Working with Files

Working with Files

All information on Athena is stored in files that are organized into directories. A file is simply a bunch of data associated with a name (the filename). Most of the things you do on Athena affect files or their contents. Papers, programs, and mail messages are all stored in files. If you are going to use the Athena system to do any substantial work, you are going to create and manipulate files.

Directories (often called "folders" on other operating systems) provide a means of organizing files into meaningful groups. When you log in, you are placed in your home directory, which contains all your files. A few directories inside your home directory have already been created for you:

- **Public** - files placed in this directory will be readable by anyone (not just at MIT, but anyone on the Internet)
- **Private** - files placed in this directory are guaranteed (unless you change the permissions) to be unreadable by anyone except you
- **www** - a location to create your personal website.
- **OldFiles** - actually, this isn't a directory at all, but a backup copy of your account from the previous night. For more information, see: Q: What is the OldFiles directory?.

Mastering the basics of the operating system means learning how to create and manipulate files (and knowing how to get out of anything you may get into). This section explains how to manipulate files. A later section, Working with Directories explains how to maneuver around directories.

File Names

Directory and filenames can be up to 256 characters in length. You can use any character on the keyboard in a filename except a slash (/), but it is best to stick to a-z, A-Z, 0-9, the period or "dot" (.), and the underscore (_). If you use any other characters, you may be setting yourself up for serious trouble. Case matters in filenames: "myfile" is a different file from "Myfile".

File Naming Conventions

Filenames often have extensions. For example, in a file named foo.txt, the ".txt" is the extension. Files have extensions for two reasons:

- An extension tells people something about a file.
- Some programs require that the files they deal with have a certain extension; programming language compilers, for instance, require that language source files have proper extensions; .c for C, and so forth.

Some file names start with a . (dot), e.g., .cshrc or .startup.X. The only significance to this is that these files do not normally show up when you ask for a list of your files. Background, utility, and "start up" files, among others, are typically named this way, so you do not usually have to see them in file listings.

Wildcards: * and ?

The system provides a mechanism called wildcard characters which lets you refer to more than one file at once.

- is one of the two most commonly used wildcard characters. It matches all files except those whose names begin with **.** (a period), or any number of characters within a filename. For example, ***.f** means "all filenames with a **.f** extension"; **h.f** and **verylongfilename.f** would both match. **a*** means "all filenames that begin with a"; **a** and **anotherfilename.c** would both match.

The other is **?**, which matches single characters. **?.f** means "all one-character filenames with a **.f** extension"; **a.f** would match, but **ab.f** would not.

Testing Wildcards with echo

One way to test out a file/directory list before you use it in an actual command is to use the **echo** command to find out what the system thinks you mean by a particular specification:

```
joeuser@athena:~$ echo specification
```

The **echo** command simply types out its arguments. When it is run, all wildcards are expanded into full filenames if any matching files exist. The actual files themselves are not touched.

For instance, suppose these were the contents of a directory:

```
joeuser@athena:~$ ls
Cold
Hot
```

```
temps
temps.old
```

If you wanted to remove the old version of the temps file (temps.old) using an abbreviated form, you could first test your file specification using the **echo** command:

```
joeuser@athena:~$ echo *old
Cold
temps.old
```

Here you would find that your directory specification did not match what you intended. You could try again with a more precise specification:

```
joeuser@athena:~$ echo *.old
temps.old
```

Now that you have confirmed that the specification refers to the files you want, you can use that specification in the actual command (such as delete) that you intended to use with confidence that the files referred to are the appropriate ones!

Creating and Displaying Files with the cat Command

The **cat** (for "concatenate") command displays the contents of a file (or any number of files) on the screen. Its command format is:

```
cat filename
```

For example, you can read the "welcome" file in your account (if you haven't already deleted it) with the following command:

```
joeuser@athena:~$ cat ~/welcome
```

The contents of the file are shown on the screen. If the file is very long, its contents will scroll past you faster than you can read them. You can type **Ctrl-s** to stop the screen from scrolling, and **Ctrl-q** to resume scrolling. To cancel a **cat** command and stop its output, type **Ctrl-c**.

If you don't specify a file, the **cat** command will wait for input from the keyboard. Using the concepts of I/O Redirection discussed earlier, this can be used as a quick way to create a small file. With this method, you just type the contents of the file at the keyboard. You can fix typing mistakes in the current line by deleting them; you can erase the current line by typing **Ctrl-u**, but you cannot change previous lines. When you are done, you finish the last line with a **Return** and then type **Ctrl-d** (it is displayed as **^D**). For example:

```
joeuser@athena:~$ cat > shopping-list
Diet Coke
Ramen
Hot Pockets (pepperoni flavor)
Mac-and-Cheese
^D
joeuser@athena:~$
```

Displaying Files With the more Command

The **more** command is another way of looking at a file. It displays a file's contents one screen at a time, pausing after every screenful so that you can read each screen. To look at a file with **more**, type:

```
more filename
```

Once **more** has filled up the screen, you must give it a subcommand to tell it to continue. There are many such subcommands. Here are the most commonly used commands and their actions:

- **Spacebar:** move forward one screenful
- **Return:** move forward one line
- **b:** move back one screenful
- ***/ string* Return:** search forward for *string*
- **q or Ctrl-c:** quit
- **?:** help

more is clever enough not to display binary files and directories.

The **more** command is often used with a pipe to view commands that generate multiple screens of output. It is frequently referred to as a "pager" because it displays multiple "pages" (or screens) of information. Using a pipe, you can use **more** to display really long directory listings so you can view them page by page:

```
joeuser@athena:~$ ls | more
```

Copying a File (cp)

To make a copy *tofile* of an existing file *fromfile*, use the **cp** command:

```
cp fromfile tofile
```

If **cp** copies the file successfully, it returns you to the prompt. At this point, two identical copies of the file exist. The file specifications can be relative names or full pathnames. The destination can also be a directory instead of a file. In that case, the **cp** command will place a copy of the file in the directory in question. The following example demonstrates both cases. The first command will make a copy of the PARTY file and name it PARTY.backup. The second command will make a copy of the PARTY file and place it in the directory called Public:

```
joeuser@athena:~$ cp PARTY PARTY.backup
joeuser@athena:~$ cp PARTY Public/
```

Be careful: if a *tofile* file already exists with the name you specify, **cp** overwrites the file without asking you (This does not apply if *tofile* is a directory). To avoid this problem, use **cp** with the **-i** switch; in this case, the system asks you whether you really want to overwrite the existing file before it tries to copy. The original *fromfile* is not affected by the **cp** operation in any case.

Moving or Renaming a File (mv)

To move or rename a file, use the **mv** command:

```
mv fromfile tofile
```

The system renames *fromfile* as *tofile*, in effect moving it from one location to another (unlike the **cp** command, there is still only one version of the file when you use **mv**). Like **cp**, the *tofile* can be a file or directory:

```
joeuser@athena:~$ mv PARTY STUDYING
joeuser@athena:~$ mv PARTY Private/
```

Like **cp**, if a *tofile* file already exists with the name you specify, **mv** overwrites the file without asking you. To avoid this problem, use **mv** with the **-i** switch – in this case, the system asks you whether you really want to overwrite the existing file.

Whether or not **mv** renames the file successfully, it returns you to the Athena prompt.

Listing the Files in a Directory (ls)

The **ls** command lists a directory's contents. Suppose you are in your home directory. If you are a new Athena user, **ls** might return something like the following:

```
joeuser@athena:~$ ls
```

Mail	OldFiles	Private
Public	welcome	www

(Actually, the directory contains other files, but **ls** does not show the others because they are "dotfiles", files with names that are prefixed with a period.)

If you have had an Athena account for a while, you no doubt have other files and directories in your home directory, which would be listed by **ls**.

The **ls** command by itself lists just the filenames, alphabetically (A-Z before a-z) in as many columns as will fit across the screen. The **ls** command has many options. To see them all, use the **man** command to look at the online manual page for **ls** by typing **man ls**. This section discusses some of the more useful ones.

To get a list of all of the files in a directory, including those whose names begin with a . (dot) character, use the **-a** option (for "all"):

```
joeuser@athena:~$ ls -a
```

.	.cshrc	Mail	OldFiles	Private
..	.login	Public	welcome	www

The file . always refers to the current directory, in this case your home directory /afs/athena.mit.edu/user/ *first-letter*/ *second-letter*/ *username*. The file .. always refers to the current directory's parent directory, in this case your home directory's parent /afs/athena.mit.edu/user/ *first-letter*/ *second-letter*. (See the section Working with Directories for more information about what these directory pathnames signify.)

To get a list of your files that shows more information about them, type **ls -l**:

```
joeuser@athena:~$ ls -l
```

drwx------	2	joeuser	mit	2048	Aug 18	17:00	Mail
drwx------	2	joeuser	mit	2048	Aug 18	17:00	OldFiles
drwx------	2	joeuser	mit	2048	Aug 18	17:00	Public
drwx------	2	joeuser	mit	2048	Aug 18	17:00	Private
-rwxr--r--	1	joeuser	mit	1915	Aug 18	17:00	welcome
drwx------	2	joeuser	mit	2048	Aug 18	17:00	www

The following table summarizes the parts of the **ls -l** output.

Element	Example	Definition
mode	rw-r--r--	file's access permission modes
links	1	number of links the file has (for directories, this is how many subdirectories exist "beneath" the file, which is always at least two: itself and its parent).
owner	jruser	username of the user who owns the file (in most cases, your username).
group	mit	The name a group to which you can assign permissions (this doesn't apply to files in AFS)
size	1915	size of file in bytes (for text files, equals number of characters in file).
modify-time	Aug 18 17:00	date and time when file was last modified (if file has never been modified, date file was created).
name	welcome	actual filename.

There are different types of files: simple files (text files, binary files, or shell scripts), and directory files. Often you need to know the type of the files you are listing. The **-F** option shows this information (this example shows a directory of someone who's used Athena for a while):

```
joeuser@athena:~$ ls -F
```

Mail/	OldFiles/	Private/	Public/
myprogram*	welcome	www/	

Notice the suffix characters (/ and *) following some of the filenames. These characters are not part of the filename, but give information about the type of file:

- **(no suffix):** regular file
- **/:** directory
- ***:** executable binary file or a shell script
- **@:** symbolic link to another file
  As with many commands, you can combine **ls** options to use more than one at the same time. For example, to get a long listing of all the files in a directory, type **ls -la**.

Removing a File (delete and rm)

It is a good idea to delete files that you no longer need. This can help keep your directories from becoming too cluttered, and keep you from exceeding your allotted disk quota.

On Athena, there are two ways to get rid of unwanted files: **delete** and **rm** (remove). The **delete** command differs from the **rm** command in that **delete** is not necessarily permanent. When you use **rm** to remove a file, the file is erased from the system immediately and permanently; when you use **delete** to remove a file, the file is removed in such a way that you can recover the file (within about three days) before it is permanently eliminated from the system.

Completely erasing the file from the system is usually what you want, but once in a while you may accidently remove a file you wanted to keep – the manuscript for a paper that's due the next morning, for example, or part of your thesis! Because of this possibility, it is a good idea to use only the **delete** command; this helps you avoid mistakenly losing any files.

Using Delete and Related Commands

The commands that let you delete, recover, or remove files are summarized in the following table. (For more information about each command, including command-line switches, see the online man pages; for example, type **man delete** for details about the **delete** command.)

- **delete:** Mark one or more files for permanent removal, making them invisible to the user (by renaming them with the prefix .#) but not

actually erasing them from the system (use **expunge** or **purge** to permanently erase files marked for deletion).
- **undelete:** Restore files marked for removal from current directory by **delete** (if not already expunged).
- **lsdel:** List files marked for removal but not yet expunged.
- **expunge:** Permanently remove specific files marked for removal.
- **purge:** Permanently remove every file marked for removal in user's home directory and all subdirectories.

If you accidentally **delete** a file and then realize that you want it back, you can get it back by using **undelete**.

Because of the way **delete** works, deleting files does not actually lower the amount of quota you are using (each file is simply renamed to a form that is invisible to your normal work, specifically from *filename* to *.#filename*). To lower your used quota, you must fully remove the deleted files from your system by using the **purge** or **expunge** commands.

For example, suppose you have a directory containing the following files:

```
advisor notes thesis.tex thesis.tex~
```

Because you are near your quota, you decide to remove the old version of your manuscript file (the one ending with ~) to create some room. However, you accidentally leave off the ~ from your command and thereby remove the newer version of the file from the directory:

```
joeuser@athena:~$ delete thesis.tex
joeuser@athena:~$ ls
advisor notes thesis.tex~
```

If you had used **rm** to do this, you would not be able to recover the lost file, and would instead have to salvage what you could from the older file. However, because you used **delete** instead of **rm**, you can recover the deleted file by using **undelete**:

```
joeuser@athena:~$ lsdel
thesis.tex
joeuser@athena:~$ undelete thesis.tex
joeuser@athena:~$ ls
advisor notes thesis.tex thesis.tex~
```

You can now remove the appropriate file, and even permanently eliminate it once you verify that you have marked the correct file for removal:

```
joeuser@athena:~$ delete thesis.tex~
joeuser@athena:~$ ls
advisor notes thesis.tex
joeuser@athena:~$ lsdel
thesis.tex~
joeuser@athena:~$ expunge thesis.tex~
joeuser@athena:~$ lsdel
joeuser@athena:~$ ls
advisor notes thesis.tex
```

Note that the **undelete** command only retrieves files removed with the **delete** command – it cannot retrieve files eliminated by **rm**. In addition, you *cannot* retrieve a deleted file that has been removed by **purge** or **expunge**. That is, you cannot **undelete** a file once it has been purged.

You can set up your system so that **delete** (rather than **rm** or **rmdir**) is automatically used whenever you want to remove files. Just put the following lines in the .cshrc.mine file in your home directory (which you can create if it does not exist):

```
alias rm delete -F
alias rmdir delete -D
```

Then when you type **rm** to get rid of a file or **rmdir** to get rid of a directory, **delete** is actually used.

Using the rm Command

The **delete** package is Athena-specific. The standard UNIX utility for removing files is **rm**, and it exists on Athena in addition to **delete**. (Athena recommends that you use **delete** rather than **rm**, since files that you **rm** accidentally usually cannot be recovered.)

To remove a file using the **rm** command, just type a command of the form:

```
rm filename
```

The **rm** command does not verify the deletion; it simply returns you to the joeuser@athena:~$ prompt. Because **rm** removes files permanently without confirmation, it is a very good idea to use the **-i** option with **rm**. The **-i** option stands for *interactive*; with this option, **rm** asks you to confirm the deletion you are performing.

Before you **rm** something, remember the old Unix adage: "**rm** is forever."

Summary of File/Directory Commands

The following commands let you list, examine, create, delete, copy, and rename files and directories:

- **ls**: list contents of directory
- **cat**: catenate and display file(s)
- **more**: display contents of file one screenful at a time
- **tee**: pipe copy of output into file
- **cp**: copy file/directory
- **mv**: move (rename) file
- **delete**: mark file/directory for later permanent removal
- **expunge**: permanently remove files marked for deletion
- **lsdel**: list files marked for deletion
- **purge**: permanently remove files under ~ marked for deletion
- **undelete**: recover files marked for deletion but not yet removed
- **rm**: permanently remove file
- **pwd**: display name of current working directory
- **cd**: change to the specified directory
- **mkdir**: create new directory
- **rmdir**: remove empty directory
- **echo**: displays the typed text, expanding wildcards