



## **Linux+ Series Supplemental Information**

**Q1:** How does Linux kernel version numbering work?

**A1:** The Linux kernel version has three numbers separated by periods (let's use 2.4.1 as an example). It's important to know what each of these numbers represents.

The first number (2 in the example given) is the major version number.

The second number (4 in the example) signifies whether the release is stable or not. If this number is even (e.g. 0,2,4,6,8) then it's a stable release. All this means is that it has been well tested and has been determined to be bug free, or at least free enough from bugs that it's ready for the general public to use. If the second number is odd then this is an experimental release which means that there could be some serious bugs in it. But eventually all the bugs will be worked out and this experimental release will be turned into the next stable release.

The third number (1 in our example above) relates to minor upgrades. Perhaps a new driver or a small bug fix prompted a new minor version to be released.

**Q2:** How do you differentiate core services from non-critical services?

**A2:** Core services have low process identification (PID) numbers because they're typically started first. Critical processes in turn will start other non-critical services which may start other services and so on. You can use the **ps - f** command to see a detailed listing of all the processes on the system. For each process you'll also see the PPID (which signifies the parent process identification number). Another slightly more intuitive way to view parent process information is with the **pstree** command. This displays the parent/child relationship for each process in the system. The higher up in the tree a process is (i.e. the fewer parents it has) the more critical it is. For example, the **init** process is the top level process in the tree. It is responsible for starting a number of other processes. So, if it stopped running many other services would be disrupted.

**Q3:** What is the proper way to document the work performed on a system?

**A3:** There is no right or wrong way to document the work you perform on a system but that doesn't mean it's an unimportant task. It just means that there are many styles of representing the important information. For instance, it's essential to document the installation of the O.S., subsequent software installations and upgrades and also backups.

In a large installation documenting the work performed on a system is good practice because people change positions and a good body of documentation will give a new person some place to look when there is a problem.

The level of detail is something that you and your colleagues will have to agree upon but the basics are: date, start time and end time, a general description of work performed, commands or GUI software packages used, what options or flags were used, and outcome of the work. Typically all of this is entered in some note book (as opposed to a file on the hard disk) so that it can be referenced easily in the event of a system crash.

**Q4:** What are core files? Why are they so big? What should I do with them?

**A4:** When a program crashes it may do what we call "dump the core" (core is an antiquated term for main memory). A core file is basically a snapshot of the system at some moment in time. For an experienced programmer a core file is an essential piece of evidence that will help him track down the reason a program is crashing. Core files are literally named "core" and they contain the entire contents of main memory when the program crashed. For modern systems with gobs of main memory that's a lot of information!!

As a system administrator you would probably not be responsible for personally tracking down the reason behind program crashes but you can help matters along by forwarding core files to their rightful owners. Tracking down the owner may be as easy as doing **ls -l** on the directory that holds the core file and looking at the owner of the file named core. However, if that owner is root or some other account that isn't an actual user then you can try to determine what program created it by using **gdb** (the GNU Debugger) as follows. If you type **gdb -c core** then you'll see some relevant information regarding the core file like the program that created it. If for instance it was a web server program that did it you could pass the core file on to the person in charge of the web server development team and it would be their responsibility to follow through.



Like I said before core files can be very large and so if there are a number of them lurking on your system they could eat up a large amount of disk space. While deleting them may solve some disk space issues that you're having you should first make sure that the owner or other interested parties are not using them to track down program crashes.

**Q5:** How do I install, configure and/or update the XFree86 server?

**A5:** You can install the X server during the system installation like we did in the video. You can also install or upgrade the X server after the fact. That would be done using the rpm command or the dpkg command that we talked about in the packages video.

A command like **rpm -Uvh XFree86-4.2.1-3.i586.rpm** would do it in the Red Hat package management world and for Debian systems you would use a command like:

**dpkg -i Xfree86-server-4.2.1-3-i586.deb**

There are two basic ways of configuring X windows once it's installed. You can either configure it manually (meaning you edit configuration files by hand) or you can configure it with the help of various tools. Let's talk about editing the configuration file by hand first. The file is named **XF86Config** (or XF86Config-4 for version 4 of XFree86) and it is usually located in the **/etc/X11** directory. If you're going to edit this file then you should have a book or some other reference in front of you to help out because it has a quite complex format to it. Like I said, the other option is to use some configuration tool that is suited specifically for this purpose which is probably the wise choice for the novice. Two notable such tools are named: **xf86config** and **Xconfigurator**.

No matter which method you decide to use the problem that many people run into is that it seems like a lot of work to edit some configuration file or change a setting in the configuration tool and then reboot the system only to realize that the configuration change you made wasn't correct.

Well, you can circumvent rebooting the O.S. over and over again if you use the telinit command that we talked about in the videos. Change the system to runlevel 3 (by typing **telinit 3**) so that X does not start automatically. You can make your configuration changes and type **startx** which will start the X server on your machine. If the changes are not satisfactory you can just go back to the configuration step and try again and type **startx** again when you're ready to test your configuration changes. If you are satisfied with the new configuration then you're done (don't forget to set the system back to a "normal" runlevel by typing **telinit 5** for instance).

**Q6:** How do I download and install patches and updates?

**A6:** The same way you downloaded and installed software from scratch, you just might have to use different settings. Suppose that you have version 1.2 of the the software package called Neatopeachykeen. If you want to upgrade this package to the newest version (say 1.3) by using the same commands you used to install the software initially but this time make sure you specify the "upgrade" flag instead of the "install from scratch" flag.

If you're using the red hat package manager the -U option takes care of it all for you. It will install the software if it's not already on your system or if it is installed already it will just update to the new version.

If you're using the debain package manager then there's an upgrade command as part of the apt-get utility that will upgrade a specified package to it's latest version.