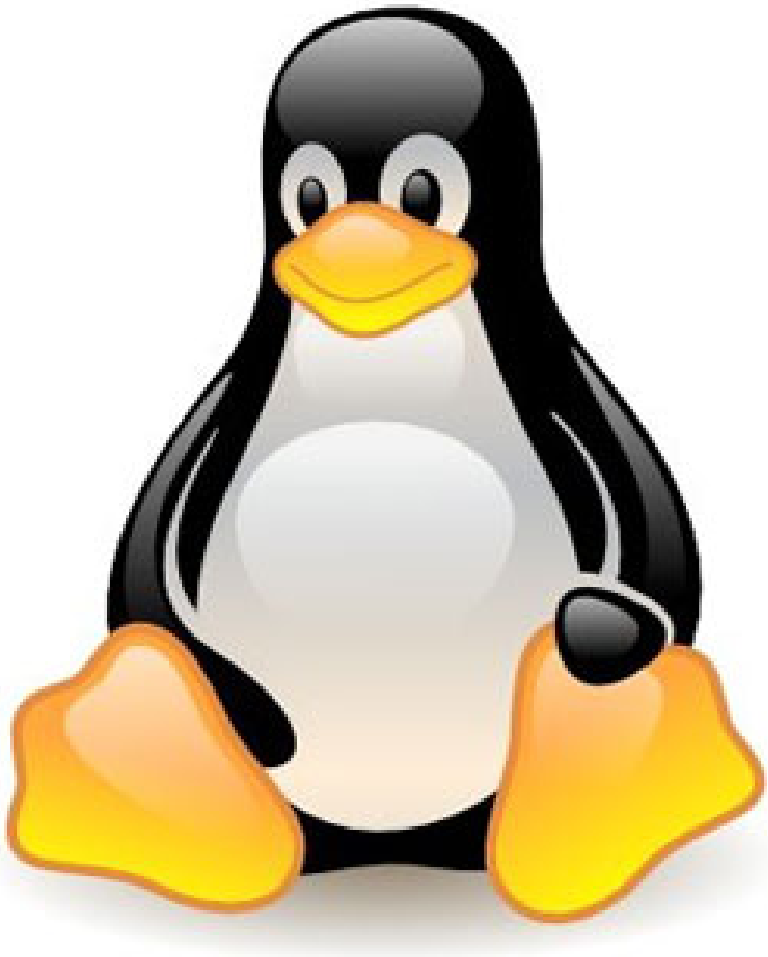
10 most useful Linux commands

For those who are new to Linux, administration will face many difficulties. Here are the 10 most useful Linux commands that make your administration a lot easier.

TipsMake.com - Maybe you don't like to learn the commands and don't care about them when using Windows. But when you "get along" with Linux, if you don't understand some of the necessary commands, your administration will be difficult. Here are the 10 most useful Linux commands that make your administration a lot easier.

1. Top



Although in practice the *top* statement lists the executing tasks. Linux users often use this command when they want to know what program is taking up memory (or how much memory the system uses). Placing frequently used tools running on the desktop will help you know what's going on on your computer at all times. Sometimes, you can use a *terminal* (usually *aterm*) to place a window where you want it, then hide their frame. When there is no frame, the *terminal* cannot be moved, so you always have quick access to the information you need.

Top is a real-time reporting system, so when there is a change in the process it is immediately reflected in the *terminal* window . *Top* integrates some useful arguments (such as the *-p* argument to help monitor the PID of specific users), but when run by default, *top* will give you all the necessary information in the list of actions. The case is in progress.

2. Ln

For many administrators, the *link* is a special tool, it not only makes it easier for users to use, but also saves memory space. Suppose you administer a number of users who have continuous access to a large folder (including lots of files) on a drive. These users log into the same system and you don't want to have to copy the entire directory to each person's own directory, instead you just need to create a link */ link to the directory* . You do not need to use memory and users will access it faster. Of course when creating links between drives you will have to use *symlinks* . Another noteworthy use of *links* is the implementation of links between multiple directories to the original Apache data directory. Not only does it save memory, but the *link* also helps keep information secure.

3. tar / zip / gzip

Tar, zip and *gzip* are compression tools that make your administration much easier. These three tools can perform the same tasks. Without these tools, installing from the source file will not be easy and creating backup files will consume a lot of memory. One of the little-known features of this tool group is the ability to extract individual files from an archive. Now *zip* and *gzip* do this easier than *tar* because if you use *tar* to extract a *file* , you must know the exact size of the file. One feature of *tar / zip / gzip* makes simpler administration is to create a *shell* script that automatically performs the backup process. All three tools together with *shell scripts* are the most reliable, best and easiest to use backup tools you've ever seen.

4. Nano, micro, emacs

Introducing a text editor here will help resolve the conflict between *vi* and *emacs* , the best way is to put the *nano* editor into these two *editors* . Many people will assume that they are not statements when they are open applications. But these editors are used as command lines so they can be considered commands. Without a good text editor, administering a Linux machine will be difficult. Suppose you are trying to edit */ etc / fstab* or */etc/samba/smb.conf* on OpenOffice. For some people, it is not difficult, but you do not know that OpenOffice will insert hidden line breaks into the text file and may lead to changes in the configuration file. So the best way to edit *bash* files or configuration files is to use the *nano, vi* or *emacs editor.*

5. Grep

Many people do not pay attention to this quite useful tool.*Grep* prints the command line according to each user's own form. For example, when you are viewing the *httpd.conf* file longer than 1000 lines, and you are looking for the *AccessFileName.htaccess entry* . You may have to look at the file just to find the entry at line 429, or you can use the command *grep -n 'AccessFileName.htaccess' / etc / httpd / conf / http.conf* . Amazingly, when you enter this command, the system will respond to *'439: AccessFileName.htaccess' to* let you know that the item you

want to search is in line 439.

The *grep* command is also useful in controlling other commands. The example uses *grep* with the *ps* command (which helps to quickly save running processes). Suppose you want to know the PID of the Firefox browser that is not executing the process. You can use the *ps aux* command and search through the entire output of the Firefox entry or use the command *ps aux | grep firefox* , then you see the following data:

```
jlwallen 17475 0.0 0.1 3604 1180?  Ss 10:54 0:00 / bin / sh / home / jwallen /
jlwallen 17478 0.0 0.1 3660 1276?  S 10:54 0:00 / bin / sh /home/jlwallen/firef
jlwallen 17484 11.0 10.7 227504 97104?  Sl 10:54 11:50 / home / jlwallenfirefox
jlwallen 17987 0.0 0.0 3112 736 pts / 0 R + 12:42 0:00 grep --color firefox
```

You now know the PID of all current Firefox commands.

6. Chmod

Linux administration and security will be very difficult without the help of *chmod* . Suppose you cannot execute a *shell* command with the command *chmod u + x [filename]* . Of course that's not just about executing a file. Many Web tools require permission before installing. If you use the command *chmod -R 666 DIRECTORY /* in this case it is wrong. When licensing issues are encountered while installing an application, many new users immediately use the *666* command instead of checking the exact licensing level that the directory or directory needs. Although this tool supports administration, it should not be used without learning about it. You need to understand *chmod* before using. Remember that *w = write* (write), *r = read* (read) and x = *execute* , UGO means User, Group and Other. UGO is the simplest way to remember licensing for each object. So when granting *rw- rw- rw-* will allow all Users, Group and Other to have read and write rights. It is best to limit the Other to the permissions.

7. Dmesg

You might think running *dmesg* every time you connect a device to a Linux machine is 'outdated', but it is actually very important. This command displays messages from intermediate memory. A lot of information will be saved when using the *dmesg* command. You can find information about system structure, cpu, network device, kernel boot options used, RAM capacity, .

Use *dmesg | tail -f* to save the last lines of *dmesg* into your *terminal* . New entries are usually at the bottom of the *tail* . Always open this window when having trouble managing or troubleshooting a system.

8. kill / killall

One of the biggest benefits of Linux is stability. But applications outside the kernel do not always have this stability. In fact, some applications may be locked, and then you just want to unlock them. The fastest way to unlock the application is to use *kill / killall* command. The difference between these two commands is that *kill* requires PID while *killall* only requires the application name. Suppose Firefox is locked. If you use the *kill* command to unlock it, you must first determine the PID using the command *ps aux | grep firefox* . Once you have the PID, use the command *kill* PID (the location of PID is the actual PID number). If you do not want to lose time searching for PID, use the command *killall firefox* (although in some cases you will have to use *killall firefoxbin* command). Of course, the *kill / killall* command cannot be applied (nor should it be applied) to Apache, Samba, .

9. man

How many times have you seen RTFM? Many people think that this word stands for 'Read the Fine Manual'. But maybe it stands for 'Read the Fine Manpage'.*Manpage* shows you how to use commands. Generally, *manpages* are written in the same format, so when you know the format you can read (and understand) them. Do not underestimate the value of *Manpage* . When you do not understand the information received, you often scroll down to see the execution parameters of each command, and that is the most important feature of *Manpage* .

10. mount / umount

Without these two commands, using mobile devices or connecting external drives will not be possible. The mount / umount command is used to install a drive (usually labeled */ dev / sda*) to a directory in the Linux file structure. Both *mount* and *umount* commands are easier to use thanks to the */ etc / fstab file* . For example, if there is an entry in the */ etc / fstab file* of */ dev / sda1* that maps to */ data* , the drive can be installed with *mount / data* . In particular, the *mount / umount* command must have root privileges (if *fstab* does not have an entry that allows users to install or uninstall it). You can also use the *mount* command without arguments and you will see all the drives that are currently installed and where they map to (as well as system files and permissions).