# Working with Filesystems

## The Filesystem Hierarchy

Slackware Linux stores all of its files and directories under a single / directory, typically referred to as *"root"*. This is in stark contract to what you may be familiar with in the form of Microsoft Windows. Different hard disk partitions, cdroms, usb flash drives, and even floppy disks can all be mounted in directories under `/`, but do not have anything like *"drive letters"*. The contents of these devices can be found almost anywhere, but there are some sane defaults that Slackware sets up for you. For example, cd-rw drives are most often found at `/mnt/cd-rw`. Here are a few common directories present on nearly all Slackware Linux installations, and what you can expect to find there.

**Table 11.1. Filesystem Layout**

| | |
|---|---|
| / | The root directory, under which all others exist |
| /bin | Minimal set of binary programs for all users |
| /boot | The kernel, initrd, and other requirements for booting Slackware |
| /etc | System configuration files |
| /dev | Collection of special files allowing direct access to hardware |
| /home | User directories where personal files and settings are stored |
| /media | Directory for auto-mounting features in DBUS/HAL |
| /mnt | Places to temporarily mount removable media |
| /opt | Directory where some (typicaly proprietary) software may be installed |
| /proc | Kernel exported filesystem for process information |
| /root | The root user's home directory |
| /sbin | Minimal set of system or superuser binaries |
| /srv | Site-specific data such as web pages served by this system |
| /sys | Special kernel implimentation details |
| /tmp | Directory reserved for temporary files for all users |
| /usr | All non-essential programs, libraries, and shared files |
| /var | Regularly changing data such as log files |

## Local Filesystem Types

The Linux kernel supports a wide variety of filesystems, which allows you to choose from a long list of features to tailor to your particular need. Fortunately, most of the default filesystem types are adequate for any needs you may have. Some filesystems are geared towards particular media. For example, the iso9660 filesystem is used almost exclusively for CD and DVD media.

### ext2

ext2 is the oldest filesystem included in Slackware Linux for storing data on hard disks. Compared to other filesystems, ext2 is simplistic. It is faster than most others for reading and writing data, but does not include any journaling capability. This means that after a hard crash, the filesystem must be exhaustively checked to discover and (hopefully) fix any errors.

### ext3

ext3 is the younger cousin of ext2. It was designed to replace ext2 in most situations and shares much the same code-base, but adds journaling support. In fact, ext3 and ext2 are so much alike that it is possible to convert one to the other on the fly without lose of data. ext3 enjoys a lot of popularity for these reasons. There are many tools available for recovering data from this filesystem in the event of catastrophic hardware failure as well. ext3 is a good general purpose filesystem with journaling support, but fails to perform as well as other journaling filesystems in specific cases. One pitfall to ext3 is that the filesystem must still go through this exhaustive check every so often. This is done when the filesystem is mounted, usually when the computer is booted, and causes an annoying delay.

### ext4

ext4 is the latest in the ext series of filesystems. It was designed to build upon ext3 with new ideas on what filesystems should do. While Slackware supports ext4, you should remember that this filesystem is still very new (particularly in file system terms) and is under heavy development. If you require stability over performance, you may wish to use a different filesystem such as ext3. With that said, ext4 does boast some major improvements over ext3 in the performance arena, but many people don't yet trust it for stable use.

### reiserfs

reiserfs is one of the oldest journaling filesystems for the Linux kernel and has been supported by Slackware for many years. It is a very fast filesystem particularly well suited for storing, retrieving, and writing lots of small files. Unfortunately there are few tools for recovering data should you experience a drive failure, and reiserfs partitions experience corruption more often than ext3.

### XFS

XFS was contributed to the Linux kernel by SGI and is one of the best filesystems for working with large volumes and large files. XFS uses more RAM than other filesystems, but if you need to work with large files its performance there is well worth the penalty in memory usage. XFS is not particularly ill-suited for desktop or laptop use, but really shines on a server that handles medium to large size files all day long. Like ext3, XFS is a fully journaled filesystem.

### JFS

JFS was contributed to the Linux kernel by IBM and is well known for its responsiveness even under extreme conditions. It can span colossal volumes making it particularly well-suited for Network Attached Storage (NAS) devices. JFS's long history and thorough testing make it one of the most reliable journaling filesystems available for Linux.

### iso9660

iso9660 is a filesystem specifically designed for optical media such as CDs and DVDs. Since optical disks are read-only media, the linux kernel does not even include write support for this filesystem. In order to create an iso9660 filesystem, you must use user-land tools like ***mkisofs***(8) or ***growisofs***(8).

### vfat

Sometimes you may need to share data between Windows and Linux computers, but can't transfer the files over a network. Instead you require a shared hard drive partition or a USB flash drive. The humble vfat filesystem is the best choice here since it is supported by the largest variety of operating systems. Unfortuantely, being a Microsoft designed filesystem, it does not store permissions in the same way as traditional Linux filesystems. This means that special options must be used to allow multiple users to access data on this filesystem.

### swap

Unlike other filesystems which hold files and directories, swap partitions hold virtual memory. This is very useful as it prevents the system from crashing should all your RAM be consumed. Instead, the kernel copies portions of the RAM into swap and frees them up for other applications to use. Think of it as adding virtual memory to your computer, very slow virtual memory. swap is typically a fail-safe and shouldn't be relied upon for continual use. Add more RAM to your system if you find yourself using lots of swap.

## Using mount

Now that we've learned what (some of) the different filesystems available in Linux are, it's time we looked at how to use them. In order to read or write data on a filesystem, that filesystem must first be mounted. To do this, we (naturally) use mount(8). The first thing we must do is decide where we want the filesystem located. Recall that there are no such things are drive letters denoting filesystems in Linux. Instead, all filesystems are mounted on directories. The base filesystem on which you install Slackware is always located at `/` and others are always located in subdirectories of `/`. `/mnt/hd` is a common place to temporarily locate a partition, so we'll use that in our first example. In order to mount a filesystem's contents, we must tell mount what kind of filesystem we have, where to mount it, and any special options to use.

```
darkstar:~# mount -t ext3 /dev/hda3 /mnt/hd -o ro
```

Let's disect this. We have an ext3 filesystem located on the third partition of the first IDE device, and we've decided to mount its contents on the directory /mnt/hd. Additionally, we have mounted it read-only so no changes can be made to these contents. The [-t ext3] argument tells mount what type of filesystem we are using, in this case it is ext3. This lets the kernel know which driver to use. Often mount can determine this for itself, but it never hurts to explicitly declare it. Second, we tell mount where to locate the filesystem's contents. Here we've chosen `/mnt/hd`. Finally, we must decide what options to use if any. These are declared with the [-o] argument. A short-list of the most common options follows.

**Table 11.2. Common mount options**

| | |
|---|---|
| ro | read-only |
| rw | read-write (default) |
| uid | user to own the contents of the filesystem |
| gid | group to own the contents of the filesystem |
| noexec | prevent execution of any files on the filesystem |
| defaults | sane defaults for most filesystems |

If this is your first Linux installation, the only options you typically need to be concerned about are *ro* and *rw*. The exception to this rule comes when you are dealing with filesystems that don't handle traditional Linux permissions such as vfat or NTFS. In those cases you'll need to use the *uid* or *gid* options to allow non-root users access to these filesystems.

```
darkstar:~# mount -t vfat /dev/hda4 /mnt/hd -o uid=alan
```

But Alan, that's appalling! I don't want to have to tell mount what filesystem or options to use everytime I load a CD. It should be easier than that. Well thankfully, it is. The `/etc/fstab` file contains all this information for filesystems that the installer sets up for you, and you can make additions to it as well. `fstab`(5) looks like a simple table containing the device to mount along with its filesystem type and optional arguments. Let's take a look.

```
darkstar:~# cat /etc/fstab
/dev/hda1        /                reiserfs    defaults               1   1
/dev/hda2        /home            reiserfs    defaults               1   2
/dev/hda3        swap             swap        defaults               0   0
/dev/cdrom       /mnt/cdrom       auto        noauto,owner,ro,users  0   0
/dev/fd0         /mnt/floppy      auto        noauto,owner           0   0
devpts           /dev/pts         devpts      gid=5,mode=620         0   0
proc             /proc            proc        defaults               0   0
```

If you have an entry in `fstab` for your filesystem, you need only tell mount the device node or the mount location.

```
darkstar:~# mount /dev/cdrom
darkstar:~# mount /home
```

One final use for ***mount*** is to tell you what filesystems are currently mounted and with what options. Simply run ***mount*** without any arguments to display these.

# Network Filesystems

In addition to local filesystems, Slackware supports a number of network filesystems as both client and server. This allows you to share data between multiple computers transparently. We'll discuss the two most common: NFS and SMB.

## NFS

NFS is the Network File System for Linux as well as several other common operating systems. It has modest performance but supports the full range of permissions for Slackware. In order to use NFS as either a client or a server, you must run the remote procedure call daemon. This is easily accomplished by setting the `/etc/rc.d/rc.rpc` file executable and telling it to start. Once it has been set executable, it will run automatically every time you boot into Slackware.

```
darkstar:~# chmod +x /etc/rc.d/rc.rpc
darkstar:~# /etc/rc.d/rc.rpc start
```

Mounting an NFS share is little different than mounting a local filesystem. Rather than specifying a local device, you must tell mount the domain name or IP address of the NFS server and the directory to mount with a colon between them.

```
darkstar:~# mount -t nfs darkstar.example.com:/home /home
```

Running an NFS server is a little bit different. First, you must configure each directory to be exported in the `/etc/exports` file. `exports`(5) contains information about what directories will be shared, who they will be shared with, and what special permissions to grant or deny.

```
# See exports(5) for a description.
# This file contains a list of all directories exported to other computers.
# It is used by rpc.nfsd and rpc.mountd.

/home/backup    192.168.1.0/24(sync,rw,no_root_squash)
```

The first column in `exports` is a list of the files to be exported via NFS. The second column is a list of what systems may access the export along with special permissions. You can specify hosts via domain name, IP address, or netblock address (as I have here). Special permissions are always a parenthetical list. For a complete list, you'll need to read the man page. For now, the only special option that matters is *no_root_squash*. Usually the root user on an NFS client cannot read or write an exported share. Instead, the root user is *“squashed”* and forced to act as the nobody user. *no_root_squash* prevents this.

You'll also need to run the NFS daemon. Starting and stopping NFS server support is done with the `/etc/rc.d/rc.nfsd` rc script. Set it executable and run it just like we did for `rc.rpc` and you are ready to go.

## SMB

SMB is the Windows network file-sharing protocol. Connecting to SMB shares (commonly called samba shares) is fairly straight forward. Unfortuantely, SMB isn't as strongly supported as NFS. Still, it offers higher performance and connectivity with Windows computers. For these reasons, SMB is the most common network file-sharing protocol deployed on local networks. Exporting SMB shares from Slackware is done through the samba daemon and configured in `smb.conf`(5). Unfortunately configuring samba as a service is beyond the scope of this book. Check online for additional documentation, and as always refer to the man page.

Thankfully mounting an SMB share is easy and works almost exactly like mounting an NFS share. You must tell mount where to find the server and what share you wish to access in exactly the same way. Additionally, you must specify a username and password.

```
darkstar:~# mount -t cifs //darkstar/home /home -o
username=alan,password=secret
```

You may be wondering why the filesystem type is cifs instead of smbfs. In older versions of the Linux kernel, smbfs was used. This has been deprecated in favor of the better performing and more secure general purpose cifs driver.

All SMB shares require the *username* and *password* arguments. This can create a security problem if you wish to place your samba share in fstab. You may avoid this problem by using the *credentials* argument. *credentials* points to a file which contains the username and password information. As long as this file is safely guarded and readable only by root, the likelyhood that your authentication credentials will be compromised is lessened.

```
darkstar:~# echo "username=alan" > /etc/creds-home
darkstar:~# echo "password=secret" >> /etc/creds-home
darkstar:~# mount -t cifs //darkstar/home -o credentials=/etc/creds-home
```

# Chapter Navigation

**Previous Chapter: Filesystem Permissions**

**Next Chapter: vi**

# Sources

- Original source: http://www.slackbook.org/beta

- Originally written by Alan Hicks, Chris Lumens, David Cantrell, Logan Johnson

slackbook, filesystem, network filesystems, nfs, smb, mount