TAR(5)

名称

tar – テープアーカイブファイルのフォーマット

解説

tar アーカイブのフォーマットは、いくつものファイル、ディレクトリやこの他のファイルシステムオブジェクト（シンボリックリンク、デバイスノードなど）を集めて、単一のバイト列のストリームとしたものです。このフォーマットは元来、固定長のブロックで操作するテープドライブと共に使用されるようにデザインされていましたが、一般的なパッケージング機構として広く用いられています。

フォーマットの概略

tar アーカイブは、512 バイトレコードの集合からなります。各ファイルシステムオブジェクトには、基本的なメタデータ（パス名、所有者、パーミッションなど）を記録したヘッダレコードと、ファイルデータを含んだ 0 個かそれ以上のレコードが必要です。アーカイブの終端は、全てがバイト 0 の 2 つのレコードで示されます。

関連項目
規格
歴史

固定ブロックサイズを使用するテープドライブとの互換性のため、tar ファイルを読み書きするプログラムは各 I/O 操作の際に、常に固定したレコード数で読み書きをします。これらの“ブロック”は、常にレコードサイズの倍数です。初期の実装によってサポートされた最大のブロックサイズは、 10240 バイトまたは 20 レコードでした。これは、ほとんどの実装に対してデフォルトですが、 1 MiB (2048 レコード) 以上のブロックサイズは、近代的な高速テープドライブで一般的に使用されています。 (注意: ここでの“ブロック”および“レコード”という言葉は完全に標準というわけではありません。このドキュメントでは、John Gilmore が **pdtar** のドキュメントで確立した慣例に沿っています。)

古い形式のアーカイブフォーマット

オリジナルの tar アーカイブフォーマットは、たくさんの実装者が必要に応じて、追加の情報を含めるために何度も拡張されてきました。このセクションは、tar プログラムの最も初期に広く使用されたバージョンであると思われる Version 7 AT&T UNIX に含まれる tar コマンドとして実装された変異型について説明しています。

古い形式の **tar** アーカイブのヘッダレコードは、次のような構成です:

```
struct header_old_tar {
 char name[100];
 char mode[8];
 char uid[8];
 char gid[8];
 char size[12];
 char mtime[12];
 char checksum[8];
 char linkflag[1];
 char linkname[100];
 char pad[255];
};
```

ヘッダレコード中の全ての未使用バイトはヌルで埋められます。

name

パス名。ヌル文字で終わる文字列として記録します。初期の tar 実装では、通常ファイル (それらへのハードリンクも含む) のみ保存しました。共通した初期の取り決めの一つとして、"/"文字を続けることでディレクトリ名を表し、ディレクトリのパーミッションと所有者情報を保存、抽出できるとされました。

mode

ファイルモード。 8 進数のアスキー文字として記録します。

uid, gid

所有者のユーザ id とグループ id。 8 進数のアスキー文字です。

size

ファイルのサイズ。 8 進数のアスキー文字です。通常ファイルに限っては、このヘッダに続くデータの量を示します。特に、初期の tar 実装ではハードリンクを展開する際、このフィールドは無視されていました。現代の作成プログラムなら、ハードリンクのエントリでは長さ 0 を記録すべきです。

mtime

ファイルの変更日時。 8 進数のアスキー文字です。これは基準時点（epoch）である協定世界時（UTC）1970 年 1 月 1 日 00:00:00 からの秒数を表します。なお、負の値は不合理に扱われるので、ここでは避けるべきです。

checksum

ヘッダのチェックサム。 8 進数のアスキー文字として記録します。チェックサムを計算するには、このチェックサムフィールドを全て空白で埋め、ヘッダの全てのバイトを符号無しの算法で合計します。このフィールドは、6 桁の 8 進数に、ヌル文字と空白文字が続く形で記録する必要があります。なお、ごく初期の tar 実装では、符号つきの算法をチェックサムフィールドに用いており、これはアーカイブをシステム間で転送する際に相互運用性の問題を引き起こします。現代の堅牢な読み込みプログラムは、両方の計算法でチェックサムを計算して、どちらの計算法によるヘッダでも受け付けます。

linkflag, linkname

ハードリンクを保存しテープを浪費しないために、複数のリンクを持つファイルは最初に表れた時のみ、アーカイブに書き込みます。次に表れた時、 **linkflag** にアスキー文字の‘1’がセットされ、 **linkname** フィールドにこのファイルが最初に表れた時のファイル名を記録します。（なお、通常のファイルの **linkflag** フィールドはヌル値です。）

初期の tar 実装の、これらのフィールドの終端方法にはさまざまなものがありました。Version 7 AT&T UNIX の tar コマンドでは、次のような取り決めをしていました（これは初期の BSD man ページにも文書化されています): パス名は、ヌル文字で終わること。mode, uid, gid フィールドは空白とヌルバイトで終わること。 size と mtime フィールドは空白で終わること。チェックサムはヌルと空白で終わらせる、です。初期の実装では、数値フィールドは空白で始める形で埋めていました。これは IEEE Std 1003.1-1988 (“POSIX.1”) 標準が発表されるまで、共通の慣習として行われていたようです。移植性を最も良くするために、現代の実装では、数値フィールドは 0 で始める形で埋めるようにするべきです。

POSIX 以前のアーカイブ

IEEE Std 1003.1-1988 (“POSIX.1”) の初期のドラフトは、John Gilmore の **pdtar** プログラムや、1980 年代終わりから 1990 年代始めにかけてのシステムの実装の基礎となり

ました。これらのアーカイブは、後に述べる POSIX ustar フォーマットに対して、次のような違いがあります:

- マジックの値は、空白が続く 5 文字の“ustar”から成ります。 version フィールドは、ヌルが続く空白文字が含まれます。
- 数値フィールドは、一般に空白で始まる形で埋められています（最終標準のように、 0 で始まる形ではない）。
- prefix フィールドはしばしば利用されず、古い形式のアーカイブのように、パス名は 100 文字までに制限されています。

POSIX ustar アーカイブ

IEEE Std 1003.1-1988 (“POSIX.1”) は、対応した tar(1) 実装で読み書きが出来る、標準的な tar ファイルフォーマットを定義しています。このフォーマットは、ヘッダ中のマジック値からしばしば“ustar”フォーマットとも呼ばれます。 （この名前は“Unix Standard TAR”の頭文字です。） これは今までのフォーマットを新しいフィールドで拡張したものです:

```
struct header_posix_ustar {
 char name[100];
 char mode[8];
 char uid[8];
 char gid[8];
 char size[12];
 char mtime[12];
 char checksum[8];
 char typeflag[1];
 char linkname[100];
 char magic[6];
 char version[2];
 char uname[32];
 char gname[32];
 char devmajor[8];
 char devminor[8];
 char prefix[155];
 char pad[12];
};
```

typeflag

エントリの種類。 POSIX は初期の **linkflag** フィールドをいくつかの新しい値で拡張しました:

“0”

通常ファイル。互換性の目的のために、NUL （ヌル文字） も同義語として扱うべきです。

“1”

ハードリンク。

“2”

シンボリックリンク。

“3”

キャラクタ型デバイスノード。

“4”

ブロック型デバイスノード。

“5”

ディレクトリ。

“6”

FIFO 型ノード。

“7”

予約。

この他

POSIX 互換実装では、理解できないあらゆる typeflag 値は通常ファイルとして扱わなければなりません。特に、作成プログラムは、対応する拡張機能をサポートしていない読み込みプログラムであっても、きちんと展開できるような確実なファイル名を全てのエントリにつけるように保証する必要があります。大文字レターの "A"から "Z"はカスタム拡張のために予約されています。なお、ソケットとホワイトアウトエントリはアーカイブできません。

size フィールドは、特にこのタイプによって意味が異なるので、ここで触れておきます。通常ファイルに対しては、もちろん、ヘッダに続くデータの量を示しています。ディレクトリに対しては、ディレクトリスペースをあらかじめ割り当てるオペレーティングシステムのために、ディレクトリ中の全てのファイルの合計サイズを示すのに使われます。これら以外のタイプでは、作成プログラムは 0 をセットするべきですし、読み込みプログラムは無視するべきです。

magic

マジック値である“ustar”が、ヌルバイトが続く形で含まれています。これが POSIX 標準アーカイブであることを示しています。 uname と gname フィールドが適切にセットされていることが求められます。

version

バージョン。 POSIX 標準アーカイブでは、“00”（アスキー数字の 0 が二つ）であるべきです。

uname, gname

ユーザ及びグループ名。ヌル文字で終わるアスキー文字列です。対応する名前がシステムに存在する時には、これらは uid/gid の値よりも優先して使用されるべきです。

devmajor, devminor

キャラクタ型デバイスエントリあるいはブロック型デバイスエントリに対する、メジャー番号とマイナ番号です。

name, prefix

パス名が長過ぎて標準フォーマットで提供される 100 バイトに収まらないなら、prefix フィールドまで最初の部分で任意の / 文字で分割することができます。prefix フィールドが空でないなら、読み込みプログラムは、フルパス名を得るために prefix の値と / 文字を通常の name フィールドの前に追加します。標準は、ディレクトリ名で後続する / 文字を必要としませんが、ほとんどの実装は、互換性の理由のためにまだこれを含んでいます。

使用していない全てのバイトは NUL（ヌル）を設定しなければならないことに注意してください。

フィールドの終端は、POSIX ではそれ以前の実装とはすこし異なる形で指定されます。**magic, uname, gname** フィールドは NUL（ヌル文字）で終わらなければなりません。**pathname, linkname, prefix** フィールドは、フィールド全体を使用する場合を除いて NUL（ヌル文字）で終わらなければなりません。（特に、156 番目の文字が / である場合、256 文字のパス名を記録することが出来ます。）POSIX は、最前部で数値フィールドを 0 で埋めることを要求し、空白か、または NUL（ヌル）文字のいずれかで終了していることを必要としています。

現在のところ、ほとんどの tar 実装は ustar フォーマットに対応しており、時折ヘッダレコードの残りの未使用部分に新しいフィールドを追加して拡張しています。

数値拡張

どのくらいの数がヘッダに格納されるかを修正することによってサポートされるサイズまたは時間の範囲を拡張するいくつかの試みがありました。

ファイルのサイズを増加させる 1 つの明白な拡張は、様々な数値フィールドから終了文字を除去することです。例えば、標準は、サイズフィールドが後続する NUL 文字のために 12 番目のバイトを予約して、11 個の 8 進数を単に含むことができます。 12 個の 8 進数であれば、ファイルサイズを最大 64 GB まで可能となります。

GNU tar、star と新しい **tar** の実装によって利用される別の拡張は、標準の数値フィールドにバイナリ数値を許可します。これは、最初のバイトの高位ビットを設定することによって示されます。フィールドの残りは、符号付きの 2 の補数値として扱われます。これは、長さと時間のフィールドのための 95 ビットの値と uid、gid とデバイス番号のための 63 ビットの

値を許可します。特に、これは、負の時間の値を扱う一貫した方法を提供しています。 GNU tar は、length、mtime、ctime と atime フィールドに対してこの拡張をサポートしています。 Joerg Schilling の star プログラムと libarchive ライブラリは、すべての数値フィールドに対してこの拡張をサポートしています。この拡張は、pax 交換形式によって提供される拡張属性レコードによって大部分は時代遅れにされることに注意してください。

別の初期の GNU 拡張は、8 進数ではなく base-64 を許可しました。この拡張は、短命で、あらゆる実装によっても、もはやサポートされていません。

Pax 交換フォーマット

POSIX ustar アーカイブに、可搬性のある形で記録できない属性が多数あります。 IEEE Std 1003.1-2001 (“POSIX.1”) では、続くエントリに適用するテキスト形式のメタデータを持った、2 つの新しいタイプのエントリを使用する“pax 交換フォーマット”を定義しています。なお、pax 交換フォーマットアーカイブは、あらゆる点で ustar アーカイブです。新しいデータは ustar 互換のアーカイブエントリに、typeflag に“x”および“g”を使用して記録されます。特に、これらの拡張機能を完全にサポートしない古い実装では、このメタデータを必要に応じて検査できるよう、通常ファイルとして展開するでしょう。

pax 交換フォーマットアーカイブのエントリは、1 つか 2 つの標準 ustar エントリを含み、それぞれヘッダとデータを持ちます。最初のオプションエントリは、これに続くエントリに対する拡張属性を記録しています。この最初のオプションエントリは typeflag "x"を持ち、size フィールドは拡張属性の合計サイズを表しています。この拡張属性自体は、可搬性のある UTF-8 エンコーディングで符号化した、テキスト形式の行の集合として記録されます。各行は 10 進数、空白、キー文字列、等号記号、値文字列、改行で構成されています。 10 進数は、行全体の長さを表し、これにはこの長さフィールド自身と改行を含んでいます。フィールドの例です:

```
25 ctime=1084839148.1212\n
```

全て小文字のキーは標準キーです。ベンダは、全て大文字のベンダ名とピリオドで始まる、独自のキーを追加することができます。なお、今までのヘッダとは異なり、数値は 8 進数ではなく、10 進数で記録されます。以下はいくつかの共通キーについて述べています。

atime, ctime, mtime

ファイルのアクセス時刻、inode 変更時刻、および修正時刻。負の値、もしくは小数点や分数値が使用できます。

hdrcharset

文字セットは、pax 拡張値によって使用されます。デフォルトで、pax 拡張属性のすべてのテキスト値は、パスネーム、ユーザ名とグループ名を含む、UTF-8 であると仮定されます。ある場合に、ローカルな仕様を UTF-8 に変換することはできません。このキーが存在し、値が 6 文字の ASCII 文字列“BINARY”であるなら、すべてのテキスト値は、プラットフォーム依存のマルチバイトエンコードであると仮定されます。このキーに対する 2 つの有効な値だけがあることに注意してください: “BINARY”または“ISO-IR 10646 2000 UTF-8”です。他の値は、標準によって許可されず、後の値は、このキ

ーが指定されないとき、デフォルトのように、一般的に使用されるべきではありません。特に、このフラグは、ファイル名を任意のエンコーディングで格納できる一般的なメカニズムとして使用されるべきではありません。

uname, uid, gname, gid

ユーザ名、グループ名、UID および GID の数値。ここに記録するユーザ名とグループ名は UTF8 で符号化されるので、非アスキー文字を含めることができます。 UID および GID フィールドは、任意の長さに出来ます。

linkpath

ファイルにリンクするフルパス。なお、これは UTF8 で符号化されるので、非アスキー文字を含めることが出来ます。

path

エントリのフルパス名。なお、これは UTF8 で符号化されるので、非アスキー文字を含めることが出来ます。

realtime.*, security.*

これらのキーは予約済みのもので、将来の標準化により利用されるでしょう。

size

ファイルのサイズ。なお、このフィールドに長さの制限はないので、アーカイブに今までの制限の 8GB よりも大きなファイルを保存させることが可能です。

SCHILY.*

Joerg Schilling の **star** 実装で使われるベンダ固有の属性。

SCHILY.acl.access, SCHILY.acl.default

POSIX.1e draft 17 に記述されているフォーマットを拡張したもので、アクセスとデフォルトの ACL を、テキスト形式の文字列で記録したもの。特に、各ユーザまたはグループのアクセス明細を、コロンで 4 つに区切ったフィールドで UID または GID 数値と共に含めることが出来ます。これにより、完全なユーザ又はグループ情報が無い場合でも、システム上で ACL の復元ができます (NIS/YP あるいは LDAP サービスが一時的に利用できない場合など)。

SCHILY.devminor, SCHILY.devmajor

デバイスノードの完全なマイナとメジャー番号。

SCHILY.fflags

ファイルフラグ。

SCHILY.realsize

ディスク上のファイルの全サイズ。 XXX 説明? XXX

SCHILY.dev, SCHILY.ino, SCHILY.nlinks

デバイス番号、inode 番号、およびエントリのリンクカウント。特に、Joerg Schilling の **SCHILY.*** 拡張を用いて、pax 交換フォーマットアーカイブに **struct stat** のデータを全て保存することが出来ます。

LIBARCHIVE.*

それを使用する **libarchive** ライブラリとプログラムによって使用されるベンダ特有の属性。

LIBARCHIVE.creationtime

ファイルが作成されたときの時間。 (これは、ファイルのメタデータが最後に変更されたときの時間を参照する、POSIX “ctime”属性と混同されるべきではありません。)

LIBARCHIVE.xattr. ***namespace.*** ***key***

libarchive は、この形式のキーを使用して POSIX.1e スタイルの拡張属性を保存します。 *key* 値は URL エンコードされています: すべての非 ASCII 文字と 2 つの特殊文字“=”と“%”は、“%”に 2 つの大文字の 16 進数が続くものとしてエンコードされます。このキーの値は base 64 でエンコードされた拡張属性値です。 XXX ここで base-64 形式を詳しく述べる XXX

VENDOR.*

XXX 他のベンダ固有拡張に関するドキュメント XXX

拡張属性により記録された値は、通常の tar ヘッダの対応する値を上書きします。なお、対応する読み込みプログラムなら、上書きがされるならば通常フィールドを無視するべきです。既存のアーカイバはこのような条件において、標準のヘッダフィールドに互換性の無い値を記録することが知られているので、この点は重要です。これらのフィールドには、長さの制限はありません。特に、数値フィールドは任意の大きさにすることが出来ます。全てのテキストフィールドは、UTF8 で符号化されます。対応する作成プログラムは、標準 ustar ヘッダには可搬性のある 7 ビットアスキー文字のみを記録するべきで、テキスト値に非アスキー文字が含まれる場合のみに、拡張属性を使用するべきです。

これまでに述べた **x** エントリに加えて、pax 交換フォーマットはまた **g** エントリもサポートしています。 **g** エントリは同様の形式ですが、これに続く全てのアーカイブエントリに与えるデフォルトの属性を指定するものです。この **g** エントリはあまり広範には用いられていません。

新しい **x** と **g** エントリの他にも、pax 交換フォーマットは初期の ustar フォーマットに対して、他の細かい変化がいくつかあります。最も厄介なことの一つは、ハードリンクがこれに続くデータを持つことを許可されていることです。これは読み込みプログラムに、最初のエント

リを探すためにアーカイブを巻戻すことなく、あるファイルに対するハードリンクを展開するのを可能にします。しかし、堅牢な読み込みプログラムに対し、ハードリンクエントリの `size` フィールドを無視するべきかどうかはっきりとしないという混乱を生じます。

GNU Tar アーカイブ

GNU tar プログラムは初期に記述された POSIX 以前のフォーマットに似たものから始まり、いくつかの異なる機構で拡張されています。ヘッダの未使用空間に新しいフィールドを追加 (それらのいくつかは、後に POSIX にて矛盾する目的で使用されています)。ヘッダが複数のレコードに跨ることを認めている。そして続くエントリを修飾する新しいエントリを追加 (大体において前述の **x** エントリに似ていますが、多目的な **x** エントリとは違い、GNU の各特別エントリは単一目的です)、といったものです。結果として、GNU tar アーカイブは、かなり寛大な POSIX 対応の読み込みプログラムならば、大抵の GNU tar アーカイブをうまく展開することができるとはいえ、POSIX とは互換性がありません。

```
struct header_gnu_tar {
 char name[100];
 char mode[8];
 char uid[8];
 char gid[8];
 char size[12];
 char mtime[12];
 char checksum[8];
 char typeflag[1];
 char linkname[100];
 char magic[6];
 char version[2];
 char uname[32];
 char gname[32];
 char devmajor[8];
 char devminor[8];
 char atime[12];
 char ctime[12];
 char offset[12];
 char longnames[4];
 char unused[1];
 struct {
  char offset[12];
  char numbytes[12];
 } sparse[4];
 char isextended[1];
 char realsize[12];
 char pad[17];
};
```

typeflag

GNU tar は、POSIX で定義されているものに加え、以下の特別エントリタイプを使用します:

7

GNU tar は、ある無名の RTOS で、ディスク上に連続したファイルをあらかじめ割り当てるために使用するのを除いて、タイプ "7"をタイプ "0"とまったく同等に扱います。

D

ディレクトリエントリを意味します。 POSIX 標準の typeflag "5"とは異なり、このヘッダにはディレクトリ内にあるファイルの名前の一覧データが続きます。もしファイルがこのアーカイブに保存されれば、ファイル名の先頭にアスキー文字の "Y"が追加されますし、保存されなければ、"N"が追加されます。名前はそれぞれヌルで終端され、名前の一覧の終端には追加のヌルがマークされます。このエントリの目的は、インクリメンタルバックアップをサポートすることにあります。プログラムがこのようなアーカイブを展開する場合、アーカイブが作成された時にディレクトリ内に存在しなかったファイルを、ディスクから削除しようとするでしょう。

なお、理解できない typeflag が通常ファイルとして展開されることを要求している POSIX に、この typeflag "D"は明確に違反しています。この場合、通常ファイルとして "D"エントリが展開されると、そのあとに同名のディレクトリを作成するのを妨げる可能性があります。

K

このエントリに対するデータは、続く通常エントリに対する長い linkname です。

L

このエントリに対するデータは、続く通常エントリに対する長いパス名です。

M

前のボリュームの最後のファイルの続きです。 GNU マルチボリュームアーカイブは、各ボリュームが正しいエントリヘッダで始まることを保証しています。これを確実にするため、ファイルの分割は、あるボリュームの最後と、これに続くボリュームの最初に保存されると言う形でなされます。この typeflag "M"は、このエントリが既存のファイルの続きであることを示しています。このエントリは、アーカイブの最初か二番目のエントリにのみ表れます（後者は最初のエントリがボリュームラベルである場合のみ)。 **size** フィールドは、このエントリのサイズを表しています。 369-380 バイト目にある **offset** フィールドは、このファイルの断片が始まるオフセットを表しています。 **realsize** フィールドは、このファイルの合計サイズを表しています（これは **size** と **offset** を足したものと同じでなければいけません)。展開の時、GNU tar はヘッダのファイル名が予測したものかどうかチェックし、ヘッダの offset が正しく続いているかチェックし、そして offset と size の合計が realsize と等しいかチェックします。

N

タイプ "N"レコードを GNU tar が生成することはもうありません。これは展開をした後に、リネーム又はシンボリックリンクを張るファイルの一覧を含んでいます。これは元々は長い名前をサポートするのに使用されました。このレコードの内容は、行われるべき操作をテキストで記述したもので、形式は“Rename %s to %s\n”または“Symlink %s to %s\n”となっています。どちらの場合も、両方のファイル名は K&R C 形式でエスケープされます。セキュリティ上の問題のために、"N"レコードは、現在、アーカイブを読み込むとき、一般的に無視されます。

S

“sparse”通常ファイルです。疎なファイルは断片の集合として保存されます。ヘッダは断片の offset/length ペアの一覧を含んでいます。もしそのようなエントリが 4 つ以上必要なら、必要に応じて“extra”ヘッダ拡張 (既に使われていない古いフォーマットです)、もしくは“sparse”拡張を用いてヘッダを拡張します。

V

この **name** フィールドは、テープまたはボリュームのヘッダ名称と解釈するべきです。このエントリは一般的に、展開の際には無視されるべきです。

magic

この magic フィールドは、“ustar”の 5 文字と空白が続く形で保持されています。なお、POSIX ustar アーカイブでは、ヌルが続いています。

version

この version フィールドは、空白文字とヌルが続く形で保持されています。なお、POSIX ustar アーカイブでは、アスキー数字の“0”を 2 つ使用しています。

atime, ctime

最後にファイルがアクセスされた時刻と、最後にファイル情報が変更された時刻です。**mtime** のように、8 進数で記録されます。

longnames

このフィールドは、どうやら既に使用されていないようです。

Sparse offset / numbytes

この構造体は、疎なファイルの断片のひとつを指定します。 2 つのフィールドは、8 進数の値を記録します。アーカイブ内で断片はそれぞれ、512 バイトの倍数にパディングされます。展開において、断片の一覧がヘッダ (拡張ヘッダも含む) から集められ、読み込んだデータを適切なオフセットで書き込みます。

isextended

もしこれが非 0 にセットされた場合、ヘッダが追加の“sparse header”レコードに続きます。それぞれのレコードは以下に示す追加の sparse ブロック 21 個などを含んでいます:

```
struct gnu_sparse_header {
 struct {
  char offset[12];
  char numbytes[12];
 } sparse[21];
 char    isextended[1];
 char    padding[7];
};
```

realsize

バイナリ形式でファイルの完全なサイズを表したもので、POSIX のファイルサイズよりも範囲がより大きくなっています。特に、 **M** タイプのファイルにおいては、現在のエントリはファイルの一部分のみです。この場合、POSIX の size フィールドはこのエントリのサイズを表しており、 **realsize** フィールドはファイルの総サイズを表しています。

GNU tar pax アーカイブ

GNU tar 1.14 (XXX これをチェック XXX) 以降は、 **--posix** フラグを指定するとき、pax 交換形式アーカイブを書き込むます。この形式は、いくつかの **SCHILY** タグを使用し、スパースファイル情報を格納するために新しいキーワードを導入している、pax 交換形式に綿密に従います。“0.0”, “0.1”と“1.0”として参照される、スパースファイルのサポートの 3 つの繰り返しがありました。

GNU.sparse.numblocks, GNU.sparse.offset, GNU.sparse.numbytes, GNU.sparse.size

“0.0” 形式は、最初にファイル中のブロックの数を示すために **GNU.sparse.numblocks** 属性を、各ブロックのオフセットとサイズを示すために **GNU.sparse.offset** と **GNU.sparse.numbytes** の組と、ファイルの全サイズを示すために、単一の **GNU.sparse.size** を使用します。これは、後者の値が任意の穴のサイズを含んでいないので、tar ヘッダのサイズと同じではありません。この形式は、標準によって公式に許可されていない、同じ属性名の複数の出現を受け付けるリーダ (reader) で保存され頼られる属性の順序を要求されます。

GNU.sparse.map

“0.1”形式は、10 進数のコンマで区切られたリストを格納する単一属性を使用します。それぞれの数値の組は、それぞれ 1 ブロックのデータのオフセットとサイズを示してい

ます。多くの pax 実装が単に認識されていない属性を捨てるので、アーカイブがこの拡張を認識しないアーカイバによって抽出されるなら、これは、うまく動作しません。

GNU.sparse.major, GNU.sparse.minor, GNU.sparse.name, GNU.sparse.realsize

“1.0”形式は、エントリ本体のファイルデータの前に追加された 1 つ以上の 512 バイトのブロックでスパースブロックマップを格納します。 pax 属性は、 （**GNU.sparse.major** と **GNU.sparse.minor** フィールドを通して）このマップの存在とファイルの全サイズを示します。 **GNU.sparse.name** は、ファイルの真の名前を保持しています。混乱を避けるために、普通の tar ヘッダに格納された名前は、変更された名前であるので、その抽出エラーで、ユーザに明らかになります。

Solaris Tar

XXX より詳細なものが必要 XXX

Solaris の tar（SunOS XXX 5.7 ?? XXX から）は、pax 交換フォーマットと基本的には良く似た“extended”フォーマットをサポートしていますが、次のような違いがあります:

- 拡張属性は pax 交換フォーマットのような **x** タイプではなく、 **X** タイプのエントリに記録されます。このエントリのフォーマットの詳細は、前述した **x** エントリと同じもののように見えます。
- 追加の **A** ヘッダは、続く通常エントリの ACL を記録するのに使用されます。このエントリの本体には、バイト 0 が続く 7 桁の 8 進数に、テキスト形式の ACL の解説が続きます。 8 進値は、ACL エントリの数と次の ACL タイプを示す定数です: POSIX.1e ACL に対しては、 01000000、NFSv4 ACL 対しては、03000000 です。

AIX Tar

XXX より詳細が必要 XXX

AIX Tar は、コード化された ACL 情報を格納するためにタイプ **A** がある ustar 形式のヘッダを使用します。 Solaris 形式と異なり、 AIX tar は、それが適用される通常ファイルの本体の後に、このヘッダを書き込みます。このヘッダのパス名は、格納された ACL のタイプを示す **NFS4** または **AIXC** のいずれかです。実際の ACL は、プラットフォーム特有のバイナリ形式で格納されます。

Mac OS X Tar

Apple's Mac OS X で配布される tar は、tar アーカイブの 2 つに分かれたファイルとして、ほとんどの通常のファイルを格納しています。 2 ファイルには、最初のものに、最後のパスの要素の先頭に追加された“._”があることを除いて、同じ名前があります。この特殊ファイルは、ACL、拡張属性とリソースを含む 2 番目のファイルに関する、追加のメタデータで AppleDouble でエンコードされたバイナリブロブ（blob）を格納します。ディスク上のオリ

ジナルのファイルを再作成するために、それぞれ個別のファイルを抽出することができ、個別のメタデータファイルをアンパックし、 th 通常ファイルにそれを適用するために、Mac OS X の **copyfile**() 関数を使用することができます。訳注: th は、意味不明、the の誤りかもしれない。反対に、同じ関数は、個別のファイルにファイルからの拡張メタデータをエンコードするために“pack”オプションを提供しています。次に、個別のファイルは、内容を tar アーカイブに入れることができます。

Apple の拡張属性は、長いファイル名でひどく相互に作用することに注意してください。各ファイルが、完全な名前で格納されるので、長い名前でファイルのために必要なオーバヘッドを 2 倍にして、拡張の個別のセットをファイルごとにアーカイブに含む必要があります。

tar タイプコードの要約

次のリストは、異なる tar 実装によって生成された tar ヘッダレコードで使用されるタイプコードの要約です。特有の実装に関するより多くの詳細を、次に見つけることができます:

NUL

初期の tar プログラムは、通常ファイルのための 0 バイトが格納されました。

0

通常ファイルのための POSIX 標準タイプコード。

1

ハードリンク記述のための POSIX 標準タイプコード。

2

シンボリックリンク記述のための POSIX 標準タイプコード。

3

キャラクタデバイスノードのための POSIX 標準タイプコード。

4

ブロックデバイスノードのための POSIX 標準タイプコード。

5

ディレクトリのための POSIX 標準タイプコード。

6

FIFO のための POSIX 標準タイプコード。

7

POSIX で予約。

7

GNU tar は、いくつかのシステムであらかじめ割り付けられたファイルを使用しました。

A

Solaris tar ACL 記述は、通常ファイルヘッダの前に格納しました。

A

AIX tar ACL 記述は、ファイル本体の後に格納しました。

D

GNU tar ディレクトリダンプ。

K

続くヘッダのための GNU tar 長いリンク名。

L

続くヘッダのための GNU tar の長いパス名。

M

ファイルが前のボリュームからのファイルの継続を示す、 GNU tar マルチボリュームマーカ。

N

GNU tar の長いファイル名のサポート。廃止予定。

S

GNU tar のスパース通常ファイル。

V

GNU tar テープ/ボリュームヘッダ名。

X

Solaris tar の汎用拡張ヘッダ。

g

POSIX pax 交換形式のグローバル拡張。

x

POSIX pax 交換形式のファイルごとの拡張。

関連項目

ar(1), pax(1), tar(1)

規格

tar ユーティリティは、もはや POSIX または Single Unix Standard の一部ではありません。これが最後に登場したのは Version 2 of the Single UNIX Specification (“SUSv2”) です。これは、後の標準で pax(1) に取って代わられました。 ustar フォーマットは、現在は、 pax(1) ユーティリティの規格の一部となっています。 pax 交換ファイルフォーマットは、 IEEE Std 1003.1-2001 (“POSIX.1”) で新しくなりました。

歴史

tar コマンドは、1979 年 1 月にリリースされた Seventh Edition Unix で登場しました。これは、First Edition Unix の **tap** プログラムを置き換えた、Fourth Edition Unix の **tp** を置き換えました。 John Gilmore によるパブリックドメイン実装の **pdtar** (1987 年頃) は、大変な影響を及ぼし、 **GNU tar** (1988 年頃) の元になりました。Joerg Shilling による **star** アーカイバは、別のオープンソース (GPL) アーカイバで (最初の開発は 1985 年頃)、 pax 交換フォーマットの完全なサポート機能があります。

この文書は、 **libarchive** と **bsdtar** プロジェクトの一環として Tim Kientzle <kientzle@FreeBSD.org>によって書かれました。

December 23, 2011 FreeBSD