xbup: a set of backup tools for Mac OSX

Victor Shoup

December 2, 2008

Abstract

This document describes some simple tools for backing up a directory on a Mac OSX HFS+ filesystem to a non-HFS+ filesystem on a remote Unix machine using `rsync`. The goals of these tools are to preserve all metadata, make backups relatively fast (and in particular, bandwidth efficient), and facilitate simple restoration of files and their metadata.

1 Introduction

Suppose you would like a way to backup your critical files on your Mac OSX laptop to a remote, non-Mac file server (and from there, they may well get backed up even further). You want the backups to be easy and fast, so that you can do them often (at least once a day, if not more often).

1.1 Possible solutions

1.1.1 `rsync`

The natural tool to use is `rsync`. However, the Mac OSX file system HFS+ associates various non-standard metadata with files, including resource forks and special information for the Finder, and if you use `rsync` "out of the box", you will simply lose this metadata. Some applications actually store important information in resource forks, so this is bad.

Apple has patched `rsync` to deal with metadata (with the `-E` option), but their implementation is notoriously buggy, and moreover, does not help with backup to non-HFS+ files systems.

The very latest versions of `rsync` (version 3.x) handles HFS+ metadata fairly well, at least for HFS+ to HFS+ backups, but there are still various problems and limitations with HFS+ to non-HFS+ backups (see §2.6 below).

There are also a few different patched versions of `rsync` floating around the web that could do the job, more or less. Here are some relevant patched `rsync`s on the web:

- `http://www.quesera.com/reynhout/misc/rsync+hfsmode/`
- `http://www.onthenet.com.au/~q/rsync/`
- `http://lartmaker.nl/rsync/`

However, it is not at all clear if these are well maintained, and they all have various limitations. Also, as `rsync` evolves, these patches will invariably fall behind.

1.1.2 rdiff-backup

A program called `rdiff-backup` (see `http://www.nongnu.org/rdiff-backup/`) can be used — in theory — to obtain similar functionality to the tools provided here. However, it is not clear how well `rdiff-backup` actually handles HFS+ metadata. Some users have reported some limitations with this tool, although these may have been fixed in later releases.

1.1.3 Commercial backup tools

One reasonable commercial backup tool is *Chronosync* (`http://www.econtechnologies.com`). While it can be used to backup to a remote filesystem, the latter must be an HFS+ filesystem, and must be "mounted" in some way. Even if this works, it would not likely be nearly as fast as `rsync`.

The same issue arises with Apple's *Time machine*.

1.1.4 xbup

The tools provided here work in conjunction with any standard `rsync`. There are several commands provided, but the most important ones are `split_xattr` and `join_xattr`.

Suppose you want to backup the directory `/Users/smith/mystuff` to the remote directory `smith@access.cims.nyu.edu:/home/smith/mystuff`. Working in your home directory (`/Users/smith`), you execute:

```
split_xattr mystuff mystuff-xattr
```

This creates a directory `mystuff-xattr`, which has the same directory structure as `mystuff`, but contains special "xattr container" files that store all the non-standard metadata. So for example, if you have a file `mystuff/path/to/foo` with funny metadata, then there will be an xattr container `mystuff-xattr/path/to/foo.__@`. Note that `mystuff/path/to/foo` may itself be a directory — directories may have funny metadata too — in which case, the corresponding xattr container is `mystuff-xattr/path/to/foo/..__@`. For efficiency, if a file has no funny metadata, then no xattr container file will be generated. Also note that `split_xattr` does not add or change any files in `mystuff`.

Now you run `rsync` twice, first to backup the files, and second to backup the xattr containers:

```
rsync -avz --delete \
   mystuff/ smith@access.cims.nyu.edu:/home/smith/mystuff
rsync -avz -c --delete \
   mystuff-xattr/ smith@access.cims.nyu.edu:/home/smith/mystuff-xattr
```

Note the `-c` option in the second `rsync`. This forces a "checksum" to determine which files are to be transferred. This is the safest way to do it; however, it is still pretty safe to leave this off (see detailed discussion below). While the checksum takes time, it will typically be significantly faster than actually transmitting all the data over the network. And anyway, the time spent doing the checksums will likely be no more than the time that `split_xattr` spent generating the data.

The next time you want to backup your files, you first remove the directory `mystuff-xattr`, and run the three commands above. So a complete backup script is:

```
rm -rf mystuff-xattr
split_xattr mystuff mystuff-xattr
rsync -avz --delete \
   mystuff/ smith@access.cims.nyu.edu:/home/smith/mystuff
rsync -avz -c --delete \
   mystuff-xattr/ smith@access.cims.nyu.edu:/home/smith/mystuff-xattr
```

To restore `mystuff`, you use `rsync` to restore the directories `mystuff` and `mystuff-xattr`, and then run the command:

```
join_xattr mystuff mystuff-xattr
```

This will set the funny metadata for all files in `mystuff`, using the information contained in `mystuff-xattr`.

The commands `split_xattr` and `join_xattr` take various optional arguments to give finer-grained control over what files and what metadata are backed up and restored (see below).

There is also a Perl script called `xbup` (see below), which provides even more functionality — but feel free to "roll your own".

The *xbup* tools require version 10.4 (or later) of OSX.

2 Macintosh Metadata Madness

This section discusses the various types of metadata associated with objects in an HFS+ filesystem.

2.1 Traditional Unix metadata

- *owner* and *group*: the user and group that "own" the object.

  For backing up one's own personal files, saving this information is not so important. However, if you want or need to backup this information, `rsync` can do this. But if you are backing up to a remote file server where you do not have root access, this may not be so straightforward. The *xbup* tools can be used to work around this.

- *permissions*: the traditional read/write/execute bits on Unix filesystems.

  `rsync` can store this metadata; however, if the permissions are too restrictive, and you are backing up to a remote file server where you do not have root access, you will have problems restoring your files. Also, in HFS+, symbolic links can have their own permission bits, while on the server's filesystem, this may not be possible. The *xbup* tools can be used to work around these (somewhat esoteric) limitations.

- *mtime*: the modification time (the last time the data was modified).

  `rsync` can preserve this metadata. Besides *mtime*, there is also *atime*, the access time (the last time the data was read), and *ctime* (the last time the data or metadata changed). Most backup tools do not preserve *atime* and *ctime* — and neither do `rsync` or the *xbup* tools.

  On version 10.5 of OSX, one can set the *mtime* of a symbolic link on HFS+. Many filesystems do not allow this, and so the *mtime* of symbolic links may be lost when backing up to such a filesystem. The *xbup* tools can be used to work around this (somewhat esoteric) limitation.

2.2 xattrs (extended attributes)

xattrs are arbitrary name/value pairs associated with a file. Two special and important *xattrs* are

- `com.apple.FinderInfo`, which encodes various Finder flags, and
- `com.apple.ResourceFork`, which represents the so-called "resource fork" of a file, which may contain lots of application-specific data, like custom icons, and other stuff.

These two *xattrs* are a sort of "legacy" from pre-OSX days. In fact, "under the hood", these are not really *xattrs* in the underlying filesystem, but are made to appear as such via the `getxattr/setxattr` function interface.

Note that a *Finder Alias*, which is a Mac-ish alternative to a symbolic link that is only interpreted by the Finder (and maybe some other Mac-ish software), is a regular file with special values of the `com.apple.FinderInfo` and `com.apple.ResourceFork` *xattrs*: `com.apple.FinderInfo` tells the Finder that the file is an alias, and `com.apple.ResourceFork` contains information about the location of the target file.

On HFS+, files, directories, and even symbolic links may have *xattrs* (although a symbolic link may not have a resource fork).[1]

It seems that other modern Unix-based filesystems are starting to provide *xattrs* as well. Unfortunately, there is very little standardization on APIs, and different filesystem impose different types of limitations (on *xattr* names and sizes, and on what types of files may have *xattrs*). Indeed, while a resource fork can easily be many kilobytes or even megabytes long[2], many filesystems place restrictions of a few kilobytes on the size of an *xattr*. Indeed, even HFS+ does not allow any *xattr* other than the resource fork to be larger than a few kilobytes (as mentioned above, the resource fork is kind of a "fake" *xattr*).

The *xbup* tools automatically backup all *xattrs*, even to filesystems that have no *xattrs* themselves.

2.3 BSD Flags

There are various so-called *BSD Flags*, which are traditionally a part of BSD Unix distributions (and OSX is partly derived from BSD Unix). These flags can be changed from the command

[1] It seems that special filesystem objects, like FIFOs and devices, cannot have *xattrs*.

[2] A resource fork can apparently be no larger than 16MB.

line using the `chflags` command. The most important of these flags is the "locked" flag (also called `uchg`). The "locked" flag prevents everyone (including root) from modifying the data or metadata, or deleting the file. The "locked" flag can also be set from the OSX GUI.

On HFS+, files, directories, and even symbolic links can have *BSD Flags* (on traditional BSD Unix filesystems, symbolic links cannot have *BSD Flags*).[3]

The *xbup* tools automatically preserve the "user level" *BSD Flags* — there are also "system level" *BSD Flags*, which the *xbup* tools do not preserve (and which are quite cumbersome: you can only "unlock" a system-locked file if you boot the machine into single-user mode).

2.4 crtime (creation time)

The *crtime* represents the time an object in the filesystem was created. It seems that *crtime* is fairly unique to HFS+. If desired, the *xbup* tools can be used to backup *crtime* (although this is not the default).

2.5 ACLs (access control lists)

ACLs are a more recent addition to HFS+. These can be used to provide more fine-grained access control to files and directories than is possible with the traditional Unix *permissions*. *ACLs* were introduced in OSX Tiger (10.4), where they were turned off by default; in OSX Leopard (10.5), they are turned on by default (you can turn them on/off for a given file system using the `fsaclctl` command).

ACLs are starting to be found on numerous filesystems. Unfortunately, there is very little standardization. *ACLs* on HFS+ are meant to compatible with Microsoft's NTFS *ACLs*, and are completely unlike POSIX *ACLs* found on other Unix filesystems. In addition, unlike *ACLs* on other Unix filesystems, on HFS+, *ACLs* and traditional *permissions* are completely independent: changing one will not affect the other.[4]

On HFS+, files, directories, and even symbolic links can have *ACLs*.[5]

The *xbup* tools can be used to backup and restore *ACLs* (although this is not the default).

2.6 `rsync` metadata limitations

The latest versions of `rsync` (versions 3.*x*), using the `--fake-super` option, can backup *xattrs*, *ACLs*, and other metadata from HFS+ to non-HFS+ filesystems. It does this by storing this metadata in *xattrs* on the destination filesystem. There are several limitations with this approach:

- The non-HFS+ filesystem must itself support *xattrs* in a way that `rsync` understands. Some *Solaris* platforms (like the one used by the author of *xbup*), for example, do not support `rsync` with `--fake-super`.

[3] It seems that special filesystem objects, like FIFOs and devices, can have *BSD flags* as well.

[4] But the algorithm to grant/deny access to an object uses both.

[5] It seems that special filesystem objects, like FIFOs and devices, cannot have *ACLs*. Also, while symbolic links can have *ACLs*, version 10.4 of OSX provides no API to actually set an *ACL* on a symbolic link (see §6).

- The resource forks on HFS+ can easily exceed the size limit for *xattrs* on non-HFS+ filesystems.
- Some (most?) non-HFS+ filesystems do not support *xattrs* on symbolic links.
- *BSD flags* and *crtime* cannot be preserved — while there are patches that make `rsync` preserve this metadata, they currently only work for HFS+ to HFS+ transfers.

In addition to the above limitations, if you want to enjoy the benefits provided by additional backup on the backup server itself, you have to hope that the backup regime of the system administrator actually backs up *xattrs*, which may not always be the case.

The *xbup* tools provide a way to work around these limitations, relying on `rsync` for what `rsync` does best: preserving data and traditional Unix metadata.

3 Installation

The tarball is called `xbup-XXX.tgz`, where `XXX` is the version number.

First, if necessary, unpack the tarball:

```
tar -xzvf xbup-XXX.tgz
```

This puts all files in a subdirectory called `xbup-XXX`. Then do:

```
cd xbup-XXX
make
make install
```

Note that `make install` will copy executables into `~/bin` by default. Edit the `makefile` to change this behavior, or copy the executables by hand. Make sure your `$PATH` environment variable includes this location.

This documentation is located in the file `doc.pdf`, and the corresponding LaTeX source is in `doc.tex`.

4 Commands

NAME: `split_xattr`

Synopsis

`split_xattr` `options datadir xattrdir`

Description

This creates a directory `xattrdir`, which acts as a repository for xattr containers corresponding to files in `datadir`. The directory `xattrdir` should *not* exist prior to invocation. xattr containers will only be created for files that have "non-traditional" metadata (by default, this means *xattrs* or *BSD Flags*). If `datadir/path/to/foo` is not a directory, its corresponding xattr container (if any) is `xattrdir/path/to/foo.__@`; otherwise, it is `xattrdir/path/to/foo/..__@`. The *mtime* of an xattr container is set to the *ctime* of the corresponding file. This means that xattr containers can be safely backed up using `rsync -a`; however, `rsync -ac` (i.e., with checksums) is even safer.[6]

If you do not have effective read permission on a file in `datadir`, you will not be able to read its extended attributes: this will result in an error. Of course, if you try to backup such a file using `rsync`, this will result in an error as well. If you do not have effective read and execute permissions on a directory, you will not be able to process its contents — this will also result in an error. To avoid these problems, you may need to run `split_xattr` (as well as `rsync`) as `root` (or run: `sudo split_xattr ...`).

Symbolic links are never followed.

Special types of files (devices, FIFOs, etc.) are not given any special treatment, and are processed like any other file.

If `split_xattr` finds any files in `datadir` whose names end with "`.__@`", an error will be reported, as such file names may conflict with names of xattr containers in `xattrdir`.

Options: these control what metadata is preserved, and what files are processed.

`--files-from` `flist`

`flist` is a file that contains a list of file/directory names, which are relative to `datadir`. Only those files (and ancestors and descendents thereof) that appear in this list will be processed. Names in `flist` should be one per line, with no extra blanks and no leading or trailing slashes. Empty lines are ignored.

For example, to backup just your `iPhoto` and `iTunes` stuff in your home directory, you could place the following in `flist`:

```
Pictures
Music
```

or possibly:

```
Pictures/iPhoto Library
Music/iTunes
```

With this second list of files, if you backup your home directory `$HOME`, then the following files will be processed:

[6] It seems that deleting the special *xattr* `com.apple.FinderInfo` will not change the *ctime* of a file. This would seem to be a bug; however, if this is the only change to the metadata, then the size of the xattr container will change, and so `rsync -a` will also detect the change, since it tests both the *mtime* and *size* of a file.

- `$HOME`
- `$HOME/Pictures`
- `$HOME/Pictures/iPhoto Library`
- all files and subdirectories in `$HOME/Pictures/iPhoto Library`
- `$HOME/Music`
- `$HOME/Music/iTunes`
- all files and subdirectories in `$HOME/Music/iTunes`

The same file `flist` can be used in conjunction with the `files-from` option in `rsync` to back up the data files. Alternatively, you can get somewhat more predictable results by using the `gen_pat` command (described below) in conjunction with `rsync`.

`--crtime`
Causes creation time to be stored (by default, this is not). This will cause an xattr container to be created for each file/directory.

`--acl`
Causes *ACLs* to be stored (by default, *ACLs* are not stored).

`--owner` `ID`
Causes the owning user to be stored. If `ID` is not `-`, then as an optimzation, this information will not be stored if the owning user is `ID`. Note that `ID` can be either a symbolic or numeric ID.

`--group` `ID`
Causes the associated group to be stored. If `ID` is not `-`, then as an optimzation, this information will not be stored if the associated group is `ID`. Note that `ID` can be either a symbolic or numeric ID.

`--perms`

`--lnkperms`

`--fixperms`
These cause traditional *permissions* to be stored.

`--perms` will force *permissions* to be stored for every file/directory. This is usually not necessary, as `rsync` will preserve this metadata, except in some cases that can be dealt with using the `--lnkperms` and `--fixperms` options.[7]

`--lnkperms` causes *permissions* to be stored for symbolic links. Many filesystems do not allow symbolic links to have their own permissions, and so backing up your files to such a filesystem will loose this information. The `--lnkperms` options will solve this problem.

`--fixperms` causes "problematic" *permissions* to be stored. Problematic permissions are defined as:

[7] Backups to a truly foreign filesystem, like FAT32, may benefit from this option.

1. permissions that do not include user read and write,
2. permissions on a directory that do not include user execute, or
3. permissions that include the `setuid`, `setgid`, or `sticky` bits.

Such permissions may cause problems when using `rsync`.

Unless you have root permission on the backup filesystem, you will not be able to recover all your files if you have problematic permissions of types (1) or (2). Problematic permissions of types (1) and (2) are traditionally quite unusual; however, with the advent of *ACLs* on HFS+, they may become more common, since one can set the traditional *permissions* to deny all access, and then use *ACLs* to grant access.

Problematic permissions of type (3) can cause problems, since these special bits have different semantics and restrictions across filesystems.

The `--fixperms` option is designed to be used in conjunction with the `rsync` option `--chmod=u+rw,u-s,g-s,-t,Du+x` , which will cause all objects on the receiving end to have user read/write permission, directories to have user execute permission, and all `setuid`, `setgid`, and `sticky` bits cleared. Note that the `--chmod` option is only available on `rsync` version 2.6.7 and later.

`--mtime`

`--lnkmtime`

These cause *mtime* to be stored.

`--mtime` will force *mtime* to be stored for every file/directory. This is usually not necessary, as `rsync` will preserve this metadata, except in some cases that can be dealt with using the `--lnkmtime` option.

`--lnkmtime` causes *mtime* to be stored for symbolic links. Many filesystems do not allow symbolic links to have their *mtime* set, and so backing up your files to such a filesystem will loose this information. The `--lnkmtime` option will solve this problem.

`--recycle` `olddir`

An experimental optimization to speed things up if you have *lots* of metadata. `split_xattr` creates an xattr container for a file with an *mtime* equal to the *ctime* of the data file. Theoretically, if the metadata of a file changes, then its *ctime* should change. Unfortunately, this is not quite true (see Footnote 6). Ignoring this limitation, the `--recycle` option can be used as follows. Suppose you run `split_xattr` once, and then move `xattrdir` to `olddir`. If you later run

```
split_xattr --recycle olddir datadir xattrdir
```

then for each file, if an xattr container needs to be generated, and if a corresponding xattr container file exists in `olddir` whose *mtime* matches the *ctime* of the data file, then instead of generating the xattr container file in `xattrdir`, the file in `olddir` is *moved* to `xattrdir`.

Note: the speed-up is not as dramatic as one would hope, and given the possibility that some changes are not properly tracked, it is not clear if the use of this flag should be recommended.

NAME: `join_xattr`

Synopsis

`join_xattr` `options datadir xattrdir`

The "opposite" of `split_xattr`: xattr container files in `xattrdir` are used to set the metadata of files in `datadir`. Note that if a file in `datadir` does not have a corresponding xattr container in `xattrdir`, that file will be stripped of its *xattrs* and *BSD Flags* — more generally, the file will be treated as if its corresponding xattr container was "empty".

You will need to have effective read and execute permissions on all directories in `datadir` in order to process their contents. If this is a problem, it can be avoided by running `join_xattr` as `root` (or run: `sudo join_xattr ...`). Also, use of the `--owner` and `--group` options will require you to run `join_xattr` as `root`.

Options: these control what metadata is restored, and what files are processed.

`--files-from` `flist`
: Just as in `split_xattr`, this can be used to restrict the files in `datadir` that are processed.

`--acl`
: Causes *ACLs* to be restored. Even if `split_xattr` was called with the `--acl` option, `join_xattr` will ignore *ACLs* unless this option is set.

  The options `--numeric-ids`, `--preserve-uuids`, `--ignore-uuids`, `--usermap`, and `--groupmap` (see below) may be used to fine-tune the behavior of this option.

`--owner` `ID`
: Causes the owning user to be restored. Even if `split_xattr` was called with the `--owner` option, `join_xattr` will ignore this data unless this option is set. If `ID` is not `-`, then if a file's corresponding xattr container is either missing, or does not contain owner data, `ID` will be used as a "default" value. Note that `ID` can be either a symbolic or numeric ID.

  The options `--numeric-ids` and `--usermap` (see below) may be used to fine-tune the behavior of this option.

`--group` `ID`
: Causes the associated group to be stored. If `ID` is not `-`, then if a file's corresponding xattr container is either missing, or does not contain group data, `ID` will be used as a "default" value. Note that `ID` can be either a symbolic or numeric ID.

  The options `--numeric-ids` and `--groupmap` (see below) may be used to fine-tune the behavior of this option.

`--numeric-ids`

`--ignore-uuids`

--preserve-uuids

The --numeric-ids option modifies the default behavior of the --owner, --group, and --acl options. The --ignore-uuids and --preserve-uuids options modify the default behavior of the --acl option.

If an xattr container stores any data at all about the owning user or group of a file, it stores a numeric ID and (if possible) a symbolic ID. By default, when such data is restored to a file, the symbolic ID will be used, if it can be translated to a numeric ID (using getpwnam/getgrnam); otherwise, the given numeric ID will be used (and a warning will be issued). The --numeric-ids option will force the given numeric ID to be used.

The situation with *ACLs* is more complicated. On the HFS+ filesystem, an ACL consists of a list of entries. Each entry grants or denies access to the file by a specified entity. An entity is either a user or a group. However, an ACL specifies this entity using a UUID (universally unique identifier), which is a 128-bit string, rather than a traditional numeric user/group ID. If the UUID belongs to a user or group that is "known" to the operating system, then there will be corresponding symbolic and numeric IDs that can be used to identify the same entity.

An xattr container will encode whether the entity is a user or group, and will also encode all three forms of ID: UUID, symbolic ID, and numeric ID — the last two IDs are included only if the entity is "known" on the machine where the xattr container was created.

When restoring an ACL, the given UUID will be used if it belongs to an entity that is "known" on the machine where the ACL is being restored; otherwise, either the given symbolic ID will be tried (if --numeric-ids is not set) or the numeric ID will be tried (if --numeric-ids is set); if these IDs do not belong to any "known" entities, the original UUID will be preserved (and a warning will be issued).

This identity translation is helpful if you are trying to move your files to a new machine: users and groups (besides special ones like root and admin) will normally have distinct UUIDs on different machines (even if the symbolic and numeric IDs are the same). Indeed, if you drop your laptop on the ground and replace it with a new laptop, then you will want to move your backed up files to a new machine, and you probably do not want to preserve all the old UUIDs, which make no sense on the new machine.

The options --ignore-uuids and --preserve-uuids modify (and simplify) the default behavior. The --ignore-uuids option forces the given UUID to be ignored completely: either the symbolic or numeric ID will be tried (depending on --numeric-ids), and if this fails to translate, then a null UUID will be used. The --preserve-uuids option forces the given UUID to be used, no matter what.

--usermap map

--groupmap map

These options modify the behavior of the --owner, --group, and --acl options. The given values of map specify maps that translate user/group IDs. For example:

```
--usermap alice:sandy,bob:tom
```

specifies that when restoring user IDs (either the owning user of of file or a user in an ACL entry), if the xattr container specifies user `alice`, then user `sandy` should be used instead, and if it specifies `bob`, then `tom` should be used instead.

In general, a `map` is a list

```
id1:id2,id3:id4,...
```

You should avoid any extraneous spaces in specifying a `map`. For each pair `id1:id2`, user `id1` will be replaced by `id2` during the restore. Each of `id1` and `id2` may be symbolic or numeric IDs. If `id1` is numeric, the pair is only relevant when interpreting numeric IDs, and if `id1` is symbolic, the pair is only relevant when interpreting symbolic IDs. Note that without the `--numeric-ids` option, a numeric ID `uid` will be translated only when restoring ownership to a file where the xattr container has no symbolic ID for the file, or the symbolic ID (after any symbolic usermap translation) does not correspond to a user on the machine where the files are being restored; a warning will be printed if no usermap translation for `uid` is given: one can suppress this warning my including the usermap pair `uid:uid`.

The `--groupmap` option works the same way, except that it maps group IDs instead of user IDs.

Note that during a restore, the default owner and group (specified by the `--owner` and `--group` options) are themselves subject to usermap and groupmap translation.

NAME: `splitf_xattr`

Synopsis

`splitf_xattr` `options datadir`

Description

This works like `split_xattr`, except that instead of creating a directory structure of xattr containers, it writes a list of file name/xattr container pairs to `stdout`. One such pair is generated for each file/directory in `datadir`, regardless of whether that file/directory has any non-standard metadata.

This is especially useful for backups to local hard drives, where there is little advantage in building up a directory structure. It can run significantly faster than `split_xattr`.

Options: these options have the same meaning as the options for `split_xattr`.

`--files-from` `flist`

`--crtime`

`--acl`

--owner ID

--group ID

--perms

--lnkperms

--fixperms

--mtime

--lnkmtime

NAME: joinf_xattr

Synopsis

joinf_xattr options datadir

The "opposite" of splitf_xattr: it reads in a list of file name/xattr container pairs from stdin (as produced by splitf_xattr), and for each such pair, restores the metadata in the container to the corresponding file in in datadir. Note that the file names are relative paths, so splitf_xattr and joinf_xattr may use different datadirs. Also note that if a file listed in stdin does not exist in datadir, the xattr container is quietly skipped (this is not considered an error).

Options: these options work just like the corresponding options for join_xattr.

--files-from flist

--acl

--owner ID

--group ID

--numeric-ids

--ignore-uuids

--preserve-uuids

--usermap map

--groupmap map

Example The following will copy a directory `dir` to `dir-copy`, transferring all metadata, including *crtime* and *ACLs*:

```
rsync -a dir/ dir-copy
splitf_xattr --crtime --acl  dir | joinf_xattr --acl  dir-copy
```

To properly preserve *permissions* on symbolic links, you should add the `--lnkperms` option to `splitf_xattr`, if you are using an `rsync` prior to v3.*x*. You may also want to use the `--lnkmtime` option for `splitf_xattr`, to preserve the *mtime* on symbolic links (whether or not this option is sufficient or necessary depends on the `rsync` version and the OS version, but it never hurts to use it).

NAME: `split1_xattr`

Synopsis

`split1_xattr` `options file`

Description

This works like `split_xattr`, except that it processes a single `file` (which may be a directory, or any other filesystem object), and it writes to `stdout` a single xattr container.

Options: these options have the same meaning as the options for `split_xattr`.

`--crtime`

`--acl`

`--owner` `ID`

`--group` `ID`

`--perms`

`--lnkperms`

`--fixperms`

`--mtime`

`--lnkmtime`

NAME: join1_xattr

Synopsis

```
join1_xattr options file
```

The "opposite" of split1_xattr: it reads an xattr container stdin, and restores the metadata in the container to file.

Options: these options work just like the corresponding options for join_xattr.

--acl

--owner ID

--group ID

--numeric-ids

--ignore-uuids

--preserve-uuids

--usermap map

--groupmap map

NAME: gen_pat

Synopsis

```
gen_pat [ -x ]
```

This command reads from stdin a list of file names (in the same format as in the --files-from option in split_xattr), and writes to stdout a list of patterns suitable for use with the --exclude-from option of rsync. The -x option makes the output also include patterns that will match xattr containers, as well.

For example, if flist contains a list of file names, one can execute

```
gen_pat < flist > pat
rsync -a --delete --exclude-from=pat src/ dst
```

This will backup only those files specified in flist (including ancestors and descendents). One can also write this more compactly as:

```
gen_pat < flist | rsync -a --delete --exclude-from=- src/ dst
```

NAME: `strip_locks`

Synopsis

```
strip_locks options datadir
```

This command can be used to strip all *BSD Flags* and (optionally) all *ACLs* from files and directories in `datadir`. This is useful if you want to restore files using `rsync` into `datadir`, as `rsync` may not otherwise be able to access the files in `datadir`.

Options:

`--files-from` `flist`
: restrict processing to files listed in `flist`

`--acl`
: strip *ACLs*, in addition to *BSD Flags*

EXAMPLES

Returning to the example in §1.1.4, a more complete backup script would like like this:

```
rm -rf mystuff-xattr
split_xattr --files-from flist mystuff mystuff-xattr
gen_pat < flist > pat
rsync -avz --delete --exclude-from=pat \
   mystuff/ smith@access.cims.nyu.edu:/home/smith/mystuff
gen_pat -x < flist > xpat
rsync -avz -c --delete --exclude-from=xpat \
   mystuff-xattr/ smith@access.cims.nyu.edu:/home/smith/mystuff-xattr
```

This would restrict the files to those listed in `flist`. A complete restore script would look like this:

```
strip_locks --files-from flist mystuff
gen_pat < flist > pat
rsync -avz --delete --exclude-from=pat \
   smith@access.cims.nyu.edu:/home/smith/mystuff/ mystuff
rm -rf mystuff-xattr
mkdir mystuff-xattr
gen_pat -x < flist > xpat
rsync -avz --exclude-from=xpat \
   smith@access.cims.nyu.edu:/home/smith/mystuff-xattr/ mystuff-xattr
join_xattr --files-from flist mystuff mystuff-xattr
```

Here is a more involved backup script. Assuming you want to backup *ACLs*, owners and groups, *permissions* and *mtime* on symbolic links, and to fix problematic permissions, the script would look like this:

```
rm -rf mystuff-xattr
split_xattr --lnkperms --lnkmtime --fixperms --acl \
   --owner smith --group smith --files-from flist \
   mystuff mystuff-xattr
gen_pat < flist > pat
rsync --chmod=u+rw,u-s,g-s,-t,Du+x --rsh='ssh -i /Users/smith/.ssh/id_rsa' \
   -avz --delete --exclude-from=pat \
   mystuff/ smith@access.cims.nyu.edu:/home/smith/mystuff
gen_pat -x < flist > xpat
rsync -avz -c --delete --exclude-from=xpat \
   --rsh='ssh -i /Users/smith/.ssh/id_rsa' \
   mystuff-xattr/ smith@access.cims.nyu.edu:/home/smith/mystuff-xattr
```

Here, the default owner and group is set to `smith`, assuming most files have owner and group `smith` (this will save on bandwidth). Also, assuming you might want to run this script as `root`, the name of your SSH secret key file should be passed to `ssh` when calling `rsync`. Here is the corresponding restore script:

```
strip_locks --acl --files-from flist mystuff
gen_pat < flist > pat
rsync -avz --delete --exclude-from=pat \
   --rsh='ssh -i /Users/smith/.ssh/id_rsa' \
   smith@access.cims.nyu.edu:/home/smith/mystuff/ mystuff
rm -rf mystuff-xattr
mkdir mystuff-xattr
gen_pat -x < flist > xpat
rsync -avz --exclude-from=xpat \
   --rsh='ssh -i /Users/smith/.ssh/id_rsa' \
   smith@access.cims.nyu.edu:/home/smith/mystuff-xattr/ mystuff-xattr
join_xattr --acl --owner smith --group smith \
   --files-from flist mystuff mystuff-xattr
```

Finally, here is an example of how the *xbup* tools can be used to backup files to an external hard drive. In this script, files and subdirectories in the home directory of user `smith`, restricted to those listed in `~smith/.bupfiles`, are backed up to an external hard drive names `lacie-80`. A "live" copy, with all metadata intact, is maintained on the backup. This particular script does not preserve *ACLs*, but this could be achieved by adding the `--acl` option to the commands `strip_locks`, `splitf_xattr`, and `joinf_xattr`:

```
SRC=/Users/smith
DST=/Volumes/lacie-80/home-backup
RSYNC=rsync
FILES=/Users/smith/.bupfiles

if ! test -d "$DST"
then
   echo "can't access $DST"
   exit 1
fi

if ! test -e "$DST/data"
then
   mkdir "$DST/data"
fi

echo "*** stripping locks"
time strip_locks "$DST/data"
echo
echo "*** syncing files"
gen_pat < "$FILES" > "$DST/pat"
time $RSYNC --stats -av --delete --delete-excluded \
   "--exclude-from=$DST/pat" \
   "$SRC/" "$DST/data"
echo
echo "*** syncing xattrs"
time splitf_xattr --crtime --files-from "$FILES" "$SRC" |
   joinf_xattr "$DST/data"
```

Notice the `--delete-excluded` option to `rsync`. With this option, if you later remove some files from `.bupfiles`, these will be deleted from the backup.

NAME: xbup

Synopsis

xbup options

This is script that invokes `split_xattr`, `join_xattr`, `strip_locks`, `gen_pat`, and `rsync` appropriately, in order to perform remote backups and restores. It provides a number of conveniences:

- quick backup of just a subdirectory, or just selected files;

- automatic archival of changed/deleted files (which themselves get automatically deleted after a specified amount of time).

Before using this script, you will have to setup a configuration file `~/.xbupconfig`, which specifies (among other things) the local source directory and the remote destination host and directory. See the file `sample-.xbupconfig` included in the distribution to get started.

By default, `xbup` backs up files — you may use the `--restore` option to restore them.

Options:

`--local`
: make effective backup directory the current working directory, rather than the root of the backup tree

`--files`
: backup only those files and directories listed in the file `.bupfiles` (located in the effective backup directory)

`--files-from` `file`
: like `--files`, but use specified `file` instead of `.bupfiles`

`--config` `file`
: read config from `file`, instead of `~/.xbupconfig`

`--checksum`
: always checksum data files. By default, no transfer occurs if *mtime* and *size* of files agrees. Note that xattr containers are always checksummed.

`--dry-run`
: just a dry run. Tip: use `--checksum --dry-run` to compare source and destination.

`--restore`
: restores files, instead of backing them up. This works with all of the above options.

The archival mechanism is very simple, and is based on `rsync`'s `--backup` option. Whenever you run `xbup` to backup your files, a directory `archive/arch.XXXXX` is created in the remote destination directory, where `XXXXX` is the current time. Any changed or deleted files will be placed in a subdirectory of of `archive/arch.XXXXX`.

To find old copies of a file `path/to/foo`, run the command

```
ls -l archive/*/data/path/to/foo
```

Finding the matching xattr container for an archived file can be a bit tricky. Suppose your old file is in `archive/arch.XXXXX/data/path/to/foo`. If there is an xattr container `archive/arch.XXXXX/xattr/path/to/foo.__@`, then this is the matching xattr container. Otherwise, run

```
ls -l archive/*/xattr/path/to/foo.__@
ls -l xattr/path/to/foo.__@
```

to see all candidates in the archives and in the main backup directory. Choose the first candidate in an archive of a later date as the old file's archive, or if there are none, the candidate in the main backup directory. Note, however, that this candidate (if any) might have been first created at a time *after* the old file was moved from the main backup to the archive, meaning that the old file had no xattr container at that time. Look at the *mtime* (from the `ls` command) of the candidate: this is the time this xattr container was first created on the remote machine; if this is later than the *mtime* of the archive directory `archive/arch.XXXXX` holding the old file, then this means the old file had no xattr container; otherwise, this is the matching xattr container.

Perhaps a shell script to find xattr containers corresponding to archived files would be helpful.

NAME: `xat`

Synopsis

```
xat options file
```

This is a general utility for inspecting/modifying extended attributes of a given `file`.

Options:

`--list`
: list xattr names and their lengths

`--get` `name`
: write value of xattr `name` to stdout

`--print` `name`
: same as above, but human readable

`--del` `name`
: delete xattr `name`

`--set` `name`
: set value of xattr `name` to value read from stdin

`--set` `name=value`
: set value of xattr `name` to `value`

`--has` `name`
: test if xattr `name` exists (useful in `find` scripts)

`--has-any`
: test if any xattrs exist (useful in `find` scripts)

5 Testing

The *xbup* tools have been tested on Mac OSX Tiger (10.4) and Leopard (10.5).

Backup bouncer v0.1.3 (see `http://www.n8gray.org/code/backup-bouncer/`), which is a tool to test preservation of metadata for Mac OSX, was used as a part of the test procedure. All tests passed successfully:

```
------------------ xbup ------------------
Verifying:    basic-permissions ... ok (Critical)
Verifying:            timestamps ... ok (Critical)
Verifying:              symlinks ... ok (Critical)
Verifying:    symlink-ownership ... ok
Verifying:             hardlinks ... ok (Important)
Verifying:       resource-forks ...
   Sub-test:              on files ... ok (Critical)
   Sub-test:  on hardlinked files ... ok (Important)
Verifying:         finder-flags ... ok (Critical)
Verifying:         finder-locks ... ok
Verifying:        creation-date ... ok
Verifying:             bsd-flags ... ok
Verifying:       extended-attrs ...
   Sub-test:              on files ... ok (Important)
   Sub-test:       on directories ... ok (Important)
   Sub-test:           on symlinks ... ok
Verifying: access-control-lists ...
   Sub-test:              on files ... ok (Important)
   Sub-test:               on dirs ... ok (Important)
Verifying:                 fifo ... ok
Verifying:              devices ... ok
Verifying:          combo-tests ...
   Sub-test:  xattrs + rsrc forks ... ok
   Sub-test:     lots of metadata ... ok
```

These results were achieved using the following backup bouncer copier test script:

```
#!/bin/sh

rsync=/usr/bin/rsync # path to rsync
xattr=... # path to xbup tools


flags="-aH --rsync-path=$rsync"

# Should exit with code 0 if the necessary programs exist, 1 otherwise
can-copy () {
    test -e $rsync
}

# Should generate some text on stdout identifying which version of the
# copier is being used, and how it's called.  This is optional.
version () {
    $rsync --version
    echo
    echo "command = sudo $rsync $flags src/ dst"
}

# Should perform a copy from $1 to $2.  Both will be directories.  Neither
# will end with '/'.  So you'll get something like:
#   backup /Volumes/Src /Volumes/Dst/99-foo
backup () {
    sudo $rsync $flags $1/ $2
    sudo $xattr/splitf_xattr --crtime --acl $1 | sudo  $xattr/joinf_xattr --acl $2
}
```

To use this script, fill in the definition of `xattr`, and place the script in the directory `copiers.d` in the backup bouncer directory. In the backup bouncer directory, run `make`, and then run `sudo ./autopilot` . This will test a bunch of copiers, including the one for *xbup*.

Note that backup bouncer v0.1.3 does not check *permissions* on symbolic links (although it does check *ownership*). If the `rsync` is older than v3.*x* (which it will be by default), then it will not preserve symbolic link permissions, and you would have to add the `--lnkperms` option to the `splitf_xattr` command in the above script to do this.

6 Implementation Notes

The functions `listxattr`, `getxattr`, `setxattr`, and `removexattr` are used to access *xattrs*. The functions `getattrlist` and `setattrlist` are used to access the *crtime*. The function `setattrlist` is also used to set *BSD flags*, *permissions*, and *mtime* (see discussion below on symlinks).

6.1 Resource Forks

If one reads the resource fork of a file, the *atime* of the file is updated. Worse, if one writes the resource fork, the *mtime* is updated. The implementation works around this by restoring the *mtime* of a file after updating any *xattrs*.

Another quirk of resource forks is that setting `com.apple.ResourceFork` via `setxattr` does not replace the resource fork, but rather overwrites it. This means if you set the resource fork to `abcd`, and then set it again to `123`, its value actually is `123d`. The solution is to first remove the resource fork using `removexattr` — this is how the current implementation works.

Resource forks can be fairly big: apparently, up to 16MB. While `get/setxattr` provide a special interface for reading/writing resource forks in small chunks, the current implementation does not use this: one big buffer is allocated via `malloc`. This should be OK, as otherwise the memory usage of these programs is fairly small. Typically, resource forks are around 30-50KB (for custom icons).

Other *xattrs* are small — `com.apple.FinderInfo` is 32 bytes, and there seems to be a 4KB limit on all other *xattrs* (but that could change).

6.2 Directories

Directories can have *xattrs*, but apparently not a resource fork. Directories can also have *BSD flags*, a *crtime*, and *ACLs*. These are all preserved.

6.3 ACLs

Files, directories, and even symbolic links can have *ACLs* (but see below regarding anomalies with *ACLs* and symbolic links). The function `acl_get_link_np` is used to fetch *ACLs*. The function `acl_set_file` is used to set the *ACL* for all files other than symbolic links (again, see below). The functions `acl_delete_file_np`, etc., are supposed to delete *ACLs*; however, they simply do not work (on either 10.4 or 10.5). To delete an *ACL*, we replace a non-empty *ACL* by an empty *ACL*. To store and retrieve *ACLs* from xattr containers, one might hope to use `acl_to_text` and `acl_from_text`; however, these are buggy and can leak memory. These routines were rewritten, both to fix the bugs, and to provide greater functionality in terms of identity mapping.

6.4 Symbolic Links

Symbolic links can have *xattrs*, but apparently not a resource fork. In fact, under OSX version 10.4, all symlinks get created with a `com.apple.FinderInfo` *xattr*, with a special "type" and "creator" values. Somewhat mysteriously, this behavior has been modified under version 10.5 of OSX, so that now these values are "masked out", and symbolic links will (normally) not have a `com.apple.FinderInfo` *xattr*. If you restore files created under version 10.4 on a version 10.5 machine, these funny *xattrs* on symbolic links will appear to vanish. If you restore files created under version 10.5 on a version 10.4 machine, all of these funny *xattrs* will appear to disappear

temporarily, but (due to some funny caching phenomenon) they will come back if you reboot or remount the filesystem); in any case, this does not seem to cause a problem: the Finder will still properly interpret symbolic links, even if the "type" and "creator" values are not set properly via `setxattr` (indeed, `GetFileInfo` will still show the correct values).

The current implementation will preserve *xattrs* for symbolic links.

Symbolic links also apparently have a *crtime*. With the `--crtime` flag, `split_xattr` will store the *crtime* of the symlink. `join_xattr` will attempt to restore it (using `setattrlist`); however, on OSX version 10.4, it quietly fails, while on version 10.5 this works.

Another quirk of symlinks is that they can have *BSD flags* on an HFS+ volume. This contrary to the BSD documentation, which says they cannot. In the implementation, `lstat` is used to obtain these flags, but they are set with `setattrlist`, rather than `chflags` (there is no `lchflags`, at least on version 10.4 of OSX).

Similarly, to set the *permissions* of a symlink, `setattrlist`, rather than `chmod`, is used, because the latter always follows through symlinks (there is no `lchmod`, at least on version 10.4 of OSX).

Similarly, to set the *mtime* of a symlink, the function `setattrlist` is called, rather than `utimes`, as again, this does not follow symlinks and also does not set the *atime* – unfortunately, just as for *crtime*, this does not work on OSX version 10.4, but does work on version 10.5.

Symbolic links can even have *ACLs*. In version 10.5 of OSX, you can easily attach *ACLs* to a symbolic link in the Info panel of the GUI, by applying the security properties of a directory to its contents. However, setting the *ACL* of a symbolic link from a `C` program is trickier. The function `acl_set_link_np` is documented to do just this; however, in versions 10.4 and 10.5, it does nothing at all. In 10.5, the way one can do this is as follows: get a file descriptor for the symbolic link using `open` with the `O_SYMLINK` flag, and then pass this file descriptor to `acl_set_fd_np`. Unfortunately, there seems to be no way to get a file descriptor for a symbolic link in 10.4 (the `O_SYMLINK` flag is not supported). Any attempt to set an ACL, or clear a non-empty ACL, on a symbolic link using the *xbup* tools under 10.4 will yield an error.

6.5 Hard links

Hard links are not treated specially by `split_xattr` or `join_xattr`, but this should not cause any problems.

If the same file appears twice in a directory structure via two different hard links, then `split_xattr` will generate two different (but identical) xattr containers; if `join_xattr` restores both of these xattr containers to the same file, the metadata will be properly restored.

If you want `rsync` to preserve hard links, you have to run it with the `-H` option; however, this can be quite expensive.

6.6 Special files

HFS+ filesystems allow traditional Unix "special" file types, such as devices and FIFOs. Apparently, such special files may have *BSD Flags* and a *crtime*, but not *xattrs* and *ACLs*.

`split_xattr` and `join_xattr` do not give these special files any special treatment, and correctly preserve and restore their metadata.

`rsync` can be used to copy FIFOs to a remote server, even if you are not `root` on that server. Unfortunately, to copy devices, you need to be `root` on the remote server. Luckily, you will never run across these special file types among normal "user" files — normally, all devices are in the directory `/dev`.

6.7 ctime anomalies

As already mentioned above, changing any metadata of a file (including *permissions*, *BSD flags*, *xattrs*, and *ACLs*) should (in theory) update the *ctime* of the file. This appears to be the case in all circumstances, except for one: removing the `com.apple.FinderInfo` extended attribute via the `removexattr` function *does not*.

6.8 32-bit crtime and mtime

crtime and *mtime* are encoded as a 32-bit quantity when stored externally in an xattr container. This makes them susceptible to the "Y2038 bug".

6.9 Unicode file names

No special processing is done with respect to unicode file names; however, this should not cause a problem.

HFS+ uses UTF8 encoding of unicode characters in file names. However, it enforces a particular "normal form": if there are two ways to encode a unicode character in UTF8, HFS+ will always internally store the file name in its preferred encoding. Other Unix file systems generally allow completely arbitrary character sequences as file names, and do not enforce any particular normal form. This can cause problems when you are backing up files from a non-HFS+ filesystem to an HFS+ filesystem (but recent versions of `rsync` provide an `--iconv` option that mitigates this problem). However, there should be no problem at all backing up HFS+ to non-HFS+ filesystems, and later restoring: the remote filesystem should preserve the encoding preferred by HFS+.

7 Copying

8 Version History

Version 2.0 (December 2, 2008) extensive modifications: preservation of ACLs and ownership, more robust and reliable, improved documentation

NOTE:

- The internal format of xattr containers created by this version is not compatible with those created with versions 1.*x*.
- The naming convention of xattr containers generated by `split_xattr` and read by `join_xattr` has changed from versions 1.*x*.

For these reasons, if you are currently backing up using version 1.*x* of the *xbup* tools, you should do a full backup using v2.0. To restore xattr contaners created by versions 1.*x*, you will have to use version 1.*x* of the software.

Version 1.3 (April 13, 2008) changed `xbup_helper` script to run `mkdir` to create subdirectories as necessary...this allows one to run `xbup --local` in a newly created directory

Version 1.2 (Jan. 17, 2008) changed behavior of `splitf_xattr`, so that now every file gets an entry in the output, even if there is no metadata — this ensures that restorating using `joinf_xattr` works more like `join_xattr`

Version 1.1 (November 1, 2007) initial release