# Mastering Linux, part 13

In this month's instalment of the Mastering Linux series, *Jarrod Spiga* discusses file system management, and examines interactive shell scripting.

**Skill level**
Advanced

**Requirements**
An installation of Linux (Fedora Core 4 and Red Hat Enterprise Linux ES3 were used in the writing of this article).

**Time to complete**
3 hours

**In this series . . .**

Part 12 — November '05
System administration

Part 13 — December '05
File systems and interactive scripts

Part 14 — January '06
Remote X-Windows

. . . and more

The ability to manage the file systems on your Linux PC is an important administration task. For instance, if you add a new hard disk to your system, it won't be accessible until you identify, partition, format and mount it within the current file system. Alternatively, if you need to remove a hard disk, it needs to be gracefully removed before pulling the plug.

## FILE SYSTEM CREATION

As mentioned above, the first step in creating a new file system is to identify the device name for your new hard disk (or other storage device). If you cast your mind back to the first instalment of this series (*APC* December 2004, page 98) , it mentioned the convention used to identify hard disks in your system. The following table shows a summary of this:

| | | |
|---|---|---|
| IDE and SATA | hda, hdb | The primary master and primary slave devices respectively. |
| | hdc, hdd | The secondary master and secondary slave devices respectively. |
| | hde, hdf, ... | Additional drives connected to other controllers in your system. |
| SCSI | sda, sdb, ... | Devices are assigned in order of controller id, followed by SCSI id. |

The parted command can be used to identify storage devices. For example:

```
parted /dev/hdb print
```

displays information relating to the device that has been assigned as hda. If you have a brand new hard disk, you may receive an Unrecognized Disk Label error. This error means the disk simply needs to be prepared for use with the following command:

```
parted /dev/hdb mklabel msdos
```

After you've identified the device, it needs to be partitioned. The fdisk utility is one of the simplest tools that you can use for this purpose. To launch it, simply pass the device as an argument to the command. For example:

```
fdisk /dev/hdb
```

Once fdisk is running, you can use single-keystroke commands to partition the disk. In general, there are

## INTERACTIVE SHELL SCRIPTING

Parts 10 and 11 of the Mastering Linux series ("Mastering the shell", *APC* September, page 116; "Scripting and customisation", *APC* October, page 116) explained how to create shell scripts to help you perform tasks more efficiently at the command line. However, shell scripts can also produce X-Windows dialogue boxes, providing additional functionality to your scripts under a GUI.

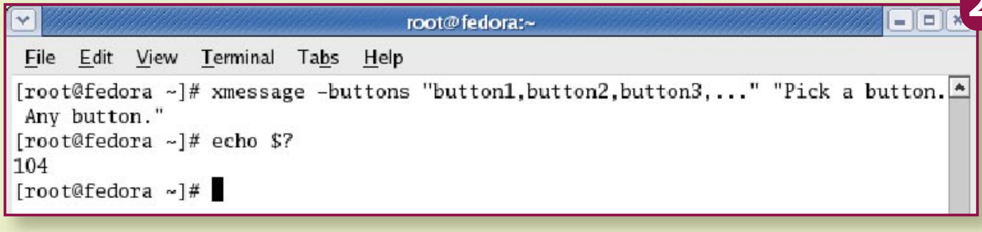
The xmessage command is used to bring up the dialogue box and requires that the message and options are passed as options to the command, as follows:

```
xmessage -buttons "button1,button2,etc"
 "message"
```

where button1,button2 and so on are the labels for the buttons in the dialogue box and message is the message that appears in the dialogue box above the buttons.

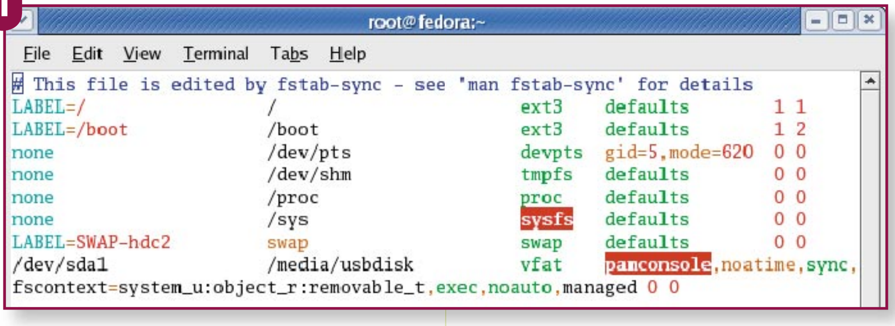
❷ A demonstration dialogue box is shown below. Depending of which button is clicked, a value is passed back to the script as the "$?" variable. If the first button is clicked, this value is 101; if the second button is clicked, 102 is the returned value; and so forth. Processing

2



```
[root@fedora ~]# xmessage -buttons "button1,button2,button3,..." "Pick a button.
 Any button."
[root@fedora ~]# echo $?
104
[root@fedora ~]# 
```

1



```
# This file is edited by fstab-sync - see 'man fstab-sync' for details
LABEL=/                 /                       ext3    defaults        1 1
LABEL=/boot             /boot                   ext3    defaults        1 2
none                    /dev/pts                devpts  gid=5,mode=620  0 0
none                    /dev/shm                tmpfs   defaults        0 0
none                    /proc                   proc    defaults        0 0
none                    /sys                    sysfs   defaults        0 0
LABEL=SWAP-hdc2         swap                    swap    defaults        0 0
/dev/sda1               /media/usbdisk          vfat    pamconsole,noatime,sync,
fscontext=system_u:object_r:removable_t,exec,noauto,managed 0 0
```

a few rules that you need to remember when creating partitions in fdisk:

- Hitting the m key will bring up help.
- A maximum of four primary partitions, or three primary and one extended, can be created on any device. Extended partitions can't be formatted, but can contain logical partitions. Partitions are created by pressing n.
- Partition Hex code 83 is for an ext2/ext3 Linux file system. Partition Hex code 82 is for a Linux Swap file system. Partition formats are changed by pressing t.
- Linux Swap file systems can't be mounted. The space assigned to these partitions is reserved for virtual memory.
- Changes won't be committed to the disk's partition table until you write your changes (by hitting the w key).

Each primary or logical partition that you create will be assigned a number to differentiate it from other partitions on the disk. Be sure to note down the numbers assigned to each partition.

## FORMATTING

After creating your partitions, you'll need to format them before they become useful. If you're reformatting an existing partition, check that it's not mounted before formatting. Swap partitions are created with the mkswap command:

```
mkswap /dev/hdb1
```

while partitions that store data are created using the mkfs series of commands. The actual command you'll use depends on the format you want for the partition. Most of the time, you'll want to create ext3 partitions for Linux (using the mkfs.ext3 command):

```
mkfs.ext3 /dev/hdb1
```

Other file system formats are supported under Linux, including FAT16, ReiserFS, NFS, iso9660 and others. While creating file systems of these types is outside the scope of this Workshop, the process is similar to that mentioned above. Just remember to ensure that you match the file system type when creating and formatting the partition.

## PREPARE FOR MOUNTING

Technically, mounting a file system is a two-step process. But if you're permanently mounting a new file system, a third step is often inserted between the others.

Step one is to create a mount point — an empty directory where your new file system will attach to your current Linux file system. You may use an existing directory to mount the new file system on, but the contents of this directory will no longer be accessible once you've mounted the file system.

The optional second step is to edit the /etc/fstab file (this is an abbreviation for file system table). The contents of this file instruct the system how to mount a number of predefined file systems. It's advantageous to add the details of the new file system to this file as it simplifies the process of physically mounting and unmounting the new file system on the current one.

❶ The image (above, left) shows the typical contents of the /etc/fstab file, while the following table details what each column in the table means and gives an example of the usage for mounting a hard disk partition:

▶

which button has been pressed is then a simple matter of checking the result — of the "$?" variable from within the script.

The basic print and backup script shown in part 10 of the Mastering Linux series could easily be extended to incorporate a confirmation screen as follows:

```
#!/bin/bash
ARCHIVE= "$2"
xmessage -buttons "Yes,No" "Are you sure
that you want to print and archive
$ORIGINAL?"
if [ $? -eq 101 ]; then
lpr "$ORIGINAL"
cp "$ORIGINAL" ~/backup/mastering_linux_
"$ARCHIVE"
fi
```

Command substitution can also be used as the button argument in the xmessage command, but remember that all of your buttons need to be comma-delimited. Many Linux commands can automatically format the output to work in this manner. For instance, the ls -m command and switch will provide a comma-separated directory listing, as can be seen in the following example:

```
#!/bin/bash
X=100
for FILE in *.txt; do
if [ $? = $INDEX ]; then
lpr "$FILE"
cp "$FILE" ~/backup/"$FILE".bak
fi
done
```

Even if the command can't format things the way you want, you can use the stream editor (sed) to do the work on the output of your command that gathers the info for your buttons.

## CALLING SCRIPTS FROM NAUTILUS

Part 10 of this series also showed that all executable scripts saved to the bin directory under your home were executed, regardless of your current working directory (thanks to the $PATH environment variable).

Another special directory may exist under your home directory for the creation of Nautilus (the GNOME file manager) scripts. This directory is located at ~/.gnome2/nautilus-scripts. Note the period before gnome2 — this indicates that the directory is hidden from general view in the file system. If this directory doesn't exist on your system, you may

▶

| | | |
|---|---|---|
| **Device** | /dev/hdb1 | The device name given to the file system that you're mounting. |
| **Mount Point** | /mnt/db | The location of the directory that is being used as the mount point. |
| **File system** | ext3 | The format of the file system being mounted. |
| **Mount options** | defaults | The options used to mount the partition (see other table). |
| **dump options** | 0 | Specifies whether a file system should be scanned for backup by the dump command. 0 = don't scan, 1 = scan. |
| **fsck options** | 1 | Specifies whether a file system should be checked when the fsck (file system check) command is run, and what order file systems are scanned in. 0 = don't scan. |

A number of different mount options can be specified in the /etc/fstab file (most of these options can also be used with the following mount commands):

| | |
|---|---|
| autolnoauto | Specifies that the file system will be automatically mounted at system boot (or not). auto is the default. |
| userlnouser | Specifies whether regular users can mount or unmount the file system, or whether this privilege is limited to the root user. nouser is the default option. |
| execlnoexec | Specifies whether executable files can be run on the file system (or not). exec is the default option. |
| rolrw | Specifies whether the file system is read-only or readable and writable. rw is the default option, but you'll want to mount optical devices as read-only (ro). |
| synclasync | Specifies whether data is committed to the file system instantly or cached for later action. Default is async, but it's probably better to use synchronous operation for removable media. |
| defaults | Specifies that the file system should be mounted using all default values. |

## MOUNTING

The final task is to attach the file system to the mount point, using the mount command. If you've added an entry for your new file system in /etc/fstab, mounting the file system is as simple as passing the mount point as an argument to the mount command:

```
mount /mnt/db
```

If you skipped the middle step, the mount command that you need to issue is more complex, since you have to manually pass the mounting options to the command. The general format of the command is:

```
mount -t fstype [-o options] device mount_
point
```

where fstype is the file system format, options are the mount options you wish to use (listed in the table above), device is the device name for the file system and mount_point is your mount point. Thus, the following command would mount your database file system in the same manner as shown above, assuming that step two was skipped:

```
mount -t ext3 /dev/hdb1 /mnt/db
```

## UNMOUNTING

The process of unmounting a file system is a no-brainer. Entering:

```
umount /mnt/db
```

simply passes the name of either the mount point or the device name to the umount command. apc

## INTERACTIVE SHELL SCRIPTING CONTINUED

simply create it to allow Nautilus scripts to be launched.

❸ A Nautilus script is just like any other shell script, except that it's also executable via a context menu from within Nautilus. The context menu can be brought up by right-clicking within Nautilus and selecting the Scripts branch.

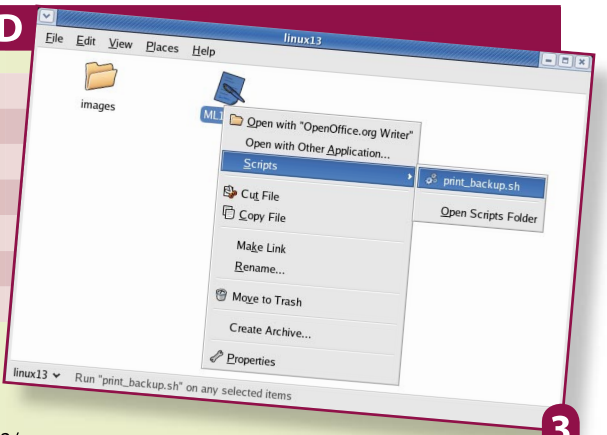
If you have one or more files selected when you bring up the context menu and execute a Nautilus script, the files are passed to the script as arguments; $1 for the first file, $2 for the second and so on. The $@ variable can also be used to access the entire list of arguments.

Extending the print and archive script above so that it operates as a Nautilus script can be done as follows (assuming that you change the archive file name to be relative to the input file name):

```
#!/bin/bash
for FILE in $@; do
if [$? -eq 101 ]; then
lpr "$FILE"
cp "$FILE" ~/backup/"$FILE".bak
fi
done
```

When you've finished writing the Nautilus scripts, ensure that they are saved under the ~/.gnome2/nautilus-scripts directory and that you make the script executable.

Of course, if you don't want to create your own, a number of useful Nautilus scripts can be downloaded from **http://g-scripts.sourceforge.net**.



3

## Next month. . .

**The next instalment of the Mastering Linux series will show you how to configure a Linux PC so that it can run in X-Windows from a remote machine — be it Windows or another Linux PC.**