|  |
| --- |
|  |
|  |  |
|  | **## BASIC BASH SHELL COMMANDS ##**  The default shell used in many Linux distributions is the GNU bash shell. This chapter describes the basic features available in the bash shell. |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **# NAVIGATING THE FILESYSTEM:** |
|  |  |
|  | When you log into the system and reach the shell command prompt, you are usually placed in your home directory. Often you want to explore other areas in the Linux system besides just your home directory. This section describes how to do that using shell commands. |
|  |  |
|  |  |
|  |  |
|  | **# ‘pwd’ - Prints working directory.** |
|  |  |
|  | The ‘pwd’ command displays the shell session's current directory location, which is called the present working directory. |
|  |  |
|  |  |
|  |  |
|  | **# ‘cd’ - Changes working directory.** |
|  |  |
|  | We use change directory command ‘cd’ to move your shell session to another directory in the Linux filesystem. |
|  |  |
|  | The ‘cd’ command may take a single parameter, "destination", which specifies the directory name you want to go to. |
|  | If you don't specify a destination on the cd command, it takes you to your home directory. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **# LISTING FILES AND DIRECTORIES:** |
|  |  |
|  |  |
|  |  |
|  |  |
|  | To see what files are available on the system, use the list command ‘ls’. |
|  | This section describes the ‘ls’ command and options available to format the information it can dispay. |
|  | **# ‘ls’ - Displaying a basic listing.** |
|  |  |
|  | - **ls -l** – display detailed content/listing. |
|  |  |
|  | - **ls -d** – content of specified directory. |
|  |  |
|  | - **ls -a** – content of directory, hidden files are included. |
|  |  |
|  | - **ls -R** - it shows files that are contained within subdirectories in the current directory. |
|  |  |
|  | Option parameters can be combined: ls -aR / ls -lR.... |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **# HANDLING FILES:** |
|  |  |
|  |  |
|  | Every once in a while you run into a situation where you need to create an empty file. For example, sometimes applications expect a log file to be present before they can write to it. |
|  |  |
|  |  |
|  | **# ‘touch’ - Creating an empty file.** |
|  |  |
|  | - IN THESE SITUATIONS, YOU CAN USE THE ‘touch’ COMMAND TO EASILY CREATE AN EMPTY FILE: |
|  |  |
|  | $ touch test\_file |
|  |  |
|  | $ ls -l test\_file |
|  |  |
|  | -rw-r--r-- 1 Dziug 197609 0 Nov 9 13:55 test\_file |
|  |  |
|  |  |
|  | The ‘touch’ command creates the new file you specify and assigns your username as the file owner. Notice in the preceding example that the file size is zero because the touch command just created an empty file. |
|  |  |
|  |  |
|  | - THE ‘touch’ COMMAND CAN ALSO BE USED TO CHANGE THE MODIFICATION TIME: |
|  |  |
|  | $ ls -l test\_file |
|  | -rw-r--r-- 1 Dziug 197609 0 Nov 9 13:55 test\_file |
|  |  |
|  | $ touch test\_file |
|  |  |
|  | $ ls -l test\_file |
|  | -rw-r--r-- 1 Dziug 197609 0 Nov 9 14:10 test\_file |
|  |  |
|  | $ |
|  |  |
|  |  |
|  | The modification time of test\_file is now updated from 13:55 to 14:10. |
|  |  |
|  |  |
|  |  |
|  | **# ‘cp’ - Copying files and folders.** |
|  |  |
|  | Copying files and directories from one location in the filesystem to another is a common practice for system administrators. The ‘cp’ command provides this feature. |
|  |  |
|  | - IN ITS MOST BASIC FORM, THE ‘cp’ COMMAND USES TWO PARAMETERS - THE SOURCE OBJECT AND THE DESTINATION OBJECT: |
|  |  |
|  | - "cp source destinantion" |
|  |  |
|  | - cp file1 file2 -- (file1 will be copied to file2) |
|  |  |
|  | - cp -r directory1 directory2 -- (directory1 will be copied to directory2) |
|  |  |
|  |  |
|  |  |
|  | **# ‘mv’ - Moving files and folders.** |
|  |  |
|  | - ITS FORM IS SIMILAR TO cp, INSTEAD IT MOVES FILES TO THE CHOSEN DESTINATION: |
|  | - mv /path1/file1 /path2/ -- (moves files and directories) |
|  | - mv file1 file2 -- (renames files and directories) |
|  |  |
|  |  |
|  |  |
|  | **# ‘rm’ - Deleting/Removing files and directories.** |
|  |  |
|  | Most likely at some point you'll want to be able to delete existing files. Whether it's to clean up a filesystem or to remove a software package, you always have opportunities to delete files. |
|  |  |
|  | - IN THE LINUX WORLD, DELETING IS CALLED REMOVING. THE COMMAND TO REMOVE FILES IN THE BASH SHELL IS ‘rm': |
|  | - rm file -- (files removal) |
|  | - rm -r directory -- (directories removal) |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **# MANAGING DIRECTORIES:** |
|  |  |
|  | Linux has a few commands that work for both files and directories (such as the ‘cp’ command), and some that work only for directories. To create a new directory, you need to use a specific command, which is covered in this section. Removing directories can get interesting, so that is covered in this section as well. |
|  |  |
|  |  |
|  |  |
|  | **# ‘mkdir’ - Creating directories.** |
|  |  |
|  | - CREATING A NEW DIRECTORY IN LINUX IS EASY - JUST USE THE ‘mkdir’ COMMAND: |
|  |  |
|  | $ mkdir New\_Dir |
|  | $ ls -ld New\_Dir |
|  | drwxr-xr-x 1 Dziug 197609 0 Nov 9 15:20 New\_Dir/ |
|  |  |
|  |  |
|  | The system creates a new directory named New\_Dir. Notice in the new directory's long listing that the directory's record begins with a “d”. This indicated that New\_Dir is not a file, but a directory. |
|  |  |
|  |  |
|  |  |
|  | - YOU CAN CREATE DIRECTORIES AND SUBDIRECTORIES IN "BULK" IF NEEDED. HOWEVER, IF YOU ATTEMPT THIS WITH JUST THE ‘mkdir’ COMMAND, YOU GET THE FOLLOWING ERROR MESSAGE: |
|  |  |
|  | $ mkdir New\_Dir/Sub\_Dir/Under\_Dir |
|  |  |
|  | mkdir: cannot create directory 'New\_Dir/Sub\_Dir/Under\_Dir': |
|  | No such file or directory |
|  | $ |
|  |  |
|  | - TO CREATE SEVERAL DIRECTORIES AND SUBDIRECTORIES AT THE SAME TIME, YOU NEED TO ADD THE ‘-p’ PARAMETER: |
|  |  |
|  | $ mkdir -p New\_Dir/Sub\_Dir/Under\_Dir |
|  | $ |
|  |  |
|  | $ ls -R New\_Dir |
|  | New\_Dir: |
|  | Sub\_Dir/ |
|  | New\_Dir/Sub\_Dir: |
|  | Under\_Dir/ |
|  | New\_Dir/Sub\_Dir/Under\_Dir: |
|  |  |
|  |  |
|  | The ‘-p’ option on the ‘mkdir’ command makes any missing parent directories as needed. A parent directory is a directory that contains other directories at the next level down the directory tree. |
|  |  |
|  | Of course, after you make something, you need to know how to delete it. This is especially useful if you created a directory in the wrong location. |
|  |  |
|  |  |
|  |  |
|  | **# ‘rmdir’ - Removing directories.** |
|  |  |
|  | Removing directories can be tricky, and for good reason. There are lots of opportunities for bad things to happen when you start deleting directories. The shell tries to protect us from accidental catastrophes as much as possible. |
|  |  |
|  | - THE BASIC COMMAND FOR REMOVING A DIRECTORY IS ‘rmdir’: |
|  |  |
|  | $ mkdir New\_Dir |
|  | $ touch New\_Dir/My\_File |
|  | $ ls -li New\_Dir/ |
|  | total 0 |
|  | 8725724278069906 -rw-r--r-- 1 Dziug 197609 0 Nov 9 16:04 My\_File |
|  | $ |
|  | $ rmdir New\_Dir |
|  | rmdir: failed to remove 'New\_Dir': Directory not empty |
|  |  |
|  |  |
|  | By default, the ‘rmdir’ command works only for removing empty directories. Because we created a file, my\_file, in the New\_dir directory, the ‘rmdir’ command refuses to remove it. |
|  | - TO FIX THIS, WE MUST REMOVE THE FILE FIRST. THEN WE CAN USE THE ‘rmdir’ COMMAND ON THE NOW EMPTY DIRECTORY: |
|  | $ rm -i New\_Dir/My\_File (-i makes a question if you really want to remove it? y/n?) |
|  | rm: remove regular empty file 'New\_Dir/My\_File'? y |
|  |  |
|  | $ |
|  |  |
|  | $ rmdir New\_Dir |
|  | $ |
|  | $ ls -ld New\_Dir |
|  | ls: cannot access 'New\_Dir': No such file or directory |
|  |  |
|  | The ‘rmdir’ has no ‘-I’ option to ask if you want to remove the directory. This is one reason it is helpful that ‘rmdir’ removes only empty directories. |
|  |  |
|  |  |
|  | - YOU CAN ALSO USE THE ‘rm’ COMMAND ON ENTIRE NON-EMPTY DIRECTORIES. USING THE ‘-r’ OPTION ALLOWS THE COMMAND TO DESCEND INTO THE DIRECTORY, REMOVE THE FILES, AND THEN REMOVE THE DIRECTORY ITSELF: |
|  |  |
|  | $ ls -l My\_Dir |
|  | total 0 |
|  | drwxr-xr-x 1 Dziug 197609 0 Nov 9 17:14 New\_Files/ |
|  |  |
|  | $ rm -ri My\_Dir |
|  | rm: descend into directory 'My\_Dir'? y |
|  | rm: descend into directory 'My\_Dir/New\_Files'? y |
|  | rm: remove directory 'my\_dir'? y |
|  | $ |
|  | $ ls -l My\_Dir |
|  |  |
|  | ls: cannot access 'My\_Dir': No such file or directory |
|  |  |
|  | - THIS ALSO WORKS FOR DESCENDING INTO MULTIPLE SUBDIRECTORIES AND IS ESPECIALLY USEFUL WHEN YOU HAVE LOTS OF DIRECTORIES AND FILES TO DELETE: |
|  |  |
|  | $ ls -FR Big\_Dir |
|  | Big\_Dir: |
|  | Another\_Med\_Dir/ file1.txt Med\_Dir/ (There are 2 directories and file1.txt in Big\_Dir) |
|  |  |
|  | Big\_Dir/Another\_Med\_Dir: (Another\_Med\_Dir is empty) |
|  |  |
|  | Big\_Dir/Med\_Dir: (There is Small\_dir/ and file2.txt in Med\_Dir) |
|  | file2.txt Small\_Dir/ |
|  | Big\_Dir/Med\_Dir/Small\_Dir: (There's only a file3.txt in Small\_dir) |
|  | file3.txt |
|  | $  --------------- |
|  | $ rm -ri Big\_Dir |
|  | rm: descend into directory 'Big\_Dir'? y |
|  | rm: remove directory 'Big\_Dir/Another\_Med\_Dir'? y |
|  | rm: remove regular empty file 'Big\_Dir/file1.txt'? y |
|  | rm: descend into directory 'Big\_Dir/Med\_Dir'? y |
|  | rm: remove regular empty file 'Big\_Dir/Med\_Dir/file2.txt'? y |
|  | rm: descend into directory 'Big\_Dir/Med\_Dir/Small\_Dir'? y |
|  | rm: remove regular empty file 'Big\_Dir/Med\_Dir/Small\_Dir/file3.txt'? y |
|  | rm: remove directory 'Big\_Dir/Med\_Dir/Small\_Dir'? y |
|  | rm: remove directory 'Big\_Dir/Med\_Dir'? y |
|  | rm: remove directory 'Big\_Dir'? y |
|  | $ |
|  |  |
|  | $ ls -FR Big\_Dir |
|  |  |
|  | ls: cannot access 'Big\_Dir': No such file or directory |
|  |  |
|  | Although this works, it's somewhat awkward. Notice that you still must verify each and every file that gets removed. For a directory with lots of files and subdirectories, this can become tedious. |
|  |  |
|  | - THE ULTIMATE SOLUTION FOR THROWING CAUTION TO THE WIND AND REMOVING AN ENTIRE DIRECTORY CONTENTS AND ALL, IS THE ‘rm’ COMMAND WITH BOTH THE ‘-r’ AND ‘-f’ PARAMETERS: |
|  |  |
|  | $ ls -FR Big\_Dir |
|  | Big\_Dir: |
|  | Another\_Med\_Dir/ file1.txt Med\_Dir/ (There are 2 directories and file1.txt in Big\_Dir) |
|  | Big\_Dir/Another\_Med\_Dir: (Another\_Med\_Dir is empty) |
|  | Big\_Dir/Med\_Dir: (There is Small\_dir/ and file2.txt in Med\_Dir) |
|  | file2.txt Small\_Dir/ |
|  | Big\_Dir/Med\_Dir/Small\_Dir: (There's only a file3.txt in Small\_dir) |
|  | file3.txt |
|  | $ |
|  | $ rm -rf Big\_Dir |
|  | $ ls -FR Big\_Dir |
|  | ls: cannot access 'Big\_Dir': No such file or directory |
|  | The rm -rf command gives no warnings and no fanfare. This, of course, is an extremely dangerous tool to have, especially if have superuser privileges. Use it sparingly, and only after triple checking to make sure you're doing exactly what you want to do! |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **# VIEWING FILE CONTENTS:** |
|  |  |
|  | You can use several commands for looking inside files without having to pull out the text editor utility (nano FileName). This section demonstrates a few of the commands you have available to help you examine files. |
|  |  |
|  | **# ‘file’ - Viewing the file type.** |
|  | Before trying to display file, try to get a handle on what type of file it is. If you try to display a binary file, you get lots of gibberish on your monitor and may even lock up your terminal emulator. |
|  |  |
|  | - THE ‘file’ COMMAND IS A HANDY LITTLE UTILITY TO HAVE AROUND. IT CAN PEEK INSIDE OF A FILE AND DETERMINE JUST WHAT KIND OF FILE IT IS: |
|  | $ file my\_file |
|  | my\_file: ASCII text |
|  | $ |
|  |  |
|  | The file in the preceding example is a text file. The file command determined not only that the file contains text but also the character code format of the text file, ASCII. |
|  | - THIS FOLLOWING EXAMPLE SHOWS A FILE THAT IS SIMPLY A DIRECTORY. THUS, THE ‘file’ COMMAND GIVES YOU ANOTHER METHOD TO DISTINGUISH A DIRECTORY: |
|  | $ file New\_Dir |
|  | New\_Dir: directory |
|  | $ |
|  |  |
|  | - THIS THIRD ‘file’ COMMAND EXAMPLE SHOWS A FILE, WHICH IS SYMBOLIC LINK. NOTE THAT THE file COMMAND EVEN TELLS YOU TO WHICH FILE IT IS SYMBOLICALLY LINKED: |
|  |  |
|  | $ file sl\_data\_file |
|  | sl\_data\_file: symbolic link to 'data\_file' |
|  | $ |
|  |  |
|  | ***--(another example)—*** |
|  | $ file RAM.pptx |
|  | RAM.pptx: Microsoft PowerPoint 2007+ |
|  | $ |
|  |  |
|  | THE FOLLOWING EXAMPLE SHOWS WHAT THE ‘file’ COMMAND RETURNS FOR A SCRIPT FILE. ALTHOUGH THE FILE IS ASCII text, BECAUSE IT'S A SCRIPT FILE, YOU CAN EXECUTE (RUN) IT ON THE SYSTEM: |
|  |  |
|  | $ file my\_script.txt |
|  | my\_script.txt: ASCII text, with CRLF line terminators |
|  | $ |
|  |  |
|  | THE FINAL EXAMPLE IS A BINARY EXECUTABLE PROGRAM. THE ‘file’ COMMAND DETERMINES THE PLATFORM THAT THE PROGRAM WAS COMPILED FOR AND WHAT TYPES OF LIBRARIES IT REQUIRES. THIS IS AN ESPECIALLY HANDY FEATURE IF YOU HAVE A BINARY EXECUTABLE PROGRAM FROM AN UNKNOWN SOURCE: |
|  |  |
|  | $ file /bin/ls |
|  | /bin/ls: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), |
|  | dynamically linked (uses shared libs), for GNU/Linux 2.6.24, |
|  | [...] |
|  | $ |
|  |  |
|  |  |
|  | **# Viewing the whole file.** |
|  |  |
|  | If you have a large text file on your hands, you may want to be able to see what's inside of it. Linux has three different commands that can help you here. |
|  |  |
|  | **# Using the 'cat' command.** |
|  |  |
|  | THE ‘cat’ COMMAND IS A HANDY TOOL FOR DISPLAYING ALL THE DATA INSIDE A TEXT FILE: |
|  | $ cat test1.txt |
|  | hello |
|  | This is a test file.txt |
|  | That we'll use to test the cat command. |
|  | $ |
|  | Nothing too exciting, just the contents of the text file. However, the ‘cat’ command has a few parameters that can help you out. |
|  | THE ‘-n’ PARAMETER NUMBERS ALL THE LINES FOR YOU: |
|  |  |
|  | $ cat -n test1.txt |
|  | 1 hello |
|  | 2 |
|  | 3 This is a test file.txt |
|  | 4 |
|  | 5 |
|  | 6 That we'll use to test the cat command. |
|  | $ |
|  |  |
|  | THAT FEATURE WILL COME IN HANDY WHEN YOU'RE EXAMINING SCRIPTS. IF YOU JUST WANT TO NUMBER THE LINES THAT HAVE TEXT IN THEM, THE ‘–b’ PARAMETER IS FOR YOU: |
|  |  |
|  | $ cat -b test1.txt |
|  | 1 hello |
|  | 2 This is a test file.txt |
|  | 3 That we'll use to test the cat command. |
|  | $ |
|  |  |
|  | FINALLY, IF YOU DON'T WANT TAB CHARACTERS TO APPEAR, USE THE ‘-T’ PARAMETER: |
|  |  |
|  | $ cat -T test1.txt |
|  | hello |
|  | This is a test file.txt |
|  | That we'll use to^I test the cat command. |
|  | $ |
|  |  |
|  | The ‘-T’ parameter replaces any tabs in the text with the ^I character combination. |
|  |  |
|  | For large files, the ‘cat’ command can be somewhat annoying. The text in the file just quickly scrolls off the display without stopping. Fortunately, we have a simple way to solve this problem. |
|  |  |
|  | **# Viewing parts of a file.** |
|  |  |
|  | Often the data you want to view is located either right at the top or burried at the bottom of a text file. If the information is at the top of a large file, you still need to wait for the 'cat' or 'more' commands to load the entire file before you can view it. If the information is located at the bottom of a file (such as a log file), you need to wade through thousands of lines of text just to get to the last few entries. Fortunately, Linux has specialized commands to solve both of these problems. |
|  |  |
|  | **# Using the 'tail' command.** |
|  |  |
|  | The 'tail' command displays the last lines in a file (the file's "tail"). By default, it shows the last 10 lines in the file. |
|  |  |
|  | FOR THESE EXAMPLES, WE CREATED A TEXT FILE CONTAINING 20 TEXT LINES. IT IS DISPLAYED HERE IN ITS ENTIRETY USING THE 'cat' COMMAND: |
|  |  |
|  | $ cat log\_file |
|  | line1 |
|  | line2 |
|  | line3 |
|  | line4 |
|  | line5 |
|  | Hello World - line 6 |
|  | line7 |
|  | line8 |
|  | line9 |
|  | line10 |
|  | line11 |
|  | Hello again - line 12 |
|  | line13 |
|  | line14 |
|  | line15 |
|  | Sweet - line16 |
|  | line17 |
|  | line18 |
|  | line19 |
|  | Last line - line20 |
|  | $ |
|  |  |
|  | NOW THAT YOU HAVE SEEN THE ENTIRE TEXT FILE, YOU CAN SEE THE EFFECT OF USING 'tail' TO VIEW THE FILE'S LAST 10 LINES: |
|  |  |
|  | $ tail log\_file |
|  | line11 |
|  | Hello again - line 12 |
|  | line13 |
|  | line14 |
|  | line15 |
|  | Sweet - line16 |
|  | line17 |
|  | line18 |
|  | line19 |
|  | Last line - line20 |
|  | $ |
|  |  |
|  | You can change the number of lines shown using tail by including the '-n' parameter. IN THIS EXAMPLE, ONLY THE LAST TWO LINES OF THE FILE ARE DISPLAYED, BY ADDING '-n' 2 TO THE 'tail' COMMAND: |
|  |  |
|  | $ tail -n 2 log\_file |
|  | line19 |
|  | Last line - line20 |
|  | $ |
|  |  |
|  | The '-f' parameter is a pretty cool feature of the 'tail' command. It allows you peek inside a file as the file is being used by other processes. The 'tail' command stays active and continues to display new lines as they appear in the text file. THIS IS A GREAT WAY TO MONITOR THE SYSTEM LOG FILES IN REAL-TIME MODE. |
|  |  |
|  | **# Using the 'head' command.** |
|  | The 'head' command does what you'd expect; it displays a file's first group of lines (the file's "head"). BY DEFAULT, IT DISPLAYS THE FIRST 10 LINES OF TEXT: |
|  | $ head log\_file |
|  | line1 |
|  | line2 |
|  | line3 |
|  | line4 |
|  | line5 |
|  | Hello World - line 6 |
|  | line7 |
|  | line8 |
|  | line9 |
|  | line10 |
|  | $ |
|  |  |
|  | Similar to the 'tail' command, the 'head' command supports the '-n' parameter so you can alter what's displayed. BOTH COMMANDS ALSO ALLOW YOU TO SIMPLY TYPE A DASH ALONG WITH THE NUMBER OF LINES TO DISPLAY, AS SHOWN HERE: |
|  |  |
|  | $ head -5 log\_file |
|  | line1 |
|  | line2 |
|  | line3 |
|  | line4 |
|  | line5 |
|  | $ |
|  |  |
|  | Usually the beginning of a file doesn't change, so the 'head' command doesn't support the '-f' parameter feature as the 'tail' command does. The 'head' command is a handy way to just peek at the beginning of a file. |