Filesystems: Section 1 Transcript

Introduction 1/3

Welcome to the Filesystems module. Based on your interactions with a UNIX system at the command line, you are used to navigating the directory hierarchy within UNIX; however, there is much more to the UNIX filesystem. UNIX manages its storage, as well as many device interactions, through a robust abstraction layer which uses several core concepts, such as open, read and write to implement data storage.

In this module, we investigate UNIX filesystems from the ground up, starting with the technical details of storage devices and how UNIX uses filesystems to aggregate and organize a variety of storage media into a unified directory tree. We also discuss some of the standards used to organize the UNIX directory hierarchy, and explain common command line tools to work with and explore the filesystem.

Throughout this module, you'll be presented with opportunities to assess and apply what you've learned.

At the end of this module, you will be able to:

- Describe how files and directories are stored and referenced on a partition, and
- Explain the general format of a filesystem,
- Explain the concepts of blocks and slack space,
- Describe the UNIX logical directory structure in relation to a storage medium,
- Summarize the relationship between devices, partitions, and volumes,
- Perform the steps to mount a partition,
- Identify common root-level directories, subdirectories, and pseudo-directories in UNIX,
- Describe objects on a UNIX filesystem, and
- Determine command usage and syntax for commands related to filesystems.

Prerequisites 2/3

As you are traversing through this module, you are also expected to use Internet search engines and UNIX `man` pages to discover and learn about a variety of UNIX commands. Some of the commands that you will need to research and become proficient with are with are: `dd`, `diskinfo`, `df`, `du`, `fdisk`, `file`, `fsck`, `fuser`, `link`, `ln`, `mkfs`, `mount`, `quota`, `stat`, `strings`, `umount`, and `unlink`.

Before continuing on, take the time to research the UNIX commands displayed - especially any that may be unfamiliar to you. You may print a copy of the commands found in Resources for this module and/or record data you learn during your research.

Bypass Exam Introduction 3/3

If you are already familiar with the subject matter presented in this module, you can choose to take a

Bypass Exam to skip this module.

The Bypass Exam option provides a single opportunity to successfully demonstrate your competence with the material presented within the module. If you pass, you'll receive credit for completing the module, unlocking the content within, and you will be free to proceed to the next module. If you do not pass, you will need to successfully complete the module, including all exercises and the Module Exam, to receive credit.

Click the Next Section button to continue.

Filesystems: Section 2 Transcript

Storage Devices

1/6

Storage devices, such as a disk or thumb drive, are physical objects that a computer can write data to and/or read data from; usually this storage medium is non-volatile, that is, data written to it will persist after power outages or reboots. Like all UNIX Input/Output peripherals, storage disks are treated as **devices** by the Operating System. Typical names include: `/dev/hda` or `/dev/sdb` for hard drives or `/dev/cdrom` for optical drives.

As a physical device, storage media stores data as streams of bits. In order to be usable to a computer, these streams of bits need a **logical structure**. There are two methods used to create a logical structure, and they are often used in combination. Let's take a few moments to examine each method.

Logical Structure: Partitioning

2/6

The first method is **partitioning**. Partitioning, most frequently associated with fixed disks, is a technology that allows a single physical storage medium to be split into several sections known as **partitions**, each being interpreted by both the Basic Input and Output System (BIOS) and the Operating System as a separate disk. There are several approaches used for partitioning, but the most commonly used technique is the **Master Boot Record** (MBR) standard.

The MBR uses the first 512 bytes of the storage device. In addition to containing code that can boot the computer, the MBR also contains a 64 byte **partition table**. This partition table defines up to four partitions of the storage device; however, other partition schemes can create additional partitions. Not all storage space on a disk needs to be within a partition; any leftover space is called **unpartitioned**, or unallocated, **space**. This is important because it is possible, though not necessarily easy, for an Operating System to read and/or write to or from an unallocated space on a storage medium, which means an adversary or hacker has the potential to hide data in this space.

Partitions

3/6

From the perspective of the OS, each partition is treated like its own storage device. For example, if /dev/sda is the storage device name, its partitions will typically be `/dev/sda1`, `/dev/sda2`, etc. Despite the numeric labeling, the partitions are treated as individual storage devices. Even when partitioned, the individual partitions are still just streams of bits that contain none of the files and directories we are used to noting. In order to structure the partitions to be usable, they need to be formatted with a filesystem.

Logical Structure: Formatting

4/6

The second method for creating a logical structure is formatting. When formatting a partition on a hard disk, you are choosing which filesystem to use within a partition. A filesystem is a standard for organizing information on a storage device. It consists of on-disk formatting and a kernel module, or code loaded into the kernel image, that knows how to interpret a structure as organized files and directories. Any storage device (disk or partition) that has a single filesystem is known as a **volume**.

When UNIX mounts storage into its directory structure, it is mounting a volume, which needs information about both the device and the filesystem to add the storage to the system. In other words, UNIX needs both elements - the device and filesystem - to successfully mount a volume. When it comes to filesystems, you will have to search for various items on a filesystem to understand how it stores and protects files as that is how you are going to find data or malware.

With this in mind, let's examine some filesystem concepts in more detail.

Filesystem Concepts

5/6

Click each tab to learn about the filesystem concepts displayed. When you have finished reviewing each concept, click the Next Slide button to continue with the module.

Blocks: The atomic unit of storage in a filesystem is known as a block. On a disk, every file consumes at least one block of storage. Within a filesystem every block has the same size. Historically the block size for UNIX had to be 512 bytes; however, modern systems commonly have block sizes that consist of 1024, 2048, or even 8192 bytes. Click another tab to continue learning about filesystem concepts. If you have finished reviewing filesystem concepts, click the Next Slide button to continue with the module.

Slack Space: When a file written to a disk is larger than the block size, the filesystem allocates enough blocks to store the contents of the file. However, there is no guarantee that these blocks will be contiguous on a disk. File contents stored in non-contiguous blocks are called **fragmented**, and fragmentation can significantly impact performance.

Since storage within a block is contiguous, one approach to speeding up disk access is to increase the block size. However, this can potentially decrease available storage space due to slack space. Slack space is the unused portion of a block that is used to store contents that do not require a full block. For example, in a filesystem with a block size of 1024 bytes, a file that is 3160 bytes will be allocated across four blocks, resulting in three full blocks and a fourth with only 88 bytes, leaving 936 bytes of slack space in the last block. Storing the same file in a filesystem with a block size of 8192 bytes requires only one block, but that block will contain 5032 bytes of slack space. Note that a clever attacker can use slack space to hide files. Click another tab to continue learning about filesystem concepts. If you have finished reviewing filesystem concepts, click the Next Slide button to continue with the module.

Superblocks: All information about a filesystem is stored within the filesystem itself, namely in the superblock. On a bootable partition, the superblock follows the boot block, while on a storage partition, the superblock is the first block of the partition. In both cases, it contains critical parameters, including the block size. A filesystem is completely unusable with a corrupt superblock, so the filesystem often creates copies of the superblock at known locations on the disk. Click another tab to continue learning about filesystem concepts. If you have finished reviewing filesystem concepts, click the Next Slide button to continue with the module.

Inode Table: An important filesystem structure is the Inode Table. Every object stored in the filesystem has an associated inode, a data structure that contains the metadata for the object, as well as pointers to the data blocks containing the contents of the object. Interestingly, the inode does not contain a name for the object. Instead, it is indexed with a number, which is often called the inode. Click another tab to continue learning about filesystem concepts. If you have finished reviewing filesystem concepts, click the Next Slide button to continue with the module.

File: A file is simply a sequence of data stored in a filesystem. The inode for the file contains all of the file metadata, including a flag indicating that the associated data is a file. The inode also contains pointers to the physical location of the blocks on the disk. For example, pointers act like addresses to the blocks on the medium containing the file contents, such as the stored data. Click another tab to continue learning about filesystem concepts. If you have finished reviewing filesystem concepts, click the Next Slide button to continue with the module.

Directory: A directory also has an inode with metadata, but includes a different flag indicating the inode is associated with a directory object. The pointers in the inode point to one or more data blocks, but in this case the data blocks contain a table which associates names of objects contained in the directory with the corresponding inodes. Notice that the inodes referenced could correspond to files or directories; this lets directories contain other directories, and creates a hierarchy. Click another tab to continue learning about filesystem concepts. If you have finished reviewing filesystem concepts, click the Next Slide button to continue with the module.

Hard Links: A single inode may be associated with multiple names in multiple directories. These associated references are called hard links. None of these referenced names is primary; however, the inode keeps track of the number of names that are hard linked to file contents. A directory object cannot have multiple hard links. Click another tab to continue learning about filesystem concepts. If you have finished reviewing filesystem concepts, click the Next Slide button to continue with the module.

Exercise Introduction

It is time for an Exercise. In order to successfully complete the Exercise, you are expected to use your notes and any available resources presented throughout the course, in addition to conducting your own Internet research.

Filesystems: Section 3 Transcript

Volumes and Mapping: Windows

1/5

UNIX and Windows treat their storage media differently. In Windows, volumes are typically either mapped to drive letters, such as `C:`, or network shares. In both cases, all storage locations are relative to the volume. In a Windows system, each logical partition comprises its own directory tree. Each volume is assigned a separate drive letter to specify the volume with which a user wishes to interact. Each subsequent volume, device, or network location is mapped to a unique drive letter by the OS. For example, there is a space on the hard disk (physical or logical volume) mapped to drive letter `D:` and a disk drive mapped to drive letter `E:`.

Volumes and Mapping: UNIX Basics

2/5

In a UNIX system, all volumes, devices, or network locations also have their own directory tree. However, in UNIX, each volume is ultimately mounted, or mapped, onto a directory forming a single directory structure, which is rooted at "/." This provides one single unified tree. When users access this single tree, the system transparently accesses the correct underlying storage media.

When a UNIX system boots up, it selects a volume (configured when the operating system was installed) to mount as "/." This volume will typically have a full directory hierarchy for a UNIX system. However, some of these directories may simply be mount points. In other words, directories whose contents are actually stored on other volumes.

For a variety of reasons, certain subdirectories such as `/boot`, `/var/log`, and `/home`, often fall into this category. In order to access these other volumes, they must be mounted into the directory tree at the mount point. Once the volume is mounted to the mount point, the mount point directly becomes identified with the root directory of the mounted volume.

Volumes and Mounting: Typical vs. Atypical Directories

3/5

Directories meant to be mount points are typically left empty on the volume mounted as "/." However, occasionally a volume may be mounted over a directory that already has its own files or subdirectories. When this atypical situation occurs, the files and subdirectories on the original volume are temporarily masked, or unavailable, until the new volume is unmounted.

Volumes and Mounting: Removable Media

4/5

Removable media are another large class of volumes that need to be mounted to allow filesystem access, and are often mounted to mount points such as `/media`, `/mnt/cdrom`, or `/mnt/usb`. It is important to note again that a mount point must exist (as a directory) within the "/" hierarchy before a volume can be mounted to it. Unlike the Windows approach, which maps different file systems and devices to individual drive letters (for example, drive `E:`), UNIX treats the removable media as just another part of the single directory tree.

Each approach has advantages and disadvantages. The advantage of the UNIX approach is that the storage volumes are abstracted away from user processes; the user process picks a location in the hierarchy to read or write; it does not care or need to know whether it is on a hard disk, a removable disk, or a network drive halfway across the world. However, this is also the disadvantage as a UNIX process may accidentally congest an entire network with what it thinks is a simple file copy!

Exercise Introduction

5/5

It is time for an Exercise. In order to successfully complete the Exercise, you are expected to use your notes and any available resources presented throughout the course, in addition to conducting your own Internet research.

Filesystems: Section 4 Transcript

UNIX Directory Hierarchy: Standard Directories 1/8

All UNIX systems maintain a single directory structure, rooted at the "/" directory. In order to set up a filing system, an administrator could create an arbitrary set of files and subdirectories under "/," but the operating system and its supporting files are already stored in this structure creating a level of predictability. Each UNIX variant, such as Solaris, has its own way of doing things; however, there are some standard files, or directories, that are typically found in most UNIX systems. Displayed are some of the most common ones and their functions.

UNIX Directory Hierarchy: Standard Subdirectories 2/8

There are also some standard subdirectories that are typically found in most UNIX systems. Take a few moments to review some of the most common ones that are useful to know.

Special Locations in the Directory Hierarchy 3/8

The designers of UNIX realized that the filesystem provided an excellent generic interface between userspace and the kernel. They decided to multi-purpose it in order to provide access to system information as well as storage resources.

The `/dev` and `/proc` directories are two pseudo-filesystems that are mounted into the directory hierarchy that provides system information rather than storage. The advantage of this approach is that the same tools used to interact with files, such as `cat` and `more`, can be used to access system information as well.

Let's take a few moments to examine the pseudo-filesystems `/dev` and `/proc` in more detail.

The /dev Directory 4/8

The `/dev` directory is where UNIX allows direct access to input/output devices. This is not true raw access since the kernel's device driver is still mediating the access; however, with storage media, this direct access does circumvent filesystems. The `/dev` directory also contains some useful pseudo-devices, such as `/dev/random` and `/dev/urandom`, which produce streams of random bytes. `/dev/null`, another pseudo-device, discards all data written to it, which is useful for suppressing the output of a command. Let's look at some of the devices in the `/dev directory`.

Using `ls -al | more`, notice that cdrom, cdrw, dvd and dvdrw all reference the optical disk, and all are symlinks to the sr0 device. Additionally, note that `/dev/mem` can access parts of memory directly, while the `/dev/ram` lets you create "disks" in memory. Furthermore, `/dev/sda` is typically a SCSI hard drive, which you can read and write to directly, bypassing (and possibly destroying) a filesystem. So, be careful!

Also, note that `/dev/tty` is a system terminal. This allows direct access to read from and write to other terminals. For example, the `tty` command prints which device your terminal is using as displayed. Here's another terminal. Let's send a message to the other guy. Note that when we type: `echo BOO! > /dev/pts/1` the typed text displays on the other terminal window.

Now that you have some basic knowledge about the `/dev`, let's explore the pseudo-filesystem `/proc`.

The /proc Directory

5/8

The `/proc` directory is where UNIX allows access to information about running processes. The subdirectories and objects within `/proc` are presented like normal files. However, instead of corresponding to storage, the OS kernel uses the filesystem structure as a convenient interface to store information about the processes that are running. In addition, it allows the user to view and change system parameters. Let's examine the `/proc` directory.

Entering `sleep 600 &` produces a process that will run for a while; the ampersand allows us get the terminal back. The return 10698 is the PID of the sleep process. We can use the corresponding `/proc` directory to find information about the process.

If we run `cat cmdline`, it shows you the command that was just run. If we run an `ls -al exe`, notice the link to the executable. Whereas running `ls -al cwd`, shows the working directory of the process; and running `ls -al fd` shows the files the process has opened, including STDIN, STDOUT, and STDERR.

Links

6/8

Links are a method UNIX provides to allow access to a single resource using multiple names within the directory hierarchy. This is often useful for several reasons, but is most often associated with programs hard-coding pathnames.

Earlier in the module we discussed hard links, which are multiple names that reference the same inode, however, hard links have some disadvantages. As noted previously, one cannot have a hard link to a directory, and hard links cannot be applied between different filesystems. To remedy these deficiencies, UNIX also allows the use of **symlinks**, or **soft links**.

A soft link is its own type of file object and has its own inode, and it generally places a pointer in the filesystem that tells any program referencing it that it should open another object instead. There are some exceptions to this, such as with some system utilities such as `rm`, which actually work on the links themselves.

Searching a Filesystem

7/8

The `find` command searches a defined portion of the directory hierarchy in a filesystem, looking for objects that meet user-defined criteria, and performing user-defined actions on them. Generically, the syntax looks like: `find <paths> <options> <tests> <actions>`. Click each tab to continue learning about syntax used to search a directory hierarchy. When you have finished reviewing each concept, click the Next Slide button to continue with the module.

paths: The paths are locations in the directory hierarchy under which find will look. If a path is a directory, by default find will search the entire directory tree under that directory, so "/" will search the entire hierarchy. Note that more than one path can be given with a `find` command.

options: In UNIX, options modify how a search is conducted. Typically, `-maxdepth` is a useful option and tells find how many directories down in the hierarchy to search.

tests: UNIX tests are logical assertions that an examined object must pass to be acted on. By default multiple tests are connected with a logical "and" (which can be made explicit with `-and` or `-a` syntax). Other corresponding logical expressions can be created using the `-or` and `-not` syntax.

actions: Last, but not least, action tells find what to do when it discovers matches. The default action is to print (explicitly `-print`). Other actions are also possible. As an example, consider the displayed syntax: `find /var -name \*.log -exec wc -l \{\} \;`

What does this command do? The first argument after `find` is a path, so this suggests that find is going to look in the `/var` directory. Next, we know that options should be listed. Is `-name` an option? How can you find this out? Hint: think `man` pages.

Quick research indicates that our example expression doesn't have any options, which means `find` will search down the entire directory structure under `/var`. The `-name` is a test that looks for filenames (not path) that match `\*.log` or actually `*.log`, since the slash is there to make the command shell-safe.

So, is `-exec` another test, or is it an expression? Isn't `find` fun?

Again we consult our references and discover `-exec` is an action. On every file we find, we will run the command `wc -l` with our filename as an argument. Again consulting our reference, we find that `wc -l` counts the number of lines in a file.

The braces, which may need to be escaped with backslashes or quoted to be shell-safe, expand to the current filename, or set of filenames, being processed.

When `-exec` is used, the arguments following the command are considered arguments to the command until an argument of ";" is encountered. Once again, the ";" may need to be escaped with a backslash to be shell-safe. Another option would be to use "+" instead of ";" for commands that take a list of files, such as `ls` and `wc`. When "+" is specified, the command is executed once for the set of files rather than once per file.

For example, if there were 3 files under `/var` ending in `.log`, the displayed command (`find /var -name \*.log -exec wc -l \{\} \;`) would result in executing 3 commands:

- wc -l file1
- wc -l file2
- wc -l file3

Note that the same find command using the "+" syntax (`find /var -name *.log -exec wc

`-l \{\} \+`) would result in executing just one command:

- wc -l file1 file2 file3

So to summarize, the `find` command looks for all files in the `/var` directory and all of its subdirectories that end in `.log`, and it will count the number of lines in each such file.

Exercise Introduction

It is time for an Exercise. In order to successfully complete the Exercise, you are expected to use your notes and any available resources presented throughout the course, in addition to conducting your own Internet research.

Filesystems: Section 5 Transcript

Summary

You have completed the Filesystems module.

You should now be able to:

- Describe how files and directories are stored and referenced on a partition,
- Explain the general format of a filesystem on a partition,
- Explain the concepts of blocks and slack space,
- Describe the UNIX logical directory structure in relation to a storage medium,
- Summarize the relationship between devices, partitions, and volumes,
- Perform the steps to mount a partition,
- Identify common root-level directories, subdirectories, and pseudo-directories in UNIX,
- Describe objects on a UNIX filesystem, and
- Determine command usage and syntax for commands related to filesystems.

To receive credit and advance to the next module, you must achieve a passing score on the Module Exam. Click the Next Section button to begin the Module Exam.