A Command Line Primer for Beginners

The How-To Geek
9/09/10 12:00pm

The command line isn't just for wise Linux beards. It's actually an awesome tool with almost limitless functionality. Here's a primer on how it works, and how you can do almost anything with it.

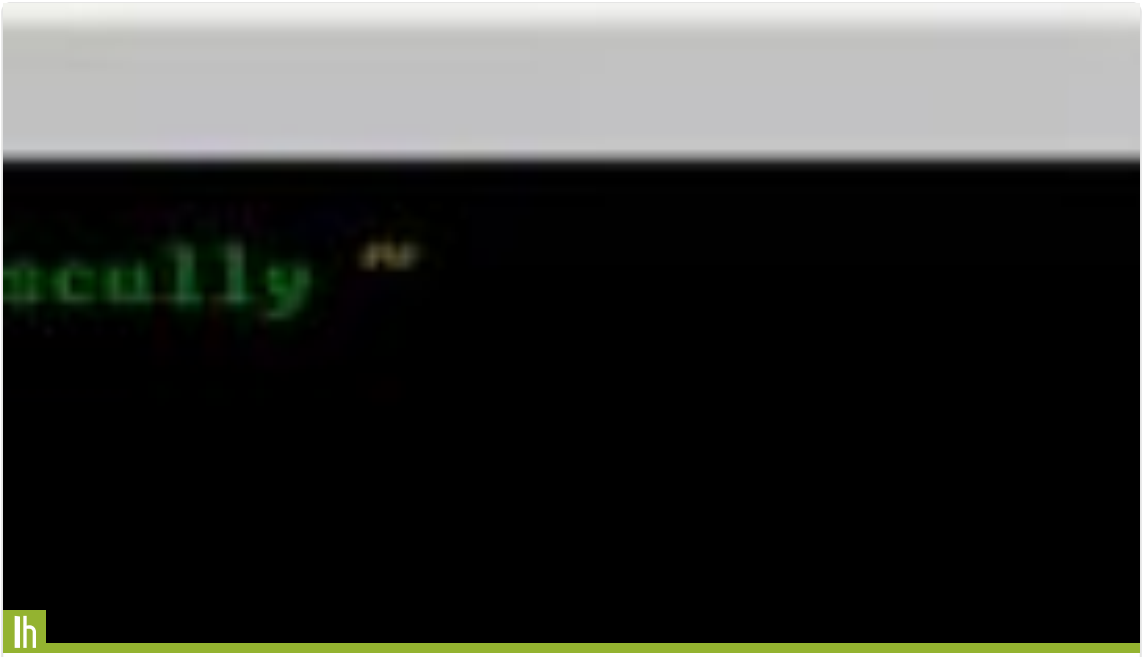
Note: This article is meant for people who are either new to the command line or only have a couple of command-line tricks up their sleeve. If you're already conversant with most basic commands, you can send this article to others that still aren't up to your skill level and spread the good word about how great the command line really is.

What Is the Command Line?

The command-line interface, sometimes referred to as the CLI, is a tool into which you can type text commands to perform specific tasks—in contrast to the mouse's pointing and clicking on menus and buttons. Since you can directly control the computer by typing, many tasks can be performed more quickly, and some tasks can be automated with special commands that loop through and perform the same action on many files—saving you, potentially, loads of time in the process.

The application or user interface that accepts your typed responses and displays the data on the screen is called a shell, and there are many different varieties that you can choose from, but the most common these days is the Bash shell, which is the default on Linux and Mac systems in the Terminal application.

By default, Windows systems only include the anemic Command Prompt application, which has nowhere near the power of Bash, so for the purposes of this article we're going to suggest you use the open source Cygwin tool as your Windows command line, since it's quite a bit more powerful. You'll also at some point want to read parts one, two, and three of our series on using the Bash shell under Cygwin—the tips apply to Linux and OS X as well.



Geek to Live: Introduction to Cygwin, Part I

Introduction to who? If that's what you thought when you saw the title of this article, then...

Read more

Basic Command Line Usage

To get started with the command line, you'll need to open up a terminal window and get ready to start typing commands. Here's a list of basic commands you can use, organized by the type of activity that you might want to perform.

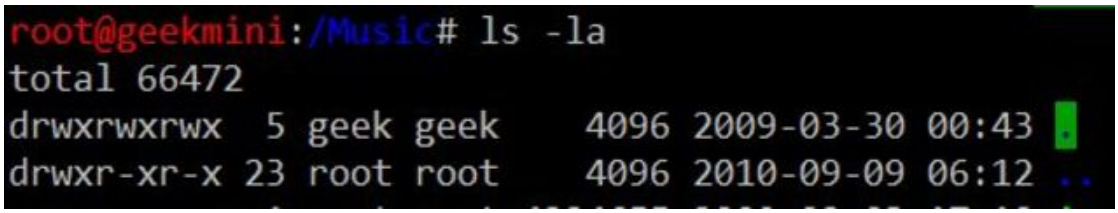
When you run your terminal application (Cygwin on Windows, Terminal on Mac and Linux), your command prompt will start up pointing to a specific folder on your hard drive. You can navigate between folders, act on files inside those folders, or perform other actions.

List Files

First, let's display a list of files inside the active folder. For this task, you'll need to use the *ls* command. You can pass a number of parameters to the command to display extra details or change the sorting. For instance, if I add `-l` to the end of my `ls` command, I'll see a detailed listing; `-t` will sort the results by file time; `-S` will sort by file size; and `-r` will reverse the sorting. You could use a combination of these together, like this command, which will show all files sorted by file size with the largest files at the bottom:

```
ls -lSr
```

If you use the –a option, you can see hidden files, and you'll also notice something else in the listing: there are two entries for "." and ".." at the beginning of the list. These represent the current folder—the "." folder—and the parent folder—the ".." folder.



Change Directories

You can change between directories using the `cd` command, and using what we just learned about the ".." folder, you can use the following command to switch to the directory directly above the current one:

```
cd ..
```

You can navigate to either full or relative paths. For example, the command above navigates to a relative path—one above the current folder. If you're in `/path/to/` and you want to navigate to the folder `stuff` inside that folder, you can simply type:

```
cd stuff
```

You can also navigate to absolute paths. Again, if I were in `/path/to/` and I want to navigate to `/another/path/`, I would simply type:

```
cd /another/path
```

To swap directories to the previous working directory, the '-' (hyphen) shortcut is handy to have on hand. For example, if you were in the `/first/folder/path/` directory and then switched over to `/etc/` to make a configuration change, you might not want to type in the full path to switch back. You can quickly switch back to the previous working directory with this command:

```
cd -
```

Creating or Removing Folders

To create a new folder, you can simply use the `mkdir <foldername>` command. You can then remove any folder with the `rmdir <foldername>` command—as long as the folder is empty. If there are files in the folder, you'll have to delete those files before you can remove the folder.

Creating and Removing Files

You can use the `touch <filename>` command to create a new, blank file, and then use the `rm <filename>` command to delete files:

```
rm filename
```

You can quickly remove all files in a directory by using the '*' (asterisk) wildcard—another simple tool that will come in very handy during your time in the command line. For example, if you're in a folder and want to delete every file inside that folder, just type:

```
rm *
```

If you want to delete a list of files and folders, including all files from *subdirectories*, without prompting you for every single entry, you can use the -r option for recursive, and the -f option for force. This command will wipe out every instance of a matching filename pattern (note the slightly different use of the wildcard) from the current directory and below:

```
rm –rf filename.*
```

Edit Plain Text Files

The command that you use to edit text files will be different based on the platform you're using and the application you prefer to use. If you're using Ubuntu Linux, you can use the nano editor to quickly edit files, which might be more suitable for beginners.

```
nano /path/to/file
```

Otherwise, the vim editor is available on just about any system and can be invoked with the vi <filename> syntax.

Displaying Files

You can display the file contents directly on the screen with the `cat` command, but the results will probably go flying past you on most large files, so it's usually better to use the `more` or `less` commands. For instance:

```
more <filename>
```

This will display the contents of a file on the screen, and prompt you to scroll through the file a screen at a time.

Command Redirection

Each command line application can accept standard input and writes to standard output, and you can use the `>` or `|` operators to redirect output from one command into another, which lets you chain commands together into much more powerful commands.

For instance, if you want to use ls –l to display a list of files but it keeps scrolling off the screen, you can pipe the output from the ls –l command into the input of the `more` command by using the | character:

```
ls –l | more
```

If you wanted to save the output of that list directly into a file instead of displaying on the console, you could use the `>` operator to redirect the output into a file instead:

```
ls -l > filename.list
```

You could then use the `cat` command to display the contents of that file, pipe that into the `grep` command (detailed further below), and then redirect that output into a separate file:

```
cat filename.list | grep keyword > filefound.list
```

Running a Script in the Current Folder

If you have an application or shell script in the current folder, you can't simply type the name of the command and expect it to start. You'll need to add a ./ to the beginning of the command in order to start it. Why? Because in the Bash shell, the current directory, or "." folder, is not included in the system path. So to launch scriptname.sh in the current folder, you'll need to use:

```
./scriptname.sh
```

Using History

You can use the ***history*** command to show a list of all the recently used commands, or the up/down arrows to loop through them. The Ctrl+R shortcut key will start a search mode where you can type the first few characters of a command to search through your recent history.

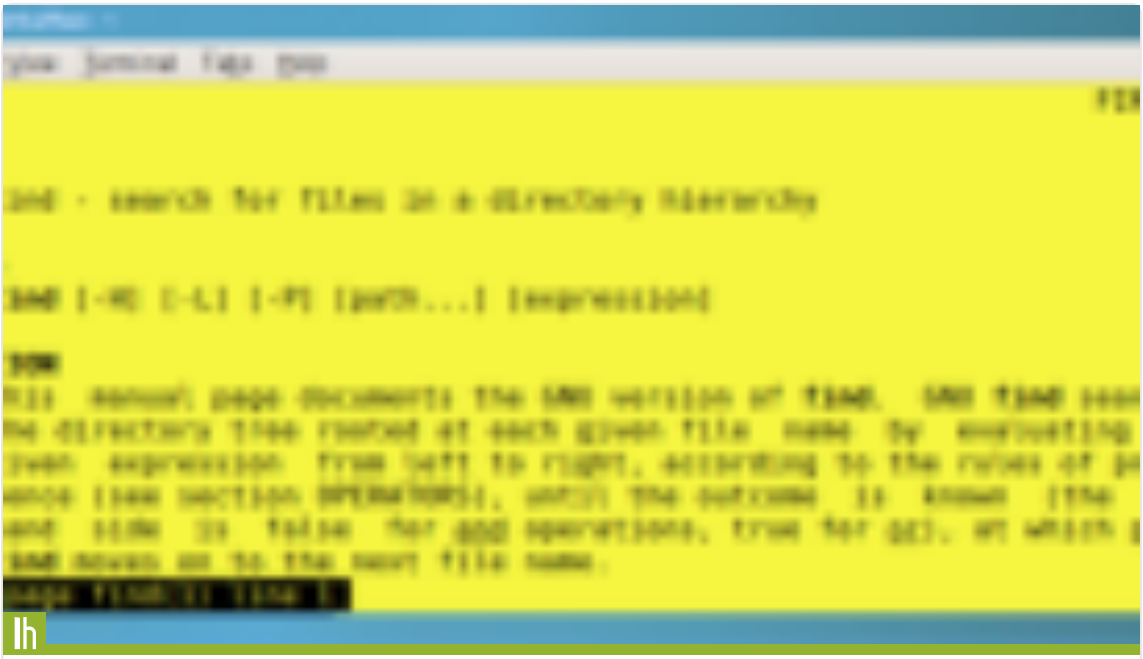
Looping Over a Set of Files

If you want to loop through a set of filenames and perform an action on each one, you can use the ***for*** command to loop through a set of files. For instance, to loop through all the .txt files in the current directory and display them on the console, you could use:

```
for f in *.txt;do echo $f;done
```

Find Files

You can use the very powerful `find` command to search for files on your system. For instance, if you wanted to find all files with .txt in the name that were modified in the last 5 days, you would use this command:



Find Anything From the Terminal

Oldie but goodie terminal command, find, can be used to locate literally any file on your system....

Read more

```
find . –name "*.txt" –mtime 5
```

Find a Text String in Files

The `grep` command can be used to quickly find text within files, even searching through subdirectories. For instance, if you wanted to search through all files in the current directory and below it for "text string", you could use this command:

```
grep –ir "text string" *
```

Batch Rename Files

You can use the rename command to quickly rename files using a regular expression pattern. For instance, if you wanted to rename all files containing foo to contain bar instead, you could use a command like this one:

```
rename –v 's/foo/bar/g' *
```

Using Bash Shortcut Keys

There are a number of very useful shortcut keys you can use in the bash shell, and it pays to master them all. Here's a couple to get you started:

- **Ctrl + U**: Clears the line from the cursor point back to the beginning.
- **Ctrl + A**: Moves the cursor to the beginning of the line.
- **Ctrl + E**: Moves the cursor to the end of the line.
- **Ctrl + R**: Allows you to search through the previous commands.

Customizing Your Command Shell

There's no need to do your work in a boring terminal when you can do all sorts of tricks to customize it, like changing the colors, fonts, and adding aliases to complicated commands to save yourself time.

You'll want to start off by reading our guide to customizing the command prompt, which will show you how to change the colors and add them to your profile so they show up when you launch a new shell.

Ask Lifehacker: How do I customize my command line prompt?

Dear Lifehacker,

Read more

Using Aliases

Aliases save you loads of time by shortening long, complicated commands down into really simple ones, or by setting the default parameters to a command so you don't have to type them every time. For instance, if you wanted to set up an alias for installing packages on your Ubuntu setup that's quicker and simpler than *sudo apt-get install packagename*, you could use something like this:

```
alias agi='sudo apt-get install'
```

This alias would make it so you could simply type `agi packagename` at the shell to install any package in fewer keystrokes. You can also use aliases to set the default arguments for a command, so if you always wanted *ls* to actually do *ls –l*, you could use this alias:

```
alias ls='ls –l'
```

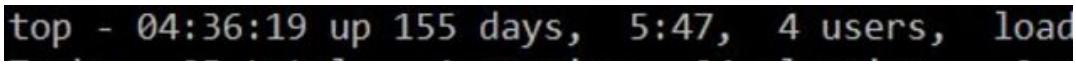
There are any number of useful aliases that you can use to personalize your setup, but if you're having trouble coming up with good ideas, check out the list of the ten most useful aliases.

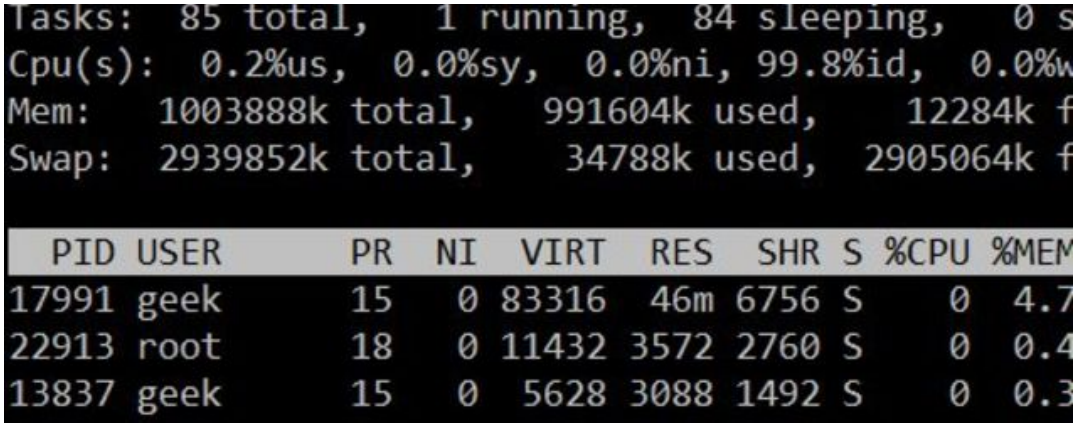
Ten Handy Bash Aliases for Linux Users

TechRepublic's 10 Things blog posts 10 shortcut ideas for Linux users (and Terminal-friendly...

Read more

Control Your System from the Shell

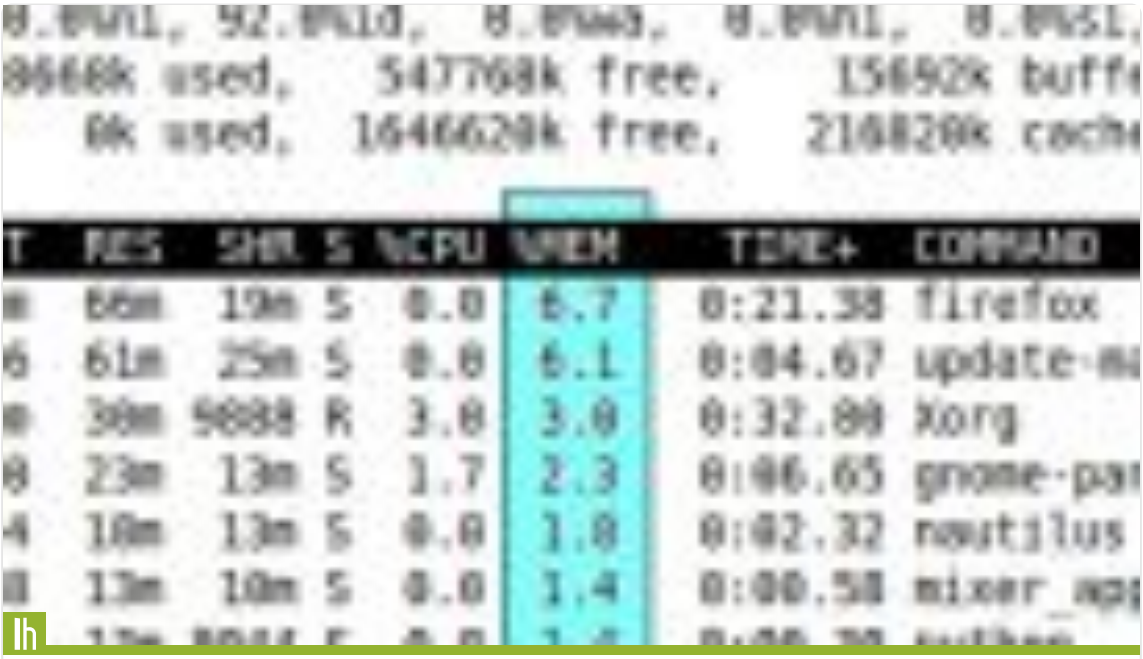




The terminal has a rich set of tools for manipulating processes and checking on system stats. You can use the `ps` command to see a list of system processes like this:

```
ps aux
```

You can then use the `kill <pid>` command to get rid of any processes that you want to kill. You can also use `top` to easily kill processes from a more graphical list of running processes by simply using the K key.



Control Your System with the Top Command