# 15 AFS File Sharing

Adapted from the ***Open AFS Guide***, *http://openafs.org/doc/*

AFS makes it easy for people to work together on the same files, no matter where the files are located. AFS users do not have to know on which computer their files are stored, and administrators can move files from computer to computer without interrupting user access. Users always identify a file by the same pathname, and AFS finds the correct file automatically. While AFS makes file-sharing easy, it does not compromise the security of the shared files.

## Client/Server Computing

AFS uses a *client-server* computing model with two types of computers. *Server* computers store data and perform services for *client* computers. Client computers perform computations for users and access data and services provided by server computers. Some computers act as both clients and servers. In most cases, you work on a client computer and access files and software stored on a file server.

## Distributed File Systems

AFS is a distributed file system that joins together the file systems of multiple file servers. A distributed file system has two main advantages over a conventional centralized file system:

* Increased availability: Copies of files can be stored on many file servers. An outage on a single server or even multiple servers does not necessarily make a file or application unavailable. Instead, user requests for the file or application are routed to accessible servers. With a centralized file system, the loss of the central file storage computer effectively shuts down the entire system.
* Increased efficiency: In a distributed file system, the workload is distributed over many smaller computers, which can be more fully utilized than the larger, and usually more expensive, file storage computer of a centralized file system.

AFS hides its distributed nature, so working with AFS files looks and feels like working with files stored on your local computer, except that you can access many more files. Also, because AFS relies on the power of the user's client workstation for computation, increasing the number of users and workstations does not slow AFS performance appreciably, making it a very efficient computing environment.

## Cells and Sites

Just like the UNIX file system, AFS uses a hierarchical file structure (a tree). Under the */afs* root directory are cells. The *cell* is the administrative domain in AFS and can be owned by a company, a university, or any defined group of users. Each cell is autonomously administered. Cell administrators determine how workstations are configured, how directories are organized, and how much storage space is available to each user. While organizing and maintaining its own file space, each cell can also connect with the file space of other cells running AFS.

The result is a huge file space that enables file sharing within and across cells. The cell to which your client computer belongs is your *local cell*. All other cells in the AFS filespace are termed *foreign cells*, such as *cmu.edu* and *umich.edu*, see following figure. An AFS *site* is a grouping of one or more related cells. For example, the *bp.ncsu.edu*, *eos.ncsu.edu*, and *unity.ncsu.edu* cells at NCSU form a single site.

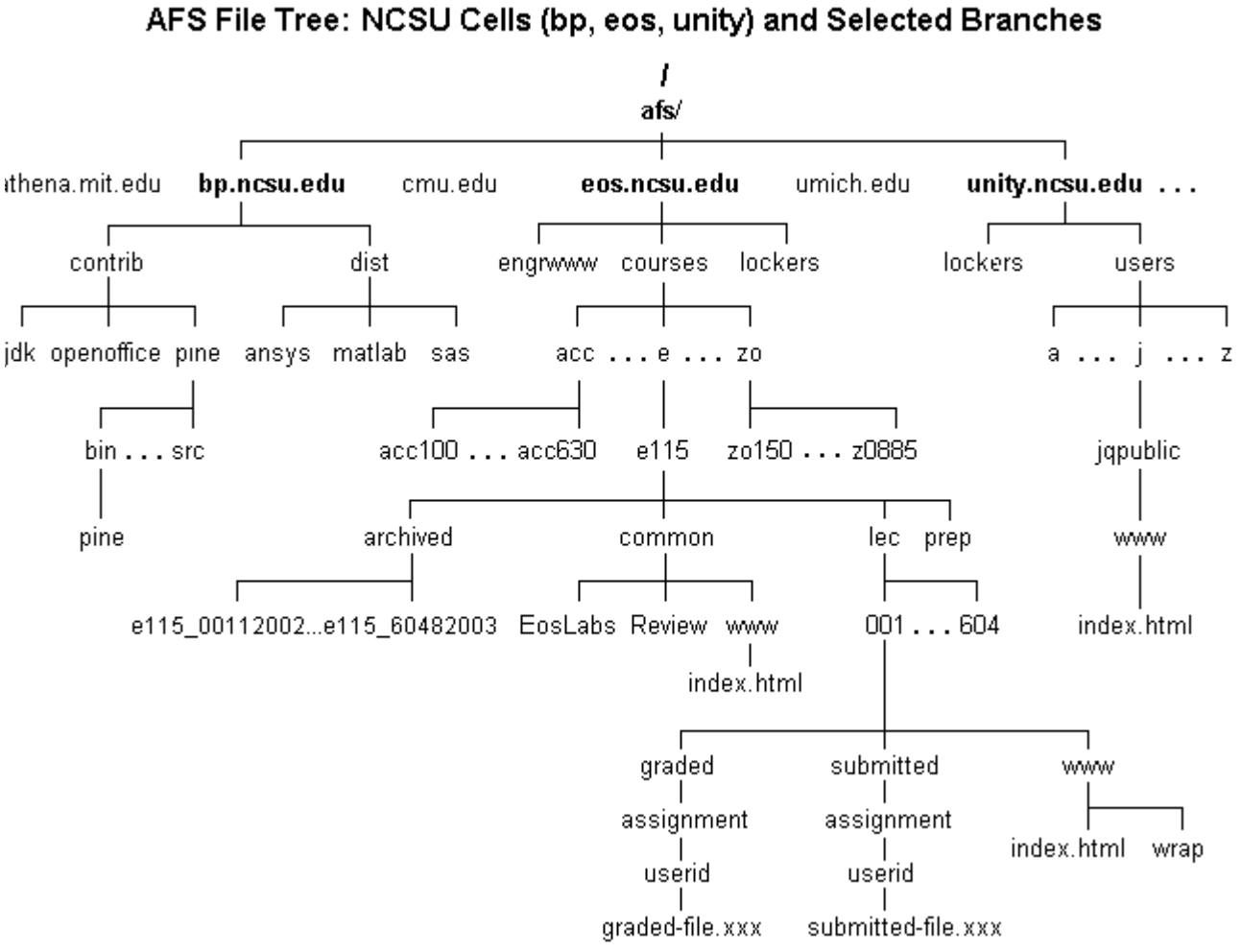
## Volumes and Mount Points

The storage disks in a computer are divided into sections called *partitions*. AFS further divides partitions into units called *volumes*.

A volume is a container for storing a subtree of related files and directories. It also has a completely independent size limit, or *quota*. Each user home directory is housed in one volume, which keeps its contents together on a file server partition. Your system administrators can move volumes from one file server to another without your noticing because AFS automatically tracks a volume's location.

AFS tracks and accesses the contents of a volume by its *mount point*. A mount point is a special file system element that looks and acts like a regular directory but tells AFS the volume's name and location. Your own volume resides on one of many file servers, and the mount point is the pointer that AFS uses to find and retrieve it for you.

For example, the volume for the user jqpublic in the unity.ncsu.edu cell is called *user.jqpublic*. A mount point exists in the */afs/unity.ncsu.edu/users/* directory named *jqpublic*. It points to the volume *user.jqpublic*. The convention NCSU follows in naming user volumes is *user.unityid*.


AFS File Tree: NCSU Cells (bp, eos, unity) and Selected Branches
/
afs/
athena.mit.edu
bp.ncsu.edu
cmu.edu
eos.ncsu.edu
umich.edu
unity.ncsu.edu . . .
contrib
dist
engrwww
courses
lockers
lockers
users
jdk
openoffice
pine
ansys
matlab
sas
acc . . . e . . . zo
a . . . j . . . z
bin . . . src
acc100 . . . acc630
e115
zo150 . . . z0885
jqpublic
pine
archived
common
lec
prep
www
e115_00112002...e115_60482003
EosLabs
Review
www
001 . . . 604
index.html
index.html
graded
submitted
www
assignment
assignment
index.html
wrap
userid
userid
graded-file.xxx
submitted-file.xxx

## Volume Quotas

Each volume has its own size restrictions, or *quota*, assigned by the system administrator. A volume's quota determines the maximum amount of disk space the volume can consume.

A volume's quota, measured in 1 kilobyte (1024 bytes) blocks, determines the storage space allowed in the volume. When users exceed their quota, they will receive error messages. As a result, users should check their quotas often by logging in at *http://sysnews.ncsu.edu/* and checking **User Info** and **Quota Manager**.

Each user's home directory (volume) is located on a disk partition with many other users. The quota command shows both percent of volume and percent of partition used, e.g.,

| Volume Name | Quota | Used | % Used | Partition |
|---|---|---|---|---|
| users.m.mcdaniel | 50000 | 25000 | 50% | 98% |

The **Used** and **% Used** of volume are the critical numbers to watch. The **Partition** does not affect your account, even if it is as high as 98%, which on a large partition still leaves ample disk space to use. However, if **Partition** reaches 100%, please contact your system administrator at *help@ncsu.edu*.

## Cache Manager

The *cache manager* is your agent in accessing information stored in AFS. When you access a file, the cache manager on your client machine requests the file from the appropriate file server machine and stores, or *caches*, a copy of it on your client machine's local disk. Application programs on your client machine use the local, cached copy of the file. This improves performance because it is much faster to use a local file than to send requests for file data across the network to the file server.

Saving your file sends the changed file back to the appropriate file server where the file is stored. In campus labs, you cannot save files to the workstation’s local drive. When you log out, the C: drive or local disk is cleared. However, because files are saved by default to your home directory in AFS, unless you change the path, your files are safely stored on network file servers, which are also backed up nightly.

Just remember to save often, which writes your data to permanent storage on the file server, and do not use the local drive for more than temporary storage.

## AFS Security and Tokens Access Control Lists

To identify yourself to AFS, you enter your Unity password to prove that you are who you say you are. When you provide this password, you become *authenticated*, and your cache manager receives a *token*.

A *token* is a package of information that is scrambled by an AFS authentication program using your AFS password as a key. Your cache manager can unscramble the token because it knows your password and AFS's method of scrambling. The token acts as proof to AFS server programs that you are an authenticated user. The token is also used for *mutual authentication*. When your cache manager contacts a file server, it also sends your token. Under mutual authentication, both parties communicating across the network prove their identities to one another. AFS requires mutual authentication whenever a server and client communicate with each other.

## UNIX and AFS

AFS is designed to be similar to the UNIX file system. For instance, many of the basic UNIX file commands (**cp** for copy, **rm** for remove, and so on) are the same in AFS as they are in UNIX.

However, AFS augments and refines the standard UNIX scheme for controlling access to files and directories. Instead of using *mode bits* to define access permissions for individual files, as UNIX does, AFS stores an *access control list* (ACL) with each directory. The ACL defines which users and groups can access the directory and the files it contains, and in what manner. The following list summarizes the differences between the two methods:

* UNIX mode bits specify three types of access permissions: r (read), w (write), and x (execute). An AFS ACL specifies seven types of access permissions: r (read), l (lookup), i (insert), d (delete), w (write), k (lock), and a (administer).
* The three sets of mode bits on each UNIX file or directory enable the user to grant permissions to three users or groups of users: the file or directory's owner, the group that owns the file or directory, and all other users. An AFS ACL, on the other hand, can accommodate 20 entries on a directory, each of which extends permissions to a user or group. Unlike standard UNIX, a user can belong to an unlimited number of groups, and groups can be defined by both users and system administrators.
* UNIX mode bits are set individually on each file and directory. An AFS ACL applies to all of the files in a directory. While at first glance the AFS method

possibly seems less precise, in actuality (given a proper directory structure) there are no major disadvantages to directory-level protections, and they are easier to establish and maintain.

* To access a file in a remote computer's UNIX file system, you must log into the remote machine or create a mount point on the local machine that points to a directory in the remote machine's UNIX file system. To access a file on a remote machine in AFS, you simply specify the file's pathname.

## Access Control Lists

AFS uses an *access control list* (*ACL*, pronounced “ackle”) to determine who can access an AFS directory and what actions they can perform on its files, e.g., read, write, administer, etc. Each directory has its own ACL, either individually defined or inherited, and up to 20 users or groups can be assigned unique rights to the space. AFS users can see and share all the files under the */afs* root directory, given the appropriate privileges.

**Remember!**

* AFS assigns permissions at the directory level, not the file level. As a result, you organize your files into directories in order to grant others access to them.
* Subdirectories inherit the ACL of the parent directory, but subdirectory ACLs can be changed by the owner to differ from the parent directory. When you grant access to a directory, you also grant access to all new subdirectories created under it. In addition, if a file is moved to a directory where the access permissions are different, the file will inherit those new settings.
* The lowest level “lookup” setting is **l** (see below), and it must be combined with the other settings for them to work. Also, a user must have **l** permission on the parent directory to reach its subdirectories. The **l** setting permits the user to move through directories to get to ones below it. Otherwise, the user will be stopped in his/her navigation with a “permission denied” error.

There are two general types of AFS commands: file server (**fs**) commands and directory protection commands (**pts**). You run them from the command line in a terminal window on both Linux and Solaris workstations.

### Levels of Access

You set permissions for directory access in the access control list (ACL). A directory's ACL is a list of users and groups and the rights they have to access and use

the files in that directory, specifically, r-Read, l-Look, i-Insert, d-Delete, w-Write, k-Lock, and a-Administer. The owner of a directory (and anyone who has administer rights) can set and manipulate the ACLs for a directory.

*Table 2: Levels of Access on AFS Directories*

| Access | Meaning |
|---|---|
| **r** | **r**ead (and copy) the contents of files in the directory. |
| **l** | **l**ook (not read access). Can list (**ls**) directory and look at its ACL. You must have **l** access to use other access rights, e.g., to read you must have **rl**. |
| **i** | **i**nsert files or subdirectories (create new files, move existing ones). |
| **d** | **d**elete files or subdirectories from the directory. |
| **w** | **w**rite or edit the contents of files in the directory. |
| **k** | **l**ock. Sets an advisory lock on a file (not used often). |
| **a** | **a**dminister or change permissions in the ACL. Owner has administer rights. |

Aliases have also been set up for common levels of access, i.e., read, write, and administer. These can be used in place of the letter abbreviations, and you can use them on both Linux and Solaris workstations.

*Table 3: Aliases for Access Settings*

| Alias | Access | Meaning |
|---|---|---|
| **read** | **rl** | read and look |
| **write** | **rlidwk** | all rights but administer |
| **all** | **rlidwka** | full owner's permissions including right to administer. Be careful giving all rights. Use write instead. |
| **none** | | remove all rights, e.g., **fa sa** *directory username none* |

## Viewing ACL Permissions

To look at the access rights on your home directory, type **cd** to return to your home directory and type the file server command, **fs la** (**f**ile **s**erver **l**ist**a**cl). By default, this command shows you the access control list for the current ( . ) directory, e.g.,

% **fs la**

```
Access list for . is
Normal rights:
www:servers l
system:administrators rlidwka
mcdaniel rlidwka
```

You can also specify a path to a directory, e.g., **fs la** */path/to/directory/*.

The output above tells you that system administrators have full rights to administer your directory, just as you do as owner of the directory (your userid would replace mcdaniel): **r**-Read, **l**-Look, **i**-Insert, **d**-Delete, **w**-Write, **k**-Lock, and **a**-Administer. The campus web servers (www:servers) also have permission to pass through your home directory to get to any subdirectories you have set up for the web, e.g., your *www* subdirectory. Remember that AFS requires that the parent directory have **l** in order to read any subdirectories below it (see ***Publishing Your Web Pages***).

There are very few people with system administrator privileges, and they are carefully screened, full-time employees of the university computing staff. It is necessary for them to have access rights in order to assist you if you have problems with your account. It is not a good idea to change or remove the administrators' permissions on your directories.

## Setting ACL Permissions

To grant someone access to a directory, you must set access to it with the **fs sa** command (**f**ile **s**erver **s**et**a**cl). Use the following command syntax to set new access rights on a directory:

**fs sa** *directory userid access*

where *directory* is the name of or path to the directory to which access is being granted, *userid* is the login name of the person to whom you are granting access, and *access* is the permission being granted to userid.

For example, if *jqpublic* wants to give *jouser* full access rights to his *~/bin* directory (except for administer rights), he would type the following at the prompt:

**fs sa** . *jouser rlidwk*
or
**fs sa** *~/bin jouser rlidwk*)

Or, he could use the alias *write* for *rlidwk*:

**fs sa** . *jouser write*

To take away or remove these rights, *jqpublic* would use the *none* access setting.

**fs sa** . *jouser none*

Fewer rights can be given than these. If *jqpublic* wants jouser to be able to read and copy his files but nothing else, he would set **rl** permission on the directory.

**fs sa** . *jouser rl*

Or, he could use the alias *read* for rl:

**fs sa** . *jouser read*

## Sharing Information with Groups

Another refinement to the standard UNIX protection scheme is that users can define their own *protection groups*, or *pts groups*. A pts group is a defined list of individual users that you can place on the ACLs of your directories. Instead of adding and removing individuals separately, you can add/remove them as a group.

A group can include both users and machines. Each user who belongs to a group inherits all of the permissions granted to the group on the ACL. AFS permits only 20 users or groups for each directory. As a result, if you want to grant access to 25 people, you could not do so unless you put them in a group.

When you create a group, you automatically become its owner. You create a group with the **pts creategroup** command (or **pts cg**):

**pts cg** *owner:group*

where *owner* is your Unity ID and *group* is a name you make up for the group. Most groups have these two parts: the part before the colon tells who owns the group, and the part after is the group's name.

Groups that you encounter that do not have an owner prefix are special groups created by system administrators. All of the groups you create must have an owner prefix and a colon before the group name.

You add a member to a group with the **pts adduser** command (or **pts ad**).

**pts ad** *userid owner:group*

where *userid* is the Unity ID of the person you want to add, and *owner:group* is the name of the group you have created. This command places that user in the group. There is no restriction on the number of members in a group.

You check who is in the group with the **pts membership** command (or **pts m**).

**pts m** *owner:group*

You remove a member from the group with the **pts removeuser** command (or **pts rem**).

**pts rem** *userid owner:group*

You delete a group with the **pts delete** command (or **pts del**):

**pts del** *owner:group*

To get a full listing of **pts** commands:

**pts help**

In the following example, *jqpublic* makes a *classproj* directory that he and three classmates (Unity IDs: *moe, larry*, and *curly*) can all work in together. *jqpublic* creates the group *jqpublic:projgroup* and adds *moe, larry*, and *curly*, to it. He adds this group to the ACL of the *classproj* directory with the **fs sa** command and gives the group write access (rlidwk).

**mkdir** *classproj*
**cd** *classproj*
**pts creategroup** *jqpublic:projgroup*
**pts adduser** *moe jqpublic:projgroup*
**pts adduser** *larry jqpublic:projgroup*
**pts adduser** *curly jqpublic:projgroup*

Or, to add all three users in one command:

**pts adduser** *-user moe larry curly -group jqpublic:projgroup*

You grant access for a group the same way you would for an individual:

**fs sa** . *jqpublic:projgroup write*

To check the membership of the group:

**pts membership** *jqpublic:projgroup*

```
Members of jqpublic:classproj (id: -1234) are:
moe
larry
curly
```

## AFS on Windows

The Windows workstations are full AFS clients, but the way they interface to the campus file system is different from Solaris and Linux. Once again, nothing is command driven in Windows. AFS control is available from the main **File -> AFS** menu on system directories. Remember that access control is set at the directory (folder) level, not on individual files.

If you have used Windows computers before, you will not have seen **AFS** on the **File** menu. It is a special customization that was done to make the Windows platform fit into the Eos/Unity AFS infrastructure so that files could be accessed and shared easily.

The following figure illustrates how to view and set access to your **K: drive**, which is mapped to your home directory in AFS. Right-clicking the **K: drive** brings up the **File** menu, and selecting **AFS** brings up a submenu of AFS functions.

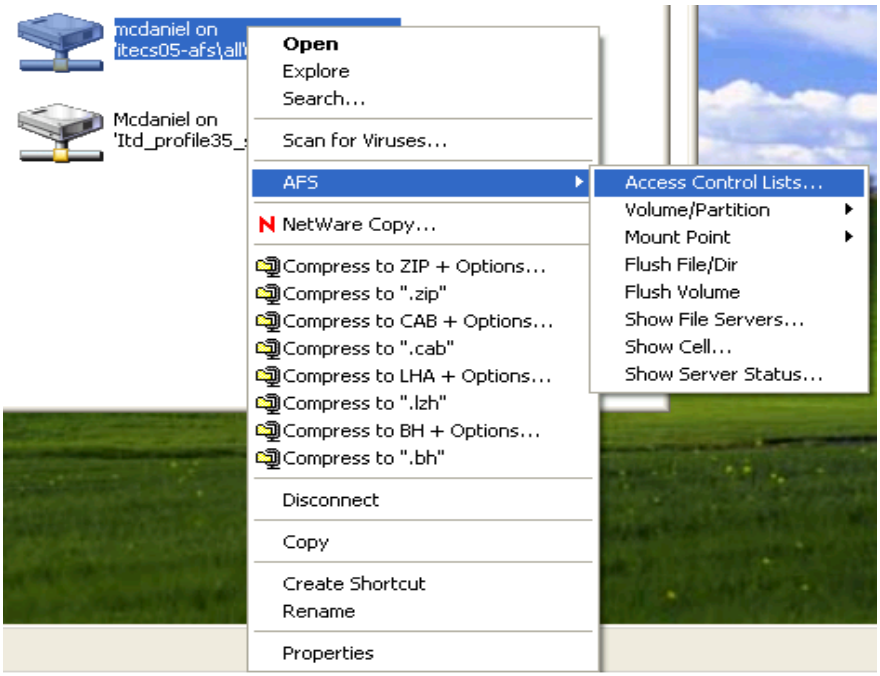
1. Select drive or folder.
2. **File -> AFS**

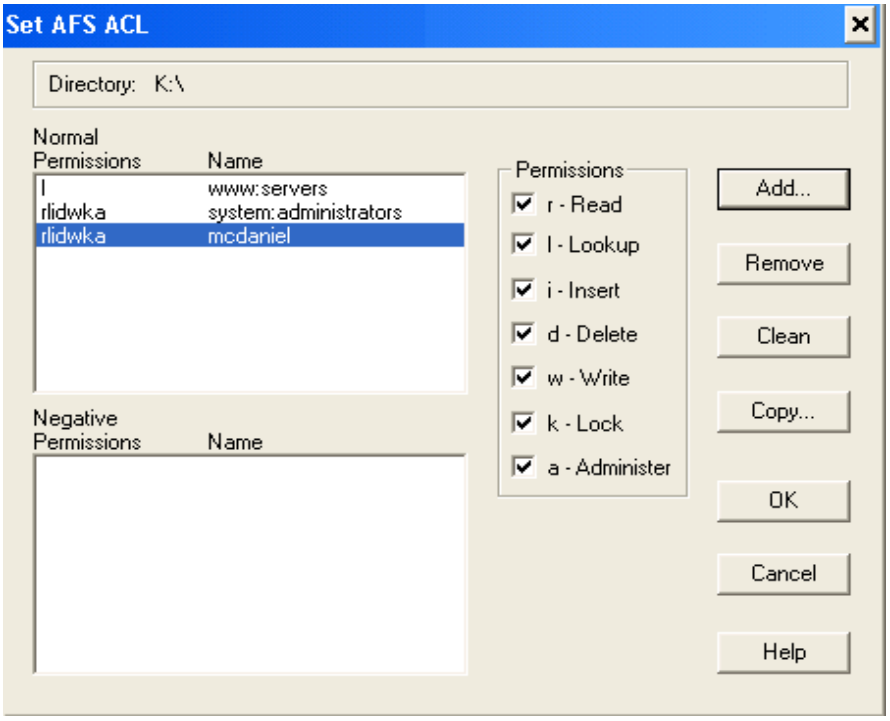
When the **Access Control List** is selected from the menu, the **AFS ACL** dialog box pops up showing the same permission settings on your home directory that you see from the command line with the **fs la** command.

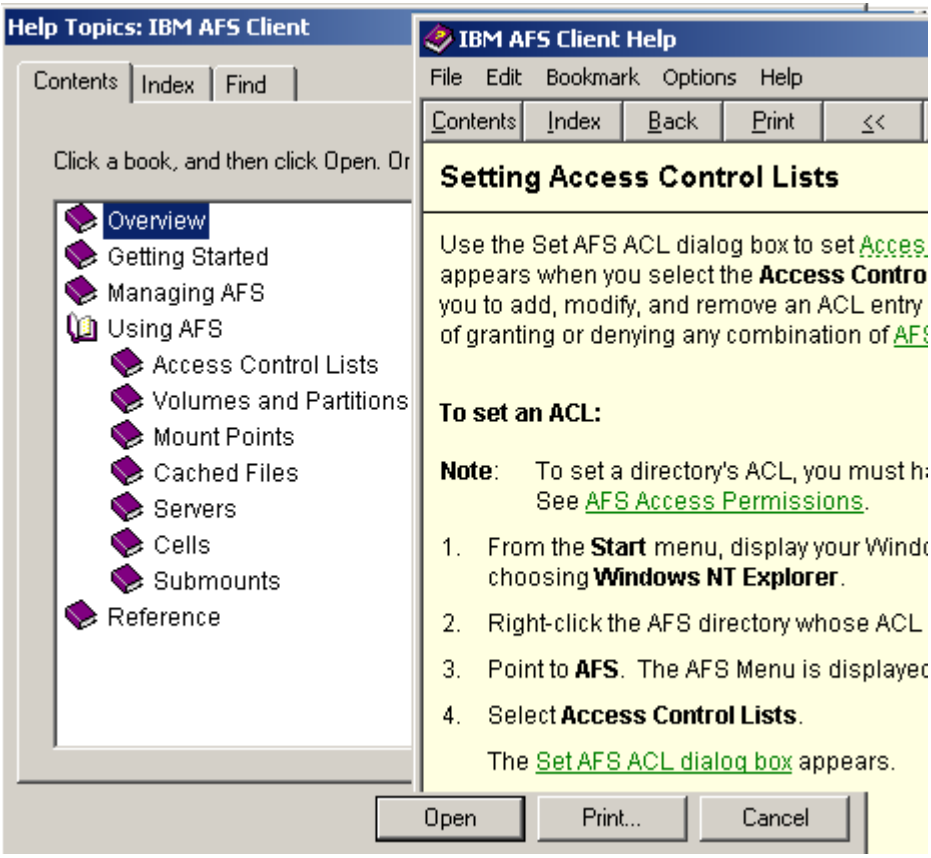
If you were to give someone read permission on your home directory or, preferably, a subdirectory, you would do the following:

1. Select **File -> AFS -> Access Control List**.
2. Type a user's Unity ID in the **Name** field.
3. Select the checkboxes for permissions (e.g., **r** and **l** for read access). Select **OK**.

Other tools and checks can also be selected from the **AFS** menu. For example, the user can check the quota on his/her home directory, or user volume, by selecting **File -> AFS -> Volume/Partition**. It is the same as typing the **quota** command on Solaris and Linux workstations, or **fs lq**.


mcdaniel on
Mcdaniel on
Open
Explore
Search...
Scan for Viruses...
AFS
NetWare Copy...
Compress to ZIP + Options...
Compress to ".zip"
Compress to CAB + Options...
Compress to ".cab"
Compress to LHA + Options...
Compress to ".lzh"
Compress to BH + Options...
Compress to ".bh"
Disconnect
Copy
Create Shortcut
Rename
Properties
Access Control Lists...
Volume/Partition
Mount Point
Flush File/Dir
Flush Volume
Show File Servers...
Show Cell...
Show Server Status...


Set AFS ACL
Directory: K:\
Normal Permissions
Name
l
www:servers
rlidwka
system:administrators
rlidwka
mcdaniel
Negative Permissions
Name
Permissions
r - Read
l - Lookup
i - Insert
d - Delete
w - Write
k - Lock
a - Administer
Add...
Remove
Clean
Copy...
OK
Cancel
Help




A **Help** system is available for the Windows AFS client with information on how to work with AFS functions and tools. It can be brought up by selecting the **Help** button on any of the AFS tool bars and dialog boxes (see figure above).

It is also possible to use the **fs** and **pts** commands from the MS-DOS command, **Start -> Programs -> Accessories -> Command Prompt**. At the command prompt, you can change to one of the AFS drives and run your AFS commands from there. Type **K:** or **J:** and press **Enter** to change to that drive.

Use the commands **dir** (not **ls**), **fs**, **pts**, **mkdir, cp**, **rm**, etc, as you are used to. Consult **Help** for more on using AFS commands at the prompt.

# OpenAFS

*The following information is reproduced in the chapter on **Remote Access Services**.*

NCSU has a large AFS network, which is used to serve user home directories, course lockers, research and project file space, and software. It is possible for users to gain access to AFS by running an **OpenAFS** client (http://www.openafs.org) on their personal computers.

The OpenAFS client, working with Kerberos, joins the file system of your local computer with the campus AFS file system. It allows you to access AFS on your personal computer in ways you are familiar with (K: and J: drives on Windows, */afs* on Unix/Linux), and you can work with the files on those drives just as you would in an Eos or Unity lab. You can open, edit, and save files as if they were on your local computer, while the client takes care of transferring them to and from the campus network. In short, the AFS file system comes to you through the OpenAFS client, and you do not have to go to a lab to have direct access to AFS.

OpenAFS is the open-source organization that maintains and distributes clients for AFS. OpenAFS provides clients for many operating systems, but the three most commonly used by students at NCSU are the ones for Windows, Linux, and Mac OS X. A high-speed connection is essential for running OpenAFS effectively. Kerberos is built in to Linux and Mac OS X, but Windows users will have to use the Kerberos for Windows software.

The following web site provides downloads, instructions for use, and configuration details for the specific client you need. At the time of this guide's publication, the AFS client for Mac Tiger (10.4) is still in development, but OpenAFS for Panther (10.3) is available. The replacement for NCSU's custom WolfCall application, whose functionality is now part of OpenAFS and Kerberos for Windows, is still in development also. As a result, the web site below will be your best and most current resource for AFS client information and downloads.

*http://www.eos.ncsu.edu/remoteaccess/afs.html*

**Important!** Establishing AFS access on your own computer is not for everyone. For the user who needs routine access to command-line tools and file transfer, the methods described in previous sections are recommended over running an AFS client, particularly if you do not have a high-speed connection. OpenAFS is useful but not essential for most users working remotely.

## AFS Glossary

**access control list (ACL)**: A list associated with an AFS directory that specifies what actions a user or group is permitted to perform on the directory and its files.

**acl entry**: An entry on an ACL that pairs one user or group of users with specific AFS access permissions. An entry can be normal, granting the user or group specific permissions, or negative, denying the user or group specific permissions.

**afs uid**: An identification number assigned to each AFS user and group. It is guaranteed to be unique.

**Andrew File System (AFS)**: A file service that joins the local file systems of several file server machines. Files are stored (distributed) on different machines in the computer network but are accessible from all machines.

**authenticated**: The state of a principal whose identity has been verified by AFS.

**authentication**: Verification that a user or process is presenting a valid identity. Authentication involves certifying that a password provided by the user is correct.

**cache manager**: The portion of an AFS client machine that communicates with AFS server processes by translating file requests made locally into remote procedure calls. It stores the requested files in a cache on the local disk, from which it makes the files available to local users.

**cell**: An administratively independent site running AFS and consisting of a set of file server machines and client machines. A machine can belong to only one cell at a time.

**file server**: A type of machine in AFS used to store files and transfer requested files to client machines.

**foreign cell**: An AFS cell other than the one to which the local (client) machine belongs. The local machine's cell is referred to as the local cell.

**local cell**: The cell to which the local client machine belongs. Even though a user can authenticate in a foreign cell or fetch files from it, the identity of the local cell remains the same throughout a logon session.

**mount point**: A special type of directory that connects a location in the AFS filespace with a volume. A mount point looks like a standard directory. Listing the directory shows the contents of the volume. Each mount point corresponds to a single volume.

**network drive**: A connection to the hard drive of a remote computer, allowing you to access shared files and directories. You can establish a network drive connection to a directory in the AFS filespace.

**partition**: A logical section of a disk in a computer.

**password**: A unique, user-defined string of characters that validates the user's system identity. The user must enter the password to become authenticated.

**quota**: The size limit of a volume assigned by the system administrator and measured in kilobyte blocks.

**token**: A set of data that is granted after a user authenticates to AFS. A token is used by the cache manager when requesting services from AFS servers. A token has an associated lifetime and expires after a set period of time. If your token expires, you no longer have authenticated access to AFS.

**username**: The name a user types in when authenticating that uniquely identifies the user in the local cell. It is mapped to the user's AFS UID.

**volume:** A "container" that keeps a set of related files and directories together on a disk partition that is specific to AFS.

**volume location server**: An AFS server process that maintains the Volume Location Database, which records location and other status information about all volumes in the cell.