mtime, ctime, and atime

unix.com/tips-and-tutorials/20526-mtime-ctime-atime.html

mtime, ctime, and atime

Unix keeps 3 timestamps for each file: mtime, ctime, and atime. Most people seem to understand atime (access time), it is when the file was last read. There does seem to be some confusion between mtime and ctime though. ctime is the inode change time while mtime is the file modification time. "Change" and "modification" are pretty much synonymous. There is no clue to be had by pondering those words. Instead you need to focus on what is being changed. mtime changes when you write to the file. It is the age of the data in the file. Whenever mtime changes, so does ctime. But ctime changes a few extra times. For example, it will change if you change the owner or the permissions on the file.

Let's look at a concrete example. We run a package called Samba that lets PC's access files. To change the Samba configuration, I just edit a file called smb.conf. (This changes mtime and ctime.) I don't need to take any other action to tell Samba that I changed that file. Every now and then Samba looks at the mtime on the file. If the mtime has changed, Samba rereads the file. Later that night our backup system runs. It uses ctime, which also changed so it backs up the file. But let's say that a couple of days later I notice that the permissions on smb.conf are 666. That's not good..anyone can edit the file. So I do a "chmod 644 smb.conf". This changes only ctime. Samba will not reread the file. But later that night, our backup program notices that ctime has changes, so it backs up the file. That way, if we lose the system and need to reload our backups, we get the new improved permission setting.

Here is a second example. Let's say that you have a data file called employees.txt which is a list of employees. And you have a program to print it out. The program not only prints the data, but it obtains the mtime and prints that too. Now someone has requested an employee list from the end of the year 2000 and you found a backup tape that has that file. Many restore programs will restore the mtime as well. When you run that program it will print an mtime from the end of the year 2000. But the ctime is today. So again, our backup program will see the file as needing to be backed up.

Suppose your restore program did not restore the mtime. You don't want your program to print today's date. Well no problem. mtime is under your control. You can set it to what ever you want. So just do:

Code:

```
$ touch -t 200012311800
employees.txt
```

This will set mtime back to the date you want and it sets ctime to now. You have complete control over mtime, but the system stays in control of ctime. So mtime is a little bit like the date on a letter while ctime is like the postmark on the envelope.

find command -mtime -ctime -atime

The find command uses arguments like:
-mtime -2
-mtime +2
-mtime 2

There are -ctime and -atime options as well. Since we now understand the differences among mtime, ctime, and atime, by understanding how find uses the -mtime option, the other two become understood as well. So I will describe find's use of the -mtime option.

As you probably know, the find command can run for minutes or hours depending on the size of the filesystem being searched. The find command makes a note of its own start time. It then looks at a file's mtime and computes how many seconds ago the file was modified. By dividing the seconds by 86,400 (and discarding any remainder), it can calculate the file's age in days:

Code:

```
0 days in seconds:       0  -
86399
1 day in seconds:    86400  -
172799
2 days in seconds:  172800  -
259159
```

So now that we know how many days ago a file was modified, we can use stuff like "-mtime 2" which specifies files that are 172800 to 259159 seconds older than the instant that the find command was started.

"-mtime -2" means files that are less than 2 days old, such as a file that is 0 or 1 days old.

"-mtime +2" means files that are more than 2 days old... {3, 4, 5, ...}

It may seem odd, but +0 is supposed to work and would mean files more than 0 days old. It is very important to recognize that find's concept of a "day" has nothing to do with midnight.

Last edited by Perderabo; 08-05-2007 at 11:40 AM..

Using perl to display the file timestamps

The ls program will display mtime if you use "ls -l". And you can get atime or ctime with "ls -lu" or "ls -lc". But ls uses a strange format. It displays the month and day in all cases. If the timestamp is recent, it also displays hour and minute. If the timestamp is older than 6 months, it display the year instead of hour and minute. A clever script can reformat this to year, month, day, hour, and minute. But ls will not display the seconds. The gnu version of ls (which is usually the only version on linux) does have extended options like --fulltime. But these extended options are non-standard and won't be available on other versions of Unix.

The perl language is also non-standard, but perl tends to be available on most versions of unix. For example, a version of perl is supplied with HP-UX and Solaris. Perl can easily display the timestamps of files. Here are some perl one-liners to display atime, mtime, and ctime.

Code:

```
$ echo hello > testfile ; date
Thu Aug 30 08:31:57 EDT 2007
$ chmod 700 testfile ; date
Thu Aug 30 08:32:48 EDT 2007
$ cat testfile ; date
hello
Thu Aug 30 08:33:30 EDT 2007
$
$
$
$
$ perl -e '@d=localtime ((stat(shift))[8]); printf "%4d%02d%02d%02d%02d%02d\n",
$d[5]+1900,$d[4]+1,$d[3],$d[2],$d[1],$d[0]' testfile
20070830083330
$ perl -e '@d=localtime ((stat(shift))[9]); printf "%4d%02d%02d%02d%02d%02d\n",
$d[5]+1900,$d[4]+1,$d[3],$d[2],$d[1],$d[0]' testfile
20070830083157
$ perl -e '@d=localtime ((stat(shift))[10]); printf "%4d%02d%02d%02d%02d%02d\n",
$d[5]+1900,$d[4]+1,$d[3],$d[2],$d[1],$d[0]' testfile
20070830083248
$
```