UNIX tips and tricks for a new user, Part 1: File maintenance tools

Skill Level: Intermediate

Tim McIntire (tm@timmcintire.net)
Consultant
Freelance Writer

26 Sep 2006

Systems administrators can use a number of programs to maintain files in a UNIX® system from the command line. In this tutorial, you'll experiment with commands, such as `cd`, `cp`, and `tar`, to navigate a UNIX file system from the command line and work with files and directories. The `cd` command changes directories, `cp` duplicates files or directories, and `tar` quickly groups files into an archive. You'll also learn how to deal with file permissions and perform simple input/output.

Section 1. Before you start

Learn what to expect from this tutorial, and how to get the most out of it.

About this series

This four-part tutorial series covers UNIX® basics from a user perspective. This initial tutorial is a good brush-up for users who have been away from UNIX-like operating systems for some time. It's also useful for brand-new UNIX users coming from a Windows® background, because it uses references and comparisons to Windows. Later tutorials in the series will cover specific applications (`vi`, for instance) in detail and discuss shell tricks and tips.

About this tutorial

Systems administrators can use a number of programs to maintain files in a UNIX system from the command line. More exist than the basic ones demonstrated here, but they're all fundamental parts of your UNIX system. You can use the `mv` command to reposition and rename files and directories. The `cp` command

duplicates one or more files or directories. An application called `tar` can quickly group files together into a single archive. This tutorial also discusses traversing the file system, dealing with file permissions, and simple input/output so that new UNIX users have a base to work from.

Objectives

The objective of this tutorial is to make new UNIX users comfortable moving around on the command line and dealing with files. It focuses on common command-line utilities that manipulate files, but file permissions and input/output are also discussed to give you a complete picture of commands you need to use on a day-to-day basis.

Prerequisites

You need a basic understanding of computers and files for this tutorial, but no experience in a UNIX-like operating system is expected. You should understand what directories (folders) and files are and be able to log in to your account on a UNIX-like operating system. If you're familiar with the DOS or Microsoft® Windows command line, you'll notice similarities, but users who have never used any sort of command line will do just fine.

System requirements

Access to a user account on any computer running any UNIX-like operating system is all you need to complete this tutorial. UNIX-like operating systems include the IBM AIX® operating system, Linux®, Berkeley Software Distribution (BSD), and Mac OS® X (using Terminal to access the command line), among many others.

Section 2. To begin

One quick caveat before you begin: Many different UNIX-like distributions are currently available, ranging from commercial distributions, such as AIX, to free distributions, such as BSD and Linux. This tutorial focuses on commands and command-line options that are available in just about every distribution; if you find differences in your distributions, check your man pages (as described later).

Log in to the UNIX-like operating system of your choice to get started. When you log in, you should automatically start in your user's home directory. The examples use the `tuser` (Test User) username.

man

Before you work with specific commands, it's important to learn about `man`. `man` stands for manual; it's a critical tool for UNIX users who pride themselves on being self-sufficient. Type `man application-name` to see an explanation of anything you want to learn about. At each page in this tutorial, you're encouraged to check the man page along with the instructions.

Try typing the following (note that $ in this tutorial refers to your command prompt; in the examples, you should type what you see after the dollar sign):

```
$ man ls
```

Section 3. Directories

The first thing you need to learn is how to traverse and examine your file system. In UNIX, directories are used to organize files in a hierarchical structure. Rather than clicking directories to enter them and seeing each file as an icon, you use a series of commands and lists to view and traverse a UNIX file system from the command line.

ls

If you're using a UNIX-like operating system for the first time but you've used a DOS or Windows command line in the past, `ls` is roughly equivalent to `dir`. It's short for *List Directory*. `ls` can be used with a variety of command-line options to get detailed lists, show hidden files, recursively list subdirectories, and so on. Enter the following examples:

```
$ ls
$ ls -l
$ ls -a
$ ls -R
```

If you're sitting on the command line thinking about your next action, a quick `ls` can help you visualize where you are and what you're doing. Think of it almost like a screen refresh in a graphical user interface, updating you on the system's current state.

cd

To traverse the file system, use `cd` to change directories. Type `cd` and then the name of the directory you want to go to. If you use a leading /, the directory you name is an *absolute* directory path: It starts from the root of your file system. If you

don't use a leading /, the directory is a *relative* path: You start from your current *working* directory. For example, if you're in your home directory, (/home/tuser), and you want to move to a subdirectory, type `cd` followed by the directory name you want to go to. For instance, if you have a Documents directory, you can type the following:

```
$ cd Documents
```

Because you start in /home/tuser/, this command puts you in /home/tuser/Documents/.

In contrast, you can use an absolute path name to specify a location. As an example, you can move into /tmp and then back to your home directory as follows:

```
$ cd /tmp
$ cd /home/tuser
```

Special directory names

In UNIX, special directory names make traversing the file system easy. The three most important ones refer to the current directory, the current directory's parent, and the user's home directory. The current directory is specified by a dot (or period). For instance, if you type `$ cd .`, you remain in the current directory, which is still /home/tuser/. This character becomes especially important when you're running executables in the current working directory. By default, many UNIX shells search application directories for applications, but it does not search the current working directory. You can always explicitly refer to files and applications in your current working directory by using ./ preceding a filename. Parent directories are referred to with two dots (or two periods). To traverse to your current working directory's parent, type the following:

```
$ cd ..
```

If you were in /home/tuser before, now you're in /home. To demonstrate the third special directory name, use the shortcut to get back to your home directory (the tilde character). Type the following command:

```
$ cd ~
```

pwd

To check your current directory, you can use `pwd`, which stands for *Print Working Directory*. It tells you where you are in the file system, which helps you determine what to use when specifying relative path names. Try experimenting with the three special directory names, absolute paths, and relative paths to traverse your file system. At each step, use `pwd` to check your current location.

Check your current directory (you should be in your home directory if you followed the steps in the previous section):

```
$ pwd
```

mkdir, rmdir

`mkdir` and `rmdir` are used to create and remove directories, respectively. `rmdir` works only if the directory is empty (it doesn't remove files).

Try the following commands:

```
$ mkdir TUTORIAL
$ cd TUTORIAL
$ pwd
$ ls
```

You're now in your newly created TUTORIAL directory, and it's empty.

You can also get to this directory by using the ~ (tilde) character. To get to /home/tuser/TUTORIAL, type:

```
$ cd ~/TUTORIAL
$ pwd
```

Directory layout

Now that you know how to move around directories and get listings, you're ready to look at the directory layout on a typical UNIX distribution. You can organize a UNIX file system several ways. This tutorial discusses a few root-level directories that are common to most UNIX-like distributions. There are other important root-level directories, but this is where you'll find yourself operating in most cases:

```
/home (or /users)
/etc
/bin
/sbin
/usr
/car
/tmp
```

/home is where user directories are found. For instance, the `tuser` user is located in /home/tuser.

/etc is the directory used to store most system-wide settings, including startup scripts and network configuration files. You need root access to edit most files in this directory.

/bin and /sbin are the directories used to store system executables (like the commands you're learning about in this tutorial). /sbin is used for system commands, such as `shutdown`, whereas /bin is used for user commands.

Applications are usually installed in /usr. A /usr/local/ subdirectory often holds applications installed that aren't part of the base distribution.

/var is the directory where things like log files, which are continually updated, are stored.

Temporary files are stored in /tmp. This directory is usually writable by all users on the system, and old files are periodically removed on some systems.

Section 4. Files

Now that you know how to move around the file system on the command line, it's time to start working with files. This section teaches you how to create an example file, copy a file, remove a file, and view and change basic file permissions. In a multiuser operating system like UNIX, it's crucial to understand ownership and permission constructs.

touch

To begin, create an empty file that you can use for this tutorial. The `touch` command can be used to create an empty file (it's normally used to update the modified date and accessed date of a file by *touching* it).

Go back to the TUTORIAL directory in your user's home, and create a file by typing the following command:

```
$ cd ~/TUTORIAL
$ touch example.txt
```

cp

The `cp` command copies files. Type `cp` followed by the name of the file you want to copy, followed by the directory you want to copy the file to (you have the option of specifying a new filename as well). For instance, try copying the example.txt file to /tmp/:

```
$ cp example.txt /tmp/
$ ls /tmp/
```

You should see example.txt in /tmp/. Now, copy the file in /tmp/ back to your current directory, but give it a new name:

```
$ cp /tmp/example.txt ./example2.txt
$ ls
```

Notice the use of a dot to specify that you want to put this new file in your current directory. It isn't necessary to include ./ in this case (because the default path for a copy is your current working directory), but it helps clearly illustrate what you intend to do. The subsequent `ls` command shows that you now have two example files in your current working directory.

mv

The `move` command is completed with `mv`. Most syntax and command-line options for move and copy are the same. If you want to move your new file, example2.txt, out of the current directory and into /tmp/, type the following:

```
$ mv example2.txt /tmp/.
```

Note again that a dot is used to explicitly call out what you're doing.

rm

Remove the files created in /tmp/ to tidy up your system. The `rm` command deletes files from your file system. This isn't like moving a file into the Recycle Bin or Trash; the command deletes the file pointer, so use the `rm` command with caution. Type the following:

```
$ rm /tmp/example.txt
$ rm /tmp/example2.txt
$ ls /tmp/
```

Both example files in /tmp/ should be gone.

Depending on which UNIX-like operating system you're using, you might have other delete commands available, such as `srm` or `can`. Try typing `man srm` and `man can` to see if you have these commands. `srm` is used as a secure version of `rm`, which writes random data over deleted files to keep them from being restored. `can` is the opposite of `srm` in some ways; `can` retains the file but moves it into a special trash directory similar to the Windows Recycle Bin.

Section 5. Ownership and permissions

File ownership and permissions are important concepts in UNIX-like operating systems. UNIX is built as a multiuser system from the ground up. Windows is also capable of acting as a multiuser system in today's versions, but Windows has a single-user ancestry dating back to early personal computers and DOS. Even today, some Windows users don't think much about multiple accounts and the file permission issues that go along with a multiuser system. It's critical for any UNIX user to understand file ownership and permissions.

chown, chgrp

File ownership in UNIX can be examined using `ls -l` and can be changed using `chown` and `chgrp`. Take a look by typing the following:

```
$ ls -l
```

Using `-l` specifies that you want to see the long format of a file listing. The long format includes information about permissions, ownership, modification date, and file size. You should see something like this:

```
tsystem:~/TUTORIAL tuser$ ls -l
total 0
-rw-r--r-- 1 tuser admin 0 Aug 13 15:35 example.txt
```

When you do a detailed listing of a directory, you can see in the third and fourth columns the user ownership and the group ownership of a file, respectively. This file is owned by the `tuser` user and the `admin` group. For now, leave this file alone; you can learn more by typing:

```
$ man chown
$ man chgrp
```

The basic syntax for this command uses the name of the user or group as the first input followed by the file or list of files you want to operate on. For example:

```
$ chown tuser example.txt
$ chgrp admin example.txt
```

chmod

Basic file permissions in UNIX are handled by a set of nine flags associated with each file. These flags are formed from three different user categories (user, group, and other) and three file operations (read, write, and execute). Take a look by typing the following command:

```
$ ls -l
```

You should see something like this:

```
-rw-r--r-- 1 tuser admin 0 Aug 13 15:35 example.txt
```

When you're reading a long listing of a directory, the first column shows the file permissions. Notice that there are ten characters (not nine) in this column. The first character denotes the type of file you're dealing with. A dash, `-`, tells you it's a regular file, for instance. A `d` denotes a directory, not a plain file. For now, concentrate on the other nine characters, each of which is one of the following: `-`, `r`, `w`, or `x`. A file that has all permissions turned off reads like this:

```
---------- 1 tuser admin 0 Aug 13 15:35 example.txt
```

A file that has all permissions turned on reads this way:

```
-rwxrwxrwx 1 tuser admin 0 Aug 13 15:35 example.txt
```

You see three sets of `rwx`. As mentioned, there are three different user categories (user, group, and other). Each of these `rwx` sets represents one of the categories:

- The first `rwx` represents the permissions the `user` set. In other words, this tells you what the owner of the file can do to it.
- The second set represents the `group` set. This tells you what members of the group can do to the file.
- The third set represents the `other` set. This tells you what all users on

the system can do to a file, regardless of who owns it.

Take a closer look at the current permissions of your example.txt file:

```
-rw-r--r-- 1 tuser admin 0 Aug 13 15:35 example.txt
```

The first set of three flags indicates that the user who owns this file can read it and write to it, but the user can't execute it (there is an `r` and a `w`, but not an `x`). You can see from the second set of three flags that the group that owns this file can read it, but can't write to it or execute it (there is an `r`, but not a `w` or an `x`). The third set of three flags shows that all other users can read it, but they can't write to it or execute it (there is an `r`, but not a `w` or an `x`). Here's a good example of where detailed `man` pages are important. Pause to type the following:

```
$ man chmod
```

Now, consider the possibility of having private information in this file that you don't want any other users to read. You probably want to remove read access for other groups and all other users. You can use `chmod` to change the permissions. Like many things in UNIX, there are multiple ways to use `chmod`; this section focuses on one. The three categories (user, group, and other) are represented by three letters (u, g, and o). The three types of permissions (read, write, and execute) are also represented by three letters (r, w, and x). To change permissions, use `chmod` followed by the letters of the categories you want to change, followed by a plus or a minus (for turn on or turn off), followed by the letters of the permissions you want to change. After this, write the name of the file (or files) you want to make the change to. This is best illustrated with an example:

```
$ chmod og-r example.txt
$ ls -l
```

You should see this result:

```
-rw------- 1 tuser admin 0 Aug 13 15:35 example.txt
```

In this example, you specify other and group (o and g) and use a minus sign to indicate that you want to turn off certain permissions for these categories. Then, you used read (r) to indicate that you're specifically turning off read access. Now, the owner (tuser) can still read and write the file, but all other users on the system (except the superuser) can't access the file. **Note**: The superuser (root) can override all file permissions.

Section 6. Dealing with multiple files

You've now learned to traverse directories and handle individual files. The next step in this tutorial is learning to deal with files in groups. Almost all UNIX commands can handle file lists as opposed to individual files. You can enter file lists by explicitly typing the name of each file you want to use or by using a wildcard to indicate that you want to use all files with a common name trait.

Wildcards

The most common method of dealing with multiple files is to use *, the wildcard character. You can select a list of files by using * to represent any character or any number of characters. To create a few more files for a demonstration, type these commands:

```
$ cp example.txt example2.txt
$ cp example.txt script.sh
```

Now, type the following:

```
$ ls *.txt
```

You should just see the files with the .txt extension. Next, type:

```
$ ls exa*
```

Again, you should see the two example files, but you should not see script.sh.

The wildcard character can be used with any command-line application that can deal with multiple files.

Recursion

Many command-line applications that deal with files have the `-R` option. When `-R` is used, the application recursively enters a directory and any subdirectories, and it performs the desired command on each file. For instance, you can go back in your home directory and copy the entire TUTORIAL directory:

```
$ cd ~
$ cp -R TUTORIAL /tmp/.
$ ls /tmp/TUTORIAL/
```

Now, remove that directory to tidy up:

```
$ rm -R /tmp/TUTORIAL/
$ ls /tmp/
```

The entire directory is removed, including all the files contained in it. Be careful: You could easily delete much more data than you plan to, especially when combining a wildcard with `-R`.

Section 7. Archives and compression

Many day-to-day file operations can be completed using single files, directories, and wildcards; but learning how to use archives and compression is important for any user interested in backups, transmitting file sets to other users, or simply saving space. A number of UNIX utilities are built into the operating system for archiving and compressing data.

tar

The most common method to group multiple files into a single file (an archive) is to use the `tar` command. `tar` is short for *Tape Archiver*, due to its roots for use with backup tapes; but today, it's commonly used for disk-to-disk operations. Try archiving what you've done so far in the TUTORIAL directory:

```
$ cd ~
$ tar cvf /tmp/tutorial.tar TUTORIAL
$ ls /tmp/
```

You should now see a file called tutorial.tar in /tmp/. This file contains both the directory TUTORIAL and the files contained in it. This example uses the command-line options `cvf`:

- `c` stands for `create`, which tells `tar` to create a new archive.
- `v` stands for verbose, telling `tar` to show a listing of each file that goes into the archive.
- `f` stands for file, telling `tar` to write the archive to a file rather than a device.

Next, go into /tmp and extract the contents of the archive:

```
$ cd /tmp/
$ ls
$ tar cvf tutorial
$ ls
```

Notice that the first `ls` command shows tutorial.tar, but it doesn't show a TUTORIAL directory. The second `ls` command, after the `tar` command, demonstrates that you now have a TUTORIAL directory in /tmp/. Remove the /tmp/TUTORIAL directory to tidy up:

```
$ rm -R /tmp/TUTORIAL
```

gzip

Once you have a tar file, you might want to compress it. A number of compression options are available in most UNIX distributions, but this tutorial focuses on using `gzip` to create something commonly called a *tarball*. A tarball is a tar file that has been compressed. Go ahead and create a compressed version of tutorial.tar so that it takes up less space:

```
$ gzip tutorial.tar
$ ls
```

Your tutorial.tar archives are now compressed and renamed tutorial.tar.gz. This is a tarball. To uncompress it, type:

```
$ gzip -d tutorial.tar.gz
```

The result is your original uncompressed tar file, tutorial.tar.

Section 8. The file system and file sizes

It's easy to learn how to deal with individual files and examine their size and content. You can use the same techniques to examine the contents of entire directories and the file system. Many new versions of UNIX can display this information in simple numbering formats, using letters to denote units.

df

`df` stands for *Display Free* disk space. Using it is as simple as typing `df`; you get information about the amount of disk space in each file system on your machine, the amount of space used, and the amount of space available. Most systems default to display the values in 512KB blocks, which is hard to read. Use `-g` to display information in gigabytes or `-m` to display information in megabytes. Some systems have a `-h` option, which stands for *human-readable*. This makes `df` use suffixes like G, M, and K, and it displays each number in three or fewer digits. Type this command:

```
$ df -h
```

This is an example of output you might see on a simple server:

```
$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             7.9G  3.7G  3.9G  50% /
none                  3.9G     0  3.9G   0% /dev/shm
/dev/sda3              24G   20G  1.9G  92% /export
```

ls -lh

If your system has `-h` for `df`, you can also use it with `ls`. Type this command to see a detailed listing with easy to read file sizes:

```
$ ls -lh
```

du

`du` is a third way to check file sizes, but it has the added advantage of summing up directory sizes. It can also be used with `-h` on some systems; otherwise, try using `-k`, which gives results in 1024-byte blocks. You can also use `-s` and a filename or wildcard to specify which directories and files you want to examine. Try this:

```
$ cd ~
$ du -sk *
$ du -sh *
```

This is an example of output you might see in a home directory:

```
$ du -sk *
316468  OLD
637940  MyData1
571788  Code
12356364        Build
3224480 Hardened
```

```
$ du -sh *
310M    OLD
623M    MyData1
559M    Code
12G     Build
3.1G    Hardened
```

/dev

The `/dev` directory holds special files called *device files*, which, among other things, are used to access disk drives on your system. To learn about the /dev directory, take another look at the output of `df`. This is different on every machine, but it's important to note that `df` shows results for each file system on your computer. Unlike Windows-based computers, each mounted file system is addressed from the system's root, which is denoted by a forward slash: /. This is different from systems that separate disks with drive letters, such as C, D, E, and so on.

In UNIX, it's common for SCSI (and SATA) disks to use device names, such as /dev/sda, /dev/sdb, /dev/sdc, and so on. A common device name for a CD-ROM drive is /dev/cdrom. These devices are *mounted* to directories so that they can be accessed without using the device name. Please consult the documentation for your flavor of UNIX to find out how devices on your system are labeled.

mount

Any device can be mounted at any location (any directory). For instance, it's common to mount a CD-ROM at /mnt/cdrom. Some UNIX-like operating systems (such as many versions of Linux and Mac OS) mount CD-ROMs automatically, but it's good to learn how to use the `mount` command either way. Insert a CD-ROM, and type the following commands:

```
$ mount -t iso9660 /dev/cdrom /mnt/cdrom
$ df
$ ls /mnt/cdrom
```

Note: This will work only if /dev/cdrom and /mnt/cdrom exist on your system. If so, you see in the output of the `df` command that the CD-ROM is now part of the file system. The `ls` command should show the contents of the CD-ROM drive you just mounted.

umount

To unmount a device, most UNIX-like operating systems use `umount`. The syntax is `umount` followed by the mount point. If your previous `mount` command succeeded, type the following:

```
$ umount /mnt/cdrom
$ df
$ ls /mnt/cdrom
```

Note: You must not be *in* the mounted file system for the device to unmount properly; otherwise, the system complains that the file system is *busy*. After a proper execution of `umount`, the `df` command no longer shows the CD-ROM drive in the file system, and the `ls` command shows that /mnt/cdrom is now empty (because nothing is mounted there -- it's a normal directory).

Section 9. Input and output

Input and output are often thought of in simple default terms: keyboard/mouse and monitor/speakers. In UNIX, access to a system of input and output streams (and an error stream) gives users and developers a way to efficiently move input and output to and from applications, which can streamline complex processes by minimizing human interaction. `stdin`, `stdout`, and `stderr` are significant parts of what make UNIX an ideal platform for scripting.

stdin, stdout

Most command-line applications can take input from `stdin` and direct output to `stdout`. `stdin` stands for *Standard Input*. `stdout` stands for *Standard Output*. By default, input from `stdin` comes from the keyboard (what you type in your terminal), and output to `stdout` goes to your display (what you see on your screen). An additional type of output, `stderr`, prints errors and debugs messages, but this tutorial focuses on `stdin` and `stdout`.

Redirection

Redirection allows users to send output that would normally go to `stdout` to another destination -- a file, for instance. Create a text file of your TUTORIAL directory listing, as follows:

```
$ cd ~/TUTORIAL
$ ls > listing.txt
$ ls
```

Another form of redirection is >>, which appends a file as opposed to creating a new file. You can also redirect `stderr` to a file with 2> or to redirect all output (`stdin`

and `stderr`) to a file with `&>`. You can combine forms of redirection; for instance, use `2>>` to append `stderr` to a file.

cat

Now that you have a file with some text in it, look at what's inside. The quickest way to examine the contents of a file is by using the `cat` command, which stands for *Concatenate*; it can be used in combination with redirection to concatenate text. Type the following command:

```
$ cat listing.txt
```

If you've executed each step in the tutorial, you should see something like this:

```
example.txt
example2.txt
listing.txt
script.sh
```

more

The previous `cat` command was executed on a relatively small file, so you can easily see all the data on a single page. If you're viewing a file that doesn't fit on a single page, the `more` command is commonly used; it pauses the output each time a full page of data is displayed to `stdout`. Pressing the Spacebar advances the output to the next page. Try creating a longer file to use for an example:

```
$ ls /etc/ > listing2.txt
$ cat listing2.txt
$ more listing2.txt
```

If you use `cat`, the file scrolls much too quickly to read; but if you use `more`, you can press the Spacebar to advance through the output step by step.

head and tail

If you quickly want to view the first or last few lines of a file, you can use `head` or `tail`. These commands are commonly used to view the top of script or the bottom of a log file. It's also handy to use them as a quick sanity check on an output file when you're debugging code. Try typing the following:

```
$ head listing2.txt
```

To view the last few lines of a file instead of the first few lines, try testing `tail`:

```
$ tail listing2.txt
```

Both commands commonly default to displaying 10 lines, but you can use the `-n` option to display any number of lines. For instance, type this command:

```
$ head -n 2 listing2.txt
```

grep

Now that you're starting to create files with more data in them, you might want to search for specific text. `grep` is a powerful search utility used here in a bare-bones example:

```
$ grep host listing2.txt
```

This command outputs all the lines in listing2.txt that contain the string `host`.

pipe

To skip the file-creation step, you can use the `pipe` character (`|`) to use the output of one command as the input for another command. This is another form of redirection and is incredibly powerful for linking long lists of commands to efficiently manage input and output. Try this simple example:

```
$ ls /etc/ | grep host
```

This command gives the same output as the two-step process listed previously. The single line takes the output of the `ls` command and uses it as the search input for the `grep` command. Here's another example:

```
$ du -sh /etc/* | more
```

In this case, you check the disk usage (`du`) for each file and directory in /etc/. The output is more than one page, so it's useful to `pipe` the result to `more`, which makes the output pause each time one page of data is displayed.

Section 10. Wrap-up

After running through this tutorial, you should have the basic knowledge to navigate a UNIX file system from the command line and work with files and directories. You should experiment with the commands listed throughout this text and closely examine the man page for any item you wish to learn more about.

Future tutorials in the series will cover specific applications (`vi`, for instance) in detail and cover shell tricks and tips. In the meantime, feel free to experiment with a test user; in short order, you'll likely find yourself whizzing through command-line options faster than you could imagine pointing and clicking with your mouse!

Resources

Learn

- UNIX tips and tricks for a new user: Check out other parts in this series.
- The Open Group: This site can teach you more about UNIX and UNIX certifications.
- Linux.org: Learn more about Linux, an open source UNIX-like operating system.
- Make UNIX work with Windows XP and Mac OS X (developerWorks, April 2006): Integrate a UNIX system into a cross-platform environment.
- FreeBSD.org: Visit this site to learn more about BSD operating systems.
- Linux Journal: Keep track of Linux developments.
- AIX and UNIX: Visit the developerWorks AIX and UNIX zone to expand your UNIX skills.
- New to AIX and UNIX: Visit the New to AIX and UNIX page to learn more about AIX and UNIX.
- developerWorks technical events and webcasts: Stay current with developerWorks technical events and webcasts.
- AIX 5L Wiki: A collaborative environment for technical information related to AIX.
- Podcasts: Tune in and catch up with IBM technical experts.

Get products and technologies

- IBM trial software: Build your next development project with software for download directly from developerWorks.

Discuss

- Participate in the AIX and UNIX forums:
  - AIX 5L -- technical
  - AIX for Developers Forum
  - Cluster Systems Management
  - IBM Support Assistant
  - Performance Tools -- technical
  - Virtualization -- technical
  - More AIX and UNIX forums
- Participate in the developerWorks blogs and get involved in the developerWorks community.

About the author

Tim McIntire

Tim McIntire works as a consultant and co-founder of Cluster Corporation, a market leader in HPCC software, support, and consulting. He also contributes periodically to IBM developerWorks and Apple Developer Connection. Tim's research, conducted while leading the computer science effort at Scripps Institution of Oceanography's Digital Image Analysis Lab, has been published in a variety of journals, including *Concurrency and Computation* and *IEEE Transactions on Geoscience and Remote Sensing.* You can visit TimMcIntire.net to learn more.