tar

The GNU tape archive

by Jay Fenlason

DRAFT!

14 August 1991

Preface

This manual describes the GNU tape archiver, `tar`, and how you can use it to store copies of a file or a group of files in an *archive*. This archive may be written directly to a magnetic tape or other storage medium, stored as a file, or sent through a pipe to another program. `tar` can also be used to add files to an already existing archive, list the files in an archive, or extract the files in the archive.

GNU `tar` was written by John Gilmore, and modified by many people. The GNU enhancements were written by Jay Fenlason.

1. The Uses of Tape Archives

The tape archiver `tar` allows you to store many files in an *archive file* or *tar file* which describes the names and contents of the constituent files. Later you can extract some or all of these files from the archive.

Tar files are not restricted to magnetic tapes. The `tar` program can equally well use an ordinary file, or a pipe, or any device, as the archive. But they were originally designed for use with magnetic tapes, and that is how the name “tar” came about.

Archive files can be used for transporting a group of files from one system to another: put all relevant files into an archive on one computer system, transfer the archive to another, and extract the contents there. The basic transfer medium might be magnetic tape, Internet FTP, or even electronic mail (though you must encode the archive with `uuencode` in order to transport it properly by mail). Both machines do not have to use the same operating system, as long as they both support the `tar` program.

A magnetic tape can store several files in sequence, but has no names for them, just relative position on the tape. A tar file or something like it is the only way to store several files on one tape and retain their names. Even when the basic transfer mechanism can keep track of names, as FTP can, the nuisance of handling multiple files, directories, and multiple links, may make a tar file a much easier method.

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

Piping one `tar` to another is an easy way to copy a directory’s contents from one disk to another, while preserving the dates, modes, owners and link-structure of all the files therein.

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

2. The Different Operations `tar` Can Perform

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

The command must always be in the first argument to `tar`. This argument can also contain options (see Chapter 3 [Options], page 6). 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 '`-c`' command to create an archive.

In addition, a set of long-named options are provided which can be used instead of or inter-mixed with the single-letter flags. The long-named options are meant to be easy to remember and logical, while the single letter flags may not always be. Long-named options are preceded by a '`+`'. In the list of single-letter commands below, each corresponding long-named option is listed next to it, in parentheses.

The remaining arguments to `tar` are either options, if they start with '`-`' or '`+`', or files to operate on.

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

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

Here is a list of all the `tar` commands:

'`-c`'
'`+create`' This command tells `tar` to create a new archive that contains the file(s) specified on the command line. If you don't specify files, all the files in the current directory are used.

If the archive file already exists, it is overwritten; the old contents are lost.

'`-d`'
'`+compare`'
'`+diff`' This command causes `tar` to compare the archive with the files in the file system. It

will report differences in file size, mode, owner, and contents. If a file exists in the archive, but not in the file system, `tar` will report this.

If you specify file names, those files are compared with the tape and they must all exist in the archive. If you don't specify files, all the files in the archive are compared.

'`-r`'

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

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

'`-t`'

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

'`-u`'

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

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

'`-x`'

'`+extract`'

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

'`-A`'

'`+catenate`'

'`+concatenate`'

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

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

'`-D`'

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

3. Options That Change How `tar` Works

Options may be specified as individual arguments starting with '`-`'. In this case, if the option wants an argument (as does, for example, '`-f`') then the argument should come after the option, separated from it by a space. All options are optional. Some options make sense with any command, while others are meaningful only with particular commands.

3.1 Options That Are Always Meaningful

'`-b` *number*'
'`+block-size` *number*'

This option is used to specify a *blocking factor* for the archive. When reading or writing the archive, `tar`, will do reads and writes of the archive in blocks of *number**512 bytes.

The default blocking factor is set when `tar` is compiled, and is typically 20.

Blocking factors larger than 20 cannot be read by very old versions of `tar`, or by some newer versions of `tar` running on old machines with small address spaces.

With a magnetic tape, larger blocks give faster throughput and fit more data on a tape (because there are fewer inter-record gaps). If the archive is in a disk file or a pipe, you may want to specify a smaller blocking factor, since a large one will result in a large number of null bytes at the end of the archive.

When writing cartridge or other streaming tapes, a much larger blocking factor (say 126 or more) will greatly increase performance. However, you must specify the same blocking factor when reading or updating the archive.

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

'`-f` *filename*'
'`+file` *filename*'

This option is used to specify the file name of the archive `tar` works on.

If this option is not given, but the environment variable `TAPE` is set, its value is used; otherwise, a default archive name (which was picked when `tar` was compiled) is used. The default is normally set up to be the "first" tape drive or other transportable I/O medium on the system.

If the filename is '`-`', `tar` reads the archive from standard input (when listing or extracting), or writes it to standard output (when creating). If the '`-`' filename 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 filename contains '`:/dev/`', it is interpreted as '`hostname:filename`'. If the *hostname* contains an "at" sign ('`@`'), it is treated as '`user@hostname:filename`'. In either case, `tar` will invoke the command `rsh` (or `remsh`) to start up an `/etc/rmt` on the remote machine. If you give an alternate login name, it will be given to the `rsh`. Naturally, the remote machine must have a copy of '`/etc/rmt`'. `/etc/rmt` is free software from the University of California, and a copy of the source code can be found with the sources for `tar`. `/etc/rmt` will have to be modified to run on non-BSD4.3 systems.

'`-C` *dir*'

'`+directory` *dir*'

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

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

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

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

Contrast this with the command

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

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

'`-M`'

'`+multi-volume`'

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.

When this option is used, `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 '`-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 ('`-xM`') starting on or before the volume where the file begins.

'`-N` *date*'

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

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

  The date is parsed using `getdate`.

'-R'
'+record-number'
: If '-R' is used, `tar` prints, along with every message it would normally produce, the record number within the archive where the message occurred. This option is especially useful when reading damaged archives, since it helps pinpoint the damaged sections.

  This can also be useful when making a log of a file-system backup tape, since the results allow you to find the file you want to retrieve on several backup tapes and choose the tape where the file appears earliest (closest to the front of the tape).

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

'-v'
'+verbose'
: This option causes `tar` to be verbose about the actions it is taking.

  Normally, the '-t' command to list an archive prints just the file names (one per line) and the other commands are silent.

  '-tv' prints a full line of information about each file, like the output of 'ls -l'. '-v' with any other command (aside from '-t') prints just the name of each file operated on.

  The output from '-v' appears on the standard output except when creating or updating an archive to the standard output, in which case the output from '-v' is sent to the standard error.

'+version'
: This option causes `tar` to print out its version number to the standard error. It has no equivalent short option name.

'-w'
'+interactive'
: This option causes `tar` to print a message for each action it intends to take, and ask for confirmation on the terminal. To confirm, you must type a line of input. If your input line begins with 'y', the action is performed, otherwise it is skipped.

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.

If `tar` is reading the archive from the standard input, `tar` will open the file '`/dev/tty`' to ask for confirmation on.

'`-X` *file*'
'`+exclude` *file*'
: This option causes `tar` to read a list of filename regular expressions, one per line, from the file *file*; `tar` will ignore files with those names. Thus if `tar` is called as '`tar -c -X foo .`' and the file '`foo`' contains '`*.o`' none of the files whose names end in '`.o`' in the current directory (or any subdirectory of the current directory) will be added to the archive. Multiple `-X` options may be given.

'`-z`'
'`-Z`'
'`+compress`'
'`+uncompress`'
: The archive should be compressed as it is written, or decompressed as it is read, using the `compress` program. This option works on physical devices (tape drives, etc.) and remote files as well as on normal files; data to or from such devices or remote files is reblocked by another copy of the `tar` program to enforce the specified (or default) block size. The default compression parameters are used; if you need to override them, avoid the '`-z`' option and run `compress` explicitly.

If the '`-z`' option is given twice, or the '`+compress-block`' option is used, `tar` will pad the archive out to the next block boundry (see Section 3.1 [General Options], page 6). This may be useful with some devices that require that all write operations be a multiple of a certain size.

Note that the '`-z`' option will not work with the '`-M`' option, or with the '`-u`', '`-r`', '`-A`', or '`-D`' commands.

3.2 Options for Creating Or Updating an Archive

These options are used to control which files `tar` puts in an archive, or to control the format the archive is written in (see Section 8.2 [Format], page 26).

Except as noted below, these options are useful with the '`-c`', '`-r`', '`-u`', '`-A`', and '`-D`' commands. Also note that the '`-B`' option, (see Section 3.3 [Extraction Options], page 11), is also useful with the '`-r`', '`-u`', '`-A`', and '`-D`' commands.

'`-G`'

'`+incremental`'
: This option should only be used when creating an incremental backup of a filesystem. When the '`-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.

'`-h`'
'`+dereference`'
: If '`-h`' 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. If the linked-to file is archived again, an entire second copy of it will be archived, instead of a link. This could be considered a bug.

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

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

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

'`-o`'
'`+old-archive`'
'`+old`'
'`+portability`'
: This option causes `tar` to write an old format archive, which does not include information about directories, pipes, fifos, contiguous files, or device files, and specifies file ownership by numeric user- and group-ids rather than by user and group names. 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. When updating an archive, do not use '`-o`' unless the archive was created with the '`-o`' option.

'`-S`'
'`+sparse`'
: This option causes all files to be put in the archive to be tested for sparseness, and handled specially if they are. The '`-S`' option is useful when many dbm files, for example, are being backed up, and running out of space on the tape is an issue. 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.

'`-V` *name*'
'`+volume` *name*'
: This option causes `tar` to write out a *volume header* at the beginning of the archive. If '`-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.

'`-W`'
'`+verify`'
: 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.

3.3 Options for Listing Or Extracting Files

The options in this section are meaningful with the '`-x`' command. Unless otherwise stated, they are also meaningful with the '`-t`' command.

'`-B`'
'`+read-full-blocks`'
: If '`-B`' is used, `tar` will not panic if an attempt to read a block from the archive does not return a full block. Instead, `tar` will keep reading until it has obtained a full block.

This option is turned on by default when `tar` is reading an archive from standard input, or from a remote machine. This is because on BSD Unix systems, a read of a pipe will return however much happens to be in the pipe, even if it is less than `tar` requested. If this option was not used, `tar` would fail as soon as it read an incomplete block from the pipe.

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

'`-G`'
'`+incremental`'
: The '`-G`' option means the archive is an incremental backup. Its meaning depends on the command that it modifies.

If the '`-G`' option is used with '`-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 filename 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 filename 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 '-G' option is used with '-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 '-G', the file system will probably fill up with files that shouldn't exist any more.

'-i'
'+ignore-zeros'

The '-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.

'-k'
'+keep-old-files'

The '-k' option prevents tar from over-writing existing files with files with the same name from the archive.

The '-k' option is meaningless with '-t'.

'-K *filename*'
'+starting-file *filename*'

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

'-m'
'+modification-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 '-t'.

'-O'

'+to-stdout'
: 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 '-t'.

'-p'
'+same-permissions'
'+preserve-permissions'
: 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 '-t'.

'-P'
'+absolute-paths'
: This option should be used when the absolute pathname of a file should be preserved in the archive. `tar` normally strips the leading '/' from the name of the file, thus making /usr/foo/bar/baz into usr/foo/bar/baz. Using the '-P' option keeps the pathname intact, and is useful in that it is not necessary to change to the root directory when extracting files.

'-s'
'+same-order'
'+preserve-order'
: This option tells `tar` that the list of filenames to be listed or extracted is sorted in the same order as the files in the archive. This allows a large list of names to be used, even on a small machine that would not otherwise be able to hold all the names in memory at the same time. Such a sorted list can easily be created by running 'tar -t' on the archive and editing its output.

  This option is probably never needed on modern computer systems.

'+preserve'
: The '+preserve' option has no equivalent short option name. It is equivalent to '-p' plus '-s'.

3.4 Old Syntax for Options

For compatibility with Unix `tar`, the first argument can contain option letters in addition to the command letter; for example, 'tar cv' specifies the option '-v' in addition to the command '-c'. The first argument to GNU `tar` is always treated as command and option letters even if it doesn't start with '-'.

Some options need their own arguments; for example, '-f' is followed by the name of the archive file. When the option is given separately, its argument follows it, as is usual for Unix programs. For example:

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

When options that need arguments are given together with the command, all the associated arguments follow, in the same order as the options. Thus, the example above could also be written in the old style as follows:

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

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

The long-named options can be used instead of the single-letter flags. They are meant to be obvious and easy to remember, possibly more so than their corresponding single-letter options. The above example using long-named options would look like this:

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

4. Using tar to Perform Full Dumps

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.

You will want to use the '-V' 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 '-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 '-l' option to prevent tar from crossing filesystem boundaries when storing (sub)directories.

The '-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 -W (verify) 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.

5. Using tar to Perform Incremental Dumps

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 (semi-)automatically late at night when people are least likely to be using the machines. This script dumps several filesystems from several machines at once (by using a network-filesystem). 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.

6. Common Problems Using tar

Unless you use the -P option, GNU tar will not allow you to create an archive that contains absolute pathnames. (An absolute pathname is one that begins with a '/'.) 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 pathnames as if the pathnames were not absolute. If the archive contained a file '/usr/bin/computoy', GNU tar would extract the file to 'usr/bin/computoy' in 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 '-P' option, or use the command 'tar -C / ...'.

Some versions of UNIX, (Ultrix 3.1 is know to have this problem) can claim that a short write near the end of a tape succeeded, when it actually failed. This will result in the -M option not working correctly. The best workaround at the moment is to use a significantly larger blocksize than the default 20.

In order to update an archive, tar must be able to backspace the archive in order to re-read or re-write a block that was just read (or written). This is currently possible only on two kinds of files: normal disk files (or any other file that can be backspaced with lseek()), and industry-standard 9-track magnetic tape (or any other kind of tape that can be backspaced with ioctl(...,MTIOCTOP,...)).

This means that the '-r', '-u', '-A', and '-D' 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 '-M', '-V', and '-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 '-G' option.

7. 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`'.

The remote tape server may need to be modified in order to run on a non-4.3BSD system.

8. The Format of a tar Archive

This chapter is based heavily on John Gilmore's *tar*(5) manual page for the public domain `tar` that GNU `tar` is based on.

8.1 The Standard Format

A *tar tape* or file contains a series of records. Each record contains `RECORDSIZE` bytes. Although this format may be thought of as being on magnetic tape, other media are often used.

Each file archived is represented by a header record which describes the file, followed by zero or more records which give the contents of the file. At the end of the archive file there may be a record filled with binary zeros as an end-of-file marker. A reasonable system should write a record of zeros at the end, but must not assume that such a record exists when reading an archive.

The records may be *blocked* for physical I/O operations. Each block of *N* records (where *N* is set by the '`-b`' option to `tar`) is written with a single `write()` operation. On magnetic tapes, the result of such a write is a single tape record. When writing an archive, the last block of records should be written at the full size, with records after the zero record containing all zeroes. When reading an archive, a reasonable system should properly handle an archive whose last block is shorter than the rest, or which contains garbage records after a zero record.

The header record is defined in C as follows:

```
/*
 * Standard Archive Format - Standard TAR - USTAR
 */
#define RECORDSIZE   512
#define NAMSIZ       100
#define TUNMLEN       32
#define TGNMLEN       32
#define SPARSE_EXT_HDR 21
#define SPARSE_IN_HDR 4

struct sparse {
char offset[12];
char numbytes[12];
};

union record {
```

```
    char        charptr[RECORDSIZE];
    struct header {
        char    name[NAMSIZ];
        char    mode[8];
        char    uid[8];
        char    gid[8];
        char    size[12];
        char    mtime[12];
        char    chksum[8];
        char    linkflag;
        char    linkname[NAMSIZ];
        char    magic[8];
        char    uname[TUNMLEN];
        char    gname[TGNMLEN];
        char    devmajor[8];
        char    devminor[8];
/* these following fields were added by JF for gnu */
/* and are NOT standard */
char atime[12];
char ctime[12];
char offset[12];
char longnames[4];
/* the next three fields were added by JK to deal with
     shrinking down sparse files */
struct sparse sp[SPARSE_IN_HDR];
char isextended;
char ending_blanks[12]; /* number of nulls at the
   end of the file, if any */
    } header;

    struct extended_header {
struct sparse sp[21];
char isextended;
    } ext_hdr;

};

/* The checksum field is filled with this while the checksum is computed. */
#define    CHKBLANKS    "        "        /* 8 blanks, no null */

/* The magic field is filled with this if uname and gname are valid. */
#define    TMAGIC    "ustar  "        /* 7 chars and a null */

/* The magic field is filled with this if this is a GNU format dump entry */
#define    GNUMAGIC  "GNUtar "        /* 7 chars and a null */

/* The linkflag defines the type of file */
#define  LF_OLDNORMAL '\0'       /* Normal disk file, Unix compatible */
#define  LF_NORMAL    '0'        /* Normal disk file */
#define  LF_LINK      '1'        /* Link to previously dumped file */
```

```
#define  LF_SYMLINK   '2'        /* Symbolic link */
#define  LF_CHR       '3'        /* Character special file */
#define  LF_BLK       '4'        /* Block special file */
#define  LF_DIR       '5'        /* Directory */
#define  LF_FIFO      '6'        /* FIFO special file */
#define  LF_CONTIG    '7'        /* Contiguous file */

/* Further link types which were defined later. */
#define LF_DUMPDIR 'D' /* This is a dir entry that contains
   the names of files that were in
   the dir at the time the dump
   was made */
#define LF_MULTIVOL 'M' /* This is the continuation
   of a file that began on another
   volume */
#define LF_SPARSE 'S' /* This is for sparse files */
#define LF_VOLHDR 'V' /* This file is a tape/volume header */
/* Ignore it on extraction */

/* Bits used in the mode field - values in octal */
#define  TSUID    04000       /* Set UID on execution */
#define  TSGID    02000       /* Set GID on execution */
#define  TSVTX    01000       /* Save text (sticky bit) */

/* File permissions */
#define  TUREAD   00400       /* read by owner */
#define  TUWRITE  00200       /* write by owner */
#define  TUEXEC   00100       /* execute/search by owner */
#define  TGREAD   00040       /* read by group */
#define  TGWRITE  00020       /* write by group */
#define  TGEXEC   00010       /* execute/search by group */
#define  TOREAD   00004       /* read by other */
#define  TOWRITE  00002       /* write by other */
#define  TOEXEC   00001       /* execute/search by other */
```

All characters in header records are represented by using 8-bit characters in the local variant of ASCII. Each field within the structure is contiguous; that is, there is no padding used within the structure. Each character on the archive medium is stored contiguously.

Bytes representing the contents of files (after the header record of each file) are not translated in any way and are not constrained to represent characters in any character set. The `tar` format does not distinguish text files from binary files, and no translation of file contents is performed.

The `name`, `linkname`, `magic`, `uname`, and `gname` are null-terminated character strings. All other fileds are zero-filled octal numbers in ASCII. Each numeric field of width *w* contains *w* minus 2 digits, a space, and a null, except `size`, and `mtime`, which do not contain the trailing null.

The name field is the pathname 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 ("stick") 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. See Section 3.3 [Extraction Options], page 11; in particular the '-G' option.

The mtime field is the modification time of the file at the time it was archived. It is the ASCII representation of the octal value of the last time the file was modified, represented as an integer number of seconds since January 1, 1970, 00:00 Coordinated Universal Time.

The chksum field is the ASCII representation of the octal value of the simple sum of all bytes in the header record. Each 8-bit byte in the header is added to an unsigned integer, initialized to zero, the precision of which shall be no less than seventeen bits. When calculating the checksum, the chksum field is treated as if it were all blanks.

The typeflag field specifies the type of file archived. If a particular implementation does not recognize or permit the specified type, the file will be extracted as if it were a regular file. As this action occurs, tar issues a warning to the standard error.

The atime and ctime fields are used in making incremental backups; they store, respectively, the particular file's access time and last inode-change time.

The offset is used by the -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 longnames field of the header belongs with something that is not yet implemented in tar, and is therefore empty.

The following fields were added to deal with sparse files. A file is *sparse* if it takes in unallocated blocks which end up being represented as zeros, i.e., no useful data. A test to see if a file is sparse is to look at the number blocks allocated for it versus the number of characters in the file; if there are fewer blocks allocated for the file than would normally be allocated for a file of that size, then the file is sparse. This is the method tar uses to detect a sparse file, and once such a file is detected, it is treated differently from non-sparse files.

Sparse files are often dbm files, or other database-type files which have data at some points and emptiness in the greater part of the file. Such files can appear to be very large when an `ls -l` is done on them, when in truth, there may be a very small amount of important data contained in the file. It is thus undesirable to have tar think that it must back up this entire file, as great quantities of room are wasted on empty blocks, which can lead to running out of room on a tape far earlier than is necessary. Thus, sparse files are dealt with so that these empty blocks are not written to the tape. Instead, what is written to the tape is a description, of sorts, of the sparse file: where the holes are, how big the holes are, and how much data is found at the end of the hole. This way, the file takes up potentially far less room on the tape, and when the file is extracted later on, it will look exactly the way it looked beforehand. The following is a description of the fields used to handle a sparse file:

The `sp` is an array of `struct sparse`. Each `struct sparse` contains two 12-character strings which represent an offset into the file and a number of bytes to be written at that offset. The offset is absolute, and not relative to the offset in preceding array element.

The header can hold four of these `struct sparse` at the moment; if more are needed, they are not stored in the header.

The `isextended` flag is set when an `extended_header` is needed to deal with a file. Note that this means that this flag can only be set when dealing with a sparse file, and it is only set in the event that the description of the file will not fit in the alloted room for sparse structures in the header. In other words, an extended_header is needed.

The `extended_header` structure is used for sparse files which need more sparse structures than can fit in the header. The header can fit 4 such structures; if more are needed, the flag `isextended` gets set and the next record is an `extended_header`.

Each `extended_header` structure contains an array of 21 sparse structures, along with a similar `isextended` flag that the header had. There can be an indeterminate number of such `extended_headers` to describe a sparse file.

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

`LF_LINK`
: This flag represents a file linked to another file, of any type, previously archived. Such files are identified in Unix by each file having the same device and inode number. The linked-to name is specified in the `linkname` field with a trailing null.

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

`LF_CHR`
`LF_BLK`
: These represent character special files and block special files respectively. In this case the `devmajor` and `devminor` fields will contain the major and minor device numbers respectively. Operating systems may map the device specifications to their own local specification, or may ignore the entry.

`LF_DIR`
: This flag specifies a directory or sub-directory. The directory name in the `name` field should end with a slash. On systems where disk allocation is performed on a directory basis, the `size` field will contain the maximum number of bytes (which may be rounded to the nearest disk block allocation unit) which the directory may hold. A `size` field of zero indicates no such limiting. Systems which do not support limiting in this manner should ignore the `size` field.

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

`LF_CONTIG`
: This specifies a contiguous file, which is the same as a normal file except that, in operating systems which support it, all its space is allocated contiguously on the disk. Operating systems which do not allow contiguous allocation should silently treat this type as a normal file.

`'A'` ...
`'Z'`
: These are reserved for custom implementations. Some of these are used in the GNU modified format, as described below.

Other values are reserved for specification in future revisions of the P1003 standard, and should not be used by any `tar` program.

The `magic` field indicates that this archive was output in the P1003 archive format. If this field contains `TMAGIC`, the `uname` and `gname` fields will contain the ASCII representation of the owner and group of the file respectively. If found, the user and group ID represented by these names will be used rather than the values within the `uid` and `gid` fields.

8.2 GNU Extensions to the Archive Format

The GNU format uses additional file types to describe new types of files in an archive. These are listed below.

`LF_DUMPDIR`
`'D'`
: This represents a directory and a list of files created by the '`-G`' option. The `size` field gives the total size of the associated list of files. Each filename 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 filename is terminated by a null. There is an additional null after the last filename.

`LF_MULTIVOL`
`'M'`
: This represents a file continued from another volume of a multi-volume archive created with the '`-M`' option. The original type of the file is not given here. The `size` field gives the maximum size of this piece of the file (assuming the volume does not end before the file is written out). The `offset` field gives the offset from the beginning of the file where this part of the file begins. Thus `size` plus `offset` should equal the original size of the file.

`LF_SPARSE`
`'S'`
: This flag indicates that we are dealing with a sparse file. Note that archiving a sparse file requires special operations to find holes in the file, which mark the positions of these holes, along with the number of bytes of data to be found after the hole.

`LF_VOLHDR`
`'V'`
: This file type is used to mark the volume header that was given with the '`-V`' option when the archive was created. The `name` field contains the `name` given after the '`-V`' 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 '`-G`', '`-M`', '`-S`', or '`-V`' 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.

Table of Contents