

Assignment 4

cpe 357 Fall 2010

The tao that can be tar(1)ed
is not the entire Tao.
The path that can be specified
is not the Full Path.

— /usr/games/fortune

Note

This is definitely a rough draft, but hopefully it'll give you enough information to get started. Luckily, because of the elimination of the former Asgn5 (poor thing) there'll be time for revision while you're programming.

Due by 11:59:59pm, Tuesday, November 16th.

Due to the Midterm, no more than one late day may be used on this assignment. (Think of this as due Sunday with two free late days.)

This assignment may be done with a partner.

Program: mytar

This assignment is to build a file archiving tool, **mytar**, that is a version of the standard utility **tar(1)**. **Tar(1)**, standing for Tape ARchive, is one of the more ancient programs from the Unix world. It bundles files and directories together in a single file so that it can be easily transferred to some other location.

Your program must be able to build and restore archives in a way that is interoperable with GNU tar.

Running mytar

Usage:

```
mytar [ctxvS]f tarfile [ path [ ... ] ]
```

Mytar is a subset of tar and only supports five options. One of 'c', 't', or 'x' is required to be present. 'f' is optional, but you are not required to support its being absent.

Options Supported:

c	Create an archive
t	Print the table of contents of an archive
x	Extract the contents of an archive
v	Increases verbosity
f	Specifies archive filename
S	Be strict about standards compliance

The 'f' option

The argument following the 'f' option specifies the name of the archive file to use. Real tar uses stdin or stdout if the 'f' is missing, but you may require 'f' to be present for this assignment.

Archive Creation (c)

In create mode, **mytar** creates a new archive. If the archive file exists, it is truncated to zero length, then all the remaining arguments on the command line are taken as paths to be added to the archive.

If a given path is a directory, that directory and all the files and directories below it are added to the archive.

If the verbose ('v') option is set, **mytar** lists files as they are added, one per line.

Archive Listing (t)

In list (Table of contents) mode, **mytar** lists the contents of the given archive file, in order, one per line. If no names are given on the command line, **mytar**, lists all the files in the archive. If a name or names are given on the command line, **mytar** will extract the given path and any and all descendents of it. That is, all files and directories beginning with the same series of directories.

If the verbose ('v') option is set, **mytar** gives expanded information about each file as it lists them. For example:

```
% mytar tvf archive.tar
drwx----- pnico/pnico          0 2010-11-02 13:49 Testdir/
-rwx--x--x pnico/pnico          72 2010-11-02 13:49 Testdir/file1
-rw----- pnico/pnico        200 2010-11-02 13:49 Testdir/file2
%
```

The elements of each listing are the permissions, the owner/group, the size, the last modified date (mtime) and the filename. The symbolic names stored in the header are preferable. If the symbolic names are absent, use the numeric ones. (See the description of the header below.)

The listing consists of a line of the following fields, each separated by a space:

Field	Width	Description
Permissions	10	(See below)
Owner/Group	17	Name of the file's owner
Size	8	Size of the file (in bytes)
Mtime	16	Last modification time (YYYY-MM-DD HH:MM)
Name	variable	Filename

The permissions string consists of 10 characters. The first gives the files type: 'd' for a directory, 'l' for a symbolic links, or '-' for any other type of file. The remaining nine characters indicate the presence or absence of read (r), write (w) or execute (x) permission for the file's owner, group, and other respectively. If a permission is not granted, write a dash (-).

Archive Extraction (x)

In extract mode, **mytar** extracts files from an new archive. If no names are given on the command line, **mytar**, extracts all the files in the archive. If a name or names are given on the command line, **mytar** will extract the given path and any and all descendents of it just like listing.

Extract restores the modification time of the extracted files.

Strict (S)

This option forces **mytar** to be strict in its interpretation of the standard. That is, it requires the magic number to be nul-terminated and checks for the version number.

Without this option, `mytar` only checks for the five characters of “ustar” in the magic number and ignores the version field. This is required to interoperate with GNU’s tar.

Optional Extensions

Not worth anything extra, but if you get into it...

- Allow `stdin/stdout` (make “-” a valid argument to `-f`, and/or make `f` optional)

Other details

These are various details that did not fit nicely in the descriptions above.

- If duplicate entries exist in an archive, use the latest one.
- If “S” is not specified, must interact with GNU’s tar.
- Directory names stored in the archive must end in `’/’`
- File types supported:
 - Regular files (Regular and alternate markings)
 - Directories
 - Symbolic links

Any file of any other type should cause an error to be reported but no action taken.

USTAR Archive Format

`Mytar` implements the POSIX-specified USTAR archive format. The format of the archive is fully specified as part of POSIX, available on the web at http://www.unix.org/single_unix_specification/¹. It is also described below.

File format documented at

http://www.opengroup.org/onlinepubs/9699919799/utilities/pax.html#tag_20_92_13_06
`/usr/include/tar.h` contains useful definitions and a description of the header fields.

Archive Format

A USTAR archive is a sequential list of records each of which consists of a header block followed by zero or more data blocks. After the last record of the file is an End of Archive marker which consists of two blocks of all zero bytes.

- All blocks are 512 bytes
- Any portion of a block not completely filled by data (e.g. the last block of a file) should be filled by zero (`’\0’`) bytes.

Field Name	Offset	Length	Notes
name	0	100	NUL-terminated if NUL fits
mode	100	8	
uid	108	8	
gid	116	8	
size	124	12	
mtime	136	12	
chksum	148	8	
typeflag	156	1	
linkname	157	100	NUL-terminated if NUL fits
magic	257	6	must be "ustar", NUL-terminated)
version	263	2	must be "00" (zero-zero)
uname	265	32	NUL-terminated
gname	297	32	NUL-terminated
devmajor	329	8	
devminor	337	8	
prefix	345	155	NUL-terminated if NUL fits

Figure 1: Fields of the USTAR header

Header Format

The header fields are described in the Figure 1.

The fields are:

- name** The name of the archived file is produced by concatenating the **prefix** (if of non-zero length), a slash, and the **name** field. If the prefix is of zero length, **name** is the complete name.
- mode** The numeric representation of the file protection modes stored as an octal number in an ASCII string terminated by one or more space or nul characters. Valid mode bits are defined in Figure 2.
- uid** The numeric user id of the file owner, encoded as an octal number in an ascii string terminated by one or more space or nul characters.
- gid** The numeric group id of the file owner, encoded as an octal number in an ascii string terminated by one or more space or nul characters.
- size** The size of the file encoded as an octal number in an ASCII string terminated by one or more space or nul characters. The size of symlinks and directories is zero.
- mtime** The last modified time of the file encoded as an octal number in an ASCII string terminated by one or more space or nul characters.
- chksum** The checksum is the result of adding up all the characters (treated as unsigned characters) in the header block, encoded as an octal number in an ASCII string terminated by one or more space or nul characters. For purposes of computing the checksum the checksum field itself is treated as if it were filled with spaces.

¹You will be required to register, but it's free.

04000	S_ISUID	Set UID on execution.
02000	S_ISGID	Set GID on execution.
01000	S_ISVTX [†]	Sticky bit.
00400	S_IRUSR	Read permission for file owner class.
00200	S_IWUSR	Write permission for file owner class.
00100	S_IXUSR	Execute/search permission for file owner class.
00040	S_IRGRP	Read permission for file group class.
00020	S_IWGRP	Write permission for file group class.
00010	S_IXGRP	Execute/search permission for file group class.
00004	S_IROTH	Read permission for file other class.
00002	S_IWOTH	Write permission for file other class.
00001	S_IXOTH	Execute/search permission for file other class.

[†]The sticky bit is not actually part of the POSIX standard, but is stored by most unix implementations.

Figure 2: Valid mode bits as defined in the standard.

typeflag A character indicating the type of the archived file. Valid file types for **mytar** are:

'0' Regular file
 '\0' Regular file (alternate)
 '2' symbolic link
 '5' directory

linkname If the file is of type symlink (or hard link should you implement that) this is the value of the link.

magic The magic number shall consist of the string “ustar”, nul-terminated.

version The version shall be the two characters “00” (zero-zero)

uname The symbolic name of the file’s owner, truncated to fit if necessary, nul-terminated.

gname The symbolic name of the file’s group, truncated to fit if necessary, nul-terminated.

devmajor Major number of the archived file, encoded as an octal number in an ASCII string terminated by one or more space or nul characters.

devminor Minor number of the archived file, encoded as an octal number in an ASCII string terminated by one or more space or nul characters.

prefix See “name” above.

When creating archives, as much of the name as will fit should be placed in the **name** field, with the overflow going to **prefix**.

Other things waiting to be integrated into the narrative:

- Permissions for extracted files
 - By default, tar does not try to restore a files’s archived permissions.
 - It offers rw permission to everyone, and the umask applies.

- If any execute bits are set in the archived permissions, tar offers execute permission to all on the extracted file.
- Differences between `mytar` and `tar(1)`:
 - you don't have to handle stdin/stdout
 - Only handles regular files, directories, and symlinks
 - verbose option can be repeated
 - “S” strict option that makes it strict on conformance.
 - mytar will archive absolute paths. Most tar implementations refuse.
- If you want to be able to try diffing against my archive files, my version of mytar goes through its arguments in order and does a preorder DFS for each while inserting. It is not required to do it this way, but a successful diff is a comforting thing.
- Error handling:
 - When reading, stop at the first corrupt record. When writing, skip files you can't read, but go on. (report, of course)
 - If you find a bad header (invalid checksum, magic number, or version (as appropriate), declare that you're lost and give up.
 - If other errors are encountered, report them with meaningful error messages and continue if possible.
 - Paths are at most 256 characters. If a path is longer than that, tar must print an error message
 - Names can only be broken on a '/'. If a name can not be partitioned, print an error and go on to the next file (if any).

Tricks and Tools

- Useful functions in Figure 3.
 - Add
 - memset
 - memcpy
 - strtol()
 - strncpy()
 - sprintf()
- Think about your data structures
- Write a function that builds directory trees (NOte: a file being in the archive does not require all its parent directories to exist.)
- Write functions that pack, unpack, and verify tar headers. (don't go on until you are sure you are reading headers accurately)

- Write a recursive traversal of the directory tree that only descends into real directories (avoid symlinks).
- Be careful not to recurse on “.” or “..”
- Be careful to remember where you started. All of the arguments are relative to the original current working directory.
- Write debugging functions early so you can observe your progress.

<code>lstat(2)</code>	to get information about a file including its type, owner, size and permissions.
<code>chdir(2)</code> <code>getcwd(3)</code>	for navigating within the filesystem.
<code>opendir(3)</code> <code>closedir(3)</code> <code>readdir(3)</code> <code>rewinddir(3)</code> etc.	for reading and manipulating directory entries.
<code>getpwuid(3)</code> <code>getgrgid(3)</code>	for help translating the user and group IDs returned by <code>lstat(2)</code> into names.
<code>readlink()</code> <code>printf(3)</code> <code>sprintf(3)</code>	to read the value of a symbolic link for generating formatted output.

Figure 3: Some potentially useful library functions

Coding Standards and Make

See the pages on coding standards and make on the cpe 357 class web page.

What to turn in

Submit via `handin` to the `asgn4` directory of the `pn-cs357` account:

- Your well-documented source files.
- A makefile (called **Makefile**) that will build your program when given the target “mytar” or no target.
- A README file that contains:
 - Your name(s). In addition to your names, please include your Cal Poly login names with it, in parentheses. E.g. (**pnico**)
 - Any special instructions for running your program.
 - Any other thing you want me to know while I am grading it.

The README file should be **plain text**, i.e, **not a Word document**, and should be named “README”, all capitals with no extension.

Sample runs

Below are some sample runs of `mytar`. I will also place executable versions in `~pn-cs357/demos` so you can run it yourself.

```
% ls -lR Test
Test:
total 8
drwx-----. 2 pnico pnico 4096 Nov  5 06:17 Subdir
-rw-----. 1 pnico pnico  96 Nov  5 06:16 file1

Test/Subdir:
total 8
-rw-----. 1 pnico pnico 156 Nov  5 06:17 file1
-rwx-----. 1 pnico pnico 135 Nov  5 06:17 file2
% mytar cvf Test.tar Test
Test
Test/Subdir
Test/Subdir/file1
Test/Subdir/file2
Test/file1
% mytar tf Test.tar
Test/
Test/Subdir/
Test/Subdir/file1
Test/Subdir/file2
Test/file1
% mytar tvf Test.tar
drwx----- pnico/pnico          0 2010-11-05 06:17 Test/
drwx----- pnico/pnico          0 2010-11-05 06:17 Test/Subdir/
-rw----- pnico/pnico        156 2010-11-05 06:17 Test/Subdir/file1
-rwx----- pnico/pnico        135 2010-11-05 06:17 Test/Subdir/file2
-rw----- pnico/pnico          96 2010-11-05 06:16 Test/file1
% mkdir Output
% cd Output
% ls
% mytar tf ../Test.tar
Test/
Test/Subdir/
Test/Subdir/file1
Test/Subdir/file2
Test/file1
% mytar tf ../Te
Test.tar Test/ Testdir/
% mytar tf ../Test.tar
Test/
Test/Subdir/
Test/Subdir/file1
Test/Subdir/file2
```



```
Test/file1
% mytar xvf ../Te
Test.tar Test/ Testdir/
% mytar xvf ../Test.tar Test/Subdir
Test/Subdir/
Test/Subdir/file1
Test/Subdir/file2
% ls -R Test/
Test/:
Subdir
```

```
Test/Subdir:
file1 file2
% mytar xvf ../Test.tar
Test/
Test/Subdir/
Test/Subdir/file1
Test/Subdir/file2
Test/file1
% ls -R Test/
Test/:
Subdir file1
```

```
Test/Subdir:
file1 file2
%
```