GNU `tar`

GNU tar: an archiver tool

FTP release, version 1.13, 16 June 1999

Melissa Weisshaus, Jay Fenlason,
Thomas Bushnell, n/BSG, Amy Gorin

1 Introduction

Welcome to the GNU `tar` manual. GNU `tar` is used to create and manipulate files (*archives*) which are actually collections of many other files; the program provides users with an organized and systematic method for controlling a large amount of data.

The first part of this chapter introduces you to various terms that will recur throughout the book. It also tells you who has worked on GNU `tar` and its documentation, and where you should send bug reports or comments.

The second chapter is a tutorial (see Chapter 2 [Tutorial], page 5) which provides a gentle introduction for people who are new to using `tar`. It is meant to be self contained, not requiring any reading from subsequent chapters to make sense. It moves from topic to topic in a logical, progressive order, building on information already explained.

Although the tutorial is paced and structured to allow beginners to learn how to use `tar`, it is not intended solely for beginners. The tutorial explains how to use the three most frequently used operations ('`create`', '`list`', and '`extract`') as well as two frequently used options ('`file`' and '`verbose`'). The other chapters do not refer to the tutorial frequently; however, if a section discusses something which is a complex variant of a basic concept, there may be a cross reference to that basic concept. (The entire book, including the tutorial, assumes that the reader understands some basic concepts of using a Unix-type operating system; see Chapter 2 [Tutorial], page 5.)

The third chapter presents the remaining five operations, and information about using `tar` options and option syntax.

The other chapters are meant to be used as a reference. Each chapter presents everything that needs to be said about a specific topic.

One of the chapters (see Chapter 7 [Date input formats], page 71) exists in its entirety in other GNU manuals, and is mostly self-contained. In addition, one section of this manual (see Section 8.4 [Standard], page 87) contains a big quote which is taken directly from `tar` sources.

In general, we give both the long and short (abbreviated) option names at least once in each section where the relevant option is covered, so that novice readers will become familiar with both styles. (A few options have no short versions, and the relevant sections will indicate this.)

1.1 Some Definitions

The `tar` program is used to create and manipulate `tar` archives. An *archive* is a single file which contains the contents of many files, while still identifying the names of the files, their owner(s), and so forth. (In addition, 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.) You can use `tar` to *create* a new archive in a specified directory.

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 *member* 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.

The term *extraction* refers to the process of copying an archive member (or multiple members) into a file in the filesystem. Extracting all the members of an archive is often called *extracting the archive*. The term *unpack* can also be used to refer to the extraction of many or all the members of an archive. Extracting an archive does not destroy the archive's structure, just as creating an archive does not destroy the copies of the files that exist outside of the archive. You may also *list* the members in a given archive (this is often thought of as "printing" them to the standard output, or the command line), or *append* members to a pre-existing archive. All of these operations can be peformed using `tar`.

1.2 What `tar` Does

The `tar` program provides the ability to create `tar` archives, as well as 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 which were already stored.

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 available devices, files, or other programs (using pipes). `tar` may even access remote devices or files (as archives).

You can use `tar` archives in many ways. We want to stress a few of them: storage, backup, and 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. However, the tape has no names for these files; it only knows their relative position on the tape. One way to store several files on one tape and retain their names is by creating a `tar` archive. 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 useful.

  Archive files are also used for long-term storage. You can think of this as transportation from the present into the future. (It is a science-fiction idiom that you can move through time as well as in space; the idea here is that `tar` can be used to move archives in all dimensions, even time!)

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. A backup puts a collection of files (possibly pertaining to many users and projects) together on a disk or a tape. This guards against accidental destruction of the information in those files. GNU `tar` has special features that allow it to be used to make incremental and full dumps of all the files in a filesystem.

Transportation
: You can create an archive on one system, transfer it to another system, and extract the contents there. This allows you to transport a group of files from one system to another.

1.3 How `tar` Archives are Named

Conventionally, `tar` archives are given names ending with '`.tar`'. This is not necessary for `tar` to operate properly, but this manual follows that convention in order to accustom readers to it and to make examples more clear.

Often, people refer to `tar` archives as "`tar` files," and archive members as "files" or "entries". For people familiar with the operation of `tar`, this causes no difficulty. However, in this manual, we consistently refer to "archives" and "archive members" to make learning to use `tar` easier for novice users.

1.4 POSIX Compliance

We make some of our recommendations throughout this book for one reason in addition to what we think of as "good sense". The main additional reason for a recommendation is to be compliant with the POSIX standards. If you set the shell environment variable `POSIXLY_CORRECT`, GNU `tar` will force you to adhere to these standards. Therefore, if this variable is set and you violate one of the POSIX standards in the way you phrase a command, for example, GNU `tar` will not allow the command and will signal an error message. You would then have to reorder the options or rephrase the command to comply with the POSIX standards.

There is a chance in the future that, if you set this environment variable, your archives will be forced to comply with POSIX standards, also. No GNU `tar` extensions will be allowed.

1.5 GNU `tar` Authors

GNU `tar` was originally written by John Gilmore, and modified by many people. The GNU enhancements were written by Jay Fenlason, then Joy Kendall, and the whole package has been further maintained by Thomas Bushnell, n/BSG, and finally François Pinard, with the help of numerous and kind users.

We wish to stress that `tar` is a collective work, and owes much to all those people who reported problems, offered solutions and other insights, or shared their thoughts and suggestions. An impressive, yet partial list of those contributors can be found in the '`THANKS`' file from the GNU `tar` distribution.

Jay Fenlason put together a draft of a GNU `tar` manual, 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. Thomas Bushnell, n/BSG and Amy Gorin worked on a tutorial and manual for GNU `tar`. François Pinard put version 1.11.8 of the manual together by taking information from all these sources and merging them. Melissa Weisshaus finally edited and redesigned the book to create version 1.12.

For version 1.12, Daniel Hagerty contributed a great deal of technical consulting. In particular, he is the primary author of Chapter 5 [Backups], page 51.

1.6 Reporting bugs or suggestions

If you find problems or have suggestions about this program or manual, please report them to 'tar-bugs@gnu.org'.

2 Tutorial Introduction to `tar`

This chapter guides you through some basic examples of three `tar` operations: '`--create`', '`--list`', and '`--extract`'. If you already know how to use some other version of `tar`, then you may not need to read this chapter. This chapter omits most complicated details about how `tar` works.

This chapter is paced to allow beginners to learn about `tar` slowly. At the same time, we will try to cover all the basic aspects of these three operations. In order to accomplish both of these tasks, we have made certain assumptions about your knowledge before reading this manual, and the hardware you will be using:

- Before you start to work through this tutorial, you should understand what the terms "archive" and "archive member" mean (see Section 1.1 [Definitions], page 1). In addition, you should understand something about how Unix-type operating systems work, and you should know how to use some basic utilities. For example, you should know how to create, list, copy, rename, edit, and delete files and directories; how to change between directories; and how to figure out where you are in the filesystem. You should have some basic understanding of directory structure and how files are named according to which directory they are in. You should understand concepts such as standard output and standard input, what various definitions of the term "argument" mean, the differences between relative and absolute path names, and .
- This manual assumes that you are working from your own home directory (unless we state otherwise). In this tutorial, you will create a directory to practice `tar` commands in. When we show path names, we will assume that those paths are relative to your home directory. For example, my home directory path is '`/home/fsf/melissa`'. All of my examples are in a subdirectory of the directory named by that path name; the subdirectory is called '`practice`'.
- In general, we show examples of archives which exist on (or can be written to, or worked with from) a directory on a hard disk. In most cases, you could write those archives to, or work with them on any other device, such as a tape drive. However, some of the later examples in the tutorial and next chapter will not work on tape drives. Additionally, working with tapes is much more complicated than working with hard disks. For these reasons, the tutorial does not cover working with tape drives. See Chapter 9 [Media], page 99, for complete information on using `tar` archives with tape drives.

In the examples, '`$`' represents a typical shell prompt. It precedes lines you should type; to make this more clear, those lines are shown in ***`this font`***, as opposed to lines which represent the computer's response; those lines are shown in `this font`, or sometimes '`like this`'. When we have lines which are too long to be displayed in any other way, we will show them like this:

```
This is an example of a line which would otherwise not fit in this space.
```

2.1 Basic `tar` Operations and Options

`tar` can take a wide variety of arguments which specify and define the actions it will have on the particular set of files or the archive. The main types of arguments to `tar` fall into one of two classes: operations, and options.

Some arguments fall into a class called *operations*; exactly one of these is both allowed and required for any instance of using `tar`; you may *not* specify more than one. People sometimes speak of *operating modes*. You are in a particular operating mode when you have specified the operation which specifies it; there are eight operations in total, and thus there are eight operating modes.

The other arguments fall into the class known as *options*. You are not required to specify any options, and you are allowed to specify more than one at a time (depending on the way you are using `tar` at that time). Some options are used so frequently, and are so useful for helping you type commands more carefully that they are effectively "required". We will discuss them in this chapter.

You can write most of the `tar` operations and options in any of three forms: long (mnemonic) form, short form, and old style. Some of the operations and options have no short or "old" forms; however, the operations and options which we will cover in this tutorial have corresponding abbreviations. We will indicate those abbreviations appropriately to get you used to seeing them. (Note that the "old style" option forms exist in GNU `tar` for compatibility with Unix `tar`. We present a full discussion of this way of writing options and operations appears in Section 3.3.3 [Old Options], page 20, and we discuss the other two styles of writing options in Section 3.3.1 [Mnemonic Options], page 19 and Section 3.3.2 [Short Options], page 20.)

In the examples and in the text of this tutorial, we usually use the long forms of operations and options; but the "short" forms produce the same result and can make typing long `tar` commands easier. For example, instead of typing

```
tar --create --verbose --file=afiles.tar apple angst aspic
```

you can type

```
tar -c -v -f afiles.tar apple angst aspic
```

or even

```
tar -cvf afiles.tar apple angst aspic
```

For more information on option syntax, see Section 4.2 [Advanced tar], page 38. In discussions in the text, when we name an option by its long form, we also give the corresponding short option in parentheses.

The term, "option", can be confusing at times, since "operations" are often lumped in with the actual, *optional* "options" in certain general class statements. For example, we just talked about "short and long forms of options and operations". However, experienced `tar` users often refer to these by shorthand terms such as, "short and long options". This term assumes that the "operations" are included, also. Context will help you determine which definition of "options" to use.

Similarly, the term "command" can be confusing, as it is often used in two different ways. People sometimes refer to `tar` "commands". A `tar` *command* is the entire command line of user input which tells `tar` what to do — including the operation, options, and any arguments (file names, pipes, other commands, etc). However, you will also sometimes hear the term "the `tar` command". When the word "command" is used specifically like this, a person is usually referring to the `tar` *operation*, not the whole line. Again, use context to figure out which of the meanings the speaker intends.

2.2 The Three Most Frequently Used Operations

Here are the three most frequently used operations (both short and long forms), as well as a brief description of their meanings. The rest of this chapter will cover how to use these operations in detail. We will present the rest of the operations in the next chapter.

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

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

`--extract`
`-x` Extract one or more members from an archive.

2.3 Two Frequently Used Options

To understand how to run `tar` in the three operating modes listed previously, you also need to understand how to use two of the options to `tar`: '`--file`' (which takes an archive file as an argument) and '`--verbose`'. (You are usually not *required* to specify either of these options when you run `tar`, but they can be very useful in making things more clear and helping you avoid errors.)

The '`--file`' Option

`--file`=archive-name
`-f` archive-name
Specify the name of an archive file.

You can specify an argument for the *`--file`=archive-name* (*`-f` archive-name*) option whenever you use `tar`; this option determines the name of the archive file that `tar` will work on.

If you don't specify this argument, then `tar` will use a default, usually 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. The 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
```

To avoid confusion, we recommend that you always specfiy an archive file name by using *`--file`=archive-name* (*`-f` archive-name*) when writing your `tar` commands. For more information on using the *`--file`=archive-name* (*`-f` archive-name*) option, see Section 6.1 [file], page 61.

The '`--verbose`' Option

`--verbose`
`-v` Show the files being worked on as `tar` is running.

`--verbose` (`-v`) shows details about the results of running `tar`. This can be especially useful when the results might not be obvious. For example, if you want to see the progress of `tar` as it writes files into the archive, you can use the '`--verbose`' option. In the beginning, you may find it useful to use '`--verbose`' at all times; when you are more accustomed to `tar`, you will likely want to use it at certain times but not at others. We will use '`--verbose`' at times to help make something clear, and we will give many examples both using and not using '`--verbose`' to show the differences.

Sometimes, a single instance of '`--verbose`' on the command line will show a full, '`ls`' style listing of an archive or files, giving sizes, owners, and similar information. Other times, '`--verbose`' will only show files or members that the particular operation is operating on at the time. In the latter case, you can use '`--verbose`' twice in a command to get a listing such as that in the former case. For example, instead of saying

```
tar -cvf afiles.tar apple angst aspic
```

above, you might say

```
tar -cvvf afiles.tar apple angst aspic
```

This works equally well using short or long forms of options. Using long forms, you would simply write out the mnemonic form of the option twice, like this:

```
$ tar --create --verbose --verbose ...
```

Note that you must double the hyphens properly each time.

Later in the tutorial, we will give examples using '`--verbose --verbose`'.

Getting Help: Using the `--help` Option

`--help`

The '`--help`' option to `tar` prints out a very brief list of all operations and option available for the current version of `tar` available on your system.

2.4 How to Create Archives

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

One of the basic operations of `tar` is `--create` (`-c`), which you use to create a `tar` archive. We will explain '`--create`' first because, in order to learn about the other operations, you will find it useful to have an archive available to practice on.

To make this easier, in this section you will first create a directory containing three files. Then, we will show you how to create an *archive* (inside the new directory). Both the directory, and the archive are specifically for you to practice on. The rest of this chapter and the next chapter will show many examples using this directory and the files you will create: some of those files may be other directories and other archives.

The three files you will archive in this example are called '`blues`', '`folk`', and '`jazz`'. The archive is called '`collection.tar`'.

This section will proceed slowly, detailing how to use '`--create`' in `verbose` mode, and showing examples using both short and long forms. In the rest of the tutorial, and in the examples in the next chapter, we will proceed at a slightly quicker pace. This section moves more slowly to allow beginning users to understand how `tar` works.

2.4.1 Preparing a Practice Directory for Examples

To follow along with this and future examples, create a new directory called 'practice' containing files called 'blues', 'folk' and 'jazz'. The files can contain any information you like: ideally, they should contain information which relates to their names, and be of different lengths. Our examples assume that 'practice' is a subdirectory of your home directory.

Now cd to the directory named 'practice'; 'practice' is now your *working directory*. (*Please note*: Although the full path name of this directory is '/*homedir*/practice', in our examples we will refer to this directory as 'practice'; the *homedir* is presumed.

In general, you should check that the files to be archived exist where you think they do (in the working directory) by running ls. Because you just created the directory and the files and have changed to that directory, you probably don't need to do that this time.

It is very important to make sure there isn't already a file in the working directory with the archive name you intend to use (in this case, 'collection.tar'), or that you don't care about its contents. Whenever you use 'create', tar will erase the current contents of the file named by *--file=archive-name* (*-f archive-name*) if it exists. tar will not tell you if you are about to overwrite a file unless you specify an option which does this . To add files to an existing archive, you need to use a different option, such as *--append* (*-r*); see Section 4.2.2 [append], page 39 for information on how to do this.

2.4.2 Creating the Archive

To place the files 'blues', 'folk', and 'jazz' into an archive named 'collection.tar', use the following command:

```
$ tar --create --file=collection.tar blues folk jazz
```

The order of the arguments is not very important, *when using long option forms*. You could also say:

```
$ tar blues --create folk --file=collection.tar jazz
```

However, you can see that this order is harder to understand; this is why we will list the arguments in the order that makes the commands easiest to understand (and we encourage you to do the same when you use tar, to avoid errors).

Note that the part of the command which says, *--file=collection.tar* is considered to be *one* argument. If you substituted any other string of characters for *'collection.tar'*, then that string would become the name of the archive file you create.

The order of the options becomes more important when you begin to use short forms. With short forms, if you type commands in the wrong order (even if you type them correctly in all other ways), you may end up with results you don't expect. For this reason, it is a good idea to get into the habit of typing options in the order that makes inherent sense. See Section 2.4.4 [short create], page 10, for more information on this.

In this example, you type the command as shown above: '--create' is the operation which creates the new archive ('collection.tar'), and '--file' is the option which lets you give it the name you chose. The files, 'blues', 'folk', and 'jazz', are now members of the archive, 'collection.tar' (they are *file name arguments* to the '--create' operation) . Now that they are are in the archive, they are called *archive members*, not files .

When you create an archive, you *must* specify which files you want placed in the archive. If you do not specify any archive members, GNU `tar` will complain.

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:

```
blues   folk   jazz   collection.tar
```

Creating the archive '`collection.tar`' did not destroy the copies of the files in the directory.

Keep in mind that if you don't indicate an operation, `tar` will not run and will prompt you for one. If you don't name any files, `tar` will complain. You must have write access to the working directory, or else you will not be able to create an archive in that directory.

Caution: Do not attempt to use *--create* (*-c*) to add files to an existing archive; it will delete the archive and write a new one. Use *--append* (*-r*) instead. See Section 4.2.2 [append], page 39.

2.4.3 Running '`--create`' with '`--verbose`'

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 `create` example above would appear as:

```
$ tar --create --verbose --file=collection.tar blues folk jazz
blues
folk
jazz
```

This example is just like the example we showed which did not use '`--verbose`', except that `tar` generated the remaining lines (note the different font styles).

In the rest of the examples in this chapter, we will frequently use `verbose` mode so we can show actions or `tar` responses that you would otherwise not see, and which are important for you to understand.

2.4.4 Short Forms with '`create`'

As we said before, the *--create* (*-c*) operation is one of the most basic uses of `tar`, and you will use it countless times. Eventually, you will probably want to use abbreviated (or "short") forms of options. A full discussion of the three different forms that options can take appears in Section 3.3 [Styles], page 19; for now, here is what the previous example (including the *--verbose* (*-v*) option) looks like using short option forms:

```
$ tar -cvf collection.tar blues folk jazz
blues
folk
jazz
```

As you can see, the system responds the same no matter whether you use long or short option forms.

One difference between using short and long option forms is that, although the exact placement of arguments following options is no more specific when using short forms, it is easier to become confused and make a mistake when using short forms. For example, suppose you attempted the above example in the following way:

```
$ tar -cfv collection.tar blues folk jazz
```

In this case, `tar` will make an archive file called '`v`', containing the files '`blues`', '`folk`', and '`jazz`', because the '`v`' is the closest "file name" to the '`-f`' option, and is thus taken to be the chosen archive file name. `tar` will try to add a file called '`collection.tar`' to the '`v`' archive file; if the file '`collection.tar`' did not already exist, `tar` will report an error indicating that this file does not exist. If the file '`collection.tar`' does already exist (e.g., from a previous command you may have run), then `tar` will add this file to the archive. Because the '`-v`' option did not get registered, `tar` will not run under '`verbose`' mode, and will not report its progress.

The end result is that you may be quite confused about what happened, and possibly overwrite a file. To illustrate this further, we will show you how an example we showed previously would look using short forms.

This example,

```
$ tar blues --create folk --file=collection.tar jazz
```

is confusing as it is. When shown using short forms, however, it becomes much more so:

```
$ tar blues -c folk -f collection.tar jazz
```

It would be very easy to put the wrong string of characters immediately following the '`-f`', but doing that could sacrifice valuable data.

For this reason, we recommend that you pay very careful attention to the order of options and placement of file and archive names, especially when using short option forms. Not having the option name written out mnemonically can affect how well you remember which option does what, and therefore where different names have to be placed. (Placing options in an unusual order can also cause `tar` to report an error if you have set the shell environment variable, `POSIXLY_CORRECT`; see Section 1.4 [posix compliance], page 3 for more information on this.)

2.4.5 Archiving Directories

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 followed the previous instructions in this tutorial, you should type:

```
$ cd ..
$
```

This will put you into the directory which contains '`practice`', i.e. your home directory. Once in the superior directory, you can specify the subdirectory, '`practice`', as a file name argument. To store '`practice`' in the new archive file '`music.tar`', type:

```
$ tar --create --verbose --file=music.tar practice
```

`tar` should output:

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

```
practice/collection.tar
```

Note that the archive thus created is not in the subdirectory 'practice', but rather in the current 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`. For example, you will probably not be able to store your home directory in an archive by invoking `tar` from the root directory; See Section 6.8.2 [absolute], page 68. (Note also that 'collection.tar', 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.tar' is extracted, the archive file 'collection.tar' will be re-written into the file system).

If you give `tar` a command such as

```
$ tar --create --file=foo.tar .
```

`tar` will report '`tar: foo.tar is the archive; not dumped`'. This happens because `tar` creates the archive 'foo.tar' in the current directory 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 normally, except for the exclusion of that one file. (*Please note:* Other versions of `tar` are not so clever; they will enter an infinite loop when this happens, so you should not depend on this behavior unless you are certain you are running GNU `tar`.)

2.5 How to List Archives

Frequently, you will find yourself wanting to determine exactly what a particular archive contains. You can use the `--list` (`-t`) operation to get the member names as they currently appear in the archive, as well as various attributes of the files at the time they were archived. For example, you can examine the archive 'collection.tar' that you created in the last section with the command,

```
$ tar --list --file=collection.tar
```

The output of `tar` would then be:

```
blues
folk
jazz
```

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

```
./birds
baboon
./box
```

Be sure to use a `--file=`*archive-name* (`-f` *archive-name*) option just as with `--create` (`-c`) to specify the name of the archive.

If you use the `--verbose` (`-v`) option with '`--list`', then `tar` will print out a listing reminiscent of '`ls -l`', showing owner, file size, and so forth.

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

```
$ tar --list --verbose --file=collection.tar folk
-rw-rw-rw- myself user 62 1990-05-23 10:55 folk
```

You can specify one or more individual member names as arguments when using '`list`'. 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`'.

Because `tar` preserves paths, file names must be specified as they appear in the archive (ie., relative to the directory from which the archive was created). Therefore, it is essential when specifying member names to `tar` that you give the exact member names. For example, *`tar --list --file=bfiles birds`* would produce an error message something like '`tar: birds: Not found in archive`', because there is no member named '`birds`', only one named '`./birds`'. While the names '`birds`' and '`./birds`' name the same file, *member* names are compared using a simplistic name comparison, in which an exact match is necessary. See Section 6.8.2 [absolute], page 68.

However, *`tar --list --file=collection.tar folk`* would respond with '`folk`', because '`folk`' is in the archive file '`collection.tar`'. If you are not sure of the exact file name, try listing all the files in the archive and searching for the one you expect to find; remember that if you use '`--list`' with no file names as arguments, `tar` will print the names of all the members stored in the specified archive.

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.tar`', type:

```
$ tar --list --verbose --file=music.tar practice
```

`tar` responds:

```
drwxrwxrwx myself user 0 1990-05-31 21:49 practice/
-rw-rw-rw- myself user 42 1990-05-21 13:29 practice/blues
-rw-rw-rw- myself user 62 1990-05-23 10:55 practice/folk
-rw-rw-rw- myself user 40 1990-05-21 13:30 practice/jazz
-rw-rw-rw- myself user 10240 1990-05-31 21:49 practice/collection.tar
```

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

2.6 How to Extract Members from an Archive

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

Creating an archive is only half the job—there is no point in storing files in an archive if you can't retrieve them. The act of retrieving members from an archive so they can be used and manipulated as unarchived files again is called *extraction*. To extract files from an archive, use the *`--extract`* (*`--get`*, *`-x`*) operation. As with *`--create`* (*`-c`*), specify the name of the archive with *`--file=archive-name`* (*`-f archive-name`*). Extracting an archive does not modify the archive in any way; you can extract it multiple times if you want or need to.

Using '`--extract`', you can extract an entire archive, or specific files. The files can be directories containing other files, or not. As with *`--create`* (*`-c`*) and *`--list`* (*`-t`*), you may use the short or the long form of the operation without affecting the performance.

2.6.1 Extracting an Entire Archive

To extract an entire archive, specify the archive file name only, with no individual file names as arguments. For example,

```
$ tar -xvf collection.tar
```

produces this:

```
-rw-rw-rw- me user     28 1996-10-18 16:31 jazz
-rw-rw-rw- me user     21 1996-09-23 16:44 blues
-rw-rw-rw- me user     20 1996-09-23 16:44 folk
```

2.6.2 Extracting Specific Files

To extract specific archive members, give their exact member names as arguments, as printed by *`--list`* (*`-t`*). If you had mistakenly deleted one of the files you had placed in the archive '`collection.tar`' earlier (say, '`blues`'), you can extract it from the archive without changing the archive's structure. It will be identical to the original file '`blues`' that you deleted.

First, make sure you are in the '`practice`' directory, and list the files in the directory. Now, delete the file, '`blues`', and list the files in the directory again.

You can now extract the member '`blues`' from the archive file '`collection.tar`' like this:

```
$ tar --extract --file=collection.tar blues
```

If you list the files in the directory again, you will see that the file '`blues`' has been restored, with its original permissions, creation times, and owner.

(These parameters will be identical to those which the file had when you originally placed it in the archive; any changes you may have made before deleting the file from the file system, however, will *not* have been made to the archive member.) The archive file, '`collection.tar`', is the same as it was before you extracted '`blues`'. You can confirm this by running `tar` with *`--list`* (*`-t`*).

Remember that as with other operations, specifying the exact member name is important. *`tar --extract --file=bfiles.tar birds`* will fail, because there is no member named '`birds`'. To extract the member named '`./birds`', you must specify *`tar --extract --file=bfiles.tar ./birds`*. To find the exact member names of the members of an archive, use *`--list`* (*`-t`*) (see Section 2.5 [list], page 12).

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

2.6.3 Extracting Files that are Directories

Extracting directories which are members of an archive is similar to extracting other files. The main difference to be aware of is that if the extracted directory has the same name as any directory already in the working directory, then files in the extracted directory will be placed into the directory of the same name. Likewise, if there are files in the pre-existing directory with the same names as the members which you extract, the files from the extracted archive will overwrite the files already in the working directory (and possible subdirectories). This will happen regardless of whether or not the files in the working directory were more recent than those extracted.

However, 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.

We can demonstrate how to use '`--extract`' to extract a directory file with an example. Change to the '`practice`' directory if you weren't there, and remove the files '`folk`' and '`jazz`'. Then, go back to the parent directory and extract the archive '`music.tar`'. You may either extract the entire archive, or you may extract only the files you just deleted. To extract the entire archive, don't give any file names as arguments after the archive name '`music.tar`'. To extract only the files you deleted, use the following command:

```
$ tar -xvf music.tar practice/folk practice/jazz
```

Because you created the directory with '`practice`' as part of the file names of each of the files by archiving the '`practice`' directory as '`practice`', you must give '`practice`' as part of the file names when you extract those files from the archive.

2.6.4 Commands That Will Fail

Here are some sample commands you might try which will not work, and why they won't work.

If you try to use this command,

```
$ tar -xvf music.tar folk jazz
```

you will get the following response:

```
tar: folk: Not found in archive
tar: jazz: Not found in archive
$
```

This is because these files were not originally *in* the parent directory '`..`', where the archive is located; they were in the '`practice`' directory, and their file names reflect this:

```
$ tar -tvf music.tar
practice/folk
practice/jazz
practice/rock
```

Likewise, if you try to use this command,

```
$ tar -tvf music.tar folk jazz
```

you would get a similar response. Members with those names are not in the archive. You must use the correct member names in order to extract the files from the archive.

If you have forgotten the correct names of the files in the archive, use *`tar --list --verbose`* to list them correctly.

2.7 Going Further Ahead in this Manual

3 Invoking GNU `tar`

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

This chapter is about how one invokes the GNU `tar` command, from the command synopsis (see Section 3.1 [Synopsis], page 17). There are numerous options, and many styles for writing them. One mandatory option specifies the operation `tar` should perform (see Section 3.4.1 [Operation Summary], page 22), other options are meant to detail how this operation should be performed (see Section 3.4.2 [Option Summary], page 24). Non-option arguments are not always interpreted the same way, depending on what the operation is.

You will find in this chapter everything about option styles and rules for writing them (see Section 3.3 [Styles], page 19). On the other hand, operations and options are fully described elsewhere, in other chapters. Here, you will find only synthetic descriptions for operations and options, together with pointers to other parts of the `tar` manual.

Some options are so special they are fully described right in this chapter. They have the effect of inhibiting the normal operation of `tar` or else, they globally alter the amount of feedback the user receives about what is going on. These are the `--help` and `--version` (see Section 3.5 [help], page 32), `--verbose` (`-v`) (see Section 3.6 [verbose], page 33) and `--interactive` (`-w`) options (see Section 3.7 [interactive], page 34).

3.1 General Synopsis of `tar`

The GNU `tar` program is invoked as either one of:

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

The second form is for when old options are being used.

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 or archive members, which specify the files or members `tar` is to act on.

You can actually type in arguments in any order, even if 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.

Each *name* in the synopsis above is interpreted as an archive member name when the main command is one of `--compare` (`--diff`, `-d`), `--delete`, `--extract` (`--get`, `-x`), `--list` (`-t`) or `--update` (`-u`). When naming archive members, you must give the exact name of the member in the archive, as it is printed by `--list` (`-t`). For `--append` (`-r`) and `--create` (`-c`), these *name* arguments specify the names of either files or directory hierarchies to place in the archive. These files or hierarchies should already exist in the file system, prior to the execution of the `tar` command.

`tar` interprets relative file names as being relative to the working directory. `tar` will make all file names relative (by removing leading slashes when archiving or restoring files), unless you specify otherwise (using the `--absolute-names` (`-P`) option). See Section 6.8.2 [absolute], page 68, for more information about `--absolute-names` (`-P`).

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 distinction between file names and archive member names is especially important when shell globbing is used, and sometimes a source of confusion for newcomers. See Section 6.5 [Wildcards], page 64, for more information about globbing. The problem is that shells may only glob using existing files in the file system. Only `tar` itself 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 *names* 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.

If you don't use any file name arguments, `--append` (`-r`), `--delete` and `--concatenate` (`--catenate`, `-A`) will do nothing, while `--create` (`-c`) will usually yield a diagnostic and inhibit `tar` execution. The other operations of `tar` (`--list` (`-t`), `--extract` (`--get`, `-x`), `--compare` (`--diff`, `-d`), and `--update` (`-u`)) will act on the entire contents of the archive.

Besides successful exits, GNU `tar` may fail for many reasons. Some reasons correspond to bad usage, that is, when the `tar` command is improperly written. Errors may be encountered later, while encountering an error processing the archive or the files. Some errors are recoverable, in which case the failure is delayed until `tar` has completed all its work. Some errors are such that it would not meaningful, or at least risky, to continue processing: `tar` then aborts processing immediately. All abnormal exits, whether immediate or delayed, should always be clearly diagnosed on `stderr`, after a line stating the nature of the error.

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` (`--diff`, `-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 Using `tar` Options

GNU `tar` has a total of eight operating modes which allow you to perform a variety of tasks. You are required to choose one operating mode each time you employ the `tar` program by specifying one, and only one operation as an argument to the `tar` command (two lists of four operations each may be found at Section 2.2 [frequent operations], page 7 and Section 4.2.1 [Operations], page 38). Depending on circumstances, you may also wish to customize how the chosen operating mode behaves. For example, you may wish to change the way the output looks, or the format of the files that you wish to archive may require you to do something special in order to make the archive look right.

You can customize and control `tar`'s performance by running `tar` with one or more options (such as `--verbose` (`-v`), which we used in the tutorial). As we said in the tutorial, *options* are arguments to `tar` which are (as their name suggests) optional. Depending on the operating mode, you may specify one or more options. Different options will have different effects, but in general they all change details of the operation, such as archive format, archive name, or level of user interaction. Some options make sense with all operating modes, while others are meaningful only with particular modes. You will likely use some options frequently, while you will only use others infrequently, or not at all. (A full list of options is available in see Section 3.4 [All Options], page 22.)

Note that `tar` options are case sensitive. For example, the options '`-T`' and '`-t`' are different; the first requires an argument for stating the name of a file providing a list of *names*, while the second does not require an argument and is another way to write `--list` (`-t`).

In addition to the eight operations, there are many options to `tar`, and three different styles for writing both: long (mnemonic) form, short form, and old style. These styles are discussed below. Both the options and the operations can be written in any of these three styles.

3.3 The Three Option Styles

There are three styles for writing operations and options to the command line invoking `tar`. The different styles were developed at different times during the history of `tar`. These styles will be presented below, from the most recent to the oldest.

Some options must take an argument. (For example, `--file=`*archive-name* (`-f` *archive-name*) takes the name of an archive file as an argument. If you do not supply an archive file name, `tar` will use a default, but this can be confusing; thus, we recommend that you always supply a specific archive file name.) Where you *place* the arguments generally depends on which style of options you choose. We will detail specific information relevant to each option style in the sections on the different option styles, below. The differences are subtle, yet can often be very important; incorrect option placement can cause you to overwrite a number of important files. We urge you to note these differences, and only use the option style(s) which makes the most sense to you until you feel comfortable with the others.

3.3.1 Mnemonic Option Style

Each option has at least one long (or mnemonic) name starting with two dashes in a row, e.g. '`list`'. The long names are more clear than their corresponding short or old names. It sometimes happens that a single mnemonic option has many different different names which are synonymous, such as '`--compare`' and '`--diff`'. In addition, long 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`'. (One way to find this out is by trying it and seeing what happens; if a particular abbreviation could represent more than one option, `tar` will tell you that that abbreviation is ambiguous and you'll know that that abbreviation won't work. You may also choose to run '`tar --help`' to see a list of options. Be aware that if you run `tar` with a unique abbreviation for the long name of an option you didn't want to use, you are stuck; `tar` will perform the command as ordered.)

Mnemonic options are meant to be obvious and easy to remember, and their meanings are generally easier to discern than those of their corresponding short options (see below). For example:

```
$ tar --create --verbose --blocking-factor=20 --file=/dev/rmt0
```

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

Mnemonic options which require arguments take those arguments immediately following the option name; they are introduced by an equal sign. For example, the '`--file`' option (which tells the name of the `tar` archive) is given a file such as '`archive.tar`' as argument by using the notation '`--file=archive.tar`' for the mnemonic option.

3.3.2 Short Option Style

Most options also have a short option name. Short options start with a single dash, and are followed by a single character, e.g. '`-t`' (which is equivalent to '`--list`'). The forms are absolutely identical in function; they are interchangeable.

The short option names are faster to type than long option names.

Short options which require arguments take their arguments immediately following the option, usually separated by white space. It is also possible to stick the argument right after the short option name, using no intervening space. For example, you 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 which indicates a specific archive, here named '`archive.tar`'.

Short options' letters may be clumped together, but you are not required to do this (as compared to old options; see below). When short options are clumped as a set, use one (single) dash for them all, e.g. '`tar -cvf`'. Only the last option in such a set is allowed to have an argument[1].

When the options are separated, the argument for each option which requires an argument directly follows that option, as is usual for Unix programs. For example:

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

If you reorder short options' locations, be sure to move any arguments that belong to them. If you do not move the arguments properly, you may end up overwriting files.

3.3.3 Old Option Style

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

Like short options, old options are single letters. However, old options must be written together as a single clumped set, without spaces separating them or dashes preceding them[2]. This set of letters must be the first to appear on the command line, after the `tar` program name and some whitespace; old options cannot appear anywhere else. 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 the mnemonic option '`--list`'. So for example, the command '`tar cv`' specifies the option '`-v`' in addition to the operation '`-c`'.

---

[1] Clustering many options, the last of which has an argument, is a rather opaque way to write options. Some wonder if GNU `getopt` should not even be made helpful enough for considering such usages as invalid.

[2] Beware that if you precede options with a dash, you are announcing the short option style instead of the old option style; short options are decoded differently.

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 given previously 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 reorder any corresponding argument appropriately.

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.

Old options are kept for compatibility with old versions of `tar`.

This second example could be corrected in many ways, among which the following are equivalent:

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

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. 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.

3.3.4 Mixing Option Styles

All three styles may be intermixed in a single `tar` command, so long as the rules for each style are fully respected[3]. Old style options and either of the modern styles of options may be mixed within a single `tar` command. However, old style options must be introduced as the first arguments only, following the rule for old options (old options must appear directly after the `tar` command and some whitespace). Modern options may be given only after all arguments to the old 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.

[3] Before GNU `tar` version 1.11.6, a bug prevented intermixing old style options with mnemonic options in some cases.

For example, all the following commands are wholly equivalent, and illustrate the 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 from what the user intended (judging based on the example in the previous set which uses long options, whose intent is therefore very clear). 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 the value '`archive.tar`'. The last example contains only old style option letters (repeating option '`c`' twice), not all of which are meaningful (eg., '`.`', '`h`', or '`i`'), with no argument value.

3.4 All `tar` Options

The coming manual sections contain an alphabetical listing of all `tar` operations and options, with brief descriptions and cross references to more in-depth explanations in the body of the manual. They also contain an alphabetically arranged table of the short option forms with their corresponding long option. You can use this table as a reference for deciphering `tar` commands in scripts.

3.4.1 Operations

--append
-r

Appends files to the end of the archive. See Section 4.2.2 [append], page 39.

--catenate
-A

Same as '`--concatenate`'. See Section 4.2.4 [concatenate], page 42.

--compare
-d

Compares archive members with their counterparts in the file system, and reports differences in file size, mode, owner, modification date and contents. See Section 4.2.6 [compare], page 44.

--concatenate
-A

Appends other `tar` archives to the end of the archive. See Section 4.2.4 [concatenate], page 42.

--create
-c

Creates a new `tar` archive. See Section 2.4 [create], page 8.

--delete

Deletes members from the archive. Don't try this on a archive on a tape! See Section 4.2.5 [delete], page 43.

--diff
-d

Same '`--compare`'. See Section 4.2.6 [compare], page 44.

--extract
-x

Extracts members from the archive into the file system. See Section 2.6 [extract], page 13.

--get
-x

Same as '`--extract`'. See Section 2.6 [extract], page 13.

--list
-t

Lists the members in an archive. See Section 2.5 [list], page 12.

--update
-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. See Section 4.2.3 [update], page 41.

3.4.2 tar Options

--absolute-names
-P

Normally when creating an archive, `tar` strips an initial '`/`' from member names. This option disables that behavior. .

--after-date
(See '`--newer`'; .)

--atime-preserve
Tells `tar` to preserve the access time field in a file's inode when dumping it. .

--backup=backup-type
Rather than deleting files from the file system, `tar` will back them up using simple or numbered backups, depending upon *backup-type*. .

--block-number
-R

With this option present, `tar` prints error messages for read errors with the block number in the archive file. .

--blocking-factor=blocking
-b blocking
Sets the blocking factor `tar` uses to *blocking* x 512 bytes per record. .

--checkpoint
This option directs `tar` to print periodic checkpoint messages as it reads through the archive. Its intended for when you want a visual indication that `tar` is still running, but don't want to see '`--verbose`' output. .

--compress
--uncompress
-Z

`tar` will use the `compress` program when reading or writing the archive. This allows you to directly act on archives while saving space. .

--confirmation
(See '`--interactive`'; .)

--dereference
-h

When creating a `tar` archive, `tar` will archive the file that a symbolic link points to, rather than archiving the symlink. .

--directory=dir
-C dir

When this option is specified, `tar` will change its current directory to *dir* before performing any operations. When this option is used during archive creation, it is order sensitive. .

--exclude=pattern
When performing operations, `tar` will skip files that match *pattern*. .

--exclude-from=file
-X file

Similar to '--exclude', except tar will use the list of patterns in the file *file*. .

--file=archive
-f archive

tar will use the file *archive* as the tar archive it performs operations on, rather than tar's compilation dependent default. .

--files-from=file
-T file

tar will use the contents of *file* as a list of archive members or files to operate on, in addition to those specified on the command-line. .

--force-local

Forces tar to interpret the filename given to '--file' as a local file, even if it looks like a remote tape drive name. .

--group=group

Files added to the tar archive will have a group id of *group*, rather than the group from the source file. *group* is first decoded as a group symbolic name, but if this interpretation fails, it has to be a decimal numeric group ID. .

Also see the comments for the *--owner=user* option.

--gunzip

(See '--gzip'; .)

--gzip
--gunzip
--ungzip
-z

This option tells tar to read or write archives through gzip, allowing tar to directly operate on several kinds of compressed archives transparently. .

--help

tar will print out a short message summarizing the operations and options to tar and exit. .

--ignore-failed-read

Instructs tar to exit successfully if it encounters an unreadable file. See Section 4.3.1 [Reading], page 45.

--ignore-umask

(See '--preserve-permissions'; see Section 4.3.2 [Writing], page 46.)

--ignore-zeros
-i

With this option, tar will ignore zeroed blocks in the archive, which normally signals EOF. See Section 4.3.1 [Reading], page 45.

--incremental
-G

Used to inform `tar` that it is working with an old GNU-format incremental backup archive. It is intended primarily for backwards compatibility only. .

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

When `tar` is performing multi-tape backups, *script-file* is run at the end of each tape. .

--interactive
--confirmation
-w

Specifies that `tar` should ask the user for confirmation before performing potentially destructive options, such as overwriting files. .

--keep-old-files
-k

When extracting files from an archive, `tar` will not overwrite existing files if this option is present. See Section 4.3.2 [Writing], page 46.

--label=name
-V name

When creating an archive, instructs `tar` to write *name* as a name record in the archive. When extracting or listing archives, `tar` will only operate on archives that have a label matching the pattern specified in *name*. .

--listed-incremental=snapshot-file
-g snapshot-file

During a '`--create`' operation, specifies that the archive that `tar` creates is a new GNU-format incremental backup, using *snapshot-file* to determine which files to backup. With other operations, informs `tar` that the archive is in incremental format. .

--mode=permissions

When adding files to an archive, `tar` will use *permissions* for the archive members, rather than the permissions from the files. The program `chmod` and this `tar` option share the same syntax for what *permissions* might be. See section "File permissions" in *GNU file utilities*. This reference also has useful information for those not being overly familiar with the Unix permission system.

Of course, *permissions* might be plainly specified as an octal number. However, by using generic symbolic modifications to mode bits, this allows more flexibility. For example, the value '`a+rw`' adds read and write permissions for everybody, while retaining executable bits on directories or on any other file already marked as executable.

--multi-volume
-M

Informs `tar` that it should create or otherwise operate on a multi-volume `tar` archive. .

--new-volume-script
: (see –info-script)

--newer=date
--after-date=date
-N
: When creating an archive, `tar` will only add files that have changed since *date*. .

--newer-mtime
: In conjunction with '`--newer`', `tar` will only add files whose contents have changed (as opposed to just '`--newer`', which will also back up files for which any status information has changed).

--no-recursion
: With this option, `tar` will not recurse into directories unless a directory is explicitly named as an argument to `tar`. .

--null
: When `tar` is using the '`--files-from`' option, this option instructs `tar` to expect filenames terminated with *NUL*, so `tar` can correctly work with file names that contain newlines. .

--numeric-owner
: This option will notify `tar` that it should use numeric user and group IDs when creating a `tar` file, rather than names. .

--old-archive
: (See '`--portability`'; .)

--one-file-system
-l
: Used when creating an archive. Prevents `tar` from recursing into directories that are on different file systems from the current directory. .

--owner=user
: Specifies that `tar` should use *user* as the owner of members when creating archives, instead of the user associated with the source file. *user* is first decoded as a user symbolic name, but if this interpretation fails, it has to be a decimal numeric user ID. .

There is no value indicating a missing number, and '`0`' usually means `root`. Some people like to force '`0`' as the value to offer in their distributions for the owner of files, because the `root` user is anonymous anyway, so that might as well be the owner of anonymous archives.

--portability
--old-archive
-o
: Tells `tar` to create an archive that is compatible with Unix V7 `tar`. .

--posix

Instructs `tar` to create a POSIX compliant `tar` archive. .

--preserve

Synonymous with specifying both '`--preserve-permissions`' and '`--same-order`'. .

--preserve-order

(See '`--same-order`'; see Section 4.3.1 [Reading], page 45.)

--preserve-permissions
--same-permissions
-p

When `tar` is extracting an archive, it normally subtracts the users' umask from the permissions specified in the archive and uses that number as the permissions to create the destination file. Specifying this option instructs `tar` that it should use the permissions directly from the archive. See Section 4.3.2 [Writing], page 46.

--read-full-records
-B

Specifies that `tar` should reblock its input, for reading from pipes on systems with buggy implementations. See Section 4.3.1 [Reading], page 45.

--record-size=size

Instructs `tar` to use *size* bytes per record when accessing the archive. .

--recursive-unlink

Similar to the '`--unlink-first`' option, removing existing directory hierarchies before extracting directories of the same name from the archive. See Section 4.3.2 [Writing], page 46.

--remove-files

Directs `tar` to remove the source file from the file system after appending it to an archive. .

--rsh-command=cmd

Notifies `tar` that is should use *cmd* to communicate with remote devices. .

--same-order
--preserve-order
-s

This option is an optimization for `tar` when running on machines with small amounts of memory. It informs `tar` that the list of file arguments has already been sorted to match the order of files in the archive. See Section 4.3.1 [Reading], page 45.

--same-owner

When extracting an archive, `tar` will attempt to preserve the owner specified in the `tar` archive with this option present. .

--same-permissions

(See '`--preserve-permissions`'; see Section 4.3.2 [Writing], page 46.)

--show-omitted-dirs
: Instructs `tar` to mention directories its skipping over when operating on a `tar` archive. .

--sparse
-S
: Invokes a GNU extension when adding files to an archive that handles sparse files efficiently. .

--starting-file=name
-K name
: This option affects extraction only; `tar` will skip extracting files in the archive until it finds one that matches *name*. See Section 4.3.3 [Scarce], page 48.

--suffix=suffix
: Alters the suffix `tar` uses when backing up files from the default '~'. .

--tape-length=num
-L num
: Specifies the length of tapes that `tar` is writing as being *num* x 1024 bytes long. .

--to-stdout
-O
: During extraction, `tar` will extract files to stdout rather than to the file system. See Section 4.3.2 [Writing], page 46.

--totals
: Displays the total number of bytes written after creating an archive. .

--touch
-m
: Sets the modification time of extracted files to the extraction time, rather than the modification time stored in the archive. See Section 4.3.2 [Writing], page 46.

--uncompress
: (See '`--compress`'; .)

--ungzip
: (See '`--gzip`'; .)

--unlink-first
-U
: Directs `tar` to remove the corresponding file from the file system before extracting it from the archive. See Section 4.3.2 [Writing], page 46.

--use-compress-program=prog
: Instructs `tar` to access the archive through *prog*, which is presumed to be a compression program of some sort. .

--verbose
-v

Specifies that `tar` should be more verbose about the operations its performing. This option can be specified multiple times for some operations to increase the amount of information displayed. .

--verify
-W

Verifies that the archive was correctly written when creating an archive. .

--version

`tar` will print an informational message about what version it is and a copyright message, some credits, and then exit. .

--volno-file=file

Used in conjunction with '`--multi-volume`'. `tar` will keep track of which volume of a multi-volume archive its working in *file*. .

3.4.3 Short Options Cross Reference

Here is an alphabetized list of all of the short option forms, matching them with the equivalent long option.

-A

'`--concatenate`'

-B

'`--read-full-records`'

-C

'`--directory`'

-F

'`--info-script`'

-G

'`--incremental`'

-K

'`--starting-file`'

-L

'`--tape-length`'

-M

'`--multi-volume`'

-N

'`--newer`'

-O

'`--to-stdout`'

-P

'`--absolute-names`'

-R
: '--block-number'

-S
: '--sparse'

-T
: '--files-from'

-U
: '--unlink-first'

-V
: '--label'

-W
: '--verify'

-X
: '--exclude-from'

-Z
: '--compress'

-b
: '--blocking-factor'

-c
: '--create'

-d
: '--compare'

-f
: '--file'

-g
: '--listed-incremental'

-h
: '--dereference'

-i
: '--ignore-zeros'

-k
: '--keep-old-files'

-l
: '--one-file-system'

-m
: '--touch'

-o
: '--portability'

-p
: '--preserve-permissions'

-r
: '--append'

-s
: '--same-order'

-t
: '--list'

-u
: '--update'

-v
: '--verbose'

-w
: '--interactive'

-x
: '--extract'

-z
: '--gzip'

3.5 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:

```
tar (GNU tar) 1.13
```

The first occurrence of '`tar`' in the result above is the program name in the package (for example, `rmt` is another program), while the second occurrence of '`tar`' is the name of the package itself, containing possibly many programs. The package is currently named '`tar`', after the name of the main program it contains[4].

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,

[4] There are plans to merge the `cpio` and `tar` packages into a single one which would be called `paxutils`. So, who knows if, one of this days, the *--version* would not yield '`tar (GNU paxutils) 3.2`'

`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`. If you know about some *keyword* which interests you and do not want to read all the `--help` output, another common idiom is doing:

```
tar --help | grep keyword
```

for getting only the pertinent lines.

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.

There is currently no `man` page for GNU `tar`. If you observe such a `man` page on the system you are running, either it does not long to GNU `tar`, or it has not been produced by GNU. Currently, GNU `tar` documentation is provided in Texinfo format only, if we except, of course, the short result of `tar --help`.

3.6 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` (`--get`, `-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 `--block-number` (`-R`), but do want visual confirmation that `tar` is actually making forward progress.

The `--show-omitted-dirs` option, when reading an archive—with `--list` (`-t`) or `--extract` (`--get`, `-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 `--block-number` (`-R`) is used, `tar` prints, along with every message it would normally produce, the block number within the archive where the message was triggered. Also, supplementary messages are triggered when reading blocks full of NULs, or when hitting end of file on the archive. As of now, if the archive if properly terminated with a NUL block, the reading of the file may stop before end of file is met, so the position of end of file will not usually show when `--block-number` (`-R`) is used. Note that GNU `tar` drains the archive before exiting when reading the archive from a pipe.

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). .

3.7 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 (see Chapter 6 [Choosing], page 61), 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.

Verbose output is normally sent to standard output, separate from other error messages. However, if the archive is produced directly on standard output, then verbose output is mixed with errors on `stderr`. Producing the archive on standard output may be used as a way to avoid using disk space, when the archive is soon to be consumed by another process reading it, say. Some people felt the need of producing an archive on stdout, still willing to segregate between verbose output and error output. A possible approach would be using a named pipe to receive the archive, and having the consumer process to read from that named pipe. This has the advantage of letting standard output free to receive verbose output, all separate from errors.

4 GNU tar Operations

4.1 Basic GNU tar Operations

The basic `tar` operations, `--create` (`-c`), `--list` (`-t`) and `--extract` (`--get`, `-x`), are currently presented and described in the tutorial chapter of this manual. This section provides some complementary notes for these operations.

`--create` (`-c`)
: Creating an empty archive would have some kind of elegance. One can initialize an empty archive and later use `--append` (`-r`) for adding all members. Some applications would not welcome making an exception in the way of adding the first archive member. On the other hand, many people reported that it is dangerously too easy for `tar` to destroy a magnetic tape with an empty archive[1]. The two most common errors are:

  1. Mistakingly using `create` instead of `extract`, when the intent was to extract the full contents of an archive. This error is likely: keys `c` and `x` are right next ot each other on the QWERTY keyboard. Instead of being unpacked, the archive then gets wholly destroyed. When users speak about *exploding* an archive, they usually mean something else :-).
  2. Forgetting the argument to `file`, when the intent was to create an archive with a single file in it. This error is likely because a tired user can easily add the `f` key to the cluster of option letters, by the mere force of habit, without realizing the full consequence of doing so. The usual consequence is that the single file, which was meant to be saved, is rather destroyed.

  So, recognizing the likelihood and the catastrophical nature of these errors, GNU `tar` now takes some distance from elegance, and cowardly refuses to create an archive when `--create` (`-c`) option is given, there are no arguments besides options, and `--files-from=`*file-of-names* (`-T` *file-of-names*) option is *not* used. To get around the cautiousness of GNU `tar` and nevertheless create an archive with nothing in it, one may still use, as the value for the `--files-from=`*file-of-names* (`-T` *file-of-names*) option, a file with no names in it, as shown in the following commands:

  ```
  tar --create --file=empty-archive.tar --files-from=/dev/null
  tar cfT empty-archive.tar /dev/null
  ```

`--extract` (`--get`, `-x`)
: A socket is stored, within a GNU `tar` archive, as a pipe.

`--list` (`-t`)
: GNU `tar` now shows dates as '`1996-11-09`', while it used to show them as '`Nov 11 1996`'. (One can revert to the old behavior by defining `USE_OLD_CTIME` in '`src/list.c`' before reinstalling.) But preferrably, people you should get used to ISO 8601 dates. Local American dates should be made available again with

[1] This is well described in *Unix-haters Handbook*, by Simson Garfinkel, Daniel Weise & Steven Strassmann, IDG Books, ISBN 1-56884-203-1.

full date localisation support, once ready. In the meantime, programs not being localisable for dates should prefer international dates, that's really the way to go.

Look up `http://www.ft.uni-erlangen.de/~mskuhn/iso-time.html` if you are curious, it contains a detailed explanation of the ISO 8601 standard.

4.2 Advanced GNU `tar` Operations

Now that you have learned the basics of using GNU `tar`, you may want to learn about further ways in which `tar` can help you.

This chapter presents five, more advanced operations which you probably won't use on a daily basis, but which serve more specialized functions. We also explain the different styles of options and why you might want to use one or another, or a combination of them in your `tar` commands. Additionally, this chapter includes options which allow you to define the output from `tar` more carefully, and provide help and error correction in special circumstances.

4.2.1 The Five Advanced `tar` Operations

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

In the last chapter, you learned about the first three operations to `tar`. This chapter presents the remaining five operations to `tar`: '`--append`', '`--update`', '`--concatenate`', '`--delete`', and '`--compare`'.

You are not likely to use these operations as frequently as those covered in the last chapter; however, since they perform specialized functions, they are quite useful when you do need to use them. We will give examples using the same directory and files that you created in the last chapter. As you may recall, the directory is called '`practice`', the files are '`jazz`', '`blues`', '`folk`', '`rock`', and the two archive files you created are '`collection.tar`' and '`music.tar`'.

We will also use the archive files '`afiles.tar`' and '`bfiles.tar`'. '`afiles.tar`' contains the members '`apple`', '`angst`', and '`aspic`'. '`bfiles.tar`' contains the members '`./birds`', '`baboon`', and '`./box`'.

Unless we state otherwise, all practicing you do and examples you follow in this chapter will take place in the '`practice`' directory that you created in the previous chapter; see Section 2.4.1 [prepare for examples], page 9. (Below in this section, we will remind you of the state of the examples where the last chapter left them.)

The five operations that we will cover in this chapter are:

`--append`
`-r`
: Add new entries to an archive that already exists.

`--update`
`-r`
: Add more recent copies of archive members to the end of an archive, if they exist.

`--concatenate`
`--catenate`
`-A`
: Add one or more pre-existing archives to the end of another archive.

`--delete` Delete items from an archive (does not work on tapes).

`--compare`
`--diff`
`-d` Compare archive members to their counterparts in the file system.

Currently, the listing of the directory using `ls` is as follows:

The archive file '`collection.tar`' looks like this:

```
$ tar -tvf collection.tar
```

The archive file '`music.tar`' looks like this:

```
$ tar -tvf music.tar
```

4.2.2 How to Add Files to Existing Archives: `--append`

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

If you want to add files to an existing archive, you don't need to create a new archive; you can use `--append` (`-r`). The archive must already exist in order to use '`--append`'. (A related operation is the '`--update`' operation; you can use this to add newer versions of archive members to an existing archive. To learn how to do this with '`--update`', see Section 4.2.3 [update], page 41.)

If you use `--append` (`-r`) to add a file that has the same name as an archive member to an archive containing that archive member, then the old member is not deleted. What does happen, however, is somewhat complex. `tar` *allows* you to have infinite numbers of files with the same name. Some operations treat these same-named members no differently than any other set of archive members: for example, if you view an archive with `--list` (`-t`), you will see all of those members listed, with their modification times, owners, etc.

Other operations don't deal with these members as perfectly as you might prefer; if you were to use `--extract` (`--get`, `-x`) to extract the archive, only the most recently added copy of a member with the same name as four other members would end up in the working directory. This is because '`--extract`' extracts an archive in the order the members appeared in the archive; the most recently archived members will be extracted last. Additionally, an extracted member will *overwrite* a file of the same name which existed in the directory already, and `tar` will not prompt you about this. Thus, only the most recently archived member will end up being extracted, as it will overwrite the one extracted before it, and so on.

There are a few ways to get around this. .

If you want to replace an archive member, use `--delete` to delete the member you want to remove from the archive, , and then use '`--append`' to add the member you want to be in the archive. Note that you can not change the order of the archive; the most recently added member will still appear last. In this sense, you cannot truely "replace" one member with another. (Replacing one member with another will not work on certain types of media, such as tapes; see Section 4.2.5 [delete], page 43 and Chapter 9 [Media], page 99, for more information.)

4.2.2.1 Appending Files to an Archive

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

The simplest way to add a file to an already existing archive is the `--append` (`-r`) operation, which writes specified files into the archive whether or not they are already among the archived files. When you use '`--append`', you *must* specify file name arguments, as 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. As with other operations, the member names of the newly added files will be exactly the 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.

'`--append`' cannot be performed on some tape drives, unfortunately, due to deficiencies in the formats those tape drives use. The archive must be a valid `tar` archive, or else the results of using this operation will be unpredictable. See Chapter 9 [Media], page 99.

To demonstrate using '`--append`' to add a file to an archive, create a file called '`rock`' in the '`practice`' directory. Make sure you are in the '`practice`' directory. Then, run the following `tar` command to add '`rock`' to '`collection.tar`':

```
$ tar --append --file=collection.tar rock
```

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

```
$ tar --list --file=collection.tar
-rw-rw-rw- me user     28 1996-10-18 16:31 jazz
-rw-rw-rw- me user     21 1996-09-23 16:44 blues
-rw-rw-rw- me user     20 1996-09-23 16:44 folk
-rw-rw-rw- me user     20 1996-09-23 16:44 rock
```

4.2.2.2 Multiple Files with the Same Name

You can use `--append` (`-r`) to add copies of files which have been updated since the archive was created. (However, we do not recommend doing this since there is another `tar` option called '`--update`'; see Section 4.2.3 [update], page 41 for more information. We describe this use of '`--append`' here for the sake of completeness.) When you extract the archive, the older version will be effectively lost. This works because files are extracted from an archive in the order in which they were archived. Thus, when the archive is extracted, a file archived later in time will overwrite a file of the same name which was archived earlier, even though the older version of the file will remain in the archive unless you delete all versions of the file.

Supposing you change the file '`blues`' and then append the changed version to '`collection.tar`'. As you saw above, the original '`blues`' is in the archive '`collection.tar`'. If you change the file and append the new version of the file to the archive, there will be two copies in the archive. When you extract the archive, the older version of the file will be extracted first, and then overwritten by the newer version when it is extracted.

You can append the new, changed copy of the file '`blues`' to the archive in this way:

```
$ tar --append --verbose --file=collection.tar blues
blues
```

Because you specified the '--verbose' option, tar has printed the name of the file being appended as it was acted on. Now list the contents of the archive:

```
$ tar --list --verbose --file=collection.tar
-rw-rw-rw- me user     28 1996-10-18 16:31 jazz
-rw-rw-rw- me user     21 1996-09-23 16:44 blues
-rw-rw-rw- me user     20 1996-09-23 16:44 folk
-rw-rw-rw- me user     20 1996-09-23 16:44 rock
-rw-rw-rw- me user     58 1996-10-24 18:30 blues
```

The newest version of 'blues' is now at the end of the archive (note the different creation dates and file sizes). If you extract the archive, the older version of the file 'blues' will be overwritten by the newer version. You can confirm this by extracting the archive and running 'ls' on the directory. See Section 4.3.2 [Writing], page 46, for more information. (*Please note:* This is the case unless you employ the `--backup` option; .)

4.2.3 Updating an Archive

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

In the previous section, you learned how to use `--append` (`-r`) to add a file to an existing archive. A related operation is `--update` (`-u`). The '--update' operation 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 newer version of the file is added to the archive (as with `--append` (`-r`)).

Unfortunately, you cannot use '--update' with magnetic tape drives. The operation will fail.

Both '--update' and '--append' 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, unless you use the `--backup` option ().

4.2.3.1 How to Update an Archive Using --update

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 and won't tell you that it didn't do anything (which may end up confusing you).

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

```
$ tar --update -v -f collection.tar 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 run 'tar --list' and look at the archive, you will see 'blues' and 'classical' at its end. There will be a total of two versions of the member 'blues'; the one at the end will be newer and larger, since you added text before updating it.

(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. See Chapter 9 [Media], page 99, for more information about tapes.

--update (*-u*) is not suitable for performing backups for two reasons: it does not change directory content entries, and it lengthens the archive every time it is used. The GNU `tar` options intended specifically for backups are more efficient. If you need to run backups, please consult Chapter 5 [Backups], page 51.

4.2.4 Combining Archives with --concatenate

Sometimes it may be convenient to add a second archive onto the end of an archive rather than adding individual files to the archive. To add one or more archives to the end of another archive, you should use the *--concatenate* (*--catenate*, *-A*) operation.

To use '`--concatenate`', name the archives to be concatenated 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 any members; all the members with the same name coexist. For information on how this affects reading the archive, .

To demonstrate how '`--concatenate`' works, create two small archives called '`bluesrock.tar`' and '`folkjazz.tar`', using the relevant files from '`practice`':

```
$ tar -cvf bluesrock.tar blues rock
blues
classical
$ tar -cvf folkjazz.tar folk jazz
folk
jazz
```

If you like, You can run '`tar --list`' to make sure the archives contain what they are supposed to:

```
$ tar -tvf bluesrock.tar
-rw-rw-rw- melissa user    105 1997-01-21 19:42 blues
-rw-rw-rw- melissa user     33 1997-01-20 15:34 rock
$ tar -tvf folkjazz.tar
-rw-rw-rw- melissa user     20 1996-09-23 16:44 folk
-rw-rw-rw- melissa user     65 1997-01-30 14:15 jazz
```

We can concatenate these two archives with `tar`:

```
$ cd ..
$ tar --concatenate --file=bluesrock.tar jazzfolk.tar
```

If you now list the contents of the '`bluesclass.tar`', you will see that now it also contains the archive members of '`jazzfolk.tar`':

```
$ tar --list --file=bluesrock.tar
blues
rock
jazz
folk
```

When you use '--concatenate', the source and target archives must already exist and must have been created using compatable format parameters (). The new, concatenated archive will be called by the same name as the first archive listed on the command line.

Like --*append* (-*r*), this operation cannot be performed on some tape drives, due to deficiencies in the formats those tape drives use.

It may seem more intuitive to you to want or try to use cat to concatenate two archives instead of using the '--concatenate' operation; after all, cat is the utility for combining files.

However, tar archives incorporate an end-of-file marker which must be removed if the concatenated archives are to be read properly as one archive. '--concatenate' removes the end-of-archive marker from the target archive before each new archive is appended. If you use cat to combine the archives, the result will not be a valid tar format archive. If you need to retrieve files from an archive that was added to using the cat utility, use the --*ignore-zeros* (-*i*) option. See [Ignore Zeros], page 45, for further information on dealing with archives improperly combined using the cat shell utility.

You must specify the source archives using --*file*=*archive-name* (-*f archive-name*) (see Section 6.1 [file], page 61). 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.

4.2.5 Removing Archive Members Using '--delete'

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

You can remove members from an archive by using the --*delete* option. Specify the name of the archive with --*file*=*archive-name* (-*f archive-name*) and then specify the names of the members to be deleted; if you list no member names, 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* (--*get*, -*x*), you must give the exact member names when using 'tar --delete'. '--delete' will remove all versions of the named file from the archive. The '--delete' operation can run very slowly.

Unlike other operations, '--delete' has no short form.

This operation will rewrite the archive. You can only use '--delete' on an archive if the archive device allows you to write to any point on the media, such as a disk; because of this, it does not work on magnetic tapes. Do not try to delete an archive member from a magnetic tape; the action will not succeed, and you will be likely to scramble the archive and damage your tape. There is no safe way (except by completely re-writing the archive) to delete files from most kinds of magnetic tape. See Chapter 9 [Media], page 99.

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

```
$ tar --list --file=collection.tar
blues
folk
jazz
rock
practice/blues
practice/folk
```

```
practice/jazz
practice/rock
practice/blues
$ tar --delete --file=collection.tar blues
$ tar --list --file=collection.tar
folk
jazz
rock
$
```

The `--delete` option has been reported to work properly when `tar` acts as a filter from `stdin` to `stdout`.

4.2.6 Comparing Archive Members with the File System

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

The '`--compare`' ('`-d`'), or '`--diff`' operation compares specified archive members against files with the same names, and then reports differences in file size, mode, owner, modification date and contents. You should *only* specify archive member names, not file names. If you do not name any members, then `tar` will compare the entire archive. If a file is represented in the archive but does not exist in the file system, `tar` reports a difference.

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

`tar` ignores files in the file system that do not have corresponding members in the archive.

The following example compares the archive members '`rock`', '`blues`' and '`funk`' in the archive '`bluesrock.tar`' with files of the same name in the file system. (Note that there is no file, '`funk`'; `tar` will report an error message.)

```
$ tar --compare --file=bluesrock.tar rock blues funk
rock
blues
tar: funk not found in archive
```

Depending on the system where you are running `tar` and the version you are running, `tar` may have a different error message, such as:

```
funk: does not exist
```

The spirit behind the `--compare` (`--diff`, `-d`) option is to check whether the archive represents the current state of files on disk, more than validating the integrity of the archive media. For this later goal, See Section 9.8 [verify], page 115.

4.3 Options Used by `--extract`

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

The previous chapter showed how to use `--extract` (`--get`, `-x`) to extract an archive into the filesystem. Various options cause `tar` to extract more information than just file contents, such as the owner, the permissions, the modification date, and so forth. This section presents options to be used with '`--extract`' when certain special considerations arise. You may review the information presented in Section 2.6 [extract], page 13 for more basic information about the '`--extract`' operation.

4.3.1 Options to Help Read Archives

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

Normally, `tar` will request data in full record increments from an archive storage device. If the device cannot return a full record, `tar` will report an error. However, some devices do not always return full records, or do not require the last record of an archive to be padded out to the next record boundary. To keep reading until you obtain a full record, or to accept an incomplete record if it contains an end-of-archive marker, specify the `--read-full-records` (`-B`) option in conjunction with the `--extract` (`--get`, `-x`) or `--list` (`-t`) operations. See Section 9.4 [Blocking], page 102.

The `--read-full-records` (`-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 record from the pipe.

If you're not sure of the blocking factor of an archive, you can read the archive by specifying `--read-full-records` (`-B`) and `--blocking-factor=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. See Section 9.4.2 [Blocking Factor], page 104.

Reading Full Records

`--read-full-records`

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

Ignoring Blocks of Zeros

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` (`--get`, `-x`) or `--list` (`-t`).

Ignore Fail Read

`--ignore-failed-read`

Do not exit with nonzero on unreadable files or directories.

4.3.2 Changing How `tar` Writes Files

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

Options to Prevent Overwriting Files

Normally, `tar` writes extracted files into the file system without regard to the files already on the system; i.e., files with the same names as archive members are overwritten when the archive is extracted. If the name of a corresponding file name is a symbolic link, the file pointed to by the symbolic link will be overwritten instead of the symbolic link itself (if this is possible). Moreover, special devices, empty directories and even symbolic links are automatically removed if they are found to be on the way of the proper extraction.

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 '`--extract`'. When this option is specified, `tar` will report an error stating the name of the files in conflict instead of overwriting the file with the corresponding extracted archive member.

The *`--unlink-first`* (*`-U`*) option removes existing files, symbolic links, empty directories, devices, etc., *prior* to extracting over them. In particular, using this option will prevent replacing an already existing symbolic link by the name of an extracted file, since the link itself is removed prior to the extraction, rather than the file it points to. On some systems, the backing store for the executable *is* the original program text. You could use the *`--unlink-first`* (*`-U`*) option to prevent segmentation violations or other woes when extracting arbitrary executables over currently running copies. Note that if something goes wrong with the extraction and you *did* use this option, you might end up with no file at all. Without this option, if something goes wrong with the extraction, the existing file is not overwritten and preserved.

If you specify the *`--recursive-unlink`* option, `tar` removes *anything* that keeps you from extracting a file as far as current permissions will allow it. This could include removal of the contents of a full directory hierarchy. For example, someone using this feature may be very surprised at the results when extracting a directory entry from the archive. This option can be dangerous; be very aware of what you are doing if you choose to use it.

Keep Old Files

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

Unlink First

`--unlink-first`
`-U`
: Try removing files before extracting over them, instead of trying to overwrite them.

Recursive Unlink

--recursive-unlink
: When this option is specified, try removing files and directory hierarchies before extracting over them. *This is a dangerous option!*

Some people argue that GNU tar should not hesitate to overwrite files with other files when extracting. When extracting a tar archive, they expect to see a faithful copy of the state of the filesystem when the archive was created. It is debatable that this would always be a proper behaviour. For example, suppose one has an archive in which 'usr/local' is a link to 'usr/local2'. Since then, maybe the site removed the link and renamed the whole hierarchy from '/usr/local2' to '/usr/local'. Such things happen all the time. I guess it would not be welcome at all that GNU tar removes the whole hierarchy just to make room for the link to be reinstated (unless it *also* simultaneously restores the full '/usr/local2', of course! GNU tar is indeed able to remove a whole hierarchy to reestablish a symbolic link, for example, but *only if* *--recursive-unlink* is specified to allow this behaviour. In any case, single files are silently removed.

Setting Modification Times

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 *--touch* (*-m*) option in conjunction with *--extract* (*--get*, *-x*).

--touch
-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* (*--get*, *-x*).

Setting Access Permissions

To set the modes (access permissions) of extracted files to those recorded for those files in the archive, use '--same-persmissions' in conjunction with the *--extract* (*--get*, *-x*) operation.

--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* (*--get*, *-x*).

Writing to Standard Output

To write the extracted files to the standard output, instead of creating the files on the file system, use *--to-stdout* (*-O*) in conjunction with *--extract* (*--get*, *-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. If you extract multiple members, they appear on standard output concatenated, in the order they are found in the archive.

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

Removing Files

`--remove-files`
Remove files after adding them to the archive.

4.3.3 Coping with Scarce Resources

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

Starting File

`--starting-file=`*name*
`-K` *name* Starts an operation in the middle of an archive. Use in conjunction with `--extract` (`--get`, `-x`) or `--list` (`-t`).

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 member *name* of the archive. This assumes, of course, that there is now free space, or that you are now extracting into a different file system. (You could also choose to 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. See Section 5.2 [Inc Dumps], page 53, See Section 3.7 [interactive], page 34, and Section 6.4 [exclude], page 63.)

Same Order

`--same-order`
`--preserve-order`
`-s` To process large lists of file names on machines with small amounts of memory. Use in conjunction with `--compare` (`--diff`, `-d`), `--list` (`-t`) or `--extract` (`--get`, `-x`).

The `--same-order` (`--preserve-order`, `-s`) 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.

4.4 Backup options

GNU `tar` offers options for making backups of files before writing new versions. These options control the details of these backups. They may apply to the archive itself before it is created or rewritten, as well as individual extracted members. Other GNU programs (`cp`, `install`, `ln`, and `mv`, for example) offer similar options.

Backup options may prove unexpectedly useful when extracting archives containing many members having identical name, or when extracting archives on systems having file name limitations, making different members appear has having similar names through the side-effect of name truncation. (This is true only if we have a good scheme for truncated backup names, which I'm not sure at all: I suspect work is needed in this area.) When any existing file is backed up before being overwritten by extraction, then clashing files are automatically be renamed to be unique, and the true name is kept for only the last file of a series of clashing files. By using verbose mode, users may track exactly what happens.

At the detail level, some decisions are still experimental, and may change in the future, we are waiting comments from our users. So, please do not learn to depend blindly on the details of the backup features. For example, currently, directories themselves are never renamed through using these options, so, extracting a file over a directory still has good chances to fail. Also, backup options apply to created archives, not only to extracted members. For created archives, backups will not be attempted when the archive is a block or character device, or when it refers to a remote file.

For the sake of simplicity and efficiency, backups are made by renaming old files prior to creation or extraction, and not by copying. The original name is restored if the file creation fails. If a failure occurs after a partial extraction of a file, both the backup and the partially extracted file are kept.

'`--backup`'
: Make backups of files that are about to be overwritten or removed. Without this option, the original versions are destroyed.

'`--suffix=`*suffix*'
: Append *suffix* to each backup file made with '`-b`'. If this option is not specified, the value of the `SIMPLE_BACKUP_SUFFIX` environment variable is used. And if `SIMPLE_BACKUP_SUFFIX` is not set, the default is '`~`', just as in Emacs.

'`--version-control=`*method*'
: Use *method* to determine the type of backups made with `--backup`. If this option is not specified, the value of the `VERSION_CONTROL` environment variable is used. And if `VERSION_CONTROL` is not set, the default backup type is '`existing`'.

  This option corresponds to the Emacs variable '`version-control`'; the same values for *method* are accepted as in Emacs. This options also more descriptive name. The valid *method*s (unique abbreviations are accepted):

  '`t`'
  '`numbered`'
  : Always make numbered backups.

`nil`
`existing`
: Make numbered backups of files that already have them, simple backups of the others.

`never`
`simple`
: Always make simple backups.

Some people express the desire to *always* use the *op-backup* option, by defining some kind of alias or script. This is not as easy as one may thing, due to the fact old style options should appear first and consume arguments a bit inpredictably for an alias or script. But, if you are ready to give up using old style options, you may resort to using something like (a Bourne shell function here):

```
tar () { /usr/local/bin/tar --backup $*; }
```

4.5 Notable `tar` Usages

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

You can easily use archive files to transport 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 system, 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.

For example, here is how you might copy a directory's contents from one disk to another, while preserving the dates, modes, owners and link-structure of all the files therein. In this case, the transfer medium is a *pipe*, which is one a Unix redirection mechanism:

```
$ cd sourcedir; tar -cf - . | (cd targetdir; tar -xf -)
```

The command also works using short option forms:

```
$ cd sourcedir; tar --create --file=- . | (cd targetdir; tar --extract --file=-)
```

This is one of the easiest methods to transfer a `tar` archive.

4.6 Looking Ahead: The Rest of this Manual

You have now seen how to use all eight of the operations available to `tar`, and a number of the possible options. The next chapter explains how to choose and change file and archive names, how to use files to store names of other files which you can then call as arguments to `tar` (this can help you save time if you expect to archive the same list of files a number of times), and how to

If there are too many files to conveniently list on the command line, you can list the names in a file, and `tar` will read that file. See Section 6.3 [files], page 62.

There are various ways of causing `tar` to skip over some files, and not archive them. See Chapter 6 [Choosing], page 61.

5 Performing Backups and Restoring Files

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

GNU `tar` is distributed along with the scripts which the Free Software Foundation uses for performing backups. There is no corresponding scripts available yet for doing restoration of files. Even if there is a good chance those scripts may be satisfying to you, they are not the only scripts or methods available for doing backups and restore. You may well create your own, or use more sophisticated packages dedicated to that purpose.

Some users are enthusiastic about `Amanda` (The Advanced Maryland Automatic Network Disk Archiver), a backup system developed by James da Silva '`jds@cs.umd.edu`' and available on many Unix systems. This is free software, and it is available at these places:

```
http://www.cs.umd.edu/projects/amanda/amanda.html
ftp://ftp.cs.umd.edu/pub/amanda
```

Here is a possible plan for a future documentation about the backuping scripts which are provided within the GNU `tar` distribution.

```
.* 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.
```

This chapter documents both the provided FSF scripts and `tar` options which are more specific to usage as a backup tool.

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*.

5.1 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`)

`--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` (`--get`, `-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. This option handles new GNU-format incremental backup. It has much the same effect as `--incremental` (`-G`), but also the time when the dump is done and the list of directories dumped is written to the given *file*. When restoring, only files newer than the saved time are restored, and the direcotyr list is used to speed up operations.

`--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.

GNU `tar` actually writes the file twice: once before the data and written, and once after.

5.2 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.

5.3 The Incremental Options

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

`--incremental` (*`-G`*) is used in conjunction with *`--create`* (*`-c`*), *`--extract`* (*`--get`*, *`-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`* (*`--get`*, *`-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.

Despite it should be obvious that a device has a non-volatile value, NFS devices have non-dependable values when an automounter gets in the picture. This led to a great deal of spurious redumping in incremental dumps, so it is somewhat useless to compare two NFS devices numbers over time. So `tar` now considers all NFS devices as being equal when it comes to comparing directories; this is fairly gross, but there does not seem to be a better way to go.

5.4 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. . . 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. , for more information on running the restoration script. , 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. . See Section 5.2 [Inc Dumps], page 53, and See Section 5.2 [Inc Dumps], page 53, before making such an attempt.

5.5 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.

, for an explanation of this syntax.

'`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.

'`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. See Section 9.4.2 [Blocking Factor], page 104.

'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.

5.5.1 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]"
```

5.5.2 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:

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.

5.6 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`' ().

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. . .

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. , 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.

5.7 Using the Restore Script

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

> **Warning:** The GNU `tar` distribution does *not* provide any such `restore` script yet. This section is only listed here for documentation maintenance purposes. In any case, all contents is subject to change as things develop.

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. , 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.

See Section 5.2 [Inc Dumps], page 53, and Section 5.2 [Inc Dumps], page 53, for an explanation of how the script makes that determination.

6 Choosing Files and Names for `tar`

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

Certain options to `tar` enable you to specify a name for your archive. Other options let you decide which files to include or exclude from the archive, based on when or whether files were modified, whether the file names do or don't match specified patterns, or whether files are in specified directories.

6.1 Choosing and Naming Archive Files

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

By default, `tar` uses an archive file name that was compiled when it was built on the system; usually this name refers to some physical tape drive on the machine. However, the person who installed `tar` on the system may not set the default to a meaningful value as far as most users are concerned. As a result, you will usually want to tell `tar` where to find (or create) the archive. The `--file=`*archive-name* (`-f` *archive-name*) option allows you to either specify or name a file to use as the archive instead of the default archive file location.

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

For example, in this `tar` command,

```
$ tar -cvf collection.tar blues folk jazz
```

'`collection.tar`' is the name of the archive. It must directly follow the '`-f`' option, since whatever directly follows '`-f`' *will* end up naming the archive. If you neglect to specify an archive name, you may end up overwriting a file in the working directory with the archive you create since `tar` will use this file's name for the archive name.

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, floppy disk, or CD write drive.

If you do not name the archive, `tar` uses the value of the environment variable `TAPE` as the file name for the archive. If that is not available, `tar` uses a default, compiled-in archive name, usually that for tape unit zero (ie. '`/dev/tu00`'). `tar` always needs an archive name.

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.

```
$ cd sourcedir; tar -cf - . | (cd targetdir; tar -xf -)
```

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.

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.

When the archive is being created to '`/dev/null`', GNU `tar` tries to minimize input and output operations. The Amanda backup system, when used with GNU `tar`, has an initial sizing pass which uses this feature.

6.2 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. See Section 4.2.1 [Operations], page 38.

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 with `--create` (`-c`), `tar` operates on all the non-directory files in the working directory. If you specify either `--list` (`-t`) or `--extract` (`--get`, `-x`), `tar` operates on all the archive members in the archive. If you specify any operation other than one of these three, `tar` does nothing.

By default, `tar` takes file names from the command line. However, there are other ways 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.

6.3 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 of files to include as the argument to '`--files-from`'. In the list, the file names should be separated by newlines. You will frequently use this option when you have generated the list of files to archive with the `find` utility.

`--files-from=`*file name*
`-T` *file name*
: Get names to extract or create from file *file name*.

If you give a single dash as a file name for '`--files-from`', (i.e., you specify either '`--files-from=-`' or '`-T -`'), then the file names are read from standard input.

Unless you are running `tar` with '`--create`', you can not use both '`--files-from=-`' and '`--file=-`' ('`-f -`') in the same command.

The following example shows how to use `find` to generate a list of files smaller than 400K in length and put that list into a file called '`small-files`'. You can then use the '`-T`' option to `tar` to specify the files from that file, '`small-files`', to create the archive '`little.tgz`'. (The '`-z`' option to `tar` compresses the archive with `gzip`; see Section 8.2.1 [gzip], page 81 for more information.)

```
$ find . -size -400 -print > small-files
$ tar -c -v -z -T small-files -f little.tgz
```

The `--null` 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`'.

`--null` Only consider `NUL` terminated file names, instead of files that terminate in a newline.

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`'.

This example shows how to use `find` to generate a list of files larger than 800K in length and put that list into a file called '`long-files`'. The '`-print0`' option to `find` just just like '`-print`', except that it separates files with a `NUL` rather than with a newline. You can then run `tar` with both the '`--null`' and '`-T`' options to specify that `tar` get the files from that file, '`long-files`', to create the archive '`big.tgz`'. The '`--null`' option to `tar` will cause `tar` to recognize the `NUL` separator between files.

```
$ find . -size +800 -print0 > long-files
$ tar -c -v --null --files-from=long-files --file=big.tar
```

6.4 Excluding Some Files

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

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.

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

The `--exclude=`*pattern* option will prevent any file or member which matches the shell wildcards (*pattern*) from being operated on (*pattern* can be a single file name or a more complex expression). 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`'. You may give multiple '`--exclude`' options.

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

Use the '`--exclude-from=`*file-of-patterns*' option to read a list of shell wildcards, 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.

Problems with Using the `exclude` Options

Some users find '`exclude`' options confusing. Here are some common pitfalls:

- The main operating mode of `tar` will always act on file names listed on the command line, no matter whether or not there is an exclusion which would otherwise affect them. In the example above, if you create an archive and exclude files that end with '`*.o`', but explicitly name the file '`catc.o`' after all the options have been listed, '`catc.o`' *will* be included in the archive.
- You can sometimes confuse the meanings of `--exclude`=*pattern* and `--exclude-from`=*file-of-patterns* (`-X` *file-of-patterns*). Be careful: use `--exclude`=*pattern* when files to be excluded are given as a pattern on the command line. Use '`--exclude-from=`*file-of-patterns*' to introduce the name of a file which contains a list of patterns, one per line; each of these patterns can exclude zero, one, or many files.
- When you use `--exclude`=*pattern*, be sure to quote the *pattern* parameter, so GNU `tar` sees wildcard characters like '`*`'. If you do not do this, the shell might expand the '`*`' itself using files at hand, so `tar` might receive a list of files instead of one pattern, or none at all, making the command somewhat illegal. This might not correspond to what you want.

  For example, write:

  ```
  $ tar -c -f archive.tar -X '*/tmp/*' directory
  ```

  rather than:

  ```
  $ tar -c -f archive.tar -X */tmp/* directory
  ```
- You must use use shell syntax, or globbing, rather than `regexp` syntax, when using exclude options in `tar`. If you try to use `regexp` syntax to describe files to be excluded, your command might fail.
- In earlier versions of `tar`, what is now the '`--exclude-from=`*file-of-patterns*' option was called '`--exclude-`*pattern*' instead. Now, '`--exclude=`*pattern*' applies to patterns listed on the command line and '`--exclude-from=`*file-of-patterns*' applies to patterns listed in a file.

6.5 Wildcards Patterns and Matching

Globbing is the operation by which *wildcard* characters, '`*`' or '`?`' for example, are replaced and expanded into all existing files matching the given pattern. However, `tar` often uses wildcard patterns for matching (or globbing) archive members instead of actual files in the filesystem. Wildcard patterns are also used for verifying volume labels of `tar` archives. This section has the purpose of explaining wildcard syntax for `tar`.

A *pattern* should be written according to shell syntax, using wildcard characters to effect globbing. Most characters in the pattern stand for themselves in the matched string, and case is significant: '`a`' will match only '`a`', and not '`A`'. The character '`?`' in the pattern matches any single character in the matched string. The character '*' in the pattern matches zero, one, or more single characters in the matched string. The character '\' says to take the following character of the pattern *literally*; it is useful when one needs to match the '?', '*', '[' or '\' characters, themselves.

The character '[', up to the matching ']', introduces a character class. A *character class* is a list of acceptable characters for the next single character of the matched string. For example, '[abcde]' would match any of the first five letters of the alphabet. Note that within a character class, all of the "special characters" listed above other than '\' lose their special meaning; for example, '[-\\[*?]]' would match any of the characters, '-', '\', '[', '*', '?', or ']'. (Due to parsing constraints, the characters '-' and ']' must either come *first* or *last* in a character class.)

If the first character of the class after the opening '[' is '!' or '^', then the meaning of the class is reversed. Rather than listing character to match, it lists those characters which are *forbidden* as the next single character of the matched string.

Other characters of the class stand for themselves. The special construction '[*a*-*e*]', using an hyphen between two letters, is meant to represent all characters between *a* and *e*, inclusive.

Periods ('.') or forward slashes ('/') are not considered special for wildcard matches. However, if a pattern completely matches a directory prefix of a matched string, then it matches the full matched string: excluding a directory also excludes all the files beneath it.

There are some discussions floating in the air and asking for modifications in the way GNU tar accomplishes wildcard matches. We perceive any change of semantics in this area as a delicate thing to impose on GNU tar users. On the other hand, the GNU project should be progressive enough to correct any ill design: compatibility at all price is not always a good attitude. In conclusion, it is *possible* that slight amendments be later brought to the previous description. Your opinions on the matter are welcome.

6.6 Operating Only on New Files

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

The --after-date=*date* (--newer=*date*, -N *date*) option causes tar to only work on files whose modification or inode-changed times are newer than the *date* given. If you use this option when creating or appending to an archive, the archive will only include new files. If you use '--after-date' when extracting an archive, tar will only extract files newer than the *date* you specify.

If you only want tar to make the date comparison based on modification of the actual contents of the file (rather than inode changes), then use the --newer-mtime=*date* option.

You may use these options with any operation. Note that these options differ from the --update (-u) operation in that they allow you to specify a particular date against which tar can compare when deciding whether or not to archive the files.

--after-date=*date*
--newer=*date*
-N *date* Only store files newer than *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` (`--newer=date`, `-N date`), but only looks at modification times.

These options limit `tar` to only operating on files which have been modified after the date specified. A file is considered to have changed if the contents have been modified, or if the owner, permissions, and so forth, have been changed. (For more information on how to specify a date, see Chapter 7 [Date input formats], page 71; remember that the entire date argument must be quoted if it contains any spaces.)

Gurus would say that `--after-date=date` (`--newer=date`, `-N date`) tests both the `mtime` (time the contents of the file were last modified) and `ctime` (time the file's status was last changed: owner, permissions, etc) fields, while `--newer-mtime=date` tests only `mtime` field.

To be precise, `--after-date=date` (`--newer=date`, `-N date`) checks *both* `mtime` and `ctime` and processes the file if either one is more recent than *date*, while `--newer-mtime=date` only checks `mtime` and disregards `ctime`. Neither uses `atime` (the last time the contents of the file were looked at).

Date specifiers can have embedded spaces. Because of this, you may need to quote date arguments to keep the shell from parsing them as separate arguments.

> **Please Note:** `--after-date=date` (`--newer=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 properly by these options. See Section 5.3 [incremental and listed-incremental], page 54.

To select files newer than the modification time of a file that already exists, you can use the '`--reference`' ('`-r`') option of GNU `date`, available in GNU shell utilities 1.13 or later. It returns the timestamp of that already existing file; this timestamp expands to become the referent date which '`--newer`' uses to determine which files to archive. For example, you could say,

```
$ tar -cf archive.tar --newer="`date -r file`" /home
```

which tells .

6.7 Descending into Directories

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

Usually, `tar` will recursively explore all directories (either those given on the command line or through the `--files-from=file-of-names` (`-T file-of-names`) option) for the various files they contain. However, you may not always want `tar` to act this way.

The `--no-recursion` option inhibits `tar`'s recursive descent into specified directories. If you specify '`--no-recursion`', you can use the `find` utility for hunting through levels of directories to construct a list of file names which you could then pass to `tar`. `find` allows you to be more selective when choosing which files to archive; see Section 6.3 [files], page 62 for more information on using `find` with `tar`, or look.

`--no-recursion`
: Prevents `tar` from recursively descending directories.

When you use '`--no-recursion`', GNU `tar` grabs directory entries themselves, but does not descend on them recursively. Many people use `find` for locating files they want to back up, and since `tar` *usually* recursively descends on directories, they have to use the '`! -d`' option to `find` as they usually do not want all the files in a directory. They then use the [No value for "op-file-from"] option to archive the files located via `find`.

The problem when restoring files archived in this manner is that the directories themselves are not in the archive; so the *`--same-permissions`* (*`--preserve-permissions`*, *`-p`*) option does not affect them—while users might really like it to. Specifying *`--no-recursion`* is a way to tell `tar` to grab only the directory entries given to it, adding no new files on its own.

6.8 Crossing Filesystem Boundaries

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

`tar` will normally automatically cross file system boundaries in order to archive files which are part of a directory tree. You can change this behavior by running `tar` and specifying *`--one-file-system`* (*`-l`*). This option only affects files that are archived because they are in a directory that is being archived; `tar` will still archive files explicitly named on the command line or through *`--files-from`=file-of-names* (*`-T` file-of-names*), regardless of where they reside.

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

The '`--one-file-system`' 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, then `tar` will not archive that file. If the file is a directory itself, `tar` will not archive anything beneath it; in other words, `tar` will not cross mount points.

It is reported that using this option, the mount point is is archived, but nothing under it.

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.

6.8.1 Changing the Working Directory

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

To change the 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.

`--directory`=directory
`-C` directory
Changes the working directory in the middle of a command line.

For example,

```
$ tar -c -f jams.tar grape prune -C food cherry
```

will place the files 'grape' and 'prune' from the current directory into the archive 'jams.tar', followed by the file 'cherry' from the directory 'food'. This option is especially useful when you have several widely separated files that you want to store in the same archive.

Note that the file 'cherry' is recorded in the archive under the precise name 'cherry', *not* 'food/cherry'. 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 -f jams.tar grape prune -C food red/cherry
```

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

You can use the '--directory' option to make the archive independent of the original name of the directory holding the files. The following command places the files '/etc/passwd', '/etc/hosts', and '/lib/libc.a' into the archive 'foo.tar':

```
$ tar -c -f foo.tar -C /etc passwd hosts -C /lib libc.a
```

However, the names of the archive members will be exactly what they were on the command line: 'passwd', 'hosts', and 'libc.a'. They will not appear to be related by file name to the original directories where those files were located.

Note that '--directory' options are interpreted consecutively. If '--directory' 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' option.

When using '--files-from' (see Section 6.3 [files], page 62), 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.)

6.8.2 Absolute File Names

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

-P
--absolute-names
: Do not strip leading slashes from file names.

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

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 a non-GNU `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.

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.

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

`tar` prints out a message about removing the '`/`' from file names. This message appears once per GNU `tar` invocation. It represents something which ought to be told; ignoring what it means can cause very serious surprises, later.

Some people, nevertheless, do not want to see this message. Wanting to play really dangerously, one may of course redirect `tar` standard error to the sink. For example, under `sh`:

```
$ tar -c -f archive.tar /home 2> /dev/null
```

Another solution, both nicer and simpler, would be to change to the '`/`' directory first, and then avoid absolute notation. For example:

```
$ (cd / && tar -c -f archive.tar home)
$ tar -c -f archive.tar -C  / home
```

7 Date input formats

> Our units of temporal measurement, from seconds on up to months, are so complicated, asymmetrical and disjunctive so as to make coherent mental reckoning in time all but impossible. Indeed, had some tyrannical god contrived to enslave our minds to time, to make it all but impossible for us to escape subjection to sodden routines and unpleasant surprises, he could hardly have done better than handing down our present system. It is like a set of trapezoidal building blocks, with no vertical or horizontal surfaces, like a language in which the simplest thought demands ornate constructions, useless particles and lengthy circumlocutions. Unlike the more successful patterns of language and science, which enable us to face experience boldly or at least level-headedly, our system of temporal calculation silently and persistently encourages our terror of time.
>
> . . . It is as though architects had to measure length in feet, width in meters and height in ells; as though basic instruction manuals demanded a knowledge of five different languages. It is no wonder then that we often look into our own immediate past or future, last Tuesday or a week from Sunday, with feelings of helpless confusion. . . .
>
> — Robert Grudin, *Time and the Art of Living*.

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.

7.1 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.

7.2 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:

```
1970-09-17           # 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 01 and 12, and *day* is a number between 01 and 31. A leading zero must be present if a number is less than ten. 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

7.3 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'. (This is the zero-oriented interpretation of '12am' and '12pm', as opposed to the old tradition derived from Latin which uses '12m' for noon and '12pm' for midnight.)

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.

7.4 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.

7.5 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.

7.6 Relative item in date strings

Relative items adjust a date (or the current date if none) forward or backward. The effects 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.

7.7 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 (see Section 7.2 [Calendar date item], page 72) 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.

7.8 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 François Pinard ('`pinard@iro.umontreal.ca`') from the '`getdate.y`' source code, and then edited by K. Berry ('`kb@cs.umb.edu`').

8 Controlling the Archive Format

8.1 Making tar Archives More Portable

Creating a `tar` archive on a particular system that is meant to be useful later on many other machines and with other versions of `tar` is more challenging than you might think. `tar` archive formats have been evolving since the first versions of Unix. Many such formats are around, and are not always comptible with each other. This section discusses a few problems, and gives some advice about 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.

8.1.1 Portable Names

Use *straight* file and directory names, made up of printable ASCII characters, avoiding colons, slashes, backslashes, spaces, 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.

8.1.2 Symbolic Links

Normally, when `tar` archives a symbolic link, it writes a block 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.

8.1.3 Old V7 Archives

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.

8.1.4 GNU `tar` and POSIX `tar`

GNU `tar` was based on an early draft of the POSIX 1003.1 `ustar` standard. GNU extensions to `tar`, such as the support for file names longer than 100 characters, use portions of the `tar` header record which were specified in that POSIX draft as unused. Subsequent changes in POSIX have allocated the same parts of the header record for other purposes. As a result, GNU `tar` is incompatible with the current POSIX spec, and with `tar` programs that follow it.

We plan to reimplement these GNU extensions in a new way which is upward compatible with the latest POSIX `tar` format, but we don't know when this will be done.

In the mean time, there is simply no telling what might happen if you read a GNU `tar` archive, which uses the GNU extensions, using some other `tar` program. So if you want to read the archive with another `tar` program, be sure to write it using the '`--old-archive`' option ('`-o`').

Traditionally, old `tar`s have a limit of 100 characters. GNU `tar` attempted two different approaches to overcome this limit, using and extending a format specified by a draft of some P1003.1. The first way was not that successful, and involved '`@MaNgLeD@`' file names, or such; while a second approach used '`././@LongLink`' and other tricks, yielding better success. In theory, GNU `tar` should be able to handle file names of practically unlimited length. So, if GNU `tar` fails to dump and retrieve files having more than 100 characters, then there is a bug in GNU `tar`, indeed.

But, being strictly POSIX, the limit was still 100 characters. For various other purposes, GNU `tar` used areas left unassigned in the POSIX draft. POSIX later revised P1003.1 `ustar` format by assigning previously unused header fields, in such a way that the upper limit for file name length was raised to 256 characters. However, the actual POSIX limit oscillates between 100 and 256, depending on the precise location of slashes in full file name (this is rather ugly). Since GNU `tar` use the same fields for quite other purposes, it became incompatible with the latest POSIX standards.

For longer or non-fitting file names, we plan to use yet another set of GNU extensions, but this time, complying with the provisions POSIX offers for extending the format, rather than conflicting with it. Whenever an archive uses old GNU `tar` extension format or POSIX extensions, would it be for very long file names or other specialities, this archive becomes non-portable to other `tar` implementations. In fact, anything can happen. The most forgiving `tar`s will merely unpack the file using a wrong name, and maybe create another file named something like '`@LongName`', with the true file name in it. `tar`s not protecting themselves may segment violate!

Compatibility concerns make all this thing more difficult, as we will have to support *all* these things together, for a while. GNU `tar` should be able to produce and read true POSIX format files, while being able to detect old GNU `tar` formats, besides old V7 format, and process them conveniently. It would take years before this whole area stabilizes...

There are plans to raise this 100 limit to 256, and yet produce POSIX conformant archives. Past 256, I do not know yet if GNU `tar` will go non-POSIX again, or merely refuse to archive the file.

There are plans so GNU `tar` support more fully the latest POSIX format, while being able to read old V7 format, GNU (semi-POSIX plus extension), as well as full POSIX. One may ask if there is part of the POSIX format that we still cannot support. This simple question has a complex answer. Maybe that, on intimate look, some strong limitations will pop up, but until now, nothing sounds too difficult (but see below). I only have these few pages of POSIX telling about 'Extended tar Format' (P1003.1-1990 – section 10.1.1), and there are references to other parts of the standard I do not have, which should normally enforce limitations on stored file names (I suspect things like fixing what `/` and ⟨NUL⟩ means). There are also some points which the standard does not make clear, Existing practice will then drive what I should do.

POSIX mandates that, when a file name cannot fit within 100 to 256 characters (the variance comes from the fact a `/` is ideally needed as the 156'th character), or a link name cannot fit within 100 characters, a warning should be issued and the file *not* be stored. Unless some `--posix` option is given (or `POSIXLY_CORRECT` is set), I suspect that GNU `tar` should disobey this specification, and automatically switch to using GNU extensions to overcome file name or link name length limitations.

There is a problem, however, which I did not intimately studied yet. Given a truly POSIX archive with names having more than 100 characters, I guess that GNU `tar` up to 1.11.8 will process it as if it were an old V7 archive, and be fooled by some fields which are coded differently. So, the question is to decide if the next generation of GNU `tar` should produce POSIX format by default, whenever possible, producing archives older versions of GNU `tar` might not be able to read correctly. I fear that we will have to suffer such a choice one of these days, if we want GNU `tar` to go closer to POSIX. We can rush it. Another possibility is to produce the current GNU `tar` format by default for a few years, but have GNU `tar` versions from some 1.*POSIX* and up able to recognize all three formats, and let older GNU `tar` fade out slowly. Then, we could switch to producing POSIX format by default, with not much harm to those still having (very old at that time) GNU `tar` versions prior to 1.*POSIX*.

POSIX format cannot represent very long names, volume headers, splitting of files in multi-volumes, sparse files, and incremental dumps; these would be all disallowed if `--posix` or `POSIXLY_CORRECT`. Otherwise, if `tar` is given long names, or '`-[VMSgG]`', then it should automatically go non-POSIX. I think this is easily granted without much discussion.

Another point is that only `mtime` is stored in POSIX archives, while GNU `tar` currently also store `atime` and `ctime`. If we want GNU `tar` to go closer to POSIX, my choice would be to drop `atime` and `ctime` support on average. On the other hand, I perceive that full dumps or incremental dumps need `atime` and `ctime` support, so for those special applications, POSIX has to be avoided altogether.

A few users requested that `--sparse` (`-S`) be always active by default, I think that before replying to them, we have to decide if we want GNU `tar` to go closer to POSIX on average, while producing files. My choice would be to go closer to POSIX in the long run. Besides possible double reading, I do not see any point of not trying to save files as sparse when creating archives which are neither POSIX nor old-V7, so the actual `--sparse` (`-S`) would become selected by default when producing such archives, whatever the reason is. So, `--sparse` (`-S`) alone might be redefined to force GNU-format archives, and recover its previous meaning from this fact.

GNU-format as it exists now can easily fool other POSIX `tar`, as it uses fields which POSIX considers to be part of the file name prefix. I wonder if it would not be a good idea, in the long run, to try changing GNU-format so any added field (like `ctime`, `atime`, file offset in subsequent volumes, or sparse file descriptions) be wholly and always pushed into an extension block, instead of using space in the POSIX header block. I could manage to do that portably between future GNU `tars`. So other POSIX `tars` might be at least able to provide kind of correct listings for the archives produced by GNU `tar`, if not able to process them otherwise.

Using these projected extensions might induce older `tars` to fail. We would use the same approach as for POSIX. I'll put out a `tar` capable of reading POSIXier, yet extended archives, but will not produce this format by default, in GNU mode. In a few years, when newer GNU `tars` will have flooded out `tar` 1.11.X and previous, we could switch to producing POSIXier extended archives, with no real harm to users, as almost all existing GNU `tars` will be ready to read POSIXier format. In fact, I'll do both changes at the same time, in a few years, and just prepare `tar` for both changes, without effecting them, from 1.*POSIX*. (Both changes: 1—using POSIX convention for getting over 100 characters; 2—avoiding mangling POSIX headers for GNU extensions, using only POSIX mandated extension techniques).

So, a future `tar` will have a `--posix` flag forcing the usage of truly POSIX headers, and so, producing archives previous GNU `tar` will not be able to read. So, *once* pretest will announce that feature, it would be particularly useful that users test how exchangeable will be archives between GNU `tar` with `--posix` and other POSIX `tar`.

In a few years, when GNU `tar` will produce POSIX headers by default, `--posix` will have a strong meaning and will disallow GNU extensions. But in the meantime, for a long while, `--posix` in GNU tar will not disallow GNU extensions like `--label=`*archive-label* (`-V` *archive-label*), `--multi-volume` (`-M`), `--sparse` (`-S`), or very long file or link names. However, `--posix` with GNU extensions will use POSIX headers with reserved-for-users extensions to headers, and I will be curious to know how well or bad POSIX `tars` will react to these.

GNU `tar` prior to 1.*POSIX*, and after 1.*POSIX* without `--posix`, generates and checks '`ustar  `', with two suffixed spaces. This is sufficient for older GNU `tar` not to recognize POSIX archives, and consequently, wrongly decide those archives are in old V7 format. It is a useful bug for me, because GNU `tar` has other POSIX incompatibilities, and I need to segregate GNU `tar` semi-POSIX archives from truly POSIX archives, for GNU `tar` should be somewhat compatible with itself, while migrating closer to latest POSIX standards. So, I'll be very careful about how and when I will do the correction.

8.1.5 Checksumming Problems

SunOS and HP-UX `tar` fail to accept archives created using GNU `tar` and containing non-ASCII file names, that is, file names having characters with the eight bit set, 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` compute checksums both ways, and accept any on read, so GNU tar can read Sun tapes even with their wrong checksums. GNU `tar` produces the standard checksum, however, raising incompatibilities with Sun. That is to say, GNU `tar` has not been modified to *produce* incorrect archives to be read by buggy `tar`'s. I've been told that more recent Sun `tar` now read standard archives, so maybe Sun did a similar patch, after all?

The story seems to be that when Sun first imported `tar` sources 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. When they later realized their mistake, they merely decided to stay compatible with it, and with themselves afterwards. Presumably, but I do not really know, HP-UX has chosen that their `tar` archives to be compatible with Sun's. The current standards do not favor Sun `tar` format. In any case, it now falls on the shoulders of SunOS and HP-UX users to get a `tar` able to read the good archives they receive.

8.2 Using Less Space through Compression

8.2.1 Creating and Reading Compressed Archives

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

-z
--gzip
--ungzip Filter the archive through `gzip`.

Some format parameters must be taken into consideration when modifying an archive: . Compressed archives cannot be modified.

You can use '`--gzip`' and '`--gunzip`' 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) record size. The default compression parameters are used; if you need to override them, avoid the *--gzip* (*--gunzip*, *--ungzip*, *-z*) option and run `gzip` explicitly. (Or set the '`GZIP`' environment variable.)

The *--gzip* (*--gunzip*, *--ungzip*, *-z*) option does not work with the *--multi-volume* (*-M*) option, or with the *--update* (*-u*), *--append* (*-r*), *--concatenate* (*--catenate*, *-A*), or *--delete* operations.

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* (*--gunzip*, *--ungzip*, *-z*).

--use-compress-program=prog
: Filter through *prog* (must accept '`-d`').

--compress (*--uncompress*, *-Z*) stores an archive in compressed format. This option is useful in saving time over networks and space in pipes, and when storage space is at a premium. *--compress* (*--uncompress*, *-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* (*--uncompress*, *-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* (*--uncompress*, *-Z*) option will not work in conjunction with the *--multi-volume* (*-M*) option or the *--append* (*-r*), *--update* (*-u*), *--append* (*-r*) and *--*

delete operations. See Section 4.2.1 [Operations], page 38, for more information on these operations.

If there is no compress utility available, `tar` will report an error. **Please note** that the `compress` program may be covered by a patent, and therefore we recommend you stop using it.

--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* (*--get*, *-x*), *--list* (*-t*) and *--compare* (*--diff*, *-d*) operations.

You can have archives be compressed by using the *--gzip* (*--gunzip*, *--ungzip*, *-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* (*--uncompress*, *-Z*) option. The GNU Project recommends you not use `compress`, because there is a patent covering the algorithm it uses. You could be sued for patent infringment merely by running `compress`.

I have one question, or maybe it's a suggestion if there isn't a way to do it now. I would like to use *--gzip* (*--gunzip*, *--ungzip*, *-z*), but I'd also like the output to be fed through a program like GNU `ecc` (actually, right now that's '`exactly`' what I'd like to use :-)), basically adding ECC protection on top of compression. It seems as if this should be quite easy to do, but I can't work out exactly how to go about it. Of course, I can pipe the standard output of `tar` through `ecc`, but then I lose (though I haven't started using it yet, I confess) the ability to have `tar` use `rmt` for it's I/O (I think).

I think the most straightforward thing would be to let me specify a general set of filters outboard of compression (preferably ordered, so the order can be automatically reversed on input operations, and with the options they require specifiable), but beggars shouldn't be choosers and anything you decide on would be fine with me.

By the way, I like `ecc` but if (as the comments say) it can't deal with loss of block sync, I'm tempted to throw some time at adding that capability. Supposing I were to actually do such a thing and get it (apparantly) working, do you accept contributed changes to utilities like that? (Leigh Clayton '`loc@soliton.com`', May 1995).

Isn't that exactly the role of the *--use-compress-prog=program* option? I never tried it myself, but I suspect you may want to write a *prog* script or program able to filter stdin to stdout to way you want. It should recognize the '`-d`' option, for when extraction is needed rather than creation.

It has been reported that if one writes compressed data (through the *--gzip* (*--gunzip*, *--ungzip*, *-z*) or *--compress* (*--uncompress*, *-Z*) options) to a DLT and tries to use the DLT compression mode, the data will actually get bigger and one will end up with less space on the tape.

8.2.2 Archiving Sparse Files

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

`-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.

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, actual disk 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.

A file is sparse if it contains blocks of zeros whose existence 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. See Section 2.4 [create], page 8, 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 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 never take more space on the media than the files take on disk (otherwise, archiving a disk filled with sparse files might take hundreds of tapes).

`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.

However, users should be well aware that at archive creation time, GNU `tar` still has to read whole disk file to locate the *holes*, and so, even if sparse files use little space on disk and in the archive, they may sometimes require inordinate amount of time for reading and examining all-zero blocks of a file. Although it works, it's painfully slow for a large (sparse) file, even though the resulting tar archive may be small. (One user reports that dumping a '`core`' file of over 400 megabytes, but with only about 3 megabytes of actual data, took about 9 minutes on a Sun Sparstation ELC, with full CPU utilisation.)

This reading is required in all cases and is not related to the fact the `--sparse` (`-S`) option is used or not, so by merely *not* using the option, you are not saving time[1].

Programs like `dump` do not have to read the entire file; by examining the file system directly, they can determine in advance exactly where the holes are and thus avoid reading through them. The only data it need read are the actual allocated data blocks. GNU `tar` uses a more portable and straightforward archiving approach, it would be fairly difficult that it does otherwise. Elizabeth Zwicky writes to '`comp.unix.internals`', on 1990-12-10:

> What I did say is that you cannot tell the difference between a hole and an equivalent number of nulls without reading raw blocks. `st_blocks` at best tells you how many holes there are; it doesn't tell you *where*. Just as programs may, conceivably, care what `st_blocks` is (care to name one that does?), they may also care where the holes are (I have no examples of this one either, but it's equally imaginable).
>
> I conclude from this that good archivers are not portable. One can arguably conclude that if you want a portable program, you can in good conscience restore files with as many holes as possible, since you can't get it right.

8.3 Handling File Attributes

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

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 (see Chapter 5 [Backups], page 51), but it is good enough for some purposes.

Handling of file attributes

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

`-m`
`--touch`
: 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`).

[1] Well! We should say the whole truth, here. When `--sparse` (`-S`) is selected while creating an archive, the current `tar` algorithm requires sparse files to be read twice, not once. We hope to develop a new archive format for saving sparse files in which one pass will be sufficient.

--same-owner

Create extracted files with the same ownership they have in the archive.

When using super-user at extraction time, ownership is always restored. So, this option is meaningful only for non-root users, when `tar` is executed on those systems able to give files away. This is considered as a security flaw by many people, at least because it makes quite difficult to correctly account users for the disk space they occupy. Also, the `suid` or `sgid` attributes of files are easily and silently lost when files are given away.

When writing an archive, `tar` writes the user id and user name separately. If it can't find a user name (because the user id is not in '`/etc/passwd`'), then it does not write one. When restoring, and doing a `chmod` like when you use *--same-permissions* (*--preserve-permissions*, *-p*) (), it tries to look the name (if one was written) up in '`/etc/passwd`'. If it fails, then it uses the user id stored in the archive instead.

--numeric-owner

The *--numeric-owner* option allows (ANSI) archives to be written without user/group name information or such information to be ignored when extracting. It effectively disables the generation and/or use of user/group name information. This option forces extraction using the numeric ids from the archive, ignoring the names.

This is useful in certain circumstances, when restoring a backup from an emergency floppy with different passwd/group files for example. It is otherwise impossible to extract files with the right ownerships if the password file in use during the extraction does not match the one belonging to the filesystem(s) being extracted. This occurs, for example, if you are restoring your files after a major crash and had booted from an emergency floppy with no password file or put your disk into another machine to do the restore.

The numeric ids are *always* saved into `tar` archives. The identifying names are added at create time when provided by the system, unless *--old-archive* (*-o*) is used. Numeric ids could be used when moving archives between a collection of machines using a centralized management for attribution of numeric ids to users and groups. This is often made through using the NIS capabilities.

When making a `tar` file for distribution to other sites, it is sometimes cleaner to use a single owner for all files in the distribution, and nicer to specify the write permission bits of the files as stored in the archive independently of their actual value on the file system. The way to prepare a clean distribution is usually to have some Makefile rule creating a directory, copying all needed files in that directory, then setting ownership and permissions as wanted (there are a lot of possible schemes), and only then making a `tar` archive out of this directory, before cleaning everything out. Of course, we could add a lot of options to GNU `tar` for fine tuning permissions and ownership. This is not the good way, I think. GNU `tar` is already crowded with options and moreover, the approach just explained gives you a great deal of control already.

-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*).

--preserve

Same as both *--same-permissions* (*--preserve-permissions*, *-p*) and *--same-order* (*--preserve-order*, *-s*).

The *--preserve* option has no equivalent short option name. It is equivalent to *--same-permissions* (*--preserve-permissions*, *-p*) plus *--same-order* (*--preserve-order*, *-s*).

8.4 The Standard Format

(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 GNU 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.

Archives are permitted to have more than one member with the same member 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, see Section 4.2.3 [update], page 41, and to learn more about having more than one archive member with the same name, see .

In addition to entries describing archive members, an archive may contain entries which `tar` itself uses to store information. See Section 9.7 [label], page 113, for an example of such an archive entry.

A `tar` archive file contains a series of blocks. Each block contains `BLOCKSIZE` 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 block which describes the file, followed by zero or more blocks which give the contents of the file. At the end of the archive file there may be a block filled with binary zeros as an end-of-file marker. A reasonable system should write a block of zeros at the end, but must not assume that such a block exists when reading an archive.

The blocks may be *blocked* for physical I/O operations. Each record of *n* blocks (where *n* is set by the `--blocking-factor=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 record. When writing an archive, the last record of blocks should be written at the full size, with blocks after the zero block containing all zeros. When reading an archive, a reasonable system should properly handle an archive whose last record is shorter than the rest, or which contains garbage records after a zero block.

The header block is defined in C as follows. In the GNU `tar` distribution, this is part of file '`src/tar.h`':

```
/* GNU tar Archive Format description.  */

/* If OLDGNU_COMPATIBILITY is not zero, tar produces archives which, by
   default, are readable by older versions of GNU tar.  This can be
   overriden by using --posix; in this case, POSIXLY_CORRECT in environment
   may be set for enforcing stricter conformance.  If OLDGNU_COMPATIBILITY
   is zero or undefined, tar will eventually produces archives which, by
   default, POSIX compatible; then either using --posix or defining
   POSIXLY_CORRECT enforces stricter conformance.

   This #define will disappear in a few years.  FP, June 1995.  */
#define OLDGNU_COMPATIBILITY 1

/*---------------------------------------------.
| 'tar' Header Block, from POSIX 1003.1-1990.  |
`---------------------------------------------*/

/* POSIX header.  */

struct posix_header
{                               /* byte offset */
  char name[100];               /*   0 */
  char mode[8];                 /* 100 */
  char uid[8];                  /* 108 */
  char gid[8];                  /* 116 */
  char size[12];                /* 124 */
  char mtime[12];               /* 136 */
  char chksum[8];               /* 148 */
  char typeflag;                /* 156 */
  char linkname[100];           /* 157 */
  char magic[6];                /* 257 */
  char version[2];              /* 263 */
  char uname[32];               /* 265 */
  char gname[32];               /* 297 */
  char devmajor[8];             /* 329 */
  char devminor[8];             /* 337 */
  char prefix[155];             /* 345 */
                                /* 500 */
};
```

```
#define TMAGIC   "ustar"        /* ustar and a null */
#define TMAGLEN  6
#define TVERSION "00"           /* 00 and no null */
#define TVERSLEN 2

/* Values used in typeflag field.  */
#define REGTYPE  '0'            /* regular file */
#define AREGTYPE '\0'           /* regular file */
#define LNKTYPE  '1'            /* link */
#define SYMTYPE  '2'            /* reserved */
#define CHRTYPE  '3'            /* character special */
#define BLKTYPE  '4'            /* block special */
#define DIRTYPE  '5'            /* directory */
#define FIFOTYPE '6'            /* FIFO special */
#define CONTTYPE '7'            /* reserved */

/* 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          /* reserved */
                                /* 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 */

/*-------------------------------------.
| 'tar' Header Block, GNU extensions.  |
`-------------------------------------*/

/* In GNU tar, SYMTYPE is for to symbolic links, and CONTTYPE is for
   contiguous files, so maybe disobeying the 'reserved' comment in POSIX
   header description.  I suspect these were meant to be used this way, and
   should not have really been 'reserved' in the published standards.  */

/* *BEWARE* *BEWARE* *BEWARE* that the following information is still
   boiling, and may change.  Even if the OLDGNU format description should be
   accurate, the so-called GNU format is not yet fully decided.  It is
   surely meant to use only extensions allowed by POSIX, but the sketch
   below repeats some ugliness from the OLDGNU format, which should rather
   go away.  Sparse files should be saved in such a way that they do *not*
   require two passes at archive creation time.  Huge files get some POSIX
   fields to overflow, alternate solutions have to be sought for this.  */
```

```
/* Descriptor for a single file hole.  */

struct sparse
{                               /* byte offset */
  char offset[12];              /*   0 */
  char numbytes[12];            /*  12 */
                                /*  24 */
};

/* Sparse files are not supported in POSIX ustar format.  For sparse files
   with a POSIX header, a GNU extra header is provided which holds overall
   sparse information and a few sparse descriptors.  When an old GNU header
   replaces both the POSIX header and the GNU extra header, it holds some
   sparse descriptors too.  Whether POSIX or not, if more sparse descriptors
   are still needed, they are put into as many successive sparse headers as
   necessary.  The following constants tell how many sparse descriptors fit
   in each kind of header able to hold them.  */

#define SPARSES_IN_EXTRA_HEADER  16
#define SPARSES_IN_OLDGNU_HEADER 4
#define SPARSES_IN_SPARSE_HEADER 21

/* The GNU extra header contains some information GNU tar needs, but not
   foreseen in POSIX header format.  It is only used after a POSIX header
   (and never with old GNU headers), and immediately follows this POSIX
   header, when typeflag is a letter rather than a digit, so signaling a GNU
   extension.  */

struct extra_header
{                               /* byte offset */
  char atime[12];               /*   0 */
  char ctime[12];               /*  12 */
  char offset[12];              /*  24 */
  char realsize[12];            /*  36 */
  char longnames[4];            /*  48 */
  char unused_pad1[68];         /*  52 */
  struct sparse sp[SPARSES_IN_EXTRA_HEADER];
                                /* 120 */
  char isextended;              /* 504 */
                                /* 505 */
};

/* Extension header for sparse files, used immediately after the GNU extra
   header, and used only if all sparse information cannot fit into that
   extra header.  There might even be many such extension headers, one after
   the other, until all sparse information has been recorded.  */

struct sparse_header
```

```
{                               /* byte offset */
  struct sparse sp[SPARSES_IN_SPARSE_HEADER];
                                /*   0 */
  char isextended;              /* 504 */
                                /* 505 */
};

/* The old GNU format header conflicts with POSIX format in such a way that
   POSIX archives may fool old GNU tar's, and POSIX tar's might well be
   fooled by old GNU tar archives.  An old GNU format header uses the space
   used by the prefix field in a POSIX header, and cumulates information
   normally found in a GNU extra header.  With an old GNU tar header, we
   never see any POSIX header nor GNU extra header.  Supplementary sparse
   headers are allowed, however.  */

struct oldgnu_header
{                               /* byte offset */
  char unused_pad1[345];        /*   0 */
  char atime[12];               /* 345 */
  char ctime[12];               /* 357 */
  char offset[12];              /* 369 */
  char longnames[4];            /* 381 */
  char unused_pad2;             /* 385 */
  struct sparse sp[SPARSES_IN_OLDGNU_HEADER];
                                /* 386 */
  char isextended;              /* 482 */
  char realsize[12];            /* 483 */
                                /* 495 */
};

/* OLDGNU_MAGIC uses both magic and version fields, which are contiguous.
   Found in an archive, it indicates an old GNU header format, which will be
   hopefully become obsolescent.  With OLDGNU_MAGIC, uname and gname are
   valid, though the header is not truly POSIX conforming.  */
#define OLDGNU_MAGIC "ustar  "  /* 7 chars and a null */

/* The standards committee allows only capital A through capital Z for
   user-defined expansion.  */

/* This is a dir entry that contains the names of files that were in the
   dir at the time the dump was made.  */
#define GNUTYPE_DUMPDIR 'D'

/* Identifies the *next* file on the tape as having a long linkname.  */
#define GNUTYPE_LONGLINK 'K'

/* Identifies the *next* file on the tape as having a long name.  */
#define GNUTYPE_LONGNAME 'L'
```

```
/* This is the continuation of a file that began on another volume.  */
#define GNUTYPE_MULTIVOL 'M'

/* For storing filenames that do not fit into the main header.  */
#define GNUTYPE_NAMES 'N'

/* This is for sparse files.  */
#define GNUTYPE_SPARSE 'S'

/* This file is a tape/volume header.  Ignore it on extraction.  */
#define GNUTYPE_VOLHDR 'V'

/*-------------------------------------.
| tar Header Block, overall structure.  |
`-------------------------------------*/

/* tar files are made in basic blocks of this size.  */
#define BLOCKSIZE 512

enum archive_format
{
  DEFAULT_FORMAT,             /* format to be decided later */
  V7_FORMAT,                  /* old V7 tar format */
  OLDGNU_FORMAT,              /* GNU format as per before tar 1.12 */
  POSIX_FORMAT,               /* restricted, pure POSIX format */
  GNU_FORMAT                  /* POSIX format with GNU extensions */
};

union block
{
  char buffer[BLOCKSIZE];
  struct posix_header header;
  struct extra_header extra_header;
  struct oldgnu_header oldgnu_header;
  struct sparse_header sparse_header;
};

/* End of Format description.  */
```

All characters in header blocks 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 block 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.

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. , 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 block. 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 block 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.

`REGTYPE`
`AREGTYPE`
: These flags represent a regular file. In order to be compatible with older versions of `tar`, a `typeflag` value of `AREGTYPE` should be silently recognized as a regular file. New archives should be created using `REGTYPE`. Also, for backward compatibility, `tar` treats a regular file whose name ends with a slash as a directory.

`LNKTYPE`
: 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.

`SYMTYPE`
: This represents a symbolic link to another file. The linked-to name is specified in the `linkname` field with a trailing null.

`CHRTYPE`
`BLKTYPE`
: 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.

`DIRTYPE`
: 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.

`FIFOTYPE`
: This specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence of this file and not its contents.

`CONTTYPE`
: 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.

For references, see ISO/IEC 9945-1:1990 or IEEE Std 1003.1-1990, pages 169-173 (section 10.1) for *Archive/Interchange File Format*; and IEEE Std 1003.2-1992, pages 380-388 (section 4.48) and pages 936-940 (section E.4.48) for *pax - Portable archive interchange*.

8.5 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.

`GNUTYPE_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.

`GNUTYPE_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.

`GNUTYPE_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.

`GNUTYPE_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.

8.6 Comparison of `tar` and `cpio`

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

The `cpio` archive formats, like `tar`, do have maximum pathname lengths. The binary and old ASCII formats have a max path length of 256, and the new ASCII and CRC ASCII formats have a max path length of 1024. GNU `cpio` can read and write archives with arbitrary pathname lengths, but other `cpio` implementations may crash unexplainedly trying to read them.

`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 present 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 record 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 present 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).

9 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.

9.1 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:file name`'. If the *hostname* contains an *at* sign (`@`), it is treated as '`user@hostname:file name`'. 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 records, 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. See Section 9.6.1 [Multi-Volume Archives], page 112.

`-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`).

9.2 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 `--absolute-names` (`-P`) 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 blocking factor 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 record 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` (`--catenate`, `-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.

9.3 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

9.4 Blocking

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

Block and *record* terminology is rather confused, and it is also confusing to the expert reader. On the other hand, readers who are new to the field have a fresh mind, and they may safely skip the next two paragraphs, as the remainder of this manual uses those two terms in a quite consistent way.

John Gilmore, the writer of the public domain `tar` from which GNU `tar` was originally derived, wrote (June 1995):

> The nomenclature of tape drives comes from IBM, where I believe they were invented for the IBM 650 or so. On IBM mainframes, what is recorded on tape are tape blocks. The logical organization of data is into records. There are various ways of putting records into blocks, including `F` (fixed sized records), `V` (variable sized records), `FB` (fixed blocked: fixed size records, *n* to a block), `VB` (variable size records, *n* to a block), `VSB` (variable spanned blocked: variable sized records that can occupy more than one block), etc. The `JCL` '`DD RECFORM=`' parameter specified this to the operating system.
>
> The Unix man page on `tar` was totally confused about this. When I wrote `PD TAR`, I used the historically correct terminology (`tar` writes data records, which are grouped into blocks). It appears that the bogus terminology made it into POSIX (no surprise here), and now François has migrated that terminology back into the source code too.

The term *physical block* means the basic transfer chunk from or to a device, after which reading or writing may stop without anything being lost. In this manual, the term *block* usually refers to a disk physical block, *assuming* that each disk block is 512 bytes in length. It is true that some disk devices have different physical blocks, but `tar` ignore these differences in its own format, which is meant to be portable, so a `tar` block is always 512 bytes in length, and *block* always mean a `tar` block. The term *logical block* often represents the basic chunk of allocation of many disk blocks as a single entity, which the operating system treats somewhat atomically; this concept is only barely used in GNU `tar`.

The term *physical record* is another way to speak of a physical block, those two terms are somewhat interchangeable. In this manual, the term *record* usually refers to a tape physical block, *assuming* that the `tar` archive is kept on magnetic tape. It is true that archives may be put on disk or used with pipes, but nevertheless, `tar` tries to read and write the archive one *record* at a time, whatever the medium in use. One record is made up of an integral number of blocks, and this operation of putting many disk blocks into a single tape block is called *reblocking*, or more simply, *blocking*. The term *logical record* refers to the logical organization of many characters into something meaningful to the application. The term *unit record* describes a small set of characters which are transmitted whole to or by the application, and often refers to a line of text. Those two last terms are unrelated to what we call a *record* in GNU `tar`.

When writing to tapes, `tar` writes the contents of the archive in chunks known as *records*. To change the default blocking factor, use the `--blocking-factor=512-size` (`-b 512-size`) option. Each record will then be composed of *512-size* blocks. (Each `tar` block is 512 bytes. See Section 8.4 [Standard], page 87.) Each file written to the archive uses at least one full record. As a result, using a larger record size can result in more wasted space for small files. On the other hand, a larger record size can often be read and written much more efficiently.

Further complicating the problem is that some tape drives ignore the blocking entirely. For these, a larger record 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.

When reading an archive, `tar` can usually figure out the record size on itself. When this is the case, and a non-standard record 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 record size itself. On most of those, you can specify a blocking factor (with `--blocking-factor=512-size` (`-b 512-size`)) larger than the actual blocking factor, and then use the `--read-full-records` (`-B`) option. (If you specify a blocking factor with `--blocking-factor=512-size` (`-b 512-size`) and don't use the `--read-full-records` (`-B`) option, then `tar` will not attempt to figure out the recording size itself.) On some devices, you must always specify the record size exactly with `--blocking-factor=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.

`tar` blocks are all fixed size (512 bytes), and its scheme for putting them into records is to put a whole number of them (one or more) into each record. `tar` records are all the same size; at the end of the file there's a block containing all zeros, which is how you tell that the remainder of the last record(s) are garbage.

In a standard `tar` file (no options), the block size is 512 and the record size is 10240, for a blocking factor of 20. What the `--blocking-factor=`*512-size* (`-b` *512-size*) option does is sets the blocking factor, changing the record size while leaving the block size at 512 bytes. 20 was fine for ancient 800 or 1600 bpi reel-to-reel tape drives; most tape drives these days prefer much bigger records in order to stream and not waste tape. When writing tapes for myself, some tend to use a factor of the order of 2048, say, giving a record size of around one megabyte.

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 record sizes, limited only by the amount of virtual memory or the physical characteristics of the tape device.

9.4.1 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 `--blocking-factor=`*512-size* (`-b` *512-size*) option specified (see Section 9.4.2 [Blocking Factor], page 104), you must specify that blocking-factor when operating on the archive. See Chapter 8 [Formats], page 77, for other examples of format parameter considerations.

9.4.2 The Blocking Factor of an Archive

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

The data in an archive is grouped into blocks, which are 512 bytes. Blocks are read and written in whole number multiples called *records*. The number of blocks in a record (ie. the size of a record in units of 512 bytes) is called the *blocking factor*. The `--blocking-factor=`*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.

Records are separated by gaps, which waste space on the archive media. If you are archiving on magnetic tape, using a larger blocking factor (and therefore larger records) 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 record. In general, the ideal record size depends on the size of the inter-record gaps on the tape you are using, and the average size of the files you are archiving. See Section 2.4 [create], page 8, for information on writing archives.

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 record size of the device containing the archive, or by the amount of available virtual memory.

Also, on some systems, not using adequate blocking factors, as sometimes imposed by the device drivers, may yield unexpected diagnostics. For example, this has been reported:

```
Cannot write to /dev/dlt: Invalid argument
```

In such cases, it sometimes happen that the `tar` bundled by the system is aware of block size idiosyncrasies, while GNU `tar` requires an explicit specification for the block size, which it cannot guess. This yields some people to consider GNU `tar` is misbehaving, because by comparison, *the bundle* `tar` *works OK*. Adding `-b 256`, for example, might resolve the problem.

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 record 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-records` (`-B`) option while specifying a blocking factor larger then the blocking factor of the archive (ie. '`tar --extract --read-full-records --blocking-factor=300`'. See Section 2.5 [list], page 12, for more information on the `--list` (`-t`) operation. See Section 4.3.1 [Reading], page 45, for a more detailed explanation of that option.

`--blocking-factor=`*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*
`--blocking-factor=`*blocks*

Set record 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 records of $block * 512$ bytes. This is true even when the archive is compressed. Some devices requires that all write operations be a multiple of a certain size, and so, `tar` pads the archive out to the next record boundary.

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 records 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.

Apparently, Exabyte drives have a physical block size of 8K bytes. If we choose our blocksize as a multiple of 8k bytes, then the problem seems to dissapper. Id est, we are using block size of 112 right now, and we haven't had the problem since we switched...

With GNU `tar` the blocking factor is limited only by the maximum record size of the device containing the archive, or by the amount of available virtual memory.

However, deblocking or reblocking is virtually avoided in a special case which often occurs in practice, but which requires all the following conditions to be simultaneously true:

- the archive is subject to a compression option,
- the archive is not handled through standard input or output, nor redirected nor piped,
- the archive is directly handled to a local disk, instead of any special device,
- `--blocking-factor=`*512-size* (`-b` *512-size*) is not explicitely specified on the `tar` invocation.

In previous versions of GNU `tar`, the '`--compress-block`' option (or even older: '`--block-compress`') was necessary to reblock compressed archives. It is now a dummy option just asking not to be used, and otherwise ignored. If the output goes directly to a local disk, and not through stdout, then the last write is not extended to a full record size. Otherwise, reblocking occurs. Here are a few other remarks on this topic:

- `gzip` will complain about trailing garbage if asked to uncompress a compressed archive on tape, there is an option to turn the message off, but it breaks the regularity of simply having to use '*prog* `-d`' for decompression. It would be nice if gzip was silently ignoring any number of trailing zeros. I'll ask Jean-loup Gailly, by sending a copy of this message to him.
- `compress` does not show this problem, but as Jean-loup pointed out to Michael, '`compress -d`' silently adds garbage after the result of decompression, which tar ignores because it already recognized its end-of-file indicator. So this bug may be safely ignored.
- '`gzip -d -q`' will be silent about the trailing zeros indeed, but will still return an exit status of 2 which tar reports in turn. `tar` might ignore the exit status returned, but I hate doing that, as it weakens the protection `tar` offers users against other possible problems at decompression time. If `gzip` was silently skipping trailing zeros *and* also avoiding setting the exit status in this innocuous case, that would solve this situation.
- `tar` should become more solid at not stopping to read a pipe at the first null block encountered. This inelegantly breaks the pipe. `tar` should rather drain the pipe out before exiting itself.

-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-records

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

If *--read-full-records* (*-B*) is used, `tar` will not panic if an attempt to read a record from the archive does not return a full record. Instead, `tar` will keep reading until it has obtained a full record.

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 record from the pipe.

This option is also useful with the commands for updating an archive.

Tape blocking

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 tape driver starts reading a record, the record 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 record. Further, if there is a reading error on a huge record, 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. We were told that for some DLT drives, the blocking should be a multiple of 4Kb, preferably 64Kb (*-b 128*) or 256 for decent performance. 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.

I might also use '`--number-blocks`' instead of '`--block-number`', so '`--block`' will then expand to '`--blocking-factor`' unambiguously.

9.5 Many Archives on One Tape

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. See Section 9.4 [Blocking], page 102.

9.5.1 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 records "rrrr"'s, and tape marks as "*"'s, a tape might look like the following:

```
rrrr*rrrrrr*rrrrr*rr*rrrrr**-------------------------
```

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. .). See Section 9.5.2 [mt], page 109, 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:

```
rrrr*rrrrrr*rrrrr*rr*rrrrr*rrr*rrrr**----------------
```

9.5.2 The `mt` Utility

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

See Section 9.4.2 [Blocking Factor], page 104.

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.

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:

`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`
`rewoffl` Rewinds the tape and takes the tape device off-line. (Ignores *number*).

`status` Prints status information about the tape unit.

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.

If you use `--extract` (`--get`, `-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. See Section 9.7 [label], page 113. . '`tar --list --label`' will cause `tar` to print the label.

9.6 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.)

GNU `tar` multi-volume archives do not use a truly portable format. You need GNU `tar` at both end to process them properly.

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* (*`--new-volume-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* (*`--new-volume-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. This option selects *`--multi-volume`* (*`-M`*) automatically. The *size* argument should then be the usable size of the tape. But for many devices, and floppy disks in particular, this option is never required for real, as far as we know.

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 be an unexisting file to be created, or else, a file already containing 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, as per Section 9.7 [label], page 113, 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* (*`--new-volume-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.

For example, let's presume someone has two tape drives on a system named '`/dev/tape0`' and '`/dev/tape1`'. For having GNU `tar` to switch to the second drive when it needs to write the second tape, and then back to the first tape, etc., just do either of:

```
$ tar --create --multi-volume --file=/dev/tape0 --file=/dev/tape1 files
$ tar cMff /dev/tape0 /dev/tape1 files
```

9.6.1 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 (see Section 2.4 [create], page 8). 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` (`--get`, `-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. See Section 2.6.1 [extracting archives], page 14, for more information about extracting archives.

`--info-script=`*script-name* (`--new-volume-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*. For example, this option can be used to eject cassettes, or 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.

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*) (see Section 9.7 [label], page 113) 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` (`--catenate`, `-A`) operation.

`--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`).

Beware that there is *no* real standard about the proper way, for a `tar` archive, to span volume boundaries. If you have a multi-volume created by some vendor's `tar`, there is almost no chance you could read all the volumes with GNU `tar`. The converse is also true: you may not expect multi-volume archives created by GNU `tar` to be fully recovered by vendor's `tar`. Since there is little chance that, in mixed system configurations, some vendor's `tar` will work on another vendor's machine, and there is a great chance that GNU `tar` will work on most of them, your best bet is to install GNU `tar` on all machines between which you know exchange of files is possible.

9.6.2 Tape Files

(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 block 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`) (), 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. See Section 9.7 [label], page 113.

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, see Section 9.5.2 [mt], page 109.

People seem to often do:

`--label="`*some-prefix* `` `date + ``*some-format*`` `" ``

or such, for pushing a common date in all volumes or an archive set.

9.7 Including a Label in the Archive

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

`-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.

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. , for information on creating multiple volume archives.

If you list or 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 list nor extract the archive. In those cases, *archive-label* argument is interpreted as a globbing-style pattern which must match the actual magnetic volume label. See Section 6.4 [exclude], page 63, for a precise description of how match is attempted[1]. If the switch `--multi-volume` (`-M`) is being used, the volume label matcher will also suffix *archive-label* by '` Volume [1-9]*`' if the initial match fails, before giving up. Since the volume numbering is automatically added in labels at creation time, it sounded logical to equally help the user taking care of it when the archive is being read.

The `--label=`*archive-label* (`-V` *archive-label*) was once called '`--volume`', but is not available under that name anymore.

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 1992-03-07 12:01 iamalabel--Volume Header--
-rw-rw-rw- ringo user 40 1990-05-21 13:30 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` (`--get`, `-x`) option.

To get a common information on all tapes of a series, use the `--label=`*archive-label* (`-V` *archive-label*) option. For having this information different in each series created through a single script used on a regular basis, just manage to get some date string as part of the label. For example:

```
$ tar cfMV /dev/tape "Daily backup for `date +%Y-%m-%d`"
$ tar --create --file=/dev/tape --multi-volume \
     --volume="Daily backup for `date +%Y-%m-%d`"
```

[1] Previous versions of `tar` used full regular expression matching, or before that, only exact string matching, instead of wildcard matchers. We decided for the sake of simplicity to use a uniform matching device through `tar`.

Also note that each label has its own date and time, which corresponds to when GNU `tar` initially attempted to write it, often soon after the operator launches `tar` or types the carriage return telling that the next tape is ready. Comparing date labels does give an idea of tape throughput only if the delays for rewinding tapes and the operator switching them were negligible, which is ususally not the case.

9.8 Verifying Data as It is Stored

-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.

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.

One can explicitely compare an already made archive with the file system by using the *--compare* (*--diff*, *-d*) option, instead of using the more automatic *--verify* (*-W*) option. See Section 4.2.6 [compare], page 44.

Note that these two options have a slightly different intent. The *--compare* (*--diff*, *-d*) option how identical are the logical contents of some archive with what is on your disks, while the *--verify* (*-W*) option is really for checking if the physical contents agree and if the recording media itself is of dependable quality. So, for the *--verify* (*-W*) operation, `tar` tries to defeat all in-memory cache pertaining to the archive, while it lets the speed optimization undisturbed for the *--compare* (*--diff*, *-d*) option. If you nevertheless use *--compare* (*--diff*, *-d*) for media verification, you may have to defeat the in-memory cache yourself, maybe by opening and reclosing the door latch of your recording unit, forcing some doubt in your operating system about the fact this is really the same volume as the one just written or read.

The *--verify* (*-W*) option would not be necessary if drivers were indeed able to detect dependably all write failures. This sometimes require many magnetic heads, some able to read after the writes occurred. One would not say that drivers unable to detect all cases are necessarily flawed, as long as programming is concerned.

9.9 Write Protection

Almost all tapes and diskettes, and in a few rare cases, even 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.

Index

N

O

P

R

S

T

U

V

W

Y

Short Contents

Table of Contents