USB Flash Memory HOWTO

Niko Sauer

Unit of Advanced Study
University of Pretoria, South Africa

< nikos@friedrichs.up.ac.za>

Revision History

Revision 0.06	2004−03−10	Revised by: ns
Added section on Linux−2.6.		
Revision 0.05	2003−12−26	Revised by: ejh
Technical review.		
Revision 0.04	2003−12−11	Revised by: ns
Revision 0.03	2003−12−02	Revised by: ejh
Technical review.		
Revision 0.02	2003−10−30	Revised by: ns
Changed 2003−11−20		

This document describes procedures for installing USB Flash Memory devices and the way of formatting them for various file systems such as vfat (the way they usually come) and ext2 (the way they fit in better with Linux). It is also described how to partition the device in two sections with different file types.

Table of Contents

Table of Contents

1. Purpose

The purpose of this document is to describe procedures for implementing USB flash memory devices (memory sticks) on Linux. This entails the following:

- making over−the−counter devices, pre−formatted for the MS Windows system, work on Linux;
- formatting the device in the ext2 filesystem on a single partition and making it work. This is intended for situations where the device will be used exclusively on Linux machines;
- formatting the device on a dual partition in the ext2 and vfat filesystems and making it work. The purpose of this is for situations where Linux + Linux and Linux + Windows are possible;
- making the device more user friendly;
- maintaining the filesystem of the device;
- using additional features of Linux−2.6.x.

The procedures described in this document should work on any USB memory stick device which can be operated as a read−write device. Read−only devices do not fall in this category. The procedures will not work if the write facility of the device is disabled (i.e. if the write protect switch is on).

The procedures described for formatting and maintenance in particular filesystems may (in principle) be adapted for other filesystems such as ext3 and reiserfs. These have not been tried and tested. Neither has any attention been paid to adaptation of the procedures to other related operating systems.

It may be dangerous to re−format the flash memory intended for a digital camera. The camera may then be unable to understand it. Use the camera's built−in menu system to do this reformatting for you.

2. Physical devices

Flash memory devices come mainly in the form of small, portable devices often referred to as *memory sticks* or *keychains*, and as part of digital cameras. They are *non−volatile devices* which operate on the principle that electric charges are used to represent data in binary format. These charges can remain unchanged almost indefinitely, but changes (such as writing to the device) limit the life span of the device (100000 writes of 8MB each).

Memory sticks plug directly into a USB−port at the back of your computer. The power it needs is supplied by the USB−port. It is sometimes convenient to use a USB−extension cable to bring the device within easy reach.

Memory sticks usually have *write protect switches* which should be turned off if you want to use it as read−write device. If the switch is in the "on" position, the device is *read−only*. The devices are usually equipped with a led (light emitting diode), which indicates that the device is operational. The led also flickers while data is being transferred to or from the device.

Digital cameras use flash memory for image storage. These are powered by the camera and connected to the computer's USB−port via an electronic interface. To be able to connect to the computer, the camera's power must be turned on for the duration of the interfacing, and should only be turned off after the device had been unmounted from the computer. It is important to remember that the device takes power from the camera's battery. The operation should therefore be terminated as soon as possible.

3. Buying a memory stick

Important considerations when buying a memory stick are the following:

The size of the memory
: This will depend on whatever use you have in mind. 128MB seems to be a popular choice.

Is it known to work on Linux?
: The established brands have web sites in which this may be stated explicitly. Alternatively, ask the supplier. But beware: the sales person may not know what you are talking about. The instruction leaflet may also mention something like: Works on Linux−2.4, or may have a picture of the Linux penguin. If these inquiries fail, it may be wise not to buy.

Extension cable and keychain
: Memory sticks usually are packaged with an extension cable (Section 2) and a keychain. The latter provides a safe and with−it way of transporting the device. Make sure that these are included. You may need an additional extension cable, depending on circumstances. Laptop and notebook computers usually don't need them.

File system
: Over−the−counter memory sticks are usually formatted in vfat (msdos), and this will work in Linux, but you will not have the strength and versatility of the standard ext2 (or other) file system. This is not a problem. It can be endowed with other file systems as will be described later in this document.

Cost
: Prices of memory sticks of a given size may vary considerably. It is a good idea to find out what that range is. The internet is excellent for the purpose.

---

4. Reading

Users of this document might want to acquaint themselves with the following: [1]

- Manual pages of the following tools: mount, umount, fdisk, mke2fs, mkdosfs, dumpe2fs> and fsck.ext2.
- Linux documentation project
- USB−Digital Camera HOWTO *
- The Linux Partition HOWTO *
- The Linux 2.4 SCSI subsystem HOWTO
- The Linux Kernel HOWTO
- The Linux USB subsystem *
- Linux filesystem hierarchy
- The Linux Kernel
- Linux Kernel 2.4.xx/Documentation (should be in `/usr/src/linux/Documentation`)

5. Hardware and kernel

5.1. Setup

This is a description of the salient features of the setup used to develop the procedures described below. All the procedures have been tried and tested, also for Linux−2.6. The screen−like displays are precise copies of what appeared on my screen.

- Hardware: Intel (R) Celeron (TM) 1100 MHz
- Distribution: RedHat Linux 7.0 (extensively modified)
- Kernel: Linux−2.4.20 (from www.kernel.org). See also Section 12 for Linux−2.6.x.
- Tools: util−linux−2.11z (mount, umount, fdisk); e2fsprogs−1.32 (mke2fs, dumpe2fs, fsck.ext2); mkdosfs−2.2

5.2. Kernel options

It is uncertain if USB−support is sufficient in kernels earlier than 2.4.xx. The following support, relevant to this document, was compiled into the kernel. A modular approach may also be followed.

5.2.1. SCSI support

- SCSI support (CONFIG_SCSI scsi_mod.o)
- SCSI disk support (CONFIG_BLK_DEV_SD sd_mod.o)

5.2.2. File systems

- DOS FAT fs support (CONFIG_FAT_FS fat.o)
- MSDOS fs support (CONFIG_MSDOS_FS msdos.o)
- VFAT (Windows 95) fs support (CONFIG_VFAT_FS vfat.o)
- /proc filesystem (CONFIG_PROC_FS)
- Second extended fs support (ext2fs) (CONFIG_EXT2_FS ext2.o)

5.2.3. USB support

- Support for USB (CONFIG_USB usbcore.o)
- Preliminary USB device filesystem (CONFIG_USB_DEVICEFS)
- USB Mass Storage support (CONFIG_USB_STORAGE usb−storage.o)

5.3. Notes

In the lists above the entities in uppercase refer to the variable names to be found in the `.config` file in the upper level directory of the kernel source (`/usr/src/linux/`). The entities `xxx.o` refer to the modules created when a modular approach is followed. When there is no reference to a module, the option can only be hard−compiled into the kernel.

Different kernel versions may have different indications of options when, for example, *make menuconfig* or *make xconfig* are run. Variables such as CONFIG_USB, which can be gleaned from the various *help* options, may be a more reliable indication.

Very recent Linux distributions such as *RedHat* and *SuSE* probably have the appropriate kernel options compiled in.

Under USB−support, options for a number of digital cameras are available.

Please consult the relevant texts as set out in Section 4 if you consider (re)compiling your kernel.

6. Some basic concepts

In this section we discuss in a very brief manner, some basic Linux concepts which should contribute to an understanding of the procedures described below.

6.1. The /proc filesystem

The /proc filesystem serves as a window through which we can see the workings of a Linux setup. The objects of most interest for this document are the directories`/proc/bus/usb/` and `/proc/scsi/`. These will be used to verify that the kernel had been set up properly for our purpose (Section 7.2).

6.2. SCSI devices

Your memory stick will be considered as a USB mass storage device posing as a removable SCSI disk (sd). SCSI disk devices are mapped to the `/dev` (devices) directory under `/dev/sda` , `/dev/sdb` , ... When different disk devices are present, they will be mapped to `/dev/sda`, `/dev/sdb`, etc. If, for example, a memory stick and a digital camera are plugged in, the one would be mapped to `/dev/sda` and the other to `/dev/sdb`. Tests have indicated that the first device to be detected is mapped to `sda` and represented in the directory `/proc/scsi/usb-storage-0`. The second will be mapped to `sdb` and represented in `/proc/scsi/usb-storage-0`. What will happen when both devices are present at boot−up, is not known. An overview of the partitions relevant to this document, the file `/proc/partitions` may be consulted. Look for entries like `sda, sda1, sdb`. Under the Linux−2.6 series of kernels this problem may be very neatly resolved (Section 12).

In the descriptions further on, it will be assumed that only one memory stick is present, and therefore mapped to `/dev/sda`.

In most distributions these points exist in abundance. You should check that this is the case in your setup (**`ls /dev/sda*`**. If there are none you can create them by doing (as root, indicated by the hash).

```
# mknod /dev/sda  b 8 0
              # mknod /dev/sda1 b 8 1
              # mknod /dev/sda2 b 8 2
```

etc, up to `/dev/sda15` if necessary. The device as a whole is mapped to `/dev/sda`, while `/dev/sdax (x = 1 ... 15)` represent the various partitions of the device. We shall describe, later on, how to create different partitions for different file systems (see Section 8 and Section 9). If you intend to use your memory stick with a single partition (filesystem), `/dev/sda1` is sufficient.

6.3. The USB device filesystem

This dynamically generated filesystem should be mounted at `/proc/bus/usb/`. It is therefore essential that this mount point exists. Once it is mounted, there should be more to be seen in `/proc/bus/usb/` and in `/proc/scsi/` (Section 7.3). To find out which USB devices are present, read the file `/proc/bus/usb/devices` (**less /proc/bus/usb/devices**). It takes a small effort to interpret the writing on the screen, but it is not difficult. A memory stick is indicated by *Mass Storage Device*.

6.4. The ext2 filesystem

The ext2 (second extended) filesystem still is the most prevalent in Linux setups. It is most versatile and sophisticated, carrying with it permissions (read−write−execute, who is permitted to do what), ownership (user, group, others), a timestamp (when last modified), etc. Moreover, tools for the proper maintenance of a device endowed with ext2, exist (Section 11). If a flash memory device is to be used exclusively within Linux setups, it is therefore desirable to format it in ext2 (see Section 8).

6.5. The vfat filesystem

Enabling vfat in the kernel makes it possible to mount filesystems created under Dos/Windows to be mounted in a Linux system. Most memory sticks are formatted for direct use on Windows and should therefore be considered as vfat−formatted. The vfat filesystem is less sophisticated than ext2 and will therefore be more economic as far as memory space is concerned. It does not carry permissions in the same way as ext2 with the result that an ext2−file saved in vfat will re−appear with its permissions changed. Nonetheless, if a memory stick is to be used for the purpose of carrying data between Linux and Windows machines, it is best to keep it in the vfat format. A compromise is to partition the memory stick in two parts: one in vfat and the other in ext2. At least the Linux machine should be able to handle both. This is described in Section 9

6.6. Explanation of command options

Here is a short list of command options that occur in this document:

- **`mount −t ext2 /dev/sda1 /mnt/memstick`** Mount in filetype ext2, device sda1 at mountpoint /mnt/memstick
- **`ls −l`** Make a complete list (modes, ownerships, etc.)
- **`mkdosfs −F 32 /dev/sda1`** Create an ms−dos filesystem, FAT−size 32, on the partition /dev/sda1
- **`ln −s /dev/sda1 /dev/flash`** link, symbolically, the existing /dev/sda1 to the symbolic /dev/flash
- **`mkdir −m 777 /mnt/memstick/superdir`** create a new directory with mode rwx for user, group and others
- **`dumpe2fs −h /dev/sda1`** Show the header of the ext2 partition /dev/sda1

7. Basic verifications

In order to make sure that the device you look at and write on is the one intended, it is best to remove similar devices from the USB−bus before starting these procedures.

7.1. A quick check

It is possible that your system is already set up for handling a USB−flash memory device. To check this, simply issue the command $ **mount** from an X−terminal. If something like

```
          none on /proc/bus/usb type usbfs (rw)
```

appears among the output, you are ready to go to Section 7.4 and proceed from there. It may be virtuous, however, to glance through the intervening sections. If the test is unsuccessful, all is not lost. Please read on.

7.2. Probing the /proc filesystem

A few things in the `/proc` directory can be checked so see if the kernel options have been properly included or the appropriate modules properly inserted. As a first check, see if the directory `/proc/bus/usb` exists. If it does, your kernel supports the USB−subsystem correctly. If this is not the case the kernel has to be re−compiled with appropriate USB support (see Section 5.2.3) or the kernel has to be updated. Next, check if the directory `/proc/scsi` exists. If it does, you're well−away. If it doesn't, SCSI support has not been compiled into your kernel (see Section 5.2.1).

In `/proc` there should be provision for a mount point for the USB filesystem. That point is `/proc/bus/usb`. If it exists, the kernel is correctly set up.

7.3. Mounting the USB filesystem

If the verifications of the previous sections were positive, the next step is to mount the USB filesystem. This is issued from root as follows:

```
# mount -t usbfs none /proc/bus/usb
```

In older kernel versions the mount command above may have to be changed by replacing *usbfs* with *usbdevfs*. In Linux−2.4.20 both forms work.

If there are no complaints, do some more tests. The first is the quick test (Section 7.1). A deeper probe would be to issue the command # **ls -l /proc/bus/usb** . This should give something similar to

```
            dr-xr-xr-x 1 root root 0  Sep 19  14:21  001
            dr-xr-xr-x 1 root root 0  Sep 19  14:21  002
            -r--r--r-- 1 root root 0  Sep 19  22:30  devices
            -r--r--r-- 1 root root 0  Sep 19  22:30  drivers
```

The path `/proc/scsi/usb-storage-0/` should now exist and one of the files at the end will show this. In my setup the command $ **`less /proc/scsi/usb-storage-0/1`** gives

```
                Host scsi1: usb-storage
                    Vendor: Generic
                   Product: Mass Storage Device
             Serial Number: None
                  Protocol: Transparent SCSI
                 Transport: Bulk
                      GUID: 0ed166800000000000000000
                  Attached: Yes/No
```

If the flashdrive is present, the last line will have the entry "Yes". If it is not plugged in, the entry will be "No".

7.4. Tests −− vfat

You are now ready to find out if the memory stick is working. Let us suppose that you bought it over the counter. Its instruction leaflet will most likely tell you how to set it up for Windows. That is an indication that it is formatted in vfat. Before you try to mount it, create a mount point for it. Do something like # **`mkdir -m 777 /mnt/memstick`**. The mounting command would most likely be

```
# mount -t vfat /dev/sda1 /mnt/memstick
```

If everything went smoothly, you should be able to see the device: # **`ls /mnt/memstick`**.

Now try some standard things like making a directory on the device and copying a favourite text file to it:

```
# mkdir /mnt/memstick/apollo
             # cp /home/myname/myfavourite_file /mnt/memstick/apollo/.
```

List again (# **`ls -l /mnt/memstick`**) and pay attention to the permissions.

Unmount the device (# **`umount /dev/sda1`**) and mount it again as above. List again and check the permissions. Most likely your favourite text file will now have an x−permission. It became executable. That is normal in the vfat filesystem. If you are happy with that, unmount the device and skip to Section 10 .

8. Setting up an Ext2 filesystem

The procedures described here will give the entire flash memory device a single partition endowed with the ext2 filesystem. This will make the device more suitable for use between Linux machines. Don't do this if the device is supposed to operate between Linux and Windows machines.

The device associated with `/dev/sda` will be re−formatted. Any data present on it will be destroyed. To make sure that the device you work with is the one intended, remove similar devices from the USB−bus before starting these procedures.

8.1. Partitioning

It is assumed that the flash memory is mapped to `/dev/sda`. In this section we treat the situation where the whole flash memory device will become devoted to a single ext2 partition. In the example procedure shown below, a 128MB flash memory was formatted for ext2. That is why figures like 131MB and 888 cylinders appear. The fdisk tool, which is used, simply reads that from the device.

All operations are done by root. The single partition will be created on `/dev/sda` (please note: not `/dev/sda1`) The procedure is described in a series of steps with comments. The standard prompt of fdisk is `Command (m for help)`**:** and you can, at any stage enter **m** to see the available commands. If you do that the result would be

```
             Command   action
                a      toggle a bootable flag
                b      edit bsd disklabel
                c      toggle the dos compatibility flag
                d      delete a partition
                l      list known partition types
                m      print this menu
                n      add a new partition
                o      create a new empty DOS partition table
                p      print the partition table
                q      quit without saving changes
                s      create a new empty Sun disklabel
                t      change a partition's system id
                u      change display/entry units
                v      verify the partition table
                w      write table to disk and exit
                x      extra functionality (experts only)
```

Your memory stick must be plugged in, but not mounted. Take care that write protect is off.

Here goes:

```
# fdisk /dev/sda
             Command (m for help):d {enter }
             Selected partition 1
             Command (m for help):n {enter}
             Command action
                e  extended
                p  primary partition (1−4)
             p {enter}
             Partition number (1−4):1 {enter}
```

```
First cylinder (1-888, default 1): {press enter}
Using default value 1
Last cylinder ... (1-888, default 888): {press enter}
Using default value 888
```

Now is a good time to verify that all went well by printing out the partition details and table.

```
Command (m for help): p
Disk /dev/sda: 131 MB, 131072000 bytes
9 heads, 32 sectors/track, 888 cylinders
Units = cylinders of 288 * 512 = 147456 bytes

Device     Boot Start End Blocks Id System

/dev/sda1          1   888 127856 83 Linux
```

Make sure that the partition is not set up as bootable. There should not be an asterisk under the heading "Boot" in the partition table. Furthermore, the headings "Id" and "System" should be as they are in the printout shown above. That indicates that you can format the device in ext2 (next section). These are the defaults. If they are not, they can be changed by

```
Command (m for help): a [toggle a bootable flag]
Command (m for help): t [change a partition's system id]
```

If (or when) the partition table is correct, you can conclude the procedure with

```
Command (m for help): w [write table to disk and exit]
```

That's it!

8.2. Making an ext2 device

Having completed the partition part, we go straight on to "formatting" the device in ext2. For this we use the tool mke2fs. This is straightforward.

```
# mke2fs /dev/sda1
```

You might like to watch the led on your memory stick flicker while this is happening. When it stops, the job is done.

8.3. Tests -- ext2

To see if you were successful, essentially repeat the procedures described in Section 7.3 and Section 7.4 with two exceptions of detail. In the first place the mount command should be

```
# mount -t ext2 /dev/sda1 /mnt/memstick
```

The second exception is that the permissions on your favourite text file should not change anymore.

9. Setting up a dual partition

9.1. Partitioning

The procedures described here will partition the flash memory device in two pieces −− one of the partitions will be formatted in ext2 and the other in vfat. This may be handy when the device is to be used to transport data between two (or more) Linux machines (ext2) and between Linux and Windows machines (vfat)

The device mapped to `/dev/sda` will be re−formatted. Any data present on it will be destroyed. To make sure that the device you work with is the one intended, remove similar devices from the USB−bus before starting these procedures.

In this section we describe how to set up a dual partition for the memory stick: One half of the device will be devoted to the vfat filesystem, and the other will be ext2. In this way your device will be more versatile, but also a little shattered. The procedure will be given in somewhat less detailed than in Section 8.1 . Note that the procedure of the previous section is repeated and that the hex code for each file type is explicitly entered (lines following the entry "t": Win95 = b; Linux = 83). When the partition will only be for Linux, this is unnecessary −− Linux is the default. Also note that the vfat partition goes on first −− that is the way Windows would have it.

```
# fdisk /dev/sda

             Command (m for help): d
             Selected partition 1

             Command (m for help): n
             Command action
                e   extended
                p   primary partition (1−4)
             p
             Partition number (1−4): 1
             First cylinder (1−888, default 1):
             Using default value 1
             Last cylinder or +size or +sizeM or +sizeK (1−888,
                   default 888):444
             Command (m for help): t
             Selected partition 1
             Hex code (type L to list codes): b
                           Changed system type of partition 1 to b (Win95 FAT32)

             Command (m for help): n
             Command action
                e   extended
                p   primary partition (1−4)
             p
             Partition number (1−4): 2
             First cylinder (445−888, default 445):
             Using default value 445
                           Last cylinder or +size or +sizeM or +sizeK (445−888,
              default 888):

             Using default value 888

             Command (m for help): t
             Partition number (1−4): 2
             Hex code (type L to list codes): 83
```

```
Command (m for help): p

                    Disk /dev/sda: 131 MB, 131072000 bytes
9 heads, 32 sectors/track, 888 cylinders
Units = cylinders of 288 * 512 = 147456 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/sda1             1       444     63920    b  Win95 FAT32
/dev/sda2           445       888     63936   83  Linux


Command (m for help):w
```

Note that we went on with the second partition before writing.

The choice of splitting the device right in the middle (1−−444; 445−−888 for a 128 MB memory stick) is arbitrary. Any other well−conceived splitting would be fine.

If you had more than one partition to begin with, the "delete" part of the procedure would have to be repeated.

9.2. Making a dual ext2/vfat system

This step is a repetition of Section 8.2. The first step makes the dos partition, the second makes the Linux ext2 partition.

```
# mkdosfs −F 32 /dev/sda1
             # mke2fs /dev/sda2
```

9.3. Tests −− ext2 and vfat

Simply carry out the tests of Section 7.4 and Section 8.3 on the two devices separately. The mount commands would be

```
# mount −t vfat /dev/sda1 /mnt/fatstick
             # mount −t ext2 /dev/sda2 /mnt/memstick
```

to mount both partitions on your device.

9.4. Configuring Linux for the dual partition

The Linux machine you did the partitioning on, will be configured for the dual partition. Evidently you need to carry the keychain to another Linux machine. There you will find that it only sees `/dev/sda1`. This is, perhaps, a shortcoming of the kernel. The workaround is as follows: On the new host machine do (keychain plugged in but not mounted)

```
# fdisk /dev/sda
                   Choose p; check partition table
```

```
          If satisfied, choose w
# less /proc/partitions
```

Your device with its partitions should be there. Data on your keychain should be untouched. For a single partition this is not necessary.

10. Making the memory stick accessible to users

What we have described so far, are simply the procedures of setting up the system and formatting memory sticks. We have to make the mounting of the USB filesystem more permanent, and we have to make mounting of such a device easier for non−root users. There are only a few things to be done.

The procedure described below is for the more complex situation of a memory stick with a dual partition (Section 9). If the partition is a single ext2, the procedure should be simplified by leaving out the line referring to the vfat part and changing `sda2` to `sda1`.

Suppose that the mount points `/mnt/memstick` and `/mnt/fatstick` have been created. You do not have to use those words and they don't even have to be sub−directories of `/mnt`

To make remembering easier, create the following symbolic links:

```
# ln -s /dev/sda1 /dev/fatflash
# ln -s /dev/sda2 /dev/flash
```

For smoother and easier mounting add the following lines to `/etc/fstab` : (*See NOTE at the end of* Section 7.3)

```
none          /proc/bus/usb usbfs       defaults       0 0
/dev/flash    /mnt/memstick ext2,vfat   rw,user,noauto 0 0
/dev/fatflash /mnt/fatstick vfat        rw,user,noauto 0 0
```

The middle line enables mounting either in ext2 or vfat. The last two lines make it possible for any user to mount the device by the command

```
$ mount /dev/flash  [for ext2 or vfat]
             $ mount /dev/fatflash  [for vfat]
```

It is possible to mount both partitions simultaneously. See Section 9.

In the case of an ext2−formatted device it does not seem possible for a user (other than root) to mount in read−write mode. The solution is to create a directory on the memory stick owned by the particular user with complete permissions. Root should do that:

```
# mount /dev/flash
             # mkdir -m 777 /mnt/memstick/superdir
             # chown charles:charles /mnt/memstick/superdir
```

If the device is in the vfat format, this step is unnecessary.

In conclusion, here is a small script I use to mount and unmount my (ext2) flash drive:

```
#!/bin/bash
EXCODE="keepit"
green='\033[0;32m'
```

```
yellow='\033[0;33m'
## ____________________________
## Function to echo in colours
echo_in_color ()
{
     message=$2
     message1=$4
     color=$1
     color1=$3
     echo -e -n $color
         echo -n $message
     echo -e -n $color1 " "
     echo  -n $message1 " "
     tput sgr0
     return
}
## ____________________________
clear
mount /dev/flash
echo_in_color $green "Flash drive mounted"
sleep 2
while [ $EXCODE != "flexit" ]
     do
        clear
        echo_in_color $yellow "Enter [flexit] to unmount and exit:"
        read EXCODE
     done
umount /dev/flash
exit
```

I call this from an FVWM menu by **`Exec exec xterm -geometry 43x2+1250+0 -e /home/nikos/bin/flashdrive`** which places it on a tiny x–term in one corner of the screen. The horizontal displacement of 1250 may have to be modified according to the horizontal resolution of your screen.

---

11. Maintenance

11.1. Filesystem inspection

It is a virtuous habit to inspect your ext2 filesystem on the flash memory regularly. To do this, the tool dumpe2fs may be used in the following way: # **`dumpe2fs -h /dev/sda1`** (you must be root and the device should not be mounted). The result should be similar to the following:

```
Filesystem volume name:   <none>
Last mounted on:          <not available>
Filesystem UUID:          c42a6963-5e6a-4cd2-b7d7-c8f09dca6c52
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      dir_index filetype sparse_super
Default mount options:    (none)
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              32000
Block count:              127856
Reserved block count:     6392
Free blocks:              116456
Free inodes:              31922
First block:              1
Block size:               1024
Fragment size:            1024
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         2000
Inode blocks per group:   250
Filesystem created:       Sat Sep 20 12:43:00 2003
Last mount time:          Tue Oct 28 14:13:03 2003
Last write time:          Tue Oct 28 14:28:27 2003
Mount count:              13
Maximum mount count:      35
Last checked:             Sat Oct 18 11:28:26 2003
Check interval:           15552000 (6 months)
Next check after:         Thu Apr 15 11:28:26 2004
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               128
Default directory hash:   tea
Directory Hash Seed:      118bee0a-efa5-4771-967e-41a0badd0355
```

A few important aspects need to be pointed out.

- When the ext2 filesystem is created, it is by default given maximal usage before it has to be checked. These can be seen *Maximum mount count* (35) and *Check interval* (expiry date).
- The usage so far: *Mount count* and *Last checked* .
- The existence of corrupted files (bad blocks): *Filesystem state* .

You might get warnings about these things when you mount the device or when you try the read files from the device.

11.2. And then?

When the usage allocation has been spent, or there is evidence of file corruption, the thing to do is to run # **`fsck.ext2 /dev/sda1`** with the device unmounted. After that, usage parameters will be freshly allocated and bad blocks will be gone.

When dealing with the vfat system, the dump does not seem to exist. The command # **`dumpe2fs -f /dev/sda1`** for filesystems other than ext2 does not work for vfat. The tool dosfsck exists (it is still Alpha), and may be risky to use on a device you have not formatted yourself.

12. The Linux−2.6 kernel

12.1. What is new?

Setting up your system as described above for the Linux−2.4.x kernels will be perfectly good, but with the Linux−2.6.x kernels (x = 0, 1, 2, 3 at the time of this writing) much more is possible:

- USB Hotplugging of devices has been improved immensely. This means that it has become possible to attach and remove a device from the USB port with full cognisance of the system. If a device is attached, it will show in the /proc filesystem. Once it is removed, it will not show anymore.
- The *sysfs* system has been introduced. By using this facility, a very tight grip on attached devices, their attributes and nodes of attachment is possible.
- Utilities for handling hotplugging have been improved or became available. With the aid of these it is possible to easily distinguish between attached devices irrespective of the order in which they were attached.

12.2. Kernel options

Kernel configuration options are very much the same as described in Section 5.2. The configuration menu (# **`make menuconfig`**) has been organized differently and more systematically. Here are the relevant choices:

12.2.1. Bus options

- Support for hot−pluggable devices (CONFIG_HOTPLUG)

12.2.2. Device drivers

- ***SCSI device support***
- SCSI device support (CONFIG_SCSI scsi_mod)
- legacy /proc/scsi support (CONFIG_SCSI_PROC_FS)
- SCSI disk support (CONFIG_BLK_DEV_SD sd_mod)
- ***USB support***
- Support for USB (CONFIG_USB usbcore)
- USB device filesystem (CONFIG_USB_DEVICEFS)
- UHCI HCD support (CONFIG_USB_UHCI_HCD uhci_hcd)
- USB Mass Storage support (CONFIG_USB_STORAGE usb_storage)

12.2.3. File systems

- Second extended (CONFIG_EXT2_FS ext2)
- ***DOS/FAT/NT filesystems***
- DOS FAT fs support (CONFIG_FAT_FS fat)
- MSDOS fs support (CONFIG_MSDOS_FS msdos)
- VFAT (Windows 95) fs support (CONFIG_VFAT_FS vfat)
- ***Pseudo filesystems***
- /proc file system support (CONFIG_PROC_FS)

12.3. Notes

It may be in your interest to consult Migrating to Linux Kernel 2.6, but it is not necessary to follow steps that cannot be followed.

12.4. Hotplugging and Naming

In Section 6.2 the situation of two USB devices plugged in at the same time was discussed. The difficulty is that it becomes hard for the user to distinguish the two devices. Progress recently made in the Linux Hotplug Project (search for hotplug) and implemented in the 2.6−series of kernels, has made it possible to address this problem efficiently. For implementation some utilities are needed:

- sysfs −− compiled by default into Linux−2.6.x
- sysfsutils−0.4.0 or later
- hotplug−2004_01_05 or later
- udev−016 or later

12.4.1. Installation

The sysfs filesystem.

The *sysfs* filesystem is intrinsic to the Linux−2.6 kernel series. It is used by *udev* and *sysfstools* to obtain information about kernel objects (devices, etc.), their attributes and mutual linkages. Activation is according to the following procedure:

```
# cd /
              # mkdir sys
              # mount −t sysfs none /sys
```

If you do **`ls −F /sys`**, a whole directory structure with entries like *block/*, *bus/*, ... will be shown. You were successful. The mounting should now be made permanent by adding the following line to `/etc/fstab`:

```
              none   /sys   sysfs   defaults   0   0
```

Sysfsutils.

This is installed in one of the usual ways without having to make special configuration options. Instructions in the README file are very clear. You should, however, first make sure that the entry `/usr/local/lib` is entered in `/etc/ld.so.conf` so that the `libsysfs.so` libraries will be recognisable by the system. After installation, check that the utilities `/usr/local/bin/lsbus` and `/usr/local/bin/systool` have been installed. Also check that the libraries mentioned above have been properly linked in (**`ldconfig −p | grep libsysfs`**).

Hotplug.

Next install the `hotplug` scripts. The first thing to do is to check that hotplugging will work on your system. Do: **`less /proc/sys/kernel/hotplug`**. The result should be `/sbin/hotplug`. If it is not so, you should add the line `echo "/sbin/hotplug" > /proc/sys/kernel/hotplug` to an appropriate initialization script, e.g. `/etc/rc.d/rc.local`. This command could also be given (as root) before checking that everything works. Installation is extremely simple. Do: `#` **`make install`**. Check afterwards that the script `/sbin/hotplug` exists.

Udev.

This is the last installation. Study the README file and follow the instructions necessary to be able to run **`make USE_KLIBC=true`**. If you have compiled the Linux−2.6.x kernel, this should not be too difficult. A quick check would be to find out if `/sbin/udev` and `/sbin/udevinfo` exist. There are other occurences as well, a very important one being `/etc/udev/udev.rules`. If you have a working setup for memory sticks (as described above) and a memory stick plugged in you could try **`ls /udev`**, you should see the device (sda, sda1, ...). If it is there, unplug the device and list again. It should be gone!

12.4.2. Naming memory sticks

With the *udev* utility in place, it is possible to identify a memory stick on the basis of some built−in attribute. One that works, is the *vendor* identity. To find this for an attached device, do **`systool -vb scsi | grep vendor`**. This should give a list of attached SCSI devices (remember that your memory stick poses as a SCSI device Section 6.2). Let us suppose that you have two memory sticks plugged in at the same time and found (as I did) two vendor Id's, namely "UFD" and "STORAGE". Then add to the file `/etc/udev/udev.rules` the following lines (at the top if you like):

```
## Flash Memory 1
BUS="scsi", SYSFS_vendor="UFD*", NAME="namib%n"

## Flash Memory 2
BUS="scsi", SYSFS_vendor="STORAGE*", NAME="kalahari%n"
```

The vendor−identities may have trailing blank spaces, and therefore the wildcard asterisks have a purpose. Do not ignore them. Remove and replace the memory sticks, then do **`ls /udev`**. Instead of the impersonal `sda`, `sda1`, etc. there should now be `kalahari`, `kalahari1`, `namib` and `namib1`. The devices have been named according to the "vendor" information in them. Thus we can identify the device, no matter where it was mapped. All that remains to be done is to creat appropriate mount points like `/mnt/namib` and `kalahari` and make entries like

```
/udev/namib1     /mnt/namib     vfat,ext2  user,noauto,rw  0 0
/udev/kalahari1  /mnt/kalahari  vfat,ext2  user,noauto,rw  0 0
```

in `/etc/fstab` to enable mounting from other places.

This is great fun ... and remember, there are other deserts on the planet.

13. Formalities

13.1. Copyright Information

If you have any questions, please contact `<nikos@friedrichs.up.ac.za >`

13.2. Disclaimer

No liability for the contents of this documents can be accepted. Use the concepts, examples and other content at your own risk. As this is a new document, there may be errors and inaccuracies, that may of course be damaging to your system. Proceed with caution, and although this is highly unlikely, the author do not take any responsibility for that.

All copyrights are held by their by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

Naming of particular products or brands should not be seen as endorsements.

You are strongly recommended to take a backup of your system before major installation and backups at regular intervals.

13.3. Credits

I wish to express my deep gratitude towards my sons Paul and Philip who in 1995 persuaded me to move to Linux. Philip gave me profound advice and insights which made this project a pleasant learning experience.

Emma Jane Hogbin, in her review, raised a number of important questions which served to improve this document immensely. My sincerest thanks to her.

13.4. Feedback

Users of this document are most welcome to email me with suggestions on the improvement of the presentation and possible additions to the scope of it.

Notes

[1] This document leans heavily on the texts marked with an asterisk