access times back to what they were before they were read, use the '`--atime-preserve`' option. This doesn't work for files that you don't own, unless you're root, and it doesn't interact with incremental dumps nicely (

FIXME: pxref Making Backups

), but it is good enough for some purposes.

7.5 The Structure of an Archive

(This message will disappear, once this node revised.)

While an archive may contain many files, the archive itself is a single ordinary file. Like any other file, an archive file can be written to a storage device such as a tape or disk, sent through a pipe or over a network, saved on the active file system, or even stored in another archive. An archive file is not easy to read or manipulate without using the `tar` utility or Tar mode in Emacs.

Physically, an archive consists of a series of file entries terminated by an end-of-archive entry, which consists of 512 zero bytes. A file entry usually describes one of the files in the archive (an *archive member*), and consists of a file header and the contents of the file. File headers contain file names and statistics, checksum information which `tar` uses to detect file corruption, and information about file types.

More than archive member can have the same file name. One way this situation can occur is if more than one version of a file has been stored in the archive. For information about adding new versions of a file to an archive,

FIXME: pxref Modifying

.

In addition to entries describing archive members, an archive may contain entries which `tar` itself uses to store information.

FIXME: xref Archive Label

, for an example of such an archive entry.

7.6 Operation mode modifiers

(This message will disappear, once this node revised.)

Operation mode modifiers

`-W`
`--verify`
Attempt to verify the archive after writing.

This option causes `tar` to verify the archive after writing it. Each volume is checked after it is written, and any discrepancies are recorded on the standard error output.

Verification requires that the archive be on a back-space-able medium. This means pipes, some cartridge tape drives, and some other devices cannot be verified.

`--remove-files`
Remove files after adding them to the archive.

`-k`
`--keep-old-files`
Do not overwrite existing files from archive.

The ‘`--keep-old-files`’ (‘`-k`’) option prevents `tar` from over-writing existing files with files with the same name from the archive.

The ‘`--keep-old-files`’ (‘`-k`’) option is meaningless with ‘`--list`’ (‘`-t`’).

`-S`
`--sparse`
Handle sparse files efficiently.

This option causes all files to be put in the archive to be tested for sparseness, and handled specially if they are. The ‘`--sparse`’ (‘`-S`’) option is useful when many `dbm` files, for example, are being backed up. Using this option dramatically decreases the amount of space needed to store such a file.

In later versions, this option may be removed, and the testing and treatment of sparse files may be done automatically with any special GNU options. For now, it is an option needing to be specified on the command line with the creation or updating of an archive.

`-O`
`--to-stdout`

Extract files to standard output.

When this option is used, instead of creating the files specified, `tar` writes the contents of the files extracted to its standard output. This may be useful if you are only extracting the files in order to send them through a pipe.

This option is meaningless with '`--list`' ('`-t`').

`--ignore-failed-read`

Do not exit with non-zero on unreadable files.

FIXME: This section needs to be written

To come: using Unix file linking capability to recreate directory structures—linking files into one subdirectory and then `tar`ring that directory.

to come: nice hairy example using absolute-names, newer, etc.

Piping one `tar` to another is an easy way to copy a directory's contents from one disk to another, while preserving the dates, modes, owners and link-structure of all the files therein.

```
cd sourcedirectory; tar cf - . | (cd targetdir; tar xf -)
```

or

FIXME: the following using standard input/output correct??

```
cd sourcedirectory; tar --create --file=- . | (cd targetdir; tar --extract --file=-)
```

Archive files can be used for transporting a group of files from one system to another: put all relevant files into an archive on one computer system, transfer the archive to another, and extract the contents there. The basic transfer medium might be magnetic tape, Internet FTP, or even electronic mail (though you must encode the archive with `uuencode` in order to transport it properly by mail). Both machines do not have to use the same operating system, as long as they both support the `tar` program.

FIXME: mention uuencode on a paragraph of its own

FIXME: end construction

8 Tapes and Other Archive Media

(This message will disappear, once this node revised.)

A few special cases about tape handling warrant more detailed description. These special cases are discussed below.

Many complexities surround the use of `tar` on tape drives. Since the creation and manipulation of archives located on magnetic tape was the original purpose of `tar`, it contains many features making such manipulation easier.

Archives are usually written on dismountable media—tape cartridges, mag tapes, or floppy disks.

The amount of data a tape or disk holds depends not only on its size, but also on how it is formatted. A 2400 foot long reel of mag tape holds 40 megabytes of data when formated at 1600 bits per inch. The physically smaller EXABYTE tape cartridge holds 2.3 gigabytes.

Magnetic media are re-usable—once the archive on a tape is no longer needed, the archive can be erased and the tape or disk used over. Media quality does deteriorate with use, however. Most tapes or disks should be disgarded when they begin to produce data errors. EXABYTE tape cartridges should be disgarded when they generate an *error count* (number of non-usable bits) of more than 10k.

Magnetic media are written and erased using magnetic fields, and should be protected from such fields to avoid damage to stored data. Sticking a floppy disk to a filing cabinet using a magnet is probably not a good idea.

8.1 Device selection and switching

(This message will disappear, once this node revised.)

`-f [`*`hostname`*`:]`*`file`*
`--file=[`*`hostname`*`:]`*`file`*

Use archive file or device *file* on *hostname*.

This option is used to specify the file name of the archive `tar` works on.

If the file name is ‘`-`’, `tar` reads the archive from standard input (when listing or extracting), or writes it to standard output (when creating). If the ‘`-`’ file name is given when updating an archive, `tar` will read the original archive from its standard input, and will write the entire new archive to its standard output.

If the file name contains a ‘`:`’, it is interpreted as ‘`hostname:filename`’. If the *hostname* contains an *at* sign (`@`), it is treated as ‘`user@hostname:filename`’. In either case, `tar` will invoke the command `rsh` (or `remsh`) to start up an ‘`/etc/rmt`’ on the remote machine. If you give an alternate login name, it will be given to the `rsh`. Naturally, the remote machine must have an executable ‘`/etc/rmt`’. This program is free software from the University of California, and a copy of the source code can be found with the sources for `tar`; it's compiled and installed by default.

If this option is not given, but the environment variable `TAPE` is set, its value is used; otherwise, old versions of `tar` used a default archive name (which was picked when `tar` was compiled). The default is normally set up to be the *first* tape drive or other transportable I/O medium on the system.

Starting with version 1.11.5, GNU `tar` uses standard input and standard output as the default device, and I will not try anymore supporting automatic device detection at installation time. This was failing really in too many cases, it was hopeless. This is now completely left to the installer to override standard input and standard output for default device, if this seems preferrable to him/her. Further, I think *most* actual usages of `tar` are done with pipes or disks, not really tapes, cartridges or diskettes.

Some users think that using standard input and output is running after trouble. This could lead to a nasty surprise on your screen if you forget to specify an output file name—especially if you are going through a network or terminal server capable of buffering large amounts of output. We had so many bug reports in that area of configuring default tapes automatically, and so many contradicting requests, that we finally consider the problem to be portably intractable. We could of course use something like '`/dev/tape`' as a default, but this is *also* running after various kind of trouble, going from hung processes to accidental destruction of real tapes. After having seen all this mess, using standard input and output as a default really sounds like the only clean choice left, and a very useful one too.

GNU `tar` reads and writes archive in blocks, I suspect this is the main reason why block devices are preferred over character devices. Most probably, block devices are more efficient too. The installer could also check for '`DEFTAPE`' in '`<sys/mtio.h>`'.

`--force-local`

Archive file is local even if it contains a colon.

`--rsh-command=`*`command`*

Use remote *command* instead of `rsh`. This option exists so that people who use something other than the standard `rsh` (e.g., a Kerberized `rsh`) can access a remote device.

When this command is not used, the shell command found when the `tar` program was installed is used instead. This is the first found of '`/usr/ucb/rsh`', '`/usr/bin/remsh`', '`/usr/bin/rsh`', '`/usr/bsd/rsh`' or '`/usr/bin/nsh`'. The installer may have overriden this by defining the environment variable `RSH` *at installation time.*

`-[0-7][lmh]`

Specify drive and density.

`-M`
`--multi-volume`

Create/list/extract multi-volume archive.

This option causes `tar` to write a *multi-volume* archive—one that may be larger than will fit on the medium used to hold it.

FIXME: xref Multi

.

`-L` *`num`*
`--tape-length=`*`num`*

Change tape after writing *num* x 1024 bytes.

This option might be useful when your tape drivers do not properly detect end of physical tapes. By being slightly conservative on the maximum tape length, you might avoid the problem entirely.

`-F` *`file`*
`--info-script=`*`file`*
`--new-volume-script=`*`file`*

Execute '`file`' at end of each tape. This implies '`--multi-volume`' ('`-M`').

The Remote Tape Server

In order to access the tape drive on a remote machine, `tar` uses the remote tape server written at the University of California at Berkeley. The remote tape server must be installed as '`/etc/rmt`' on any machine whose tape drive you want to use. `tar` calls '`/etc/rmt`' by running an `rsh` or `remsh` to the remote machine, optionally using a different login name if one is supplied.

A copy of the source for the remote tape server is provided. It is Copyright © 1983 by the Regents of the University of California, but can be freely distributed. Instructions for compiling and installing it are included in the '`Makefile`'.

Unless you use the [No value for "absolue-names"] option, GNU `tar` will not allow you to create an archive that contains absolute file names (a file name beginning with '`/`'.) If you try, `tar` will automatically remove the leading '`/`' from the file names it stores in the archive. It will also type a warning message telling you what it is doing.

When reading an archive that was created with a different `tar` program, GNU `tar` automatically extracts entries in the archive which have absolute file names as if the file names were not absolute. This is an important feature. A visitor here once gave a `tar` tape to an operator to restore; the operator used Sun `tar` instead of GNU `tar`, and the result was that it replaced large portions of our '`/bin`' and friends with versions from the tape; needless to say, we were unhappy about having to recover the file system from backup tapes.

For example, if the archive contained a file '`/usr/bin/computoy`', GNU `tar` would extract the file to '`usr/bin/computoy`', relative to the current directory. If you want to extract the files in an archive to the same absolute names that they had when the archive was created, you should do a '`cd /`' before extracting the files from the archive, or you should either use the '`--absolute-names`' ('`-P`') option, or use the command '`tar -C / ...`'.

Some versions of Unix (Ultrix 3.1 is know to have this problem), can claim that a short write near the end of a tape succeeded, when it actually failed. This will result in the -M option not working correctly. The best workaround at the moment is to use a significantly larger blocksize than the default 20.

In order to update an archive, `tar` must be able to backspace the archive in order to reread or rewrite a block that was just read (or written). This is currently possible only on two kinds of files: normal disk files (or any other file that can be backspaced with '`lseek`'), and industry-standard 9-track magnetic tape (or any other kind of tape that can be backspaced with the `MTIOCTOP` `ioctl`.

This means that the '`--append`' ('`-r`'), '`--update`' ('`-u`'), '`--concatenate`' ('`-A`'), and '`--delete`' commands will not work on any other kind of file. Some media simply cannot be backspaced, which means these commands and options will never be able to work on them. These non-backspacing media include pipes and cartridge tape drives.

Some other media can be backspaced, and `tar` will work on them once `tar` is modified to do so.

Archives created with the '`--multi-volume`' ('`-M`'), '`--label=`*`archive-label`*' ('`-V` *`archive-label`*'), and '`--incremental`' ('`-G`') options may not be readable by other version of `tar`. In particular, restoring a file that was split over a volume boundary will require some careful work with `dd`, if it can be done at all. Other versions of `tar` may also create an empty file whose name is that of the volume header. Some versions of `tar` may create normal files instead of directories archived with the '`--incremental`' ('`-G`') option.

Some Common Problems and their Solutions:

errors from system:
permission denied
no such file or directory
not owner

errors from `tar`:
directory checksum error
header format error

errors from media/system:
i/o error
device busy

8.2 Blocking

(This message will disappear, once this node revised.)

When writing to tapes, `tar` writes the contents of the archive in chunks known as *blocks*. To change the default blocksize, use the '`--block-size=`*`512-size`*' ('`-b` *`512-size`*') option. Each block will then be composed of *size* records. (Each `tar` record is 512 bytes.

FIXME: xref Archive Format

.) Each file written to the archive uses at least one full block. As a result, using a larger block size can result in more wasted space for small files. On the other hand, a larger block size can ofter be read and written much more efficiently.

Further complicating the problem is that some tape drives ignore the blocking entirely. For these, a larger block size can still improve performance (because the software layers above the tape drive still honor the blocking), but not as dramatically as on tape drives that honor blocking.

Wher reading an archive, `tar` can usually figure out the block size on itself. When this is the case, and a non-standard block size was used when the archive was created, `tar` will print a message about a non-standard blocking factor, and then operate normally. On some tape devices, however, `tar` cannot figure out the block size itself. On most of those, you can specify a blocking factor (with '`--block-size=`*`512-size`*' ('`-b` *`512-size`*')) larger than the actual blocking factor, and then use the '`--read-full-blocks`' ('`-B`') option. (If you specify a blocking factor with '`--block-size=`*`512-size`*' ('`-b` *`512-size`*') and don't use the '`--read-full-blocks`' ('`-B`') option, then `tar` will not attempt to figure out the blocking size itself.) On some devices, you must always specify the block size exactly with '`--block-size=`*`512-size`*' ('`-b` *`512-size`*') when reading, because `tar` cannot figure it out. In any case, use '`--list`' ('`-t`') before doing any extractions to see whether `tar` is reading the archive correctly.

If you use a blocking factor larger than 20, older `tar` programs might not be able to read the archive, so we recommend this as a limit to use in practice. GNU `tar`, however, will support arbitrarily large block sizes, limited only by the amount of virtual memory or the physical characteristics of the tape device.

If you are writing a compressed archive to tape with '`--compress`' ('`-Z`') or '`--gzip`' ('`-z`') (

FIXME: pxref Input and Output

), `tar` will not block the archive correctly. This doesn't matter if you are writing the archive to a normal file or through a pipe, but if you are writing it to a tape drive, then this causes problems. Use '`--compress-blocks`' or '`--gzip-block`' instead, to cause `tar` to arrange to have blocking work correctly.

8.2.1 Format Variations

(This message will disappear, once this node revised.)

Format parameters specify how an archive is written on the archive media. The best choice of format parameters will vary depending on the type and number of files being archived, and on the media used to store the archive.

To specify format parameters when accessing or creating an archive, you can use the options described in the following sections. If you do not specify any format parameters, `tar` uses default parameters. You cannot modify a compressed archive. If you create an archive with the '`--block-size=`*`512-size`*' ('`-b` *`512-size`*') option specified (

FIXME: pxref Blocking
Factor

), you must specify that block-size when operating on the archive.

FIXME: xref Matching Format Parameters

, for other examples of format parameter considerations.

8.2.2 The Blocking Factor of an Archive

(This message will disappear, once this node revised.)

The data in an archive is grouped into records, which are 512 bytes. Records are read and written in whole number multiples called *blocks*. The number of records in a block (ie. the size of a block in units of 512 bytes) is called the *blocking factor*. The '`--block-size=`*`512-size`*' ('`-b` *`512-size`*') option specifies the blocking factor of an archive. The default blocking factor is typically 20 (ie. 10240 bytes), but can be specified at installation. To find out the blocking factor of an existing archive, use '`tar --list --file=`*`archive-name`*'. This may not work on some devices.

Blocks are seperated by gaps, which waste space on the archive media. If you are archiving on magnetic tape, using a larger blocking factor (and therefore larger blocks) provides faster throughput and allows you to fit more data on a tape (because there are fewer gaps). If you are archiving on cartridge, a very large blocking factor (say 126 or more) greatly increases performance. A smaller blocking factor, on the other hand, may be usefull when archiving small files, to avoid archiving lots of nulls as `tar` fills out the archive to the end of the block. In general, the ideal block size depends on the size of the inter-block gaps on the tape you are using, and the average size of the files you are archiving.

FIXME: xref Creating
Archives

, for information on writing archives.

FIXME: need example of using a cartridge with blocksize=126 or more

Archives with blocking factors larger than 20 cannot be read by very old versions of `tar`, or by some newer versions of `tar` running on old machines with small address spaces. With GNU `tar`, the blocking factor of an archive is limited only by the maximum block size of the device containing the archive, or by the amount of available virtual memory.

If you use a non-default blocking factor when you create an archive, you must specify the same blocking factor when you modify that archive. Some archive devices will also require you to specify the blocking factor when reading that archive, however this is not typically the case. Usually, you can use '`--list`' ('`-t`') without specifying a blocking factor—`tar` reports a non-default block size and then lists the archive members as it would normally. To extract files from an archive with a non-standard blocking factor (particularly if you're not sure what the blocking factor is), you can usually use the '`--read-full-blocks`' ('`-B`') option while specifying a blocking factor larger then the blocking factor of the archive (ie. '`tar --extract --read-full-blocks --block-size=300`'.

FIXME: xref Listing Contents

for more information on the '`--list`' ('`-t`') operation.

FIXME: xref read-full-blocks

for a more detailed explanation of that option.

'`--block-size=`*`number`*'
'`-b` *`number`*'

Specifies the blocking factor of an archive. Can be used with any operation, but is usually not necessary with '`--list`' ('`-t`').

Device blocking

`-b` *`blocks`*
`--block-size=`*`blocks`*

Set block size to *blocks* ∗ 512 bytes.

This option is used to specify a *blocking factor* for the archive. When reading or writing the archive, `tar`, will do reads and writes of the archive in blocks of *block* ∗ 512 bytes.

The default blocking factor is set when `tar` is compiled, and is typically 20.

Blocking factors larger than 20 cannot be read by very old versions of `tar`, or by some newer versions of `tar` running on old machines with small address spaces.

With a magnetic tape, larger blocks give faster throughput and fit more data on a tape (because there are fewer inter-record gaps). If the archive is in a disk file or a pipe, you may want to specify a smaller blocking factor, since a large one will result in a large number of null bytes at the end of the archive.

When writing cartridge or other streaming tapes, a much larger blocking factor (say 126 or more) will greatly increase performance. However, you must specify the same blocking factor when reading or updating the archive.

With GNU `tar` the blocking factor is limited only by the maximum block size of the device containing the archive, or by the amount of available virtual memory.

`--block-compress`

Block the output of compression for tapes.

`-i`
`--ignore-zeros`

Ignore blocks of zeros in archive (means EOF).

The '`--ignore-zeros`' ('`-i`') option causes `tar` to ignore blocks of zeros in the archive. Normally a block of zeros indicates the end of the archive, but when reading a damaged archive, or one which was created by `cat`-ing several archives together, this option allows `tar` to read the entire archive. This option is not on by default because many versions of `tar` write garbage after the zeroed blocks.

Note that this option causes `tar` to read to the end of the archive file, which may sometimes avoid problems when multiple files are stored on a single physical tape.

`-B`
`--read-full-blocks`

Reblock as we read (for reading 4.2BSD pipes).

If '`--read-full-blocks`' ('`-B`') is used, `tar` will not panic if an attempt to read a block from the archive does not return a full block. Instead, `tar` will keep reading until it has obtained a full block.

This option is turned on by default when `tar` is reading an archive from standard input, or from a remote machine. This is because on BSD Unix systems, a read of a pipe will return however much happens to be in the pipe, even if it is less than `tar` requested. If this option was not used, `tar` would fail as soon as it read an incomplete block from the pipe.

This option is also useful with the commands for updating an archive.

Tape blocking

FIXME: Appropriate options should be moved here from elsewhere.

When handling various tapes or cartridges, you have to take care of selecting a proper blocking, that is, the number of disk blocks you put together as a single tape block on the tape, without intervening tape gaps. A *tape gap* is a small landing area on the tape with no information on it, used for decelerating the tape to a full stop, and for later regaining the reading or writing speed. When the driver starts reading a tape block, the tape block has to be read whole without stopping, as a tape gap is needed to stop the tape motion without loosing information.

Using higher blocking (putting more disk blocks per tape block) will use the tape more efficiently as there will be less tape gaps. But reading such tapes may be more difficult for the system, as more memory will be required to receive at once the whole block. Further, if there is a reading error on a huge tape block, this is less likely that the system will succeed in recovering the information. So, blocking should not be too low, nor it should be too high. `tar` uses by default a blocking of 20 for historical reasons, and it does not really matter when reading or writing to disk. Current tape technology would easily accomodate higher blockings. Sun recommends a blocking of 126 for Exabytes and 96 for DATs. Other manufacturers may use different recommendations for the same tapes. This might also depends of the buffering techniques used inside modern tape controllers. Some imposes a minimum blocking, or a maximum blocking. Others request blocking to be some exponent of two.

So, there is no fixed rule for blocking. But blocking at read time should ideally be the same as blocking used at write time. At one place I know, with a wide variety of equipment, they found it best to use a blocking of 32 to guarantee that their tapes are fully interchangeable.

I was also told that, for recycled tapes, prior erasure (by the same drive unit that will be used to create the archives) sometimes lowers the error rates observed at rewriting time.

8.3 Many archives on one tape

FIXME: Appropriate options should be moved here from elsewhere.

Most tape devices have two entries in the '`/dev`' directory, or entries that come in pairs, which differ only in the minor number for this device. Let's take for example '`/dev/tape`', which often points to the only or usual tape device of a given system. There might be a corresponding '`/dev/nrtape`' or '`/dev/ntape`'. The simpler name is the *rewinding* version of the device, while the name having '`nr`' in it is the *no rewinding* version of the same device.

A rewinding tape device will bring back the tape to its beginning point automatically when this device is opened or closed. Since `tar` opens the archive file before using it and closes it afterwards, this means that a simple:

```
tar cf /dev/tape directory
```

will reposition the tape to its beginning both prior and after saving *directory* contents to it, thus erasing prior tape contents and making it so that any subsequent write operation will destroy what has just been saved.

So, a rewinding device is normally meant to hold one and only one file. If you want to put more than one `tar` archive on a given tape, you will need to avoid using the rewinding version of the tape device. You will also have to pay special attention to tape positioning. Errors in positionning may overwrite the valuable data already on your tape. Many people, burnt by past experiences, will only use rewinding devices and limit themselves to one file per tape, precisely to avoid the risk of such errors. Be fully aware that writing at the wrong position on a tape loses all information past this point and most probably until the end of the tape, and this destroyed information *cannot* be recovered.

To save *directory-1* as a first archive at the beginning of a tape, and leave that tape ready for a second archive, you should use:

```
mt -f /dev/nrtape rewind
tar cf /dev/nrtape directory-1
```

Tape marks are special magnetic patterns written on the tape media, which are later recognizable by the reading hardware. These marks are used after each file, when there are many on a single tape. An empty file (that is to say, two tape marks in a row) signal the logical end of the tape, after which no file exist. Usually, non-rewinding tape device drivers will react to the close request issued by `tar` by first writing two tape marks after your archive, and by backspacing over one of these. So, if you remove the tape at that time from the tape drive, it is properly terminated. But if you write another file at the current position, the second tape mark will be erased by the new information, leaving only one tape mark between files.

So, you may now save *directory-2* as a second archive after the first on the same tape by issuing the command:

```
tar cf /dev/nrtape directory-2
```

and so on for all the archives you want to put on the same tape.

Another usual case is that you do not write all the archives the same day, and you need to remove and store the tape between two archive sessions. In general, you must remember how many files are already saved on your tape. Suppose your tape already has 16 files on it, and that you are ready to write the 17th. You have to take care of skipping the first 16 tape marks before saving *directory-17*, say, by using these commands:

```
mt -f /dev/nrtape rewind
mt -f /dev/nrtape fsf 16
tar cf /dev/nrtape directory-17
```

In all the previous examples, we put aside blocking considerations, but you should do the proper things for that as well.

FIXME: xref Blocking

.

8.3.1 Tape Positions and Tape Marks

(This message will disappear, once this node revised.)

Just as archives can store more than one file from the file system, tapes can store more than one archive file. To keep track of where archive files (or any other type of file stored on tape) begin and end, tape archive devices write magnetic *tape marks* on the archive media. Tape drives write one tape mark between files, two at the end of all the file entries.

If you think of data as a series of "0000"'s, and tape marks as "x"'s, a tape might look like the following:

```
0000x000000x00000x00x00000xx-------------------------
```

Tape devices read and write tapes using a read/write *tape head*—a physical part of the device which can only access one point on the tape at a time. When you use `tar` to read or write archive data from a tape device, the device will begin reading or writing from wherever on the tape the tape head happens to be, regardless of which archive or what part of the archive the tape head is on. Before writing an archive, you should make sure that no data on the tape will be overwritten (unless it is no longer needed). Before reading an archive, you should make sure the tape head is at the beginning of the archive you want to read. (The `restore` script will find the archive automatically.

FIXME: xref Scripted
Restoration

).

FIXME: xref mt

, for an explanation of the tape moving utility.

If you want to add new archive file entries to a tape, you should advance the tape to the end of the existing file entries, backspace over the last tape mark, and write the new archive file. If you were to add two archives to the example above, the tape might look like the following:

```
0000x000000x00000x00x00000x000x0000xx----------------
```

8.3.2 The `mt` Utility

(This message will disappear, once this node revised.)

FIXME: is it true that this only works on non-block devices? should
FIXME: explain the difference, xref to block-size (fixed or variable).

You can use the `mt` utility to advance or rewind a tape past a specified number of archive files on the tape. This will allow you to move to the beginning of an archive before extracting or reading it, or to the end of all the archives before writing a new one.

FIXME: why isn't there an "advance 'til you find two tape marks together"?

The syntax of the `mt` command is:

```
mt [-f tapename] operation [number]
```

where *tapename* is the name of the tape device, *number* is the number of times an operation is performed (with a default of one), and *operation* is one of the following:

FIXME: is there any use for record operations?

`eof` `weof`	Writes *number* tape marks at the current position on the tape.
`fsf`	Moves tape position forward *number* files.
`bsf`	Moves tape position back *number* files.
`rewind`	Rewinds the tape. (Ignores *number*).
`offline` `rewoff1`	Rewinds the tape and takes the tape device off-line. (Ignores *number*).
`status`	Prints status information about the tape unit.

FIXME: is there a better way to frob the spacing on the list?

If you don't specify a *tapename*, `mt` uses the environment variable TAPE; if TAPE does not exist, `mt` uses the device '`/dev/rmt12`'.

`mt` returns a 0 exit status when the operation(s) were successful, 1 if the command was unrecognized, and 2 if an operation failed.

FIXME: new node on how to find an archive?

If you use '`--extract`' ('`-x`') with the '`--label=`*`archive-label`*' ('`-V` *`archive-label`*') option specified, `tar` will read an archive label (the tape head has to be positioned on it) and print an error if the archive label doesn't match the *archive-name* specified. *archive-name* can be any regular expression. If the labels match, `tar` extracts the archive.

FIXME: xref Archive Label

.

FIXME: xref Matching Format Parameters

.

FIXME: fix cross references

'`tar --list --label`' will cause `tar` to print the label.

FIXME: program to list all the labels on a tape?

8.4 Using Multiple Tapes

(This message will disappear, once this node revised.)

Often you might want to write a large archive, one larger than will fit on the actual tape you are using. In such a case, you can run multiple `tar` commands, but this can be inconvenient, particularly if you are using options like '`--exclude=`*`pattern`*' or dumping entire filesystems. Therefore, `tar` supports multiple tapes automatically.

Use '`--multi-volume`' ('`-M`') on the command line, and then `tar` will, when it reaches the end of the tape, prompt for another tape, and continue the archive. Each tape will have an independent archive, and can be read without needing the other. (As an exception to this, the file that `tar` was archiving when it ran out of tape will usually be split between the two archives; in this case you need to extract from the first archive, using '`--multi-volume`' ('`-M`'), and then put in the second tape when prompted, so `tar` can restore both halves of the file.)

When prompting for a new tape, `tar` accepts any of the following responses:

'`?`'	Request `tar` to explain possible responses
'`q`'	Request `tar` to exit immediately.
'`n` *`file name`*'	Request `tar` to write the next volume on the file *file name*.
'`!`'	Request `tar` to run a subshell.
'`y`'	Request `tar` to begin writing the next volume.

(You should only type '`y`' after you have changed the tape; otherwise `tar` will write over the volume it just finished.)

If you want more elaborate behavior than this, give `tar` the '`--info-script=`*`script-name`*' ('`-F` *`script-name`*') option. The file *script-name* is expected to be a program (or shell script) to be run instead of the normal prompting procedure. When the program finishes, `tar` will immediately begin writing the next volume. The behavior of the '`n`' response to the normal tape-change prompt is not available if you use '`--info-script=`*`script-name`*' ('`-F` *`script-name`*').

The method `tar` uses to detect end of tape is not perfect, and fails on some operating systems or on some devices. You can use the '`--tape-length=`*`1024-size`*' ('`-L` *`1024-size`*') option if `tar` can't detect the end of the tape itself. The *size* argument should be the size of the tape.

The volume number used by `tar` in its tape-change prompt can be changed; if you give the '`--volno-file=`*`file-of-number`*' option, then *file-of-number* should contain a decimal number. That number will be used as the volume number of the first volume written. When `tar` is finished, it will rewrite the file with the now-current volume number. (This does not change the volume number written on a tape label (

FIXME: pxref Special Options for Archiving

; it *only* affects the number used in the prompt.)

If you want `tar` to cycle through a series of tape drives, then you can use the '`n`' response to the tape-change prompt. This is error prone, however, and doesn't work at all with '`--info-script=`*`script-name`*' ('`-F` *`script-name`*'). Therefore, if you give `tar` multiple '`--file=`*`archive-name`*' ('`-f` *`archive-name`*') options, then the specified files will be used, in sequence, as the successive volumes of the archive. Only when the first one in the sequence needs to be used again will `tar` prompt for a tape change (or run the info script).

Multi-volume archives

With '`--multi-volume`' ('`-M`'), `tar` will not abort when it cannot read or write any more data. Instead, it will ask you to prepare a new volume. If the archive is on a magnetic tape, you should change tapes now; if the archive is on a floppy disk, you should change disks, etc.

Each volume of a multi-volume archive is an independent `tar` archive, complete in itself. For example, you can list or extract any volume alone; just don't specify '`--multi-volume`' ('`-M`'). However, if one file in the archive is split across volumes, the only way to extract it successfully is with a multi-volume extract command '`--extract --multi-volume`' ('`-xM`') starting on or before the volume where the file begins.

8.4.1 Archives Longer than One Tape or Disk

(This message will disappear, once this node revised.)

To create an archive that is larger than will fit on a single unit of the media, use the '`--multi-volume`' ('`-M`') option in conjunction with the '`--create`' ('`-c`') option (

FIXME: pxref Creating Archives

). A *multi-volume* archive can be manipulated like any other archive (provided the '`--multi-volume`' ('`-M`') option is specified), but is stored on more than one tape or disk.

When you specify '`--multi-volume`' ('`-M`'), `tar` does not report an error when it comes to the end of an archive volume (when reading), or the end of the media (when writing). Instead, it prompts you to load a new storage volume. If the archive is on a magnetic tape, you should change tapes when you see the prompt; if the archive is on a floppy disk, you should change disks; etc.

You can read each individual volume of a multi-volume archive as if it were an archive by itself. For example, to list the contents of one volume, use '`--list`' ('`-t`'), without '`--multi-volume`' ('`-M`') specified. To extract an archive member from one volume (assuming it is described that volume), use '`--extract`' ('`-x`'), again without '`--multi-volume`' ('`-M`').

If an archive member is split across volumes (ie. its entry begins on one volume of the media and ends on another), you need to specify '`--multi-volume`' ('`-M`') to extract it successfully. In this case, you should load the volume where the archive member starts, and use '`tar --extract --multi-volume`'—`tar` will prompt for later volumes as it needs them.

FIXME: xref Extracting From Archives

for more information about extracting archives.

'`--info-script=`*`script-name`*' ('`-F` *`script-name`*') is like '`--multi-volume`' ('`-M`'), except that `tar` does not prompt you directly to change media volumes when a volume is full—instead, `tar` runs commands you have stored in *script-name*. This option can be used to broadcast messages such as '`Someone please come change my tape`' when performing unattended backups. When *script-name* is done, `tar` will assume that the media has been changed.

FIXME: There should be a sample program here, including an exit before
FIXME: end.

'`--multi-volume`'
'`-M`' Creates a multi-volume archive, when used in conjunction with '`--create`' ('`-c`'). To perform any other operation on a multi-volume archive, specify '`--multi-volume`' ('`-M`') in conjunction with that operation.

'`--info-script=`*`program-file`*'
'`-F` *`program-file`*'
Creates a multi-volume archive via a script. Used in conjunction with '`--create`' ('`-c`').

8.4.2 Tape Files

(This message will disappear, once this node revised.)

When `tar` writes an archive to tape, it creates a single tape file. If multiple archives are written to the same tape, one after the other, they each get written as separate tape files. When extracting, it is necessary to position the tape at the right place before running `tar`. To do this, use the `mt` command. For more information on the `mt` command and on the organization of tapes into a sequence of tape files.

FIXME: see ***.

8.5 Including a Label in the Archive

(This message will disappear, once this node revised.)

FIXME: Should the arg to –label be a quoted string?? no - ringo

To avoid problems caused by misplaced paper labels on the archive media, you can include a *label* entry—an archive member which contains the name of the archive—in the archive itself. Use the '`--label=`*`archive-label`*' ('`-V` *`archive-label`*') option in conjunction with the '`--create`' ('`-c`') operation to include a label entry in the archive as it is being created.

If you create an archive using both '`--label=`*`archive-label`*' ('`-V` *`archive-label`*') and '`--multi-volume`' ('`-M`'), each volume of the archive will have an archive label of the form '`archive-label Volume n`', where *n* is 1 for the first volume, 2 for the next, and so on.

FIXME: xref Multi-Volume Archives

, for information on creating multiple volume archives.

If you extract an archive using '`--label=archive-label`' ('`-V archive-label`'), `tar` will print an error if the archive label doesn't match the *archive-label* specified, and will then not extract the archive. You can include a regular expression in *archive-label*, in this case only.

FIXME: why is a reg. exp. useful here? (to limit extraction to a
FIXME: specific group? ie for multi-volume???

To find out an archive's label entry (or to find out if an archive has a label at all), use '`tar --list --verbose`'. `tar` will print the label first, and then print archive member information, as in the example below:

```
% tar --verbose --list --file=iamanarchive
V--------- 0/0        0 Mar  7 12:01 1992 iamalabel--Volume Header--
-rw-rw-rw- ringo/user 40 May 21 13:30 1990 iamafilename
```

'`--label=archive-label`'
'`-V archive-label`'

Includes an *archive-label* at the beginning of the archive when the archive is being created, when used in conjunction with the '`--create`' ('`-c`') option. Checks to make sure the archive label matches the one specified (when used in conjunction with the '`--extract`' ('`-x`') option.

FIXME: was –volume

9 Performing Backups and Restoring Files

(This message will disappear, once this node revised.)

```
.* dumps
. + what are dumps

. + different levels of dumps
.  - full dump = dump everything
.  - level 1, level 2 dumps etc, -
A level n dump dumps everything changed since the last level
n-1 dump (?)

. + how to use scripts for dumps  (ie, the concept)
.  - scripts to run after editing backup specs (details)

. + Backup Specs, what is it.
.  - how to customize
.  - actual text of script  [/sp/dump/backup-specs]

. + Problems
.  - rsh doesn't work
.  - rtape isn't installed
.  - (others?)

. + the --incremental option of tar

. + tapes
.  - write protection
.  - types of media
.    : different sizes and types, useful for different things
.  - files and tape marks
     one tape mark between files, two at end.
.  - positioning the tape
     MT writes two at end of write, backspaces over one when writing again.
```

To *back up* a file system means to create archives that contain all the files in that file system. Those archives can then be used to restore any or all of those files (for instance if a disk crashes or a file is accidently deleted). File system *backups* are also called *dumps*.

9.1 Using `tar` to Perform Full Dumps

(This message will disappear, once this node revised.)

Full dumps should only be made when no other people or programs are modifying files in the filesystem. If files are modified while `tar` is making the backup, they may not be stored properly in the archive, in which case you won't be able to restore them if you have to. (Files not being modified are written with no trouble, and do not corrupt the entire archive.)

You will want to use the '`--label=`*`archive-label`*' ('`-V` *`archive-label`*') option to give the archive a volume label, so you can tell what this archive is even if the label falls off the tape, or anything like that.

Unless the filesystem you are dumping is guaranteed to fit on one volume, you will need to use the '`--multi-volume`' ('`-M`') option. Make sure you have enough tapes on hand to complete the backup.

If you want to dump each filesystem separately you will need to use the '`--one-file-system`' ('`-l`') option to prevent `tar` from crossing filesystem boundaries when storing (sub)directories.

The '`--incremental`' ('`-G`') option is not needed, since this is a complete copy of everything in the filesystem, and a full restore from this backup would only be done onto a completely empty disk.

Unless you are in a hurry, and trust the `tar` program (and your tapes), it is a good idea to use the '`--verify`' ('`-W`') option, to make sure your files really made it onto the dump properly. This will also detect cases where the file was modified while (or just after) it was being archived. Not all media (notably cartridge tapes) are capable of being verified, unfortunately.

'`--listed-incremental=`*`snapshot-file`*' ('`-g` *`snapshot-file`*') take a file name argument always. If the file doesn't exist, run a level zero dump, creating the file. If the file exists, uses that file to see what has changed.

'`--incremental`' ('`-G`')

FIXME: look it up

'`--incremental`' ('`-G`') handle old GNU-format incremental backup.

This option should only be used when creating an incremental backup of a filesystem. When the '`--incremental`' ('`-G`') option is used, `tar` writes, at the beginning of the archive, an entry for each of the directories that will be operated on. The entry for a directory includes a list of all the files in the directory at the time the dump was done, and a flag for each file indicating whether the file is going to be put in the archive. This information is used when doing a complete incremental restore.

Note that this option causes `tar` to create a non-standard archive that may not be readable by non-GNU versions of the `tar` program.

The '`--incremental`' ('`-G`') option means the archive is an incremental backup. Its meaning depends on the command that it modifies.

If the '`--incremental`' ('`-G`') option is used with '`--list`' ('`-t`'), `tar` will list, for each directory in the archive, the list of files in that directory at the time the archive was created. This information is put out in a format that is not easy for humans to read, but which is unambiguous for a program: each file name is preceded by either a '`Y`' if the file is present in the archive, an '`N`' if the file is not included in the archive, or a '`D`' if the file is a directory (and is included in the archive). Each file name is terminated by a null character. The last file is followed by an additional null and a newline to indicate the end of the data.

If the '`--incremental`' ('`-G`') option is used with '`--extract`' ('`-x`'), then when the entry for a directory is found, all files that currently exist in that directory but are not listed in the archive *are deleted from the directory.*

This behavior is convenient when you are restoring a damaged file system from a succession of incremental backups: it restores the entire state of the file system to that which obtained when the backup was made. If you don't use '`--incremental`' ('`-G`'), the file system will probably fill up with files that shouldn't exist any more.

'`--listed-incremental=`*`snapshot-file`*' ('`-g` *`snapshot-file`*') handle new GNU-format incremental backup.

'`--listed-incremental=`*`snapshot-file`*' ('`-g` *`snapshot-file`*') acts like '`--incremental`' ('`-G`'), but when used in conjunction with '`--create`' ('`-c`') will also cause `tar` to use the file *file*, which contains information about the state of the filesystem at the time of the last backup, to decide which files to include in the archive being created. That file will then be updated by `tar`. If the file *file* does not exist when this option is specified, `tar` will create it, and include all appropriate files in the archive.

The file, which is archive independent, contains the date it was last modified and a list of devices, inode numbers and directory names. `tar` will archive files with newer mod dates or inode change times, and directories with an unchanged inode number and device but a changed directory name. The file is updated after the files to be archived are determined, but before the new archive is actually created.

9.2 Using `tar` to Perform Incremental Dumps

(This message will disappear, once this node revised.)

Performing incremental dumps is similar to performing full dumps, although a few more options will usually be needed.

You will need to use the '`-N` *`date`*' option to tell `tar` to only store files that have been modified since *date*. *date* should be the date and time of the last full/incremental dump.

A standard scheme is to do a *monthly* (full) dump once a month, a *weekly* dump once a week of everything since the last monthly and a *daily* every day of everything since the last (weekly or monthly) dump.

Here is a copy of the script used to dump the filesystems of the machines here at the Free Software Foundation. This script is run via `cron` late at night when people are least likely to be using the machines. This script dumps several filesystems from several machines at once (via NFS). The operator is responsible for ensuring that all the machines will be up at the time the dump happens. If a machine is not running, its files will not be dumped, and the next day's incremental dump will *not* store files that would have gone onto that dump.

```
#!/bin/csh
# Dump thingie
set now = `date`
set then = `cat date.nfs.dump`
/u/hack/bin/tar -c -G -v\
 -f /dev/rtu20\
 -b 126\
 -N "$then"\
 -V "Dump from $then to $now"\
 /alpha-bits/gp\
 /gnu/hack\
 /hobbes/u\
 /spiff/u\
 /sugar-bombs/u
echo $now > date.nfs.dump
mt -f /dev/rtu20 rew
```

Output from this script is stored in a file, for the operator to read later.

This script uses the file '`date.nfs.dump`' to store the date/time of the last dump.

Since this is a streaming tape drive, no attempt to verify the archive is done. This is also why the high blocking factor (126) is used. The tape drive must also be rewound by the `mt` command after the dump is made.

9.3 The Incremental Options

(This message will disappear, once this node revised.)

'`--incremental`' ('`-G`') is used in conjunction with '`--create`' ('`-c`'), '`--extract`' ('`-x`') or '`--list`' ('`-t`') when backing up and restoring file systems. An archive cannot be extracted or listed with the '`--incremental`' ('`-G`') option specified unless it was created with the option specified. This option should only be used by a script, not by the user, and is usually disregarded in favor of '`--listed-incremental=`*`snapshot-file`*' ('`-g` *`snapshot-file`*'), which is described below.

'`--incremental`' ('`-G`') in conjunction with '`--create`' ('`-c`') causes `tar` to write, at the beginning of the archive, an entry for each of the directories that will be archived. The entry for a directory includes a list of all the files in the directory at the time the archive was created and a flag for each file indicating whether or not the file is going to be put in the archive.

Note that this option causes `tar` to create a non-standard archive that may not be readable by non-GNU versions of the `tar` program.

'`--incremental`' ('`-G`') in conjunction with '`--extract`' ('`-x`') causes `tar` to read the lists of directory contents previously stored in the archive, *delete* files in the file system that did not exist in their directories when the archive was created, and then extract the files in the archive.

This behavior is convenient when restoring a damaged file system from a succession of incremental backups: it restores the entire state of the file system to that which obtained when the backup was made. If '`--incremental`' ('`-G`') isn't specified, the file system will probably fill up with files that shouldn't exist any more.

'`--incremental`' ('`-G`') in conjunction with '`--list`' ('`-t`'), causes `tar` to print, for each directory in the archive, the list of files in that directory at the time the archive was created. This information is put out in a format that is not easy for humans to read, but which is unambiguous for a program: each file name is preceded by either a '`Y`' if the file is present in the archive, an '`N`' if the file is not included in the archive, or a '`D`' if the file is a directory (and is included in the archive). Each file name is terminated by a null character. The last file is followed by an additional null and a newline to indicate the end of the data.

'`--listed-incremental=`*`snapshot-file`*' ('`-g` *`snapshot-file`*') acts like '`--incremental`' ('`-G`'), but when used in conjunction with '`--create`' ('`-c`') will also cause `tar` to use the file *snapshot-file*, which contains information about the state of the file system at the time of the last backup, to decide which files to include in the archive being created. That file will then be updated by `tar`. If the file *file* does not exist when this option is specified, `tar` will create it, and include all appropriate files in the archive.

The file *file*, which is archive independent, contains the date it was last modified and a list of devices, inode numbers and directory names. `tar` will archive files with newer mod dates or inode change times, and directories with an unchanged inode number and device but a changed directory name. The file is updated after the files to be archived are determined, but before the new archive is actually created.

FIXME: this section needs to be written

9.4 Levels of Backups

(This message will disappear, once this node revised.)

An archive containing all the files in the file system is called a *full backup* or *full dump*. You could insure your data by creating a full dump every day. This strategy, however, would waste a substantial amount of archive media and user time, as unchanged files are daily re-archived.

It is more efficient to do a full dump only occasionally. To back up files between full dumps, you can a incremental dump. A *level one* dump archives all the files that have changed since the last full dump.

A typical dump strategy would be to perform a full dump once a week, and a level one dump once a day. This means some versions of files will in fact be archived more than once, but this dump strategy makes it possible to restore a file system to within one day of accuracy by only extracting two archives—the last weekly (full) dump and the last daily (level one) dump. The only information lost would be in files changed or created since the last daily backup. (Doing dumps more than once a day is usually not worth the trouble).

GNU `tar` comes with scripts you can use to do full and level-one dumps. Using scripts (shell programs) to perform backups and restoration is a convenient and reliable alternative to typing out file name lists and `tar` commands by hand.

Before you use these scripts, you need to edit the file '`backup-specs`', which specifies parameters used by the backup scripts and by the restore script.

FIXME: xref Script Syntax

. Once the backup parameters are set, you can perform backups or restoration by running the appropriate script.

The name of the restore script is `restore`. The names of the level one and full backup scripts are, respectively, `level-1` and `level-0`. The `level-0` script also exists under the name `weekly`, and the `level-1` under the name `daily`—these additional names can be changed according to your backup schedule.

FIXME: xref Scripted Restoration

, for more information on running the restoration script.

FIXME: xref Scripted Backups

, for more information on running the backup scripts.

Please Note: The backup scripts and the restoration scripts are designed to be used together. While it is possible to restore files by hand from an archive which was created using a backup script, and to create an archive by hand which could then be extracted using the restore script, it is easier to use the scripts.

FIXME: xref incremental
and listed-incremental

, before making such an attempt.

FIXME: shorten node names

9.5 Setting Parameters for Backups and Restoration

(This message will disappear, once this node revised.)

The file '`backup-specs`' specifies backup parameters for the backup and restoration scripts provided with `tar`. You must edit '`backup-specs`' to fit your system configuration and schedule before using these scripts.

FIXME: This about backup scripts needs to be written: BS is a shell
FIXME: script .... thus ... '`backup-specs`' is in shell script
FIXME: syntax. xref Script Syntax, for an explanation of this syntax.
FIXME:
FIXME: whats a parameter .... looked at by the backup scripts ... which
FIXME: will be expecting to find ... now syntax ... value is linked to
FIXME: lame ... '`backup-specs`' specifies the following parameters:

`ADMINISTRATOR`
The user name of the backup administrator.

`BACKUP_HOUR`
The hour at which the backups are done. This can be a number from 0 to 23, or the string '`now`'.

`TAPE_FILE`
The device `tar` writes the archive to. This device should be attached to the host on which the dump scripts are run.

FIXME: examples for all ...

`TAPE_STATUS`
The command to use to obtain the status of the archive device, including error count. On some tape drives there may not be such a command; in that case, simply use 'TAPE_STATUS=false'.

`BLOCKING`
The blocking factor `tar` will use when writing the dump archive.

FIXME: xref Blocking Factor

.

`BACKUP_DIRS`
A list of file systems to be dumped. You can include any directory name in the list—subdirectories on that file system will be included, regardless of how they may look to other networked machines. Subdirectories on other file systems will be ignored.

The host name specifies which host to run `tar` on, and should normally be the host that actually contains the file system. However, the host machine must have GNU `tar` installed, and must be able to access the directory containing the backup scripts and their support files using the same file name that is used on the machine where the scripts are run (ie. what `pwd` will print when in that directory on that machine). If the host that contains the file system does not have this capability, you can specify another host as long as it can access the file system through NFS.

`BACKUP_FILES`

A list of individual files to be dumped. These should be accessible from the machine on which the backup script is run.

FIXME: same file name, be specific. through nfs ...

9.5.1 An Example Text of '`Backup-specs`'

(This message will disappear, once this node revised.)

The following is the text of '`backup-specs`' as it appears at FSF:

```
# site-specific parameters for file system backup.

ADMINISTRATOR=friedman
BACKUP_HOUR=1
TAPE_FILE=/dev/nrsmt0
TAPE_STATUS="mts -t $TAPE_FILE"
BLOCKING=124
BACKUP_DIRS="
albert:/fs/fsf
apple-gunkies:/gd
albert:/fs/gd2
albert:/fs/gp
geech:/usr/jla
churchy:/usr/roland
albert:/
albert:/usr
apple-gunkies:/
apple-gunkies:/usr
gnu:/hack
gnu:/u
apple-gunkies:/com/mailer/gnu
apple-gunkies:/com/archive/gnu"

BACKUP_FILES="/com/mailer/aliases /com/mailer/league*[a-z]"
```

9.5.2 Syntax for '`Backup-specs`'

(This message will disappear, once this node revised.)

'`backup-specs`' is in shell script syntax. The following conventions should be considered when editing the script:

FIXME: "conventions?"

A quoted string is considered to be contiguous, even if it is on more than one line. Therefore, you cannot include commented-out lines within a multi-line quoted string. BACKUP_FILES and BACKUP_DIRS are the two most likely parameters to be multi-line.

A quoted string typically cannot contain wildcards. In '`backup-specs`', however, the parameters BACKUP_DIRS and BACKUP_FILES can contain wildcards.

9.6 Using the Backup Scripts

(This message will disappear, once this node revised.)

The syntax for running a backup script is:

`'script-name' [time-to-be-run]`

where *time-to-be-run* can be a specific system time, or can be *`now`*. If you do not specify a time, the script runs at the time specified in '`backup-specs`' (

FIXME: pxref Script Syntax

).

You should start a script with a tape or disk mounted. Once you start a script, it prompts you for new tapes or disks as it needs them. Media volumes don't have to correspond to archive files—a multi-volume archive can be started in the middle of a tape that already contains the end of another multi-volume archive. The `restore` script prompts for media by its archive volume, so to avoid an error message you should keep track of which tape (or disk) contains which volume of the archive.

FIXME: xref Scripted Restoration

.

FIXME: have file names changed?

The backup scripts write two files on the file system. The first is a record file in '`/etc/tar-backup/`', which is used by the scripts to store and retrieve information about which files were dumped. This file is not meant to be read by humans, and should not be deleted by them.

FIXME: xref incremental and listed-incremental

, for a more detailed explanation of this file.

The second file is a log file containing the names of the file systems and files dumped, what time the backup was made, and any error messages that were generated, as well as how much space was left in the media volume after the last volume of the archive was written. You should check this log file after every backup. The file name is '`log-`*`mmm-ddd-yyyy`*`-level-1`' or '`log-`*`mmm-ddd-yyyy`*`-full`'.

The script also prints the name of each system being dumped to the standard output.

FIXME: the section on restore scripts is commented out.
FIXME: a section on non-scripted testore mya be a good idea

9.7 Using the Restore Script

FIXME: subject to change as things develop

(This message will disappear, once this node revised.)

(This node was `@`ignore'd—merely listing it for now.)

To restore files that were archived using a scripted backup, use the `restore` script. The syntax for the script is:

where ***** are the file systems to restore from, and ***** is a regular expression which specifies which files to restore. If you specify –all, the script restores all the files in the file system.

You should start the restore script with the media containing the first volume of the archive mounted. The script will prompt for other volumes as they are needed. If the archive is on tape, you don't need to rewind the tape to to its beginning—if the tape head is positioned past the beginning of the archive, the script will rewind the tape as needed.

FIXME: xref Media

, for a discussion of tape positioning.

If you specify '`--all`' as the *files* argument, the `restore` script extracts all the files in the archived file system into the active file system.

> **Warning:**The script will delete files from the active file system if they were not in the file system when the archive was made.

FIXME: xref incremental and listed-incremental

, for an explanation of how the script makes that determination.

FIXME: this may be an option, not a given

10 Date input formats

This section describes the textual date representations that GNU programs accept. These are the strings you, as a user, can supply as arguments to the various programs. The C interface (via the `getdate` function) is not described here.

Although the date syntax here can represent any possible time since zero A.D., computer integers are not big enough for such a (comparatively) long time. The earliest date semantically allowed on Unix systems is midnight, 1 January 1970 UCT.

10.1 General date syntax

A *date* is a string, possibly empty, containing many items separated by whitespace. The whitespace may be omitted when no ambiguity arises. The empty string means the beginning of today (i.e., midnight). Order of the items is immaterial. A date string may contain many flavors of items:

- calendar date items
- time of the day items
- time zone items
- day of the week items
- relative items
- pure numbers.

We describe each of these item types in turn, below.

A few numbers may be written out in words in most contexts. This is most useful for specifying day of the week items or relative items (see below). Here is the list: ‘`first`’ for 1, ‘`next`’ for 2, ‘`third`’ for 3, ‘`fourth`’ for 4, ‘`fifth`’ for 5, ‘`sixth`’ for 6, ‘`seventh`’ for 7, ‘`eighth`’ for 8, ‘`ninth`’ for 9, ‘`tenth`’ for 10, ‘`eleventh`’ for 11 and ‘`twelfth`’ for 12. Also, ‘`last`’ means exactly −1.

When a month is written this way, it is still considered to be written numerically, instead of being “spelled in full”; this changes the allowed strings.

Alphabetic case is completely ignored in dates. Comments may be introduced between round parentheses, as long as included parentheses are properly nested. Hyphens not followed by a digit are currently ignored. Leading zeros on numbers are ignored.

10.2 Calendar date item

A *calendar date item* specifies a day of the year. It is specified differently, depending on whether the month is specified numerically or literally. All these strings specify the same calendar date:

```
1970-9-17            # ISO 8601.
70-9-17              # This century assumed by default.
70-09-17             # Leading zeros are ignored.
9/17/72              # Common U.S. writing.
24 September 1972
24 Sept 72           # September has a special abbreviation.
24 Sep 72            # Three-letter abbreviations always allowed.
```

```
Sep 24, 1972
24-sep-72
24sep72
```

The year can also be omitted. In this case, the last specified year is used, or the current year if none. For example:

```
9/17
sep 17
```

Here are the rules.

For numeric months, the ISO 8601 format '`year-month-day`' is allowed, where *year* is any positive number, *month* is a number between 1 and 12, and *day* is a number between 1 and 31. If *year* is less than 100, then 1900 is added to it to force a date in this century. The construct '`month/day/year`', popular in the United States, is accepted. Also '`month/day`', omitting the year.

Literal months may be spelled out in full: '`January`', '`February`', '`March`', '`April`', '`May`', '`June`', '`July`', '`August`', '`September`', '`October`', '`November`' or '`December`'. Literal months may be abbreviated to their first three letters, possibly followed by an abbreviating dot. It is also permitted to write '`Sept`' instead of '`September`'.

When months are written literally, the calendar date may be given as any of the following:

```
day month year
day month
month day year
day-month-year
```

Or, omitting the year:

```
month day
```

10.3 Time of day item

A *time of day item* in date strings specifies the time on a given day. Here are some examples, all of which represent the same time:

```
20:02:0
20:02
8:02pm
20:02-0500      # In EST (Eastern U.S. Standard Time).
```

More generally, the time of the day may be given as '`hour:minute:second`', where *hour* is a number between 0 and 23, *minute* is a number between 0 and 59, and *second* is a number between 0 and 59. Alternatively, '`:second`' can be omitted, in which case it is taken to be zero.

If the time is followed by '`am`' or '`pm`' (or '`a.m.`' or '`p.m.`'), *hour* is restricted to run from 1 to 12, and '`:minute`' may be omitted (taken to be zero). '`am`' indicates the first half of the day, '`pm`' indicates the second half of the day. In this notation, 12 is the predecessor of 1: midnight is '`12am`' while noon is '`12pm`'.

The time may alternatively be followed by a timezone correction, expressed as '`shhmm`', where *s* is '`+`' or '`-`', *hh* is a number of zone hours and *mm* is a number of zone minutes. When a timezone correction is given this way, it forces interpretation of the time in UTC, overriding any previous specification for the timezone or the local timezone. The *minute* part of the time of the day may not be elided when a timezone correction is used. This is the only way to specify a timezone correction by fractional parts of an hour.

Either ‘`am`’/‘`pm`’ or a timezone correction may be specified, but not both.

10.4 Timezone item

A *timezone item* specifies an international timezone, indicated by a small set of letters. Any included period is ignored. Military timezone designations use a single letter. Currently, only integral zone hours may be represented in a timezone item. See the previous section for a finer control over the timezone correction.

Here are many non-daylight-savings-time timezones, indexed by the zone hour value.

+000	‘`GMT`’ for Greenwich Mean, ‘`UT`’ or ‘`UTC`’ for Universal (Coordinated), ‘`WET`’ for Western European and ‘`Z`’ for militaries.
+100	‘`WAT`’ for West Africa and ‘`A`’ for militaries.
+200	‘`AT`’ for Azores and ‘`B`’ for militaries.
+300	‘`C`’ for militaries.
+400	‘`AST`’ for Atlantic Standard and ‘`D`’ for militaries.
+500	‘`E`’ for militaries and ‘`EST`’ for Eastern Standard.
+600	‘`CST`’ for Central Standard and ‘`F`’ for militaries.
+700	‘`G`’ for militaries and ‘`MST`’ for Mountain Standard.
+800	‘`H`’ for militaries and ‘`PST`’ for Pacific Standard.
+900	‘`I`’ for militaries and ‘`YST`’ for Yukon Standard.
+1000	‘`AHST`’ for Alaska-Hawaii Standard, ‘`CAT`’ for Central Alaska, ‘`HST`’ for Hawaii Standard and ‘`K`’ for militaries.
+1100	‘`L`’ for militaries and ‘`NT`’ for Nome.
+1200	‘`IDLW`’ for International Date Line West and ‘`M`’ for militaries.
-100	‘`CET`’ for Central European, ‘`FWT`’ for French Winter, ‘`MET`’ for Middle European, ‘`MEWT`’ for Middle European Winter, ‘`N`’ for militaries and ‘`SWT`’ for Swedish Winter.
-200	‘`EET`’ for Eastern European, USSR Zone 1 and ‘`O`’ for militaries.
-300	‘`BT`’ for Baghdad, USSR Zone 2 and ‘`P`’ for militaries.
-400	‘`Q`’ for militaries and ‘`ZP4`’ for USSR Zone 3.
-500	‘`R`’ for militaries and ‘`ZP5`’ for USSR Zone 4.
-600	‘`S`’ for militaries and ‘`ZP6`’ for USSR Zone 5.
-700	‘`T`’ for militaries and ‘`WAST`’ for West Australian Standard.
-800	‘`CCT`’ for China Coast, USSR Zone 7 and ‘`U`’ for militaries.

-900	‘`JST`’ for Japan Standard, USSR Zone 8 and ‘`V`’ for militaries.
-1000	‘`EAST`’ for East Australian Standard, ‘`GST`’ for Guam Standard, USSR Zone 9 and ‘`W`’ for militaries.
-1100	‘`X`’ for militaries.
-1200	‘`IDLE`’ for International Date Line East, ‘`NZST`’ for New Zealand Standard, ‘`NZT`’ for New Zealand and ‘`Y`’ for militaries.

Here are many DST timezones, indexed by the zone hour value. Also, by following a non-DST timezone by the string ‘`DST`’ in a separate word (that is, separated by some whitespace), the corresponding DST timezone may be specified.

0	‘`BST`’ for British Summer.
+400	‘`ADT`’ for Atlantic Daylight.
+500	‘`EDT`’ for Eastern Daylight.
+600	‘`CDT`’ for Central Daylight.
+700	‘`MDT`’ for Mountain Daylight.
+800	‘`PDT`’ for Pacific Daylight.
+900	‘`YDT`’ for Yukon Daylight.
+1000	‘`HDT`’ for Hawaii Daylight.
-100	‘`MEST`’ for Middle European Summer, ‘`MESZ`’ for Middle European Summer, ‘`SST`’ for Swedish Summer and ‘`FST`’ for French Summer.
-700	‘`WADT`’ for West Australian Daylight.
-1000	‘`EADT`’ for Eastern Australian Daylight.
-1200	‘`NZDT`’ for New Zealand Daylight.

10.5 Day of week item

The explicit mention of a day of the week will forward the date (only if necessary) to reach that day of the week in the future.

Days of the week may be spelled out in full: ‘`Sunday`’, ‘`Monday`’, ‘`Tuesday`’, ‘`Wednesday`’, ‘`Thursday`’, ‘`Friday`’ or ‘`Saturday`’. Days may be abbreviated to their first three letters, optionally followed by a period. The special abbreviations ‘`Tues`’ for ‘`Tuesday`’, ‘`Wednes`’ for ‘`Wednesday`’ and ‘`Thur`’ or ‘`Thurs`’ for ‘`Thursday`’ are also allowed.

A number may precede a day of the week item to move forward supplementary weeks. It is best used in expression like ‘`third monday`’. In this context, ‘`last` *day*’ or ‘`next` *day*’ is also acceptable; they move one week before or after the day that *day* by itself would represent.

A comma following a day of the week item is ignored.

10.6 Relative item in date strings

Relative items adjust a date (or the current date if none) forward or backward. The effect of relative items accumulate. Here are some examples:

```
1 year
1 year ago
3 years
2 days
```

The unit of time displacement may be selected by the string '`year`' or '`month`' for moving by whole years or months. These are fuzzy units, as years and months are not all of equal duration. More precise units are '`fortnight`' which is worth 14 days, '`week`' worth 7 days, '`day`' worth 24 hours, '`hour`' worth 60 minutes, '`minute`' or '`min`' worth 60 seconds, and '`second`' or '`sec`' worth one second. An '`s`' suffix on these units is accepted and ignored.

The unit of time may be preceded by a multiplier, given as an optionally signed number. Unsigned numbers are taken as positively signed. No number at all implies 1 for a multiplier. Following a relative item by the string '`ago`' is equivalent to preceding the unit by a multiplicator with value -1.

The string '`tomorrow`' is worth one day in the future (equivalent to '`day`'), the string '`yesterday`' is worth one day in the past (equivalent to '`day ago`').

The strings '`now`' or '`today`' are relative items corresponding to zero-valued time displacement, these strings come from the fact a zero-valued time displacement represents the current time when not otherwise change by previous items. They may be used to stress other items, like in '`12:00 today`'. The string '`this`' also has the meaning of a zero-valued time displacement, but is preferred in date strings like '`this thursday`'.

When a relative item makes the resulting date to cross the boundary between DST and non-DST (or vice-versa), the hour is adjusted according to the local time.

10.7 Pure numbers in date strings

The precise intepretation of a pure decimal number is dependent of the context in the date string.

If the decimal number is of the form *yyyymmdd* and no other calendar date item (

FIXME: pxref Calendar date item

) appears before it in the date string, then *yyyy* is read as the year, *mm* as the month number and *dd* as the day of the month, for the specified calendar date.

If the decimal number is of the form *hhmm* and no other time of day item appears before it in the date string, then *hh* is read as the hour of the day and *mm* as the minute of the hour, for the specified time of the day. *mm* can also be omitted.

If both a calendar date and a time of day appear to the left of a number in the date string, but no relative item, then the number overrides the year.

10.8 Authors of `getdate`

`getdate` was originally implemented by Steven M. Bellovin ('`smb@research.att.com`') while at the University of North Carolina at Chapel Hill. The code was later tweaked by a couple of people on Usenet, then completely overhauled by Rich $alz ('`rsalz@bbn.com`') and

Jim Berets (‘`jberets@bbn.com`’) in August, 1990. Various revisions for the GNU system were made by David MacKenzie, Jim Meyering, and others.

This chapter was originally produced by Francois Pinard (‘`pinard@iro.umontreal.ca`’) from the ‘`getdate.y`’ source code, and then edited by K. Berry (‘`kb@cs.umb.edu`’).

11 Format of `tar` archives

(This message will disappear, once this node revised.)

11.1 The Standard Format

(This message will disappear, once this node revised.)

A `tar` archive file contains a series of records. Each record contains `RECORDSIZE` bytes. Although this format may be thought of as being on magnetic tape, other media are often used.

Each file archived is represented by a header record which describes the file, followed by zero or more records which give the contents of the file. At the end of the archive file there may be a record filled with binary zeros as an end-of-file marker. A reasonable system should write a record of zeros at the end, but must not assume that such a record exists when reading an archive.

The records may be *blocked* for physical I/O operations. Each block of *n* records (where *n* is set by the '`--block-size=512-size`' ('`-b 512-size`') option to `tar`) is written with a single '`write ()`' operation. On magnetic tapes, the result of such a write is a single tape record. When writing an archive, the last block of records should be written at the full size, with records after the zero record containing all zeroes. When reading an archive, a reasonable system should properly handle an archive whose last block is shorter than the rest, or which contains garbage records after a zero record.

The header record is defined in C as follows. In the GNU `tar` distribution, this is part of file '`src/tar.h`':

```
/* Standard Archive Format - Standard TAR - USTAR.  */

/* Header block on tape.

   We use traditional DP naming conventions here.  A "block" is a big chunk
   of stuff that we do I/O on.  A "record" is a piece of info that we care
   about.  Typically many "record"s fit into a "block".  */

#define RECORDSIZE      512
#define NAMSIZ          100
#define TUNMLEN         32
#define TGNMLEN         32
#define SPARSE_EXT_HDR  21
#define SPARSE_IN_HDR   4

struct sparse
  {
    char offset[12];
    char numbytes[12];
  };

union record
```

```
  {
    char charptr[RECORDSIZE];

    struct header
      {
        char arch_name[NAMSIZ];
        char mode[8];
        char uid[8];
        char gid[8];
        char size[12];
        char mtime[12];
        char chksum[8];
        char linkflag;
        char arch_linkname[NAMSIZ];
        char magic[8];
        char uname[TUNMLEN];
        char gname[TGNMLEN];
        char devmajor[8];
        char devminor[8];

        /* The following fields were added for GNU and are not standard.  */

        char atime[12];
        char ctime[12];
        char offset[12];
        char longnames[4];
        /* Some compilers would insert the pad themselves, so pad was
           once autoconfigured.  It is simpler to always insert it!  */
        char pad;
        struct sparse sp[SPARSE_IN_HDR];
        char isextended;
        char realsize[12];      /* true size of the sparse file */
#if 0
        char ending_blanks[12]; /* number of nulls at the end of the file,
                                   if any */
#endif
      }
    header;

    struct extended_header
      {
        struct sparse sp[21];
        char isextended;
      }
    ext_hdr;
  };
```

```
/* The checksum field is filled with this while the checksum is computed.  */
#define CHKBLANKS       "        "      /* 8 blanks, no null */

/* The magic field is filled with this value if uname and gname are valid,
   marking the archive as being in standard POSIX format (though GNU tar
   itself is not POSIX conforming).  */
#define TMAGIC "ustar  "        /* 7 chars and a null */

/* The magic field is filled with this if this is a GNU format dump entry.
   But I suspect this is not true anymore.  */
#define GNUMAGIC "GNUtar "      /* 7 chars and a null */

/* The linkflag defines the type of file.  */
#define LF_OLDNORMAL    '\0'    /* normal disk file, Unix compat */
#define LF_NORMAL       '0'     /* normal disk file */
#define LF_LINK         '1'     /* link to previously dumped file */
#define LF_SYMLINK      '2'     /* symbolic link */
#define LF_CHR          '3'     /* character special file */
#define LF_BLK          '4'     /* block special file */
#define LF_DIR          '5'     /* directory */
#define LF_FIFO         '6'     /* FIFO special file */
#define LF_CONTIG       '7'     /* contiguous file */
/* Further link types may be defined later.  */

/* Note that the standards committee allows only capital A through
   capital Z for user-defined expansion.  This means that defining
   something as, say '8' is a *bad* idea.  */

/* This is a dir entry that contains the names of files that were in the
   dir at the time the dump was made.  */
#define LF_DUMPDIR      'D'

/* Identifies the NEXT file on the tape as having a long linkname.  */
#define LF_LONGLINK     'K'

/* Identifies the NEXT file on the tape as having a long name.  */
#define LF_LONGNAME     'L'

/* This is the continuation of a file that began on another volume.  */
#define LF_MULTIVOL     'M'

/* For storing filenames that didn't fit in 100 characters.  */
#define LF_NAMES        'N'

/* This is for sparse files.  */
#define LF_SPARSE       'S'
```

```
/* This file is a tape/volume header.  Ignore it on extraction.  */
#define LF_VOLHDR       'V'

#if 0
/* The following two blocks of #define's are unused in GNU tar.  */

/* Bits used in the mode field - values in octal */
#define  TSUID    04000         /* set UID on execution */
#define  TSGID    02000         /* set GID on execution */
#define  TSVTX    01000         /* save text (sticky bit) */

/* File permissions */
#define  TUREAD   00400         /* read by owner */
#define  TUWRITE  00200         /* write by owner */
#define  TUEXEC   00100         /* execute/search by owner */
#define  TGREAD   00040         /* read by group */
#define  TGWRITE  00020         /* write by group */
#define  TGEXEC   00010         /* execute/search by group */
#define  TOREAD   00004         /* read by other */
#define  TOWRITE  00002         /* write by other */
#define  TOEXEC   00001         /* execute/search by other */

#endif

/* End of Standard Archive Format description.  */
```

All characters in header records are represented by using 8-bit characters in the local variant of ASCII. Each field within the structure is contiguous; that is, there is no padding used within the structure. Each character on the archive medium is stored contiguously.

Bytes representing the contents of files (after the header record of each file) are not translated in any way and are not constrained to represent characters in any character set. The `tar` format does not distinguish text files from binary files, and no translation of file contents is performed.

The `name`, `linkname`, `magic`, `uname`, and `gname` are null-terminated character strings. All other fileds are zero-filled octal numbers in ASCII. Each numeric field of width *w* contains *w* minus 2 digits, a space, and a null, except `size`, and `mtime`, which do not contain the trailing null.

The `name` field is the file name of the file, with directory names (if any) preceding the file name, separated by slashes.

FIXME: how big a name before field overflows?

The `mode` field provides nine bits specifying file permissions and three bits to specify the Set UID, Set GID, and Save Text (*sticky*) modes. Values for these bits are defined above. When special permissions are required to create a file with a given mode, and the user restoring files from the archive does not hold such permissions, the mode bit(s) specifying those special permissions are ignored. Modes which are not supported by the operating system restoring files from the archive will be ignored. Unsupported modes should be faked up when creating or updating an archive; e.g. the group permission could be copied from the *other* permission.

The `uid` and `gid` fields are the numeric user and group ID of the file owners, respectively. If the operating system does not support numeric user or group IDs, these fields should be ignored.

The `size` field is the size of the file in bytes; linked files are archived with this field specified as zero.

FIXME: xref Modifiers

, in particular the '`--incremental`' ('`-G`') option.

The `mtime` field is the modification time of the file at the time it was archived. It is the ASCII representation of the octal value of the last time the file was modified, represented as an integer number of seconds since January 1, 1970, 00:00 Coordinated Universal Time.

The `chksum` field is the ASCII representation of the octal value of the simple sum of all bytes in the header record. Each 8-bit byte in the header is added to an unsigned integer, initialized to zero, the precision of which shall be no less than seventeen bits. When calculating the checksum, the `chksum` field is treated as if it were all blanks.

The `typeflag` field specifies the type of file archived. If a particular implementation does not recognize or permit the specified type, the file will be extracted as if it were a regular file. As this action occurs, `tar` issues a warning to the standard error.

The `atime` and `ctime` fields are used in making incremental backups; they store, respectively, the particular file's access time and last inode-change time.

The `offset` is used by the '`--multi-volume`' ('`-M`') option, when making a multi-volume archive. The offset is number of bytes into the file that we need to restart at to continue the file on the next tape, i.e., where we store the location that a continued file is continued at.

The following fields were added to deal with sparse files. A file is *sparse* if it takes in unallocated blocks which end up being represented as zeros, i.e., no useful data. A test to see if a file is sparse is to look at the number blocks allocated for it versus the number of characters in the file; if there are fewer blocks allocated for the file than would normally be allocated for a file of that size, then the file is sparse. This is the method `tar` uses to detect a sparse file, and once such a file is detected, it is treated differently from non-sparse files.

Sparse files are often `dbm` files, or other database-type files which have data at some points and emptiness in the greater part of the file. Such files can appear to be very large when an '`ls -l`' is done on them, when in truth, there may be a very small amount of important data contained in the file. It is thus undesirable to have `tar` think that it must back up this entire file, as great quantities of room are wasted on empty blocks, which can lead to running out of room on a tape far earlier than is necessary. Thus, sparse files are dealt with so that these empty blocks are not written to the tape. Instead, what is written to the tape is a description, of sorts, of the sparse file: where the holes are, how big the holes are, and how much data is found at the end of the hole. This way, the file takes up potentially far less room on the tape, and when the file is extracted later on, it will look exactly the way it looked beforehand. The following is a description of the fields used to handle a sparse file:

The `sp` is an array of `struct sparse`. Each `struct sparse` contains two 12-character strings which represent an offset into the file and a number of bytes to be written at that offset. The offset is absolute, and not relative to the offset in preceding array element.

The header can hold four of these `struct sparse` at the moment; if more are needed, they are not stored in the header.

The `isextended` flag is set when an `extended_header` is needed to deal with a file. Note that this means that this flag can only be set when dealing with a sparse file, and it is only set in the event that the description of the file will not fit in the alloted room for sparse structures in the header. In other words, an extended_header is needed.

The `extended_header` structure is used for sparse files which need more sparse structures than can fit in the header. The header can fit 4 such structures; if more are needed, the flag `isextended` gets set and the next record is an `extended_header`.

Each `extended_header` structure contains an array of 21 sparse structures, along with a similar `isextended` flag that the header had. There can be an indeterminate number of such `extended_header`s to describe a sparse file.

`LF_NORMAL`
`LF_OLDNORMAL`

These flags represent a regular file. In order to be compatible with older versions of `tar`, a `typeflag` value of `LF_OLDNORMAL` should be silently recognized as a regular file. New archives should be created using `LF_NORMAL`. Also, for backward compatibility, `tar` treats a regular file whose name ends with a slash as a directory.

`LF_LINK`

This flag represents a file linked to another file, of any type, previously archived. Such files are identified in Unix by each file having the same device and inode number. The linked-to name is specified in the `linkname` field with a trailing null.

`LF_SYMLINK`

This represents a symbolic link to another file. The linked-to name is specified in the `linkname` field with a trailing null.

`LF_CHR`
`LF_BLK`

These represent character special files and block special files respectively. In this case the `devmajor` and `devminor` fields will contain the major and minor device numbers respectively. Operating systems may map the device specifications to their own local specification, or may ignore the entry.

`LF_DIR`

This flag specifies a directory or sub-directory. The directory name in the `name` field should end with a slash. On systems where disk allocation is performed on a directory basis, the `size` field will contain the maximum number of bytes (which may be rounded to the nearest disk block allocation unit) which the directory may hold. A `size` field of zero indicates no such limiting. Systems which do not support limiting in this manner should ignore the `size` field.

`LF_FIFO`

This specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence of this file and not its contents.

`LF_CONTIG`
: This specifies a contiguous file, which is the same as a normal file except that, in operating systems which support it, all its space is allocated contiguously on the disk. Operating systems which do not allow contiguous allocation should silently treat this type as a normal file.

`A ... Z` These are reserved for custom implementations. Some of these are used in the GNU modified format, as described below.

Other values are reserved for specification in future revisions of the P1003 standard, and should not be used by any `tar` program.

The `magic` field indicates that this archive was output in the P1003 archive format. If this field contains `TMAGIC`, the `uname` and `gname` fields will contain the ASCII representation of the owner and group of the file respectively. If found, the user and group IDs are used rather than the values in the `uid` and `gid` fields.

11.2 GNU Extensions to the Archive Format

(This message will disappear, once this node revised.)

The GNU format uses additional file types to describe new types of files in an archive. These are listed below.

`LF_DUMPDIR`
`'D'` This represents a directory and a list of files created by the '`--incremental`' ('`-G`') option. The `size` field gives the total size of the associated list of files. Each file name is preceded by either a '`Y`' (the file should be in this archive) or an '`N`'. (The file is a directory, or is not stored in the archive.) Each file name is terminated by a null. There is an additional null after the last file name.

`LF_MULTIVOL`
`'M'` This represents a file continued from another volume of a multi-volume archive created with the '`--multi-volume`' ('`-M`') option. The original type of the file is not given here. The `size` field gives the maximum size of this piece of the file (assuming the volume does not end before the file is written out). The `offset` field gives the offset from the beginning of the file where this part of the file begins. Thus `size` plus `offset` should equal the original size of the file.

`LF_SPARSE`
`'S'` This flag indicates that we are dealing with a sparse file. Note that archiving a sparse file requires special operations to find holes in the file, which mark the positions of these holes, along with the number of bytes of data to be found after the hole.

`LF_VOLHDR`
`'V'` This file type is used to mark the volume header that was given with the '`--label=`*`archive-label`*' ('`-V` *`archive-label`*') option when the archive was created. The `name` field contains the `name` given after the '`--label=`*`archive-label`*' ('`-V` *`archive-label`*') option. The `size` field is zero. Only the first file in each volume of an archive should have this type.

You may have trouble reading a GNU format archive on a non-GNU system if the options '`--incremental`' ('`-G`'), '`--multi-volume`' ('`-M`'), '`--sparse`' ('`-S`'), or '`--label=`*`archive-label`*' ('`-V` *`archive-label`*') were used when writing the archive. In general, if `tar` does not use the GNU-added fields of the header, other versions of `tar` should be able to read the archive. Otherwise, the `tar` program will give an error, the most likely one being a checksum error.

11.3 Comparison of `tar` and `cpio`

(This message will disappear, once this node revised.)

Here is a summary of differences between `tar` and `cpio`. The accuracy of the following information has not been verified. The following people contributed to this section, mainly through a survey conducted in 1991. The remainder of this section does not otherwise try to relate topics to people.

Bent Bertelsen	dmdata@login.dkuug.dk
David Hoopes	talgras!david
Guy Harris	guy@auspex.com
Kai Petzke	wpp@marie.physik.tu-berlin.de
Kristen Nielsen	dmdata@login.dkuug.dk
Leslie Mikesell	les@chinet.chi.il.us

FIXME: Reorganize the following material

`tar` handles symbolic links in the form in which it comes in BSD; `cpio` doesn't handle symbolic links in the form in which it comes in System V prior to SVR4, and some vendors may have added symlinks to their system without enhancing `cpio` to know about them. Others may have enhanced it in a way other than the way I did it at Sun, and which was adopted by AT&T (and which is, I think, also present in the `cpio` that Berkeley picked up from AT&T and put into a later BSD release—I think I gave them my changes).

(SVR4 does some funny stuff with `tar`; basically, its `cpio` can handle `tar` format input, and write it on output, and it probably handles symbolic links. They may not have bothered doing anything to enhance `tar` as a result.)

`cpio` handles special files; traditional `tar` doesn't.

`tar` comes with V7, System III, System V, and BSD source; `cpio` comes only with System III, System V, and later BSD (4.3-tahoe and later).

`tar`'s way of handling multiple hard links to a file can handle file systems that support 32-bit inumbers (e.g., the BSD file system); `cpio`s way requires you to play some games (in its `"binary"` format, i-numbers are only 16 bits, and in its `"portable ASCII"` format, they're 18 bits—it would have to play games with the `"file system ID"` field of the header to make sure that the file system ID/i-number pairs of different files were always different), and I don't know which `cpio`s, if any, play those games. Those that don't might get confused and think two files are the same file when they're not, and make hard links between them.

`tar`s way of handling multiple hard links to a file places only one copy of the link on the tape, but the name attached to that copy is the *only* one you can use to retrieve the file; `cpio`s way puts one copy for every link, but you can retrieve it using any of the names.

`>`What type of check sum (if any) is used, and how is this calculated.

See the attached manual pages for `tar` and `cpio` format. `tar` uses a checksum which is the sum of all the bytes in the `tar` header for a file; `cpio` uses no checksum.

> `>`If anyone knows why `cpio` was made when `tar` was prasent `>`at the unix scene,

It wasn't. `cpio` first showed up in PWB/UNIX 1.0; no generally-available version of UNIX had `tar` at the time. I don't know whether any version that was generally available *within AT&T* had `tar`, or, if so, whether the people within AT&T who did `cpio` knew about it.

On restore, if there is a corruption on a tape `tar` will stop at that point, while `cpio` will skip over it and try to restore the rest of the files.

The main difference is just in the command syntax and header format.

`tar` is a little more tape-oriented in that everything is blocked to start on a block boundary.

> `>`Is there any differences between the ability to recover crashed `>`archives between the two of them. (Is there any chance of recovering `>`crashed archives at all.)

Theoretically it should be easier under `tar` since the blocking lets you find a header with some variation of '`dd skip=`*`nn`*'. However, modern `cpio`'s and variations have an option to just search for the next file header after an error with a reasonable chance of re-syncing. However, lots of tape driver software won't allow you to continue past a media error which should be the only reason for getting out of sync unless a file changed sizes while you were writing the archive.

> `>`If anyone knows why `cpio` was made when `tar` was prasent `>`at the unix scene, please tell me about this too.

Probably because it is more media efficient (by not blocking everything and using only the space needed for the headers where `tar` always uses 512 bytes per file header) and it knows how to archive special files.

You might want to look at the freely available alternatives. The major ones are `afio`, GNU `tar`, and `pax`, each of which have their own extensions with some backwards compatibility.

Sparse files were `tar`red as sparse files (which you can easily test, because the resulting archive gets smaller, and GNU `cpio` can no longer read it).

Index

Short Contents

Table of Contents