Installing a USB external hard drive backup (Linux)

RWD Nickalls,
dick@nickalls.org
www.nickalls.org

www.nickalls.org/dick/papers/linux/usbdrive.pdf
revision 5b (July 2015)

1 Introduction

The following account derives from my experience installing a USB 2·0 powered 320 GB drive (Iomega Compact portable drive). Since I have only about 50 GB of files to backup on my `/home/` directory I felt this particular device was more than adequate for my needs, especially as I was keen for it to be completely powered via the USB interface, i.e., avoiding the need to carry around a dedicated power transformer as well.

Iomega Compact portable USB 2 harddrive, 320 GB
(uses a two-headed USB cable).

Size: When selecting a portable USB harddrive one of the main features to decide upon (apart from memory capacity) is whether or not you want it to be powered entirely via the USB port. In general USB-powered drives are relatively small physically (laptop size drive), and with a capacity $\leq$ 1TB. Larger capacity harddrives ($\geq$ 2TB) are, not surprisingly, more power hungry and typically require a separate power cable & transformer.

Filesystem: Virtually all portable USB harddrives come pre-formatted using the MS-Windows NTFS[1], and so will require reformatting to one of the Linux-based file systems, for example EXT3 (for relatively old Linux systems) or the newer EXT4 (since approximately 2008). The filesystem you will need to install therefore depends on the filesystem being used on the Linux-box you are intending to backup.

So, what filesystem does my Linux-box actually use? I have an Intel Duo processor SATA system with an EXT3 filesystem, running a Mandriva-2006 32-bit operating system (a 'power-pack' DVD installation). In short, I have an oldish PC using EXT3. Note, however, the EXT3 information is not listed under hardware, but in special system files which I will describe later.

In the event, my portable HDD turned out to be already formatted with NTFS, and so I did need to reformat it with the EXT3 filesystem.

Plan: We will therefore describe how to do the following: (a) determine the filesystem being used by the PC, (b) set up an EXT3 filesystem on the new portable harddrive using the command ⟨`mkfs`⟩, (c) use the Linux ⟨`rsync`⟩ command to backup all the directories and files, and finally (d) write a couple of simple BASH scripts to automate the backup process.

Caution: Its a good idea when venturing into new territory to proceed cautiously in small stages, testing the commands and learning how to use them safely. As a general

[1] New Technology File System (NTFS)—the standard file system used in MS-Windows-NT and subsequent systems. See the entry in Wikipedia for details (http://en.wikipedia.org/wiki/NTFS.html).

principle always read the Linux 'man' pages on any new command you are intending to use, as these often give some useful hints, and sometimes indicate known pitfalls associated with using certain command-line options. I would also strongly recommend buying a copy of the book *Linux in a nutshell*[2], as this gives an excellent breakdown of all the Linux commands and much more, and is an invaluable resource in this situation. Finally, *Wikipedia* is an excellent resource giving quite detailed information on the various filesystems. All the books mentioned are also listed in the References section at the end.

Linux platform: While this account relates specifically to my Mandriva Linux platform, the principles of approach are essentially the same for all platforms. Note however, that while Mandriva has a so-called standard directory system, other platforms may have small variations as regards exactly which directories key files are kept in (eg. `mtab, fstab, partitions`). To find these, simply use either the ⟨`locate`⟩ command or the ⟨`whereis`⟩ command. For example, in my case the command

```
$ locate /mtab
```

returns the line

```
/etc/mtab
```

while the command

```
$ whereis /mtab
```

returns the line

```
mtab: /etc/mtab
```

Preliminaries: Before we set about installing a new filesystem we need to briefly review the process of mounting and unmounting filesystems generally, as this is central to what we shall be doing later. Note that in the following, both input commands and output screen-code (both highlighted in blue) occasionally extend on to the following page.

In the next section we look at how and where filesystem data is held, the various commands which allow us to look at this data, and determine the filesystem being used by the PC.

2 Devices and filesystems

The PC can only transfer files to and from a storage device if its filesystem is 'mounted'. A suitable analogy might be a car—the engine can only transfer power and traction if the gear-box is engaged.

[2] Siever E, Figgins S, Love R and Robbins A (2009). *Linux in a nutshell*, 6th edition (O'Reilly).

Consequently, Linux maintains a record of all currently mounted devices (in the file `/etc/mtab`), and whenever a new device is mounted or unmounted, Linux updates this file accordingly. Linux also holds all the information on how a given device should be mounted—this is in the file `/etc/fstab`. We now look at these two important files.

2.1 File `/etc/fstab` (filesystem table)

This important file is written by the ROOT program `fstab-sync` on startup while Linux is looking for devices, or when Linux discovers a new device has been plugged in or removed (see the man-pages on `fstab-sync` and also `fstab` for details).

To view the `fstab` file we can use the ⟨`less`⟩ command, as follows:

```
$  less  /etc/fstab
```

The default form of the `fstab` file (i.e., with no devices plugged in to the PC) is typically something along the following format, depending on what partitions have been created:

```
# This file is edited by fstab-sync - see 'man fstab-sync' for details
/dev/sda1 / ext3 defaults 1 1
/dev/sda6 /home ext3 defaults 1 2
/dev/hda /mnt/cdrom auto
    umask=0,user,iocharset=iso8859-15,codepage=850,noauto,ro,exec,users
    0 0
/dev/sda10 /opt ext3 defaults 1 2
none /proc proc defaults 0 0
/dev/sda11 /tmp ext3 defaults 1 2
/dev/sda7 /usr ext3 defaults 1 2
/dev/sda8 /usr/local ext3 defaults 1 2
/dev/sda9 /var ext3 defaults 1 2
/dev/sda5 swap swap defaults 0 0
```

Note that each device has a one-line entry (sometimes wrapped in the above display). With the exception of the CDROM device, each of these one-line entries consists of six blocks or classes of information, each separated by at least one space (if a block contains several items, then these items must be comma-separated), as follows:

```
[device name] [mount point] [filesystem] [default params] [n] [n]
```

Thus the third block of each entry is reserved for the filesystem, and so we can see that in our case (above) all the `/dev/sdaXX` devices (except the `swap` partition) are setup for the `ext3` filesystem.

USB-device: If we now plug in a 'regular' USB memory stick and repeat the above command ⟨`$ less /etc/fstab`⟩, we find that Linux has detected the new device and rewritten the file `/etc/fstab`, which now has an extra entry (`/dev/sdb1/...`) added at the end, as follows:

```
# This file is edited by fstab-sync - see 'man fstab-sync' for details
/dev/sda1 / ext3 defaults 1 1
/dev/sda6 /home ext3 defaults 1 2
/dev/hda /mnt/cdrom auto
    umask=0,user,iocharset=iso8859-15,codepage=850,noauto,ro,exec,users
    0 0
```

```
/dev/sda10 /opt ext3 defaults 1 2
none /proc proc defaults 0 0
/dev/sda11 /tmp ext3 defaults 1 2
/dev/sda7 /usr ext3 defaults 1 2
/dev/sda8 /usr/local ext3 defaults 1 2
/dev/sda9 /var ext3 defaults 1 2
/dev/sda5 swap swap defaults 0 0
/dev/sdb1   /mnt/removable   vfat
    pamconsole,exec,noauto,noatime,codepage=850,iocharset=iso8859-15,
    managed 0 0
```

Thus we can see from the last entry above (which is wrapped across three lines) the USB stick has the device name `/dev/sdb1`, mount point and label `/mnt/removable`, and is formatted for the filesystem `vfat`.

Device label: It is important to appreciate that an external/portable device is known to the computer and its filesystem by its 'label'—a sort of name which is allocated to the device when its filesystem is formatted. In the case of the USB stick above, its device-label is 'removable'. For example, the portable harddrive we are going to format later, will have been allocated a device-label when it was formatted by the manufacturers. We can therefore give it a new device-label when we reformat it, if we wish.

2.2 File `/etc/mtab` (mount table)

The file `/etc/mtab` (mount table) holds similar information about all the currently mounted devices. As before we can read the file using the ⟨`less`⟩ command, as follows:

```
$  less /etc/mtab
```

which gives:

```
/dev/sda1 / ext3 rw 0 0
none /proc proc rw 0 0
none /sys sysfs rw 0 0
/dev/sda6 /home ext3 rw 0 0
/dev/sda10 /opt ext3 rw 0 0
/dev/sda11 /tmp ext3 rw 0 0
/dev/sda7 /usr ext3 rw 0 0
/dev/sda8 /usr/local ext3 rw 0 0
/dev/sda9 /var ext3 rw 0 0
none /proc/bus/usb usbfs rw,devmode=0664,devgid=43 0 0
/dev/sdb1 /mnt/removable vfat
    rw,nosuid,nodev,noatime,codepage=850,iocharset=iso8859-15,user=dick
    0 0
```

2.3 File `/proc/partitions`

Some rather more detailed device information is held in the file `/proc/partitions`. As before we can read the file using the ⟨`less`⟩ command, as follows:

```
$  less  /proc/partitions
```

which gives:

```
major minor  #blocks  name

   8     0  244198584 sda
   8     1   20900533 sda1
   8     2          1 sda2
   8     5    9558643 sda5
   8     6  112647748 sda6
   8     7   20265966 sda7
   8     8   20386453 sda8
   8     9   20555136 sda9
   8    10   10241406 sda10
   8    11   10345828 sda11
   8    16    1000944 sdb
   8    17     999813 sdb1
```

2.4 The ⟨df⟩ command

In fact most of the above information, as well as the memory status of all the devices, can also be accessed using the useful linux ⟨df⟩ command (disk filesystem-usage) in conjunction with the options -h (human readable) and -T (filetype), as in the following command:

```
$ df -hT
```

which then shows the following information.

```
Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/sda1     ext3     20G  224M   19G   2% /
/dev/sda6     ext3    106G   59G   48G  56% /home
/dev/sda10    ext3    9.7G  129M  9.1G   2% /opt
/dev/sda11    ext3    9.8G  129M  9.1G   2% /tmp
/dev/sda7     ext3     20G  2.5G   16G  14% /usr
/dev/sda8     ext3     20G  4.5G   14G  25% /usr/local
/dev/sda9     ext3     20G  597M   18G   4% /var
/dev/sdb1     vfat    977M  107M  870M  11% /mnt/removable
[dick@localhost ~]$
```

Alternatively, one can use the ⟨KDF⟩ command (KDiskFree; available on KDE systems), which essentially just implements the command ⟨df -hT⟩ and then displays the data graphically in a separate window (see Figure 1).

2.5 The ⟨tail⟩ command

It can be very instructive to look at the /var/log/messages file (as ROOT) before and after plugging in a new USB device—see the BasicallyTech (2012) blog-note on this. To do this type the following command (as ROOT):

```
#   tail -f /var/log/messages
```

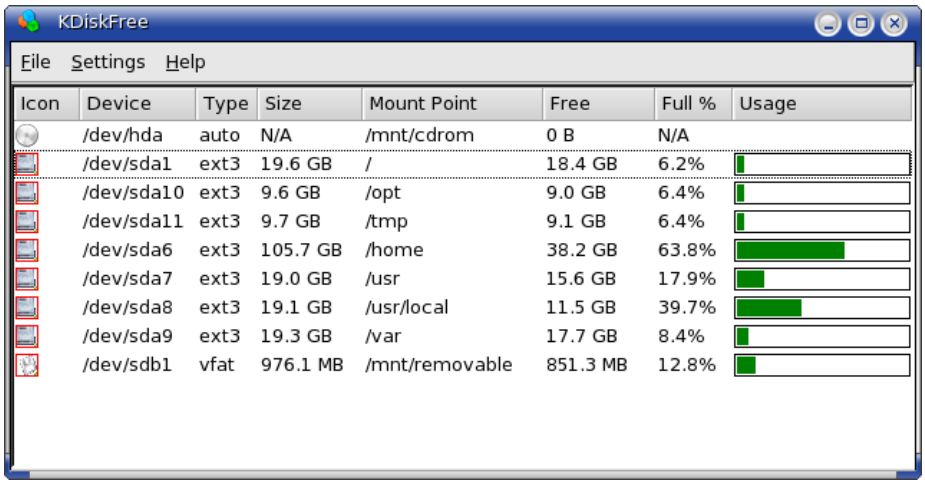

Figure 1: KDF utility showing the details of the USB memory stick (last line).

Note that when the ⟨`-f`⟩ option ('Follow') is used the ⟨`tail`⟩ command shows all the new messages in real-time as they are appended to the file. Use the command ⟨`Ctrl-c`⟩ to terminate the command.

When you view the messages, you will see (a) the PC makes a new mount-point entry in `/etc/fstab`: we can see this happening by using the ⟨`tail`⟩ command to look at the last few entries in the "messages" log at `/var/log/messages` (see below), (b) we can then go and look at `/etc/fstab`.

```
# tail -f /var/log/messages
...
...
Aug 28 13:30:55 localhost kernel: usb 5-8: new high speed USB device
    using ehci_hcd and address 3
Aug 28 13:30:55 localhost kernel: Initializing USB Mass Storage
    driver...
Aug 28 13:30:55 localhost kernel: scsi2 : SCSI emulation for USB Mass
    Storage devices
Aug 28 13:30:55 localhost kernel: usbcore: registered new driver
    usb-storage
Aug 28 13:30:55 localhost kernel: USB Mass Storage support registered.
Aug 28 13:30:55 localhost udev[5826]: run_program: exec of program
    failed
Aug 28 13:30:56 localhost hald[3286]: Timed out waiting for hotplug
    event 1058. Rebasing to 1060
Aug 28 13:31:00 localhost kernel:   Vendor: SanDisk   Model: Cruzer
    Micro      Rev: 0.1
Aug 28 13:31:00 localhost kernel:   Type:   Direct-Access
            ANSI SCSI revision: 02
Aug 28 13:31:00 localhost kernel: SCSI device sdb: 2001888 512-byte
    hdwr sectors (1025 MB)
Aug 28 13:31:00 localhost kernel: sdb: Write Protect is off
Aug 28 13:31:00 localhost kernel: sdb: assuming drive cache: write
    through
```

```
Aug 28 13:31:00 localhost kernel: SCSI device sdb: 2001888 512-byte
    hdwr sectors (1025 MB)
Aug 28 13:31:00 localhost kernel: sdb: Write Protect is off
Aug 28 13:31:00 localhost kernel: sdb: assuming drive cache: write
    through
Aug 28 13:31:00 localhost kernel:  /dev/scsi/host2/bus0/target0/lun0:
    p1
Aug 28 13:31:00 localhost kernel: Attached scsi removable disk sdb at
    scsi2, channel 0, id 0, lun 0
Aug 28 13:31:00 localhost pam_console.dev[5866]: Restoring console
    permissions for /dev/sdb  /dev/discs/disc1/disc /dev/scs
                ost2/bus0/target0/lun0/disc
Aug 28 13:31:00 localhost pam_console.dev[5885]: Restoring console
    permissions for /dev/sdb1  /dev/discs/disc1/part1 /dev/s
                /host2/bus0/target0/lun0/part1
Aug 28 13:31:00 localhost fstab-sync[5893]: added mount point
    /mnt/removable for /dev/sdb1
```

Now when I unmount the device, and then remove it Linux responds with the following two messages.

```
Aug 28 13:34:45 localhost kernel: usb 5-8: USB disconnect, address 3
Aug 28 13:34:45 localhost fstab-sync[5966]: removed mount point
    /mnt/removable for /dev/sdb1
```

3 Mounting & unmounting a filesystem

In practice the mounting process is usually hidden from us, since (a) we mostly tend to work within a GUI environment (typically the X windows system) with file-managers which generally mount new devices automatically and seamlessly. For example, Mandriva Linux uses the 'automount' system by default, so any device is automatically mounted whenever it is first plugged in.

However, when the GUI is not running, we then have to implement mounting using the dedicated Linux ⟨`mount`⟩ command, including the 'label' name of the device, as for example with the following command I might use for my USB memory stick which has the label 'removable':

```
$ mount  -v  /mnt/removable
```

which responds with a lot of useful information (because I have used the `-v` option), all from the `/etc/fstab` file we saw earlier:

```
$ mount  -v  /mnt/removable
/dev/sdb1 on /mnt/removable type vfat
    (rw,nosuid,nodev,noatime,codepage=850,iocharset=iso8859-15,user=dick)
```

In contrast, the unmounting process is usually much more apparent, since we are all familiar with the need to unmount a removable device before unplugging it, in order to avoid possible data loss[3].

If the GUI is already running, then unmounting is typically achieved by clicking on the appropriate icon. When the GUI is not running we then have to use the Linux ⟨`umount`⟩ command. If you are the OWNER of the device then you can run these commands from the USER prompt, otherwise this has to be done as ROOT.

For example, the Linux command to unmount my USB memory stick is

```
$ umount -v /mnt/removable
```

which responds as follows:

```
$ umount -v /mnt/removable
/dev/sdb1 umounted
```

Although the process of unmounting is generally the prelude to removing a device, this is not always the case, since it is sometimes necessary to have a device plugged in but at the same time be unmounted. A useful analogy is that of having the gearbox of a car in neutral, while keeping the engine running.

3.1 Checking the filesystem of the portable harddrive

When I plug in the new portable harddrive and repeat the ⟨`df`⟩ command I now obtain the following:

```
$ df  -hT
Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/sda1     ext3     20G  224M   19G   2% /
/dev/sda6     ext3    106G   59G   48G  56% /home
/dev/sda10    ext3    9.7G  129M  9.1G   2% /opt
/dev/sda11    ext3    9.8G  129M  9.1G   2% /tmp
/dev/sda7     ext3     20G  2.5G   16G  14% /usr
/dev/sda8     ext3     20G  4.5G   14G  25% /usr/local
/dev/sda9     ext3     20G  597M   18G   4% /var
/dev/sdb1     ntfs    299G   74M  299G   1% /mnt/Iomega_HDD
```

The entry for the new device now appears as the last line, showing (a) its device-name is `/dev/sdb1`, (b) it is formatted using NTFS, and (c) its 'label' is *Iomega_HDD*.

4 Setting up an EXT3 filesystem

Having familiarised ourselves with the process of mounting and unmounting filesystems, we are now in a position to address the process of installing an EXT3 filesystem on the new USB portable harddrive device (portable HDD).

[3]This is because, unless you deliberately flush the input and output buffers (say, by giving a ⟨`sync`⟩ command at the command-line), the PC may well not have finished moving all the data, since it has to fit this in with all the other demands on its time etc. When you right-click on a removable-device icon and select the "safely remove' option the PC then sends its own ⟨`sync`⟩ command, and once the buffers have been flushed, it then says you can 'safely remove the device' etc.

4.1 The device-name

The first thing to determine is the 'device-name' your computer uses for the portable HDD device (on a SATA system this is typically something like `/dev/sdXX` where the 's' stands for SATA). A simple way of determining this is to use the Linux ⟨`df -hT`⟩ command. We did this in the previous section, which returned the following line for the portable Iomega harddrive:

```
...
/dev/sdb1      ntfs    299G   74M  299G   1% /mnt/Iomega_HDD
```

4.2 Using the ⟨`mkfs`⟩ command

Linux uses the rather general ⟨`mkfs`⟩ command to construct a file-system on a device (eg, on a harddrive partition), and to give the device a label or name for defining the device. In our case the device-name is `/dev/sdb1`, and we wish to use the EXT3 file-system, and, for convenience, let us allocate it an easy to remember name without any spaces or underscores, for example as "*iomegaHDD*". The command, which needs to be run as ROOT and with the device unmounted, is then

```
# mkfs -v   -t ext3   -L iomegaHDD  /dev/sdb1
```

where `-v` means verbose mode, `-t` indicates the file-system type to be used, and `-L` indicates the label to be used.

In fact, the command ⟨`mkfs`⟩ is actually a front-end utility command which, in this case, just invokes the more specific command ⟨`mkfs.ext3`⟩, so we can instead just use this command directly, as follows (but it is probably slightly better to actually use the initial ⟨`mkfs`⟩ above, as we shall see below):

```
# mkfs.ext3 -v  -L iomegaHDD  /dev/sdb1
```

Note that you can easily find all the other ⟨`mkfs`⟩ commands just by typing the command ⟨`locate /mkfs`⟩ and looking for the entries in the `/sbin/` directory, as follows:

```
$ locate /mkfs
...
/sbin/mkfs.ext3
/sbin/mkfs.bfs
/sbin/mkfs
/sbin/mkfs.cramfs
/sbin/mkfs.minix
/sbin/mkfs.msdos
/sbin/mkfs.vfat
```

So, lets run this command (as ROOT) and see what happens.

```
[root@localhost ~]# mkfs.ext3  -v -L iomegaHDD /dev/sdb1
mke2fs 1.38 (30-Jun-2005)
/dev/sdb1 is mounted; will not make a filesystem here!
[root@localhost ~]#
```

Ahh... Linux is pointing out the important fact that the USB device is actually still mounted, and since we did not use the potentially dangerous `-F` option (forces the ⟨`mkfs.ext3`⟩ command to run even if the device is mounted) Linux did not proceed further[4].

Remember that the device was automatically mounted by Mandriva Linux (this is a useful Mandriva facility) when we plugged it in via the USB cable, as was indicated when we used the ⟨`df -hT`⟩ command earlier. We therefore need to unmount it now (since we can't proceed until its unmounted), using the Linux ⟨`umount`⟩ command[5], as follows:

```
$ umount -vl /dev/sdb1
```

where the `-l` implements the useful so-called 'lazy' form of the command (see the man page). We can now check that the device has been unmounted by using the ⟨`df`⟩ command we used earlier, as follows:

```
$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1              20G  224M   19G   2% /
/dev/sda6             106G   59G   48G  56% /home
/dev/sda10            9.7G  129M  9.1G   2% /opt
/dev/sda11            9.8G  129M  9.1G   2% /tmp
/dev/sda7              20G  2.5G   16G  14% /usr
/dev/sda8              20G  4.5G   14G  25% /usr/local
/dev/sda9              20G  597M   18G   4% /var
```

which confirms that the device has now been unmounted (i.e., the entry for the new USB HDD device has disappeared).

OK, so we can now proceed with the ⟨`mkfs.ext3`⟩ command (as ROOT), as follows (note this generates a lot of information since we are using the `-v` option):

```
[root@localhost]# mkfs.ext3  -v -L iomegaHDD /dev/sdb1
mke2fs 1.38 (30-Jun-2005)
Filesystem label=iomegaHDD
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
39075840 inodes, 78142160 blocks
3907108 blocks (5.00%) reserved for the super user
First data block=0
2385 block groups
32768 blocks per group, 32768 fragments per group
16384 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632,
    2654208,
        4096000, 7962624, 11239424, 20480000, 23887872, 71663616
```

[4]This highlights the important point that it is generally safer to use the more restricted ⟨`mkfs`⟩ command rather than the ⟨`mkfs.ext3`⟩ command which does allow the use of some potentially dangerous options. This also reinforces the importance of reading all the relevant 'man' pages for such commands, and keeping an eye out for any possible hazards.

[5]Note that this command is 'umount' and not 'unmount' as you might have expected.

```
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 37 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@localhost]#
```

This is telling us that the EXT3 file-system has now been setup and so, as a check, we can view the current status by using the ⟨`df`⟩ command again. In order to remount the device at this stage (we don't know just now whether our device has been labelled as we specified or not), it is easiest (with Mandriva Linux) just to disconnect and reconnect the USB cables (since Madriva uses its auto-mount facility), and then run the ⟨`df`⟩ command again (as USER), as follows:

```
[dick@localhost ~]$ df -h
Filesystem     Type    Size  Used Avail Use% Mounted on
/dev/sda1      ext3     20G  224M   19G   2% /
/dev/sda6      ext3    106G   59G   48G  56% /home
/dev/sda10     ext3    9.7G  129M  9.1G   2% /opt
/dev/sda11     ext3    9.8G  129M  9.1G   2% /tmp
/dev/sda7      ext3     20G  2.5G   16G  14% /usr
/dev/sda8      ext3     20G  4.5G   14G  25% /usr/local
/dev/sda9      ext3     20G  597M   18G   4% /var
/dev/sdb1      ext3    294G  129M  279G   1% /mnt/iomegaHDD
[dick@localhost ~]$
```

From this we can see (last line) that our new USB device `/dev/sdb1` is now formatted as EXT3 and is labelled `iomegaHDD` as specified. Note that the fact that this new last line has appeared also implies that Linux has successfully mounted the device—always a good sign!

4.3 Setting the ownership of the device

If we wish to use the new device freely as USER, then we now need to set the user permissions accordingly. Since I am the only user of my PC, I set both the OWNER and GROUP to myself (dick) as follows (needs to be done as ROOT):

```
# chown dick  /mnt/iomegaHDD
# chgrp dick  /mnt/iomegaHDD
```

Checking this, by moving to the `/mnt` directory and using the ⟨`dir -l`⟩ command, we then find that we now have 'dick' as both OWNER and GROUP for the iomegaHDD device, as follows:

```
[dick@localhost ~]$ cd /mnt
[dick@localhost mnt]$ dir -l
total 12
drwxr-xr-x   2 root root 4096 Nov 30  2009 cdrom
drwxr-xr-x   2 root root 4096 Nov 30  2009 floppy
drwxrwxr-x  39 dick dick 4096 May 20 17:09 iomegaHDD
[dick@localhost mnt]$
```

Since we now have a working EXT3 filesystem on the portable harddrive, we can turn our attention to the process of maintaining a backup copy of the files and directory structure on it, and writing a simple BASH file to coordinate this.

5 Writing a data backup program

Of course one can always just copy files or directories to or from the new portable drive using your regular file-manager (the KDE *Konqueror* system in my case). However, this is both complicated and somewhat unreliable if you are wanting a regular 'backup' of all your files.

In order to allow flexibility, we will therefore describe writing two small BASH scripts: the first to backup a single directory, and the second to coordinate the backing-up of a list of directories, by invoking the first script.

Since one may well wish to backup different lists of directories on different occasions, one may therefore create different files with different lists etc. For example you may wish to backup directory-list A at the end of each day, and backup a more comprehensive set of directories (say directory-list B) at the end of each month.

Having a good BASH programming book is of course extremely useful, and as you will see from the programs below I have found the book by Johnson (2009) to be an excellent guide for just this sort of thing (see References at the end).

However, since its important to be familiar with the extremely important Linux command ⟨`rsync`⟩, we address this first.

5.1 The ⟨`rsync`⟩ command

The Linux ⟨`rsync`⟩ command is used for synchronising (mirroring) the file content of directories in different locations, either within the same computer or across a network. It has the virtue of being particularly fast, since it only transfers the 'blocks' of data which are different between the local and remote locations. Consequently, this command is particularly useful for maintaining a remote backup of your harddrive on, say, a portable USB harddrive. The format of the ⟨`rsync`⟩ command is as follows:

```
$ rsync  [options]  localdir   remotedir
```

There are lots of possible command-line options, so do read the man page. The `-n` option enables everything *except* actually transfer data, and is therefore useful for testing purposes. The `-b` option forces a single backup copy of files (rather than just overwriting). The `--delete` option allows for the removal of remote files no longer present in the local directories.

For example, in order to update the files and sub-directories of, say, directory *A* to its backup/mirror location, then a typical command which preserves user permissions, creates backup copies of files, and giving statistics of the data moved, would be something like the following:

```
$ rsync -avb --stats  /localPATH/directoryA/    /remotePATH/directoryA
```

5.1.1 Word of caution

When using the ⟨rsync⟩ command some care needs to be taken as to whether a trailing backslash is used with the local directory (see the example command shown above), since this influences how the *remote* directory is structured. This rather confusing aspect is addressed in the following extract from the rsync man page.

> A trailing slash on the source changes this behaviour to avoid creating an additional directory level at the destination. You can think of a trailing / on a source as meaning "copy the contents of this directory" as opposed to "copy the directory by name", but in both cases the attributes of the containing directory are transferred to the containing directory on the destination. In other words, each of the following commands copies the files in the same way, including their setting of the attributes of /foo:
>
> rsync -av /src/foo /dest
>
> rsync -av /src/foo/ /dest/foo

The following example shows the screen output following a command to update the files and subdirectories associated with my local PC ~/a1test directory to the new portable *iomegaHDD* harddrive device. Since this directory did not initially exist on the *remote* device (portable HDD), ⟨rsync⟩ first creates the /mnt/iomegaHDD/a1test directory, then makes a list of files to copy, and finally sets about transferring the files.

```
[dick@localhost ~]$ rsync -avb --stats ~/a1test/  /mnt/iomegaHDD/a1test
building file list ... done
created directory /mnt/iomegaHDD/a1test
./
backup-dat/
backup-dat/cubic.dat
backup-dat/cubic1993.dat
backup-dat/data-figB.dat
...
...
backup-dat/testing.dat
backup-dat/texdata.dat
backup-dat/texfiledata.dat
backup-dat/timefile.dat

Number of files: 142
Number of files transferred: 140
Total file size: 2511761 bytes
Total transferred file size: 2511761 bytes
Literal data: 2511761 bytes
Matched data: 0 bytes
File list size: 2553
File list generation time: 0.001 seconds
File list transfer time: 0.000 seconds
Total bytes sent: 2521050
Total bytes received: 3112

sent 2521050 bytes  received 3112 bytes  5048324.00 bytes/sec
total size is 2511761  speedup is 1.00
[dick@localhost ~]$
```

5.1.2 Buffer activity

Note that once screen activity stops and the screen-prompt is returned, data will usually continue to be transferred for a while, as the PC's output buffer is gradually emptied (the lights on both the PC and the portable harddrive will be flashing). One must be careful, therefore, not to disconnect the portable device until you have confirmation that all the data has been copied over (see the ⟨`umount`⟩ command below). Consequently, when updating a large filesystem all at once, it is probably a good idea to do it in stages, performing a ⟨`sync`⟩ command (not to be confused with the ⟨`rsync`⟩ command) after backing-up one set of directories and before starting the next—all of which is easily arranged by using a dedicated script (BASH file) to coordinate the backup, as described below.

The ⟨`sync`⟩ command simply forces the PC to flush the buffer there and then, rather than waiting (as it may otherwise do) and doing it later at its own convenience.

5.1.3 Directory permissions

For the ⟨`rsync`⟩ command to work the directory permissions must be appropriate; ie., permission is required for the directories to be accessed etc.

For example, the ⟨`rsync`⟩ command will fail to copy the contents of a directory having permissions set as `rw-r--r--`. This problem can easily be fixed, however, using the following command (as USER).

```
$  chmod  a+x  <full dir path>
```

(where: a(ALL), +(ADD), x(EXECUTE)) which will generate the following permission set `rwxr-xr-x`, and hence allow ⟨`rsync`⟩ to work OK. For an excellent section on Linux permissions see the book by Smith (2003).

5.2 BASH program to run the backup

In order to facilitate automating the backups we are actually going to describe making two small BASH programs as follows: (a) the first (program 1), backs-up a particular directory-name given in the command-line, and (b) the second (program 2), which essentially contains a list of directories, then calls the first program to back-up each of the specified directories in the list. Both the programs given here are very simple and have worked well for several years on my PC.

Since the ⟨`rsync`⟩ command requires you to specify both the *local* and *remote* directory names, the big advantage of writing a dedicated program (or script) to perform the backup, apart from the obvious convenience, is that it avoids the hazard of making a typo while writing the remote directory-name at the command-line. Such an error would then result in the data being inadvertently deposited into a strange remote directory (one with the typo name), i.e., you may well not find the data again. Making a typo with a local directory name, however, is less problematic since one is unlikely to have a directory having the typo name, and so Linux will abort the command, and generate a *directory does not exist...* error message.

5.2.1 Program 1: to back-up a single directory

This program accepts a directory name as input, checks that the specified local directory actually exists, and if it does, it then copies the directory and contents to the portable

harddrive. Finally it calls the command ⟨sync⟩ to flush the buffers before terminating.

```
## backupdir2HDD.sh
##
## backups up a single dir

echo " ====================="
echo " running script dnbackupdir2HDD.sh"
echo " backing up files to iomega_HDD ..."

space="   "
printf " the first entered param =  $1 \n"
##dir=$1
localdir="/home/dick/$1"

## first check localdir is a valid directory
## see page 22 in ProBash Programming
if [ -d $localdir ]
then
    ## it is true
    echo " check:  $localdir  exists OK"
    echo " scanning dir:  $localdir ..."
else
     ## it is false
     echo " ERROR: dir $localdir does NOT exist"
     echo " ...quiting prog now ...."
     echo " ====================="
     ## quit prog using a valid exit code 1
     exit 1
fi

## now define the source and destination dirs
## include trailing / on sourcedir to force inclusion of  subdirs also
sourcedir="$localdir/"
destdir="/mnt/iomegaHDD/$1"
printf "command:  rsync -avb --stats  $sourcedir $space $destdir\n"
rsync -avb --stats  $space  $sourcedir $space $destdir

echo " waiting for sync..."
sync
echo " ...................done"
echo " ====================="
```

To use this program, all we have to do is to specify one of the main directories in the USER's 'home' directory. For example, if I have a directory structure as follows:

```
/home/dick/dir1
/home/dick/dir2
....
....
```

then as I am USER 'dick', all I need to do is type the command

```
$ bash backupdir2HDD.sh  dir1
```

which results in the directory `/dir1` being copied to the portable harddrive as

`/mnt/iomegaHDD/dir1`

If this directory does not currently exist on the portable harddrive, then it will automatically be created.

The 'delete' option: Note that the program as written simply backs-up all files to the portable HDD. We have written it this way simply because if you decide to delete a file from the PC, the back-up program will not then automatically also remove it from the back-up HDD. This is a safety idea, since you may wish to access the file again at some stage from the backup.

However, if periodically you wish to clean out from your backup all those files which no longer exist on your PC (you are sure you don't want them any more), then you can include the `--delete` option in the ⟨`rsync`⟩ command (see line 35), as follows:

```
rsync -avb --stats --delete  $space  $sourcedir $space $destdir
```

Now, each time you run the program, it will (in addition to backing up all new files) delete all files which do not currently exist on the PC. For safety, therefore, I keep the 'delete' option in a separate program just so I can keep my deletions nicely controlled.

5.2.2 Program 2: list of directories to backup

The following program makes it easy to back-up a number of specified directories, and calls the above program to do the backing up. Once we are happy that the above program works reliably, we can then simply call the following program to implement the backup. The specific list of directories for backing-up can easily be changed as necessary.

```
1  ## backup2HDD.sh
2  ##
3  echo " backing up files to portable HDD:"
4
5  ##====check whether portable HDD is mounted==========
6
7  mtabentry=$( grep   /iomegaHDD  /etc/mtab )
8  printf " /mtab entry is: $mtabentry\n"
9  ## test whether returned string length is zero
10 if  [ -z  "$mtabentry" ]
11 then
12      ## length of string is zero
13      echo " ERROR: HDD is NOT mounted (no entry in /mtab)"
14      echo " ...quiting prog now ...."
15      echo " ======================"
16      ## quit prog using a valid exit code 1
17      exit 1
18
19 else
20     ## length of string is NOT zero
21     ## grep has returned a result
22     echo " portable HDD is mounted OK"
23 fi
24
25 ##====list of directories to backup=======
```

```
26
27 bash backupdir2HDD.sh  dir1
28 echo " =======next section================"
29 bash backupdir2HDD.sh  dir2
30 echo " =======next section================"
31 ...
32 ...
33 echo "...all done"
```

The above program first checks to see whether the portable HDD is mounted—if not, it aborts and issues an error message. If it is mounted, it then proceeds to backup all the listed directories by calling the first program (passing the directory name to it) each time to actually do the backup. It is easy now to edit this program and adjust the list of directories to suit your need, or even just place the directory list in a separate file and call it into the program, or pass the filename as a command-line option.

Mount status: The initial mount status is checked by using the ⟨`grep`⟩ command to look for an appropriate entry in the `/mtab` file. For example, if the HDD is already mounted, then the following ⟨`grep`⟩ command finds a string which includes the label of the HDD, and hence it returns an error code of zero (note that the command's return code is held in the BASH variable `$?`).

```
[dick@localhost ~]$ grep iomegaHDD  /etc/mtab; echo $?
/dev/sdb1 /mnt/iomegaHDD ext3 rw,nosuid,nodev,user=dick 0 0
0
[dick@localhost ~]$
```

Conversely, if the HDD is not currently mounted, then the above ⟨`grep`⟩ command will fail to find a string in the `/mtab` file which includes the label of the HDD, and hence it will return an error code of 1.

```
[dick@localhost ~]$ grep iomegaHDD  /etc/mtab; echo $?
1
[dick@localhost ~]$
```

6 Maintenance

Once the portable hard-drive is 'up-and-running' it is important to periodically check the filesystem integrity; i.e., weeding out old files (files which have already been deleted from the PC) and checking for filesystem errors (using the ⟨`fsck`⟩ tools).

6.1 Program for deleting old files

Its a good idea to use a separate program (for safety reasons) for deleting from the portable backup hard-drive those 'old' files which have already been deleted from the PC (and hence no longer need to be backed-up). The following program is essentially the same as 'program 1' (described in Section 5.2.1), except for a change in line 33 to include the ⟨`--delete`⟩ option for the ⟨`rsync`⟩ command. This program deletes all files from the destination directory which do not exist in the associated PC directory.

So, for example, if we wish to cleanup the directory `~/dirA`, then we use the command

```
$ bash dnbackupdir2HDDdelete.sh  dirA
```

```
1  ## backupdir2HDDdelete.sh
2  echo " running script dnbackupdir2HDDdelete.sh"
3  echo " backing up files to iomega_HDD ..."
4
5  space="   "
6  printf " input directory name =  $1 \n"
7  ##dir=$1
8  localdir="/home/dick/$1"
9
10 ## first check localdir is a valid directory
11 ## see page 22 in ProBash Programming
12 if [ -d $localdir ]
13 then
14     ## it is true
15     echo " check:  $localdir  exists OK"
16     echo " scanning dir:  $localdir ..."
17 else
18      ## it is false
19      echo " ERROR: dir $localdir does NOT exist"
20      echo " ...quiting prog now ...."
21      echo " ====================="
22      ## quit prog using a valid exit code 1
23      exit 1
24 fi
25
26 ## now define the source and destination dirs
27 ## include trailing / on sourcedir to force inclusion of  subdirs also
28 ## use --stats  if want extra statistics with rsync command
29 sourcedir="$localdir/"
30 destdir="/mnt/iomegaHDD/$1"
31 ### WARNING -- this DELETES all remote files NOT in the local dir
32 printf "command:  rsync -avb --delete  $sourcedir $space $destdir\n"
33 rsync -avb --delete  $space  $sourcedir $space $destdir
34
35 echo " waiting for sync..."
36 sync
37 echo " ...................done"
38 echo " OK to unmount portable Iomega_HDD drive"
39 echo " ====================="
```

6.2 Filesystem check

Its also a good idea to periodically run the ⟨`fsck`⟩ program in order to keep the back-up filesystem in good order. Since the possibility of filesystem errors or problems increases with the number of read/write events, Linux issues a warning message (see the file `/var/log/messages`) once a filesystem has been mounted more than a certain number of times, and suggests you run one of the filesystem checking tools.

For example, looking at the ⟨`dmesg`⟩ output (device messages) from time-to-time using the command

```
$ dmesg  | tail  --lines=50
```

after plugging in the portable hard-drive device may reveal something like the following [6] (see lines 13–14):

```
1  usb-storage: device found at 4
2  usb-storage: waiting for device to settle before scanning
3    Vendor: TOSHIBA   Model: MK3259GSX         Rev:
4    Type:   Direct-Access                      ANSI SCSI revision: 02
5  SCSI device sdb: 625142448 512-byte hdwr sectors (320073 MB)
6  sdb: assuming drive cache: write through
7  SCSI device sdb: 625142448 512-byte hdwr sectors (320073 MB)
8  sdb: assuming drive cache: write through
9   /dev/scsi/host3/bus0/target0/lun0: p1
10 Attached scsi disk sdb at scsi3, channel 0, id 0, lun 0
11 usb-storage: device scan complete
12 kjournald starting.  Commit interval 5 seconds
13 EXT3-fs warning: maximal mount count reached, running e2fsck is
       recommended
14 EXT3 FS on sdb1, internal journal
15 EXT3-fs: mounted filesystem with ordered data mode.
```

Notice the warning in line 13

```
EXT3-fs warning: ......., running e2fsck is recommended
```

which suggests that we should run the filesystem-check program ⟨`e2fsck`⟩ on our portable hard-drive (filesystem name = `/dev/sdb1`). Since the ⟨`e2fsck`⟩ command is actually CALLed by the more familiar ⟨`fsck`⟩ command, it is sufficient to just run ⟨`fsck`⟩ itself.

Before proceeding its a good idea just to check the filesystem name for the portable USB hard-drive (`iomegaHDD`)—and while it is still mounted—using the linux ⟨`df`⟩ command as follows:

```
$ df
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1              20G  241M   19G   2% /
/dev/sda6             106G   88G   19G  83% /home
/dev/sda10            9.7G  129M  9.1G   2% /opt
/dev/sda11            9.8G  129M  9.1G   2% /tmp
/dev/sda7              20G  2.4G   16G  13% /usr
/dev/sda8              20G   11G  7.4G  60% /usr/local
/dev/sda9              20G  1.1G   18G   6% /var
/dev/sdb1             294G   76G  203G  28% /mnt/iomegaHDD
```

and the last line shows that the `iomegaHDD` filesystem name is indeed `/dev/sdb1`.

So after first *unmounting* the USB filesystem [7] using the ⟨`umount ...`⟩ command

[6] Note that one obtains essentially the same information by reading the last 50 lines or so of the 'messages' log (as ROOT), using the command `# tail -f --lines=50 /var/log/messages`

[7] You will see from the 'man' page of the ⟨`fsck`⟩ command that one should never run it on a mounted filesystem unless you really know what you are doing.

```
$ umount -v  /dev/sdb1
/dev/sdb1 umounted
```

and then (as ROOT) run the fsck command (using the `-v` (verbose) option adds lots of useful extra information at the end):

```
#  fsck  -v  /dev/sdb1
```

the screen output will show something like the following:

```
[root@localhost backup]# fsck  -v  /dev/sdb1
fsck 1.38 (30-Jun-2005)
e2fsck 1.38 (30-Jun-2005)
iomegaHDD has been mounted 40 times without being checked, check
    forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information

  334974 inodes used (0%)
    4367 non-contiguous inodes (1.3%)
         # of inodes with ind/dind/tind blocks: 55119/1511/0
20629266 blocks used (26%)
       0 bad blocks
       0 large files

  317030 regular files
   17631 directories
       0 character device files
       0 block device files
       0 fifos
       0 links
     304 symbolic links (304 fast symbolic links)
       0 sockets
--------
  334965 files
[root@localhost backup]#
```

Finally we need to exit back to USER mode

```
[root@localhost backup]# exit
exit
[dick@localhost backup]$
```

The total time for ⟨`fsck`⟩ to process my 320 GB drive was approximately 10 minutes. Also, fortunately, there were no warning or error messages, so all is well!

A useful check is to now mount the portable USB hard-drive (use the ⟨`mount ...`⟩ command) and then look at the last few ⟨`dmesg`⟩ lines (new), as follows:

```
$ mount -v  /dev/sdb1
/dev/sdb1 on /mnt/iomegaHDD type ext3 (rw,nosuid,nodev,user=dick)
$
$ dmesg  | tail  --lines=50
```

we will find that the "`EXT3-fs warning:`" no longer appears, so all is well.

```
...
kjournald starting.  Commit interval 5 seconds
EXT3 FS on sdb1, internal journal
EXT3-fs: mounted filesystem with ordered data mode.
```

7 References

- Johnson CFA (2009). *Pro Bash Programming: scripting the GNU/Linux shell.* (Apress) [www.apress.com]
- Shotts WE (2009). *The Linux Command Line*
  [an excellent free book on the basics of Linux, 522 pages (PDF version downloadable from: http://www.linuxcommand.org/tlcl.php)]
- Siever E, Figgins S, Love R and Robbins A (2009). *Linux in a nutshell*, 6th edition (O'Reilly).
- Smith RW (2003). *Linux power tools* (Sybex, London).
  [see pp. 102–105 for permissions]
- Basically Tech (2012). *Using a USB external hard disk for backups with Linux*. Downloaded March 2012: http://www.basicallytech.com/blog/index.php?/archives/73-Using-a-USB-external-hard-disk-for-backups-with-Linux.html