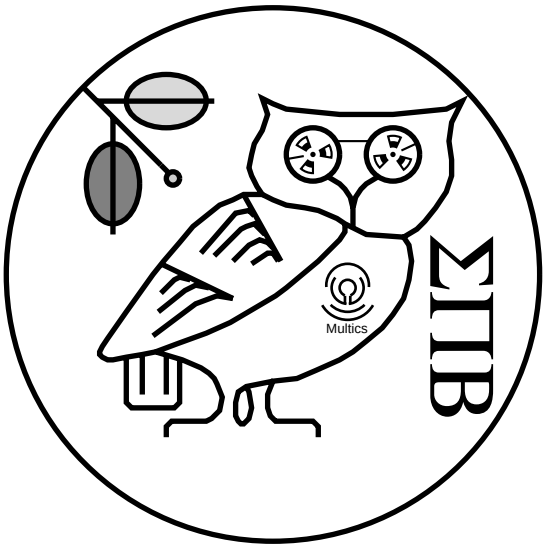

Inessential AFS†

Version 2.00

The Student Information Processing Board

Emanuel Jay Berkenbilt
Barbara Christine Manganis
Albert Dvornik
Eric Jonas

November 1 2004

†Copyright © 1992 Student Information Processing Board of the Massachusetts Institute of Technology

Contents

0.1 Filesystems: An Introduction 1

0.1.1 Why Should I Care About All This? 1

0.1.2 Carrying Your Files Around 1

0.2 UFS and NFS 1

0.2.1 File and Directory Permissions in UFS and NFS 2

0.3 AFS 3

0.3.1 The NFS to AFS migration 3

0.3.2 File Permissions in AFS 4

0.3.3 AFS Groups 5

0.4 AFS organization: Cells and Volumes 6

0.4.1 Backup and Read-only Volumes 7

0.4.2 `OldFiles` 8

0.5 Disk Quotas 8

0.6 AFS Commands 8

0.6.1 Yeah, this is all neat, but.... 8

0.6.2 fs 9

0.6.3 pts 10

0.6.4 aklog 11

0.6.5 vos 12

0.6.6 Special string `@sys` 13

0.7 Where to Go For More Information 14

0.8 Credits 15

0.1 Filesystems: An Introduction

Working on a computer involves creating and manipulating files. The data and the programs cannot be continuously stored in a computer's memory: they have to be stored in files that can be read by the computer when needed. The various methods for handling files are called **filesystems.** `AFS` stands for *Andrew File System.* It is a distributed filesystem developed at CMU in the early 1990s.

0.1.1 Why Should I Care About All This?

You don't need to know much about filesystems to start using a computer. However, with a bit of information, you can:

- let other people read your files
- let *some* other people read *some* of your files, and not the others
- get to know the basics of how filesystems actually work

0.1.2 Carrying Your Files Around

Original unix systems in the 1970s defined a hierarchal filesystem with a root node (/) underneath which all files would live. The resulting tree of files and directories had associated *permissions which governed who could read, write, and view which files.*

Traditional unix filesystems were designed to run on one computer, with multiple users accessing that one computer. Distributed filesystems (such as AFS) were developed to allow multiple users with multiple computers to access the same set of files, and to encourage sharing of files.

Instead of storing your files on a local hard disk, AFS stores files in a remote fileserver, often a fast computer with a fast network connection and lots of storage space. On Athena, all personal files are stored on fileservers and are accessible from any Athena workstation. However, since all the files have to pass through the network to get to your workstation, access may be slower than for local files.

0.2 Access Control Lists

Nearly all modern filesytems (including AFS) incorporate the notion of textitaccess control: restricting the ability to read and manipulate files to a certain set of users. An Access Control List (ACL) is associated with a file or directory and says who can do what to that file (or directory).

AFS offers a sophisticated access control mechanism. However, understanding the complexities of AFS ACLs requires a good understanding of traditional UNIX filesystem permissions

0.2.1 Traditional UNIX Permissions

Traditional UNIX file and directory access control is so basic that one often simply refers to it as "unix file permissions". [reword this] that The original UNIX filesystem defined an access model which differentiated between three types of users:

- *user (u) The owner of the file.*
- *group (g) Members of the UNIX group associated with the file*
- *(others (o) Everyone else.*

Each file had a single owner and a single group. Each class of user (u, g, or o)) can perform some subset of the following operations:

***read (r)**: Having read permission on a file lets you read the contents of the file. Read permission on a directory implies the ability to list all the files in the directory.*

***write (w)**: Write permission let you change the contents of the file (for a file) or create new files in the directory (for a directory).*

***execute (x)**: Only files which are programs should have execute permissions; these tell the operating system that this file can be run. For directories, execute permission allows you to enter the directory (i.e., `cd` into it), and to access any of its files.*[1]

Any file you create will (by default) have you as its owner, and will have its group set to the group of its parent directory.

The permissions, owner and group associated with a file or directory can be checked by looking at the output of `ls -lg`. The permissions are listed in the first field of the output. If the first character is `d`, the entry is a directory; the character `-` or the letter `f` signifies a normal file. The next three characters signify the user permissions: their values will be `r`, `w` or `x` (respectively) if the permission is granted, or `-` if it isn't. The following three characters represent the group permissions in the same way, and the last three represent others. The owner of the file is listed in the third field of the output; the fourth field lists the group associated with the file. So if a file notes looked like this:

[1] If you have "r" and not "x" on a directory you have permission to list the directory, but not to access its files.

```
-rw-r-----  1 joeuser   joepals      10460 jun 17 11:27  notes
```

it would mean that the user joeuser has read and write permissions on the file notes, the members of the group joepals have only read permission, while everybody else has none. Note that the username root signifies the ***superuser****. Anyone with the superuser privileges implicitly has read and write permissions, as well as the ability to change permissions,*[2] *on all the files on the local disks.*

The permissions can be changed by using the `chmod` *command. Granting permissions is done by* `chmod who+permissions`*, where who is any combination of* `u`*,* `g` *and* `o`*, and permissions any combination of* `r`*,* `w` *and* `x`*. Similarly, permissions are withdrawn by using* `chmod who-permissions`*. You can change the group associated with a file by* `chgrp groupname file(s)`*. For more information on the* `chmod` *command, read the corresponding manual page.*

0.2.2 AFS Permissions

Traditional UNIX filesystems let you can set permissions on a file-by-file basis. AFS is different: only directories can have ACLs. The directory is the finest level of access control in AFS. Each subdirectory in the directory tree has its own ACL.

There are seven types of access that you can grant:

Lookup (l)*: With lookup access on a directory, it is possible to look at the directory's ACL and to list the contents of the directory (i.e., what files and directories are in it). It does not imply read access to the files.* **You must have lookup permission to use any other permission** *(except administer).*

Read (r)*: Read access on a directory implies permission to read the contents of all the files in a directory. (This says nothing about the right to read files in its subdirectories.)*

Write (w)*: Write access on a directory grants permission to modify existing files and subdirectories within a directory, and to change permissions on the files in that directory. It implies neither insert nor delete access to the directory.*

Insert (i)*: Insert access on a directory implies permission to create files or subdirectories in the directory. It does not imply the ability to modify the files once they are created, however. Insert access without write access is useful mainly for the case when you want to allow someone to create files or subdirectories in a given directory but not to modify files that are already there.*

[2] The superuser can also change the ownership of a file by using the `chown` command. There are also several "special" permissions that haven't been listed here. For more information, see the `chown` and `chmod` manual pages.

Delete (d)*: Delete access on a directory gives the ability to remove files or empty subdirectories from the directory. Like insert, delete access does not imply write access.*

Administer (a)*: With administer access on a directory, it is possible to change the ACL of the directory. Administer access does not imply any other kind of access. As with all other rights, setting or resetting administer access on a directory only affects that particular directory. It does* not *affect pre-existing subdirectories.*

Lock (l)*: A user with lock access on a directory can put advisory locks on files within the directory. This is typically useful only to programmers.*

0.2.3 Subdirectory permissions

ACLs on directories are not recursive. Consider a directory `dir1` *with a subdirectory* `dir2`*, with files in* `dir1/dir2`*. Changing the ACL on* `dir1` *does not automatically change the ACL on it's subdirectory* `dir2`*. However, any newly created subdirectory will inherit the permissions of its parent direcotry.*

To change the the ACL for a file or directory, you can use the `fs` *command, mentioned in section 0.6 of this document.*

Traditional UNIX file permissions still exist for files in `AFS`*, but their meaning is different.*

The user field can be used to further restrict the access rights on the `AFS` *files and directories (Group and other fields are generally not used.) Removing the user permissions (*`chmod u-r filename`*)will deny the right of access to* all *the users who would normally have that right, including the owner of the file.*

For example, if the user read permission for a file is set (as it normally is), then anyone who has an `AFS` *read permission on the directory can access the file. However, if you remove the user read permission, then no one (including you) will be able to read the file,* even if they have `AFS` read permission on the directory.

It is also possible to give users "negative permissions" on a directory, thus specifically denying them the corresponding rights. Negative permissions are stronger than the positive ones. Their use is described in more detail in section 0.3.3 of this document.

Each directory has its own ACL. Whenever you create a new directory, it "inherits" the ACL of its parent. You always have the administer rights on the top-level directories in your locker (or any other AFS volume[3] *you own). You cannot take this right away from yourself.*

[3]For more information on `AFS` volumes, see section 0.4 of this document.

0.2.4 AFS Groups

If several users need to appear together on many ACLs, it might be easier to regulate permissions if you put them in a ***group.*** *A group is simply a list of users. You can give permissions to a group in the same way that you give permissions to users, and these permissions will then apply to all members of the group.*

You can create your own ***user groups****, which will have names of the form user:groupname, where user is your username, and groupname is a name you choose for the the group. For instance, user jis could create group jis:friends for his friends Jim, Mary and Bill, and another group jis:6.001 for his 6.001 classmates, Dave, Barbara and Tom. Then he could, for instance, give group jis:friends read and lookup permissions on most of his directories (jis trusts his friends). Group jis:6.001 could get read and lookup permissions on directories* `/mit/jis/scheme` *and* `/mit/jis/scheme/6.001`*, and read, lookup and insert on* `/mit/jis/scheme/hacks`*. And so on. For some examples of how to create and manipulate groups, see 0.6 of this document.*

If you want to give permission to everyone in a group except *someone, you can exclude them using negative permissions, described in section 0.3.2 of this document. Negative permissions deny rights to users or groups. They are stronger than positive rights, so if jis gives a read permission to jis:6.001, but denies it to Tom, who is a member of the group, Tom will not be able to read the files.*

`AFS` *groups also include* ***system groups.*** *System groups have names of the form system: groupname. The group system:anyuser includes any AFS user. Giving read and lookup permissions to system:anyuser will, therefore, make the directory* ***world-readable:*** *anyone with access to* `AFS` *(literally around the globe!) will be able to read the files. The group system:authuser includes*[4] *any user that has authenticated themselves with* `AFS` *locally*[5] *(this includes all Athena users).*

Any other `AFS` *group system:groupname corresponds to a Moira group groupname under Athena. You cannot create Moira groups without assistance. More information on Moira groups can be found in the Moira section of An Inessential Guide to Athena, available from SIPB.*

0.3 AFS organization: Cells and Volumes

The largest element in the `AFS` *structure is a* ***cell****. A cell constitutes a separate administrative domain. Each cell keeps its own list of users, groups, and system administrators. That means that a user from one cell might not exist in another cell. In that case, the user will only be able to access the files in directories that have the appropriate permission set to system:anyuser. An example of a*

[4] These are not real lists of users, but they behave as if they were.

[5] It does not include users whose authentication originated outside of MIT (e.g., CMU). Reciprocally, for files at CMU, it doesn't include users from MIT...

cell is the `athena.mit.edu` *cell (this is the cell that contains Athena user home directories, along with course lockers and most Athena software), or SIPB's own* `sipb.mit.edu` *cell.*

Each cell is made up of ***volumes****. A volume is a collection of files and directories that are grouped together and given a name. Your home directory is a volume, the volume* `user.username`*. Once created, each volume can (as a unit) be moved from one server to another, backed up, replicated or destroyed. Files and directories can be created, modified or deleted only in an existing volume.*

The whole multi-cell `AFS` *directory structure is accessible through the directory* `/afs`*.*

The volume in `/afs` *is named* `root.afs`*. The directory* `/afs` *contains the mountpoints to the root volumes for each cell, which are usually named* `cellname:root.cell`*. These volumes act like directories, and may in turn contain the mountpoints to other volumes.*

Thus you can `cd` *to* `/afs/athena.mit.edu/user/a/u/autumn`*, and be "connected to" the volume* `user.autumn` *in the cell* `athena.mit.edu`*. For the Athena cell, you can use just* `/afs/athena/...` *instead of* `/afs/athena.mit.edu/...` *since the* `root.afs` *directory contains a symbolic link from one to the other. The same goes for other* `.mit.edu` *cells, too:* `/afs/sipb.mit.edu/...` *and* `/afs/sipb/...` *are the same.*

Because of the way `AFS` *works, you do not have to explicitly attach any volume or filesystem that is on* `AFS` *in order to have access to it. All you need, in order to access a file, is the pathname of the file. For example, if I wanted to get to the games locker, I could type* `cd /afs/athena/contrib/games`*, without having to do* `attach games`*. However, this does not mean that you shouldn't, in some circumstances, attach or add a volume. For one thing, attaching a volume subscribes you to some classes of Zephyr messages, so you will be automatically notified if a portion of* `AFS` *you are using is becoming unavailable (e.g., due to a server shutdown). Some programs will not work unless certain lockers are attached. Also, volumes that are in other cells (outside the* `athena` *cell) will not recognize you or give you your proper permissions unless you are authenticated to that cell (except for volumes where the appropriate permissions are given to system:anyuser). To do this you need to get tokens for the cell, which are analogous to separate Kerberos tickets for individual* `NFS` *fileservers. Attaching or adding a volume will automatically get you tokens (read the manual pages for add and attach for more information). Another way to get tokens is to use the command* `aklog`*, described later in this document.*

0.3.1 Backup and Read-only Volumes

There are three types of volumes: ***read-write****,* ***read-only****, and* ***backup****. A read-write volume is a regular volume that can be read and written—just as the name implies. A read-write volume may have associated with it zero, one, or many read-only volumes. Read-only volumes cannot be modified by normal users. They have special properties, the most important of which is that many copies of a read-only volume can exist at once. If an* `AFS` *mountpoint is read-only and a read-only volume exists with the right name,* `AFS` *just picks one read-only volume to read from. If that volume disappears or*

somehow becomes unreachable, `AFS` will start using another one without the user ever knowing the difference. Backup volumes are also special. There can be only one backup volume for a read-write volume. Read-only volumes cannot have backup volumes. In other words, a backup volume can be associated only with a read-write volume. A backup volume is a read-only copy of a read-write volume that actually shares the same disk space as the read-write volume. *These volumes are often known as* ***clones****. When a volume is backed up, that volume initially takes a very small amount of space on disk. As the read-write volume and the backup volume get further out of synchronization, data is actually copied. The next time the volume is backed up, the old copied data is destroyed. That means that, once a volume has been backed up once, subsequent backups of the volume may actually reduce the total amount of disk spaced used!*

0.3.2 `OldFiles`

You should have a backup volume mountpoint in your home directory, called `OldFiles`. As with the volumes in the directory `/afs`, this is not an actual directory. It is a mountpoint for the volume `user.username.backup`, but for all intents and purposes it behaves like a directory. (Of course, it doesn't use up your quota.) All your files are backed up a few times every week. If you want to retrieve a file that you have accidentally removed, all you have to do is cd to the appropriate directory and use the command `cp`[6] to copy the file into your home directory.

If you don't have an `OldFiles` mountpoint, you can create one using the `fs mkmount` command. Type:

```
fs mkmount ~/OldFiles user.userusername.backup
```

For more information about the command `fs`, see section 0.6 of this document.

0.4 Disk Quotas

In `AFS`, quotas are set on a per-volume basis. That means it doesn't matter who writes into an `AFS` volume, it only has a fixed amount of space. The quota on an `AFS` volume can be examined with the `fs listquota` command (see section 0.6 of this document).

You should note that the space your `OldFiles` backup volume takes does not count against your user quota.

[6] Read the manual pages.

0.5 AFS Commands

0.5.1 Yeah, this is all neat, but....

You may be wondering how you actually do all this neat stuff, like change permissions and check quota and create groups.

0.5.2 fs

The command `fs` *is actually not a single command, but a whole group of commands*[7] *that allow you to query the fileserver and set permissions.*

One important command to know is `fs listquota`*. This returns the disk usage and quota for whatever volume you happen to be in (if there are no arguments) or for whatever volume you give it. For example,*

```
athena% fs lq /afs/sipb/project/sipb
Volume Name            Quota    Used    % Used   Partition
project.sipb.readonly  110000  109849      100%<<       87%     <<WARNING
```

This shows you the disk quota and the usage for the volume ***volume.sipb.readonly****. Note that this is a readonly volume, as described in section 0.4.1.*

Two other commands are `fs setacl`*, and* `fs listacl`*. These commands allow you to change and list the permissions on any particular directory. For example, suppose user jis wanted to make his* `scheme` *directory world-readable. To find out what the current permissions on the directory are, he would do the following:*

```
athena% fs la ~/scheme
Access list for /mit/jis/scheme is
Normal rights:
  system:expunge ld
  jis rlidwka
```

This shows that jis has all permissions on his `scheme` *directory, while the group system:*`expunge` *(the magical little demons who clean out your deleted files every once in a while) has list and delete permissions. Now, he wants give anyone the ability to read his scheme directory. While he is at it, he wants to give his trusted friend srz write permission on his scheme directory. He would type:*

[7] Many AFS commands are of this type, i.e., a suite of related commands rather than many distinct commands.

```
athena% fs sa ~/scheme system:anyuser read
athena% fs sa ~/scheme srz write
```

Note that the syntax of the `setacl` *command is* `fs sa directory who permissions`*. Note also that in this example we used the aliases* ***read*** *and* ***write****. In all there are four such aliases: read, which is the same r and l access; write, which expands to r, l, i, d, w, and k access; all, which means all kinds of access (r, l, i, d, w, k,* and *a); and none, which sets no access.*

Now jis wants to see his new list of acls, so he types:

```
athena% fs la ~/scheme
Access list for /mit/jis/scheme is
Normal rights:
  system:expunge ld
  system:anyuser rl
  srz rlidwk
  jis rlidwka
```

In addition to the `fs listacl` *and* `fs setacl` *commands, there are other* `fs` *command arguments which allow you to do everything from create mountpoints (see section 0.4.2) to query the servers. For example:*

- `fs mkmount`*, or* `fs mkm`*: As mentioned before, this command allows you to create a mountpoint, such as the one used for* `OldFiles`*.*
- `fs lsmount`*, or* `fs lsm`*: This lists information about a mountpoint, most importantly the specific name of the mounted volume. For example,* `fs lsm /afs/sipb` *gives the following:*

  ```
  '/mit/sipb' is a symbolic link, leading to a mount point for volume
    '#project.sipb'
  ```

- `fs whereis`*: This command tells you on which server a file or a directory is located. You can use it in the form* `fs whereis ~` *to find out on which server your home directory is stored.*
- `fs checkservers`*, or* `fs checks`*: This command will tell you what servers are currently down.*
- `fs help`*: This will give you information about various* `fs` *arguments and what each one does. You can also type* `fs help` `argument` *for more information about a particular argument.*

0.5.3 pts

The command `pts` *is used to contact the* `AFS Protection Server`*, an AFS server that stores all the information on* `AFS` *users and groups. You can create your own groups and add members to*

them with the commands `pts creategroup` *and* `pts adduser`*. You can remove users from groups with* `pts removeuser` *and delete groups with* `pts delete`*. You can get the members of a group with* `pts membership` *and find out information about a group with* `pts examine`*.*

Suppose jis wants to create the group called jis:6.001. He will first check to see whether the group exists. Then he will create it and add some people to it.

```
athena% pts examine jis:6.001
pts: User or group doesn't exist so couldn't look up id for jis:6.001
athena% pts creategroup jis:6.001
group jis:6.001 has id -65574
athena% pts adduser jis jis:6.001
athena% pts adduser lnp jis:6.001
athena% pts adduser cuban jis:6.001
athena% pts membership jis:6.001
Members of jis:6.001 are:
  lnp
  jis
  cuban
```

Note that jis must add himself to the group jis:6.001; you are not *by default a member of any group you create.*

Now that he has created the group, he can give its members access to his scheme directory.

```
athena% fs sa scheme jis:6.001 all
athena% fs la
Access list for scheme is
Normal rights:
  jis:6.001 rlidwka
  system:expunge ld
  system:anyuser rl
  jis rlidwka
  srz rlidwk
```

You can use `pts membership` *to get the members of most system-controlled lists*[8] *as well as of user-controlled lists. Even though* `Moira` *can give you this information as well, it is sometimes faster to get the information from* `AFS`*.*

Note that unlike `NFS` *groups, you do not have to wait for system updates for changes to* `AFS` *groups to take effect. They take effect as soon as the user re-authenticates to the cell. The easiest way to do that is to type* `renew`*, and then enter your Athena password at the* `Password:` *prompt.*

[8]The exceptions are specifically "hidden" system lists.

0.5.4 aklog

As mentioned before, you do not need to explicitly attach a volume to access its files. However, if the volume is in another cell, the cell may not "recognize" you and grant you proper permissions. You can use the command `aklog` *to identify, or* ***authenticate*** *yourself to an MIT-based cell*[9]*. Basically,* `aklog` *uses your Kerberos tickets*[10] *to get you* ***tokens*** *for the cell. To use aklog, you can type* `aklog -c cellname`*, or you can cd to a volume in the cell and type* `aklog .` *(".." is an abbreviation for the current directory). For more information, you can read the manual page for* `aklog`*. Note that if you attach or add a volume, the command* `aklog` *will be called as a part of the attach process.*

0.5.5 vos

The command `vos` *is primarily of interest to system administrators, but it does have a few functions of interest to general users. Specifically, it is used to query and manipulate volumes. The* `vos` *is in the* `afsuser` *locker. To use it, you should first* `add afsuser`*.*

The command `vos examine` *is used to find out general information about a volume. This information includes the volume identification number of the volume itself, as well as the volume identification number of the backup and read-only volumes associated with it (if they exist). It also tells what server the volume is located on and when the volume has last been released or backed up. In addition, it tells you when the volume was last changed, and what the maximum quota of the volume is. Here is an example:*

```
athena% vos examine user.autumn
user.autumn                        536956932 RW       2077 K  On-line
    CIRCE.MIT.EDU /vicepa
    Parent  536956932 Clone          0 Backup  536956934
    MaxQuota       5000 K
    Creation    Fri May 22 15:52:21 1992
    Last Update Fri Jun 26 15:42:15 1992
    1539 accesses in the past day

        readWriteID 536956932   valid
        readOnlyID  0           invalid
        backUpID    536956934   valid
    number of sites -> 1
    server CIRCE.MIT.EDU partition /vicepa RW Site
```

[9]To authenticate with a non-MIT cell, you need to have an account there. World-readable files are accessible without authentication.

[10]For more information about Kerberos, see *An Inessential Guide to Athena*, available from SIPB.

Suppose I want to know when my account was last backed up. I can use the same command on `user.autumn.backup` *to find out:*

```
athena% vos examine user.autumn.backup
user.autumn.backup                536956934 BK       2087 K  On-line
    CIRCE.MIT.EDU /vicepa
    Parent  536956932 Clone          0 Backup  536956934
    MaxQuota       5000 K
    Creation    Sun Jun 28 23:03:11 1992
    Last Update Sun Jun 28 23:03:11 1992
    0 accesses in the past day

        readWriteID 536956932   valid
        readOnlyID  0           invalid
        backUpID    536956934   valid
    number of sites -> 1
    server CIRCE.MIT.EDU partition /vicepa RW Site
```

This tells me that my backup volume was last updated on Sunday, June 28.

Another useful command is ***vos listvldb****. In the form* `fs listvldb -name volumename` *(where volumename is the name of the volume in the form given by* `fs lsm`*), this command will tell you what server the volume volumename is on (among other things), even if that server is down.*

For more information on the command `vos`*, you can type* `man vos` *at the* `athena%` *prompt.*

0.5.6 Special string `@sys`

`@sys` *is not really a command, but a special string. Since* `AFS` *is designed to look the same on different kinds of machines, the special string* `@sys`*, when it appears in a pathname, has a different meaning on each machine/operating system pair. When the system parses a pathname, it automatically replaces the string "*`@sys`*" with a string that is of the form* ***machine-type_operating-system****. For example, on a IBM RT running AOS 4.3 (as Athena RT's do),* `@sys` *has the legal value* `rt_aos4` *in the athena cell. On an Athena VAXstation,* `@sys` *has the value* `vax_bsd43`*. On an Athena DECstation, it's* `pmax_ul4`*; on an Athena RS6000, it's* `rs_aix31`[11]*. Thus the directory* `/mit/sipb/@sys` *on different machines corresponds to different directories. If I am on a VAXstation, I will end up in* `/mit/sipb/vax_bsd43`*; if I am on an IBM RT/PC, I will end up in* `/mit/sipb/rt_aos4`*. And so on.*

One use for `@sys`*, as we have seen above, is in the naming of directories to contain binary files. Binary files are programs that have been compiled and are ready to run. These files are different for*

[11] The values of `@sys` for various Athena machines can be found in the file `/afs/athena/service/systypes`.

different machine architectures; VAX binaries will not run on a Sun or a DEC or an RT. Therefore, these files are often kept in separate directories. Many lockers use `@sys` *itself as the names of these directories. Thus a single command, such as* `/afs/sipb/project/sipb/@sys/xscreensaver`*, can be used to start an application, such as xscreensaver, on any machine. However, you should note that there is no fixed convention for the naming of these directories. Although using* `@sys` *might be "most standard," some lockers use different names, such as* `bin/@sys`*,* `bin.@sys`*, or* `bin` *as a symbolic link to* `.bin/@sys`*. Since the convention has not been fully established, be aware that different lockers sometimes use different methods. Note again that attaching a locker with the command* `add` *puts the correct pathname into your path; you will be able to execute the correct binary by simply typing the filename*[12]*.*

0.6 Where to Go For More Information

`UNIX` ***manual pages****, commonly known as* ***manpages****, are an on-line copy of the documentation from the* `UNIX` *Reference Manual and other documents. They give out information on* `UNIX` *commands and most other software (including all Athena and SIPB programs),* `UNIX` *system calls, standard library functions and file formats. Manpages are an extremely useful source of information*[13]*. To access a manpage about a command, type* `man commandname`*. If you are looking for a command that does something specific you need, use the* `apropos` *command*[14]

On-Line Consultants *are provided by Athena to answer users' questions. To contact an on-line consultant, type* `olc`*.*

OLC stock answers *are a collection of consultants' answers to frequently asked questions. It can be reached by typing* `olc answers`*.*

Online Help *is an interactive help program. Enter* `olh &` *to start.*

Athena documents AFS on Athena and NFS2AFS Fact Sheet give additional information on AFS.

Commands `fs`*,* `pts` *and* `vos` *have built-in help. To list the options, type* `command help`*; to get help on a particular option, use* `command option(s) -help`*.*

SIPB office *in W20-557 has free copies of all SIPB documents, including this document and An Inessential Guide to Athena. We also answer user questions.*

[12] For more information, read the manual page for `add`.

[13] Older users will often point this fact out to the less experienced ones by using a word of sage advice, "RTFM", which stands for "Read The Friendly Manual".

[14] Further information can be found in the manpage for the `apropos` command.

0.7 Credits

This document was originally written by E. Jay Berkenbilt and revised by Barbara Manganis and Albert Dvornik. We would like to thank the people who helped us by reading over this document in its many stages and making helpful suggestions. We would also like to thank the SIPB as a whole, and in particular those members who gave us AFS tutorials and moral support late at night over Zephyr. Please send comments about this document, including corrections and suggestions for improvement, to `bug-sipb@ATHENA.MIT.EDU`*, or bring them to the SIPB office in W20-557.*