COMP09024 Unix System Administration

Lecture 2: Files, Filesystems and File Management

Duncan Thomson/Hector Marco

UWS

Trimester 1 2020/21

Outline

2.1 File and Filesystem Structure

What is a file?

- A file comprises two types of data:
  - An *inode*, containing information about the file
  - Blocks containing information *in* the file
- But in Unix, 'everything is a file'
  - So directories ('folders') are files (whose contents are relative filenames with references to inodes)
  - Links to other files are files (whose contents are the path to the other file)
  - (most) Devices are (special) files which can be read or written
- File data lives in a filesystem: could be on a hard disk, a USB key, a CDROM, over the network...

What's stored in an inode?

- An inode number (a unique integer on each filesystem)
- Type of file (file, directory, device, symbolic link. . .)
- Size of file in bytes
- Pointers to blocks where data is stored (may be direct, indirect, double indirect, or triple indirect, depending on file size)
- ID of user and group who own the file
- Timestamps of last access, modification and change time
- File permissions (or mode)
- Link count (how many hard links exist to this file)

Notice what's **not** stored in an inode: file name!

The Virtual Filesystem (VFS)

- The overall Unix filesystem is hierarchical, made of multiple filesystems grafted together
- The first filesystem accessed is known as the *root* filesystem — it becomes the structure onto which other filesystems can be attached
- The process of making a physical filesystem available (whether the root FS or a later one) is known as *mounting*
- Entries — usually directories — are used as *mount points* for subsequent filesystems attached after the root system
- The overall hierarchy resulting from a series of such mounts is known as the *virtual filesystem* (VFS)

Filesystem Hierarchy Standard (FHS)

- Unix standard specifying purpose of directories
- `/bin` — essential binaries
- `/sbin` — essential system binaries
- `/lib` — libraries required for either of above
- `/boot` – boot loader files and kernel images
- `/etc` — configuration files (often in subdirectories)
- `/home` — users' home directories
- `/root` — `root` user's home directory
- `/usr` — user utilities and applications (including `bin`, `sbin`, `lib`, `include`, `share`...subdirectories)
- `/var` — variable files (eg log files, cached data, etc)
- `/media` — removable media mount points
- `/mnt` — general purpose mount points
- `/tmp` — temporary files

2.2 File Management

The `cd` and `pwd` Commands

- `cd` changes the current working directory
- `pwd` shows the current directory
- Can take relative or absolute path names
- Relative names can include several elements, and may use `..` for upward relative movement (to parent directory)
- With no parameter changes to user's home directory
- For example, if user `alice` is currently in the directory `/home/bob/reports`, all of the following commands would change the working directory to `/home/alice`:

```
cd /home/alice
cd ../../alice
cd
```

The `ls` Command

- Lists directory entries (sorted by name, in columns, by default)
- `-l` gives long listing (owners, timestamp, permissions...):

```
-rwxr-xr-- 1 alice admin 32768 Feb 01 11:13 eg.t
```

- `-a` includes filenames starting with a dot (usually omitted)
- `-S` sort by file size
- `-t` sort by time accessed
- `-u` and `-c` use access time or inode change time instead of modification time for sorting by or displaying timestamps
- `-i` includes inode number
- `-r` reverses sort order
- `-m` print a comma-separated list instead of columns

Copying Files with `cp`

- `cp` makes a copy of a file
- First parameter is source, last parameter is destination:

```
cp myfile.txt newfile
```

- Can also copy multiple files into a directory:

```
cp file1 file2 file3 directory
cp * /home/alice/subdir          # * matches all
```

- By default overwrites destination file(s) — prevent this with `-n` (noclobber) or `-i` (interactive)
- `-a` preserves all information possible ('archive')
- `--preserve` lists what information to preserve
- `-r` performs recursive copy (copy directory contents too)

Moving & Renaming Files with `mv`

- `mv` moves a file to another location or name
- Source file(s) copied to destination (last parameter)
- If destination is a filename, the file is renamed and moved; if destination is a directory, the file is moved:

```
mv myfile.txt myfilenew.txt
mv *.docx wordfiles
```

- `-i` and `-n` can prevent overwriting destination (as with `cp`)
- If a directory is specified as a source, the whole directory and its contents is moved (no need for `-r` flag for recursion)

Deleting Files with `rm`

- Files can be removed with `rm`
- List of files specified all removed if permissions allow:
- Used non-recursively, `rm` will not remove a directory
- `-r` recursive removal (remove directories and contents)
- `-i` asks for confirmation about every file (interactive)
- `-f` forces removal, even if would ask otherwise
- The most dangerous command: `rm -rf *`
- Remember: Unix assumes you know what you're doing...what if you were `root` in the `/` directory?

Managing Directories with `mkdir` and `rmdir`

- `mkdir` creates one (or more) empty directory
  - `-m` allows setting of permissions (mode)
  - `-p` creates any missing parent directories required

```
mkdir reports
mkdir -m 700 secrets
```

- `rmdir` removes an (empty) directory
- Remember that a directory may contain 'hidden' files — you can see these with `ls -a`

Links: Hard Links and Symbolic Links

- Hard links are directory entries pointing to the same inode
  - Must be on same physical filesystem
  - Create with `ln` command: `ln file linkname`
  - File remains until last link is removed
  - Number of hard links to a file is shown in `ls -l` output
- Symbolic links are directory entries containing a VFS path
  - May span physical filesystems
  - Created with `ln -s`
  - If original file is removed, link is left 'dangling'
  - Indicated by `l` in `ls -l` listings

```
$ ln file hard ;   ln -s file soft ;    ls -l
-rw-r--r-- 2 bob bob 0 Jan 10 12:00 file
-rw-r--r-- 2 bob bob 0 Jan 10 12:00 hard
lrwxrwxrwx 1 bob bob 4 Jan 10 12:01 soft -> file
```

2.3 Unix File Permissions

File Permissions

- Three main permission types exist for Unix files:
- Read (`r`, octal 4)
  - For files, allow reading of file contents
  - For directories, allow list names of directory contents
- Write (`w`, octal 2)
  - For files, allow appending or writing to file
  - For directories, allow creation, renaming and removal of files in directory
- Execute (`x`, octal 1)
  - For files, allow execution as a program (or script)
  - For directories, allow access to file and inode contents in a directory (provided name is known)

User Classes and File Owners

- Each file in Unix is owned by a user and a group
- These can be seen using the `-l` option to `ls` (or the `-o` or `-g` options)
- When a user attempts to access a file, they are considered to belong to one of three classes:
  - The *user* who owns the file
  - The *group* who owns the file
  - All *other* users
- The user and group who own a file can be changed using:
  - `chown` to change user and/or group
  - `chgrp` to change group only

Permission Notation — Symbolic

- Nine characters are used to indicate the mode (permissions) of a file by `ls -l`
- For each of three classes of user (user, group, other), there are three characters:
  - `r` if readable by that class, `-` if not
  - `w` if writable by that class, `-` if not
  - `x` if executable by that class, `-` if not
- For example, `rwxr-x---` means:
  - The owner can read, write and execute the file
  - The group owners can read and execute the file
  - Other users have no permissions

Permission Notation — Octal

- Permissions can also be expressed as a 3-digit octal (base-8) number
- Each digit represents the permissions for each of three classes of user: user, group, other
- Each digit is then made up of the permissions which that class of user has added together:
  - 4 for read permission
  - 2 for write permission
  - 1 for execute permission
- For example, 750 would be the same permissions as in the previous slide:
  - The owner can read, write and execute the file
  - The group owners can read and execute the file
  - Other users have no permissions

The `chmod` Command

- `chmod` changes the permissions on a file
- The first parameter is the permissions
- Remaining parameters are file names
- The permissions can be octal (eg 750, 644) which will set the file permissions to that overall value
- Permissions can also be symbolic, with each group of changes separated by commas, and including:
  - User class (`u`, `g`, `o` or `a` (all))
  - Addition (+), removal (−) or setting (=) of permissions
  - Permissions (`r`, `w`, or `x`)
- For example: `chmod u+w,g-x,o= file2` adds write permissions for the owner, removes execute permission for the group, and sets other permissions to be none

Examples of `chmod`

- To set the permissions to 750 on `file1`:

```
chmod 750 file1
```

- To remove read permission for others on `file2`:

```
chmod o-r file1
```

- To add execute permission for all on `file3` and `file4`:

```
chmod a+x file3 file4
```

- To set user permissions to `r-x` and remove write permission for group and other users on `file5` a:

```
chmod u=rx,go-w file5
```

Setting Default Permissions with `umask`

- Files and directories are first created with default permissions
- These are controlled by the value of the *umask*
- `umask` command shows the current umask value (eg 0022)
- The umask value is an octal number showing which permissions are **not** given by default
- Files will have permissions of 666, and directories of 777, masked by umask value (masked by in a bitwise sense)
- Example: If umask value is 027, files will have permissions of 640, and directories of 750, by default
- `umask` command can be used to change the umask value
- `umask -s` shows mask symbolically (as the permissions which remain)

Additional Permissions

- There are in fact 12 mode bits in total in Unix
- The set UID bit (SUID) allows executable files to be run with the user permissions of their owners (shown by an `s` instead of an `x` for the user permissions)
- The set GID bit (SGID) allows executable files to be run with the group permissions of their group owners; when applied to a directory, new files created will have the group owner of the directory (shown by an `s` instead of an `x` for the group permissions)
- The sticky bit (T), on a directory, prevents users from moving or deleting other users' files in writable directories (shown by a `t` instead of an `x` for the other permissions)
- `chmod` can be used to add and remove these permissions (symbolically `s` or `t`)

Extended Permissions Systems

- Some Unix systems include extended permissions (or security) systems
- eg Linux include access control lists (ACLs), which:
  - Allow fine-grained per-user and per-group permissions
  - These can be examined with `getfacl` and changed with `setfacl`
- Many Unix variants implement a similar ACL system (from a draft POSIX standard)
- eg SELinux adds Mandatory Access Control (MAC) to Linux, and includes the idea of security contexts to all files
  - These can be examined with `-Z` option to `ls`

2.4 Filesystem Management

Mounting Filesystems with `mount`

- The `mount` command is used to mount filesystems into the VFS
- It takes (usually) two parameters with a type flag (and options)
  - Filesystem type specified with `-t` flag
  - Device (or other source) where the filesystem is stored
  - Mount point — directory in the VFS where the FS is to be attached
- Example: `mount -t ext4 /dev/sda3 /home`
- A wide range of options can be used with `mount`, including:
  - `-w` mount read/write (default)
  - `-r` mount readonly
  - `-o remount` remount (eg with different options)

(Some) Filesystem Types

- `ufs` — Unix File System, used by most Unices
- `ext2` — Second extended filesystem (Linux 0.99)
- `ext3` — Third extended filesystem (Linux 2.4.15)
- `ext4` — Second extended filesystem (Linux 2.6.28)
- `jfs` — Journaled File System (from IBM)
- `xfs` — XFS, developed by Silicon Graphics
- `zfs` — ZFS, developed by Sun/Oracle
- `iso9660` — used on CDROMs
- `fat` — File Allocation Table, original DOS filesystem
- `vfat` — Virtual FAT, DOS/Windows filesystem (USB disks)
- `ntfs` — New Techology File System, MS Windows NT
- `nfs` — Network File System
- `cifs` — Common Internet File System

Device Files or Filesystem 'Sources'

- Device files are usually stored in the `/dev` directory
- Hard disks (and their partitions) appears as *block special* files (with a `b` in the first column of `ls -l` output)
- Different Unices have different naming conventions
- Modern Linux systems usually name hard disks `sd`*X*, where *X* is `a` for first disk, `b` for second, etc
- Partitions on disks are numbered: `sda1`, `sda2`, etc
- Other physical devices will have files in `/dev`
- Networked filesystems specify server name and 'shared' or 'exported' name:
  - NFS example (on server `mozart`): `mozart:/home`
  - CIFS example (on server `satie`): `//satie/home`

The `umount` Command

- To unattach a physical filesystem from VFS, use the `umount` command (note the spelling!)
- Requires either the filesystem source or the mount point as a parameter
- When a filesystem is in use, the `umount` command will not work
- Other commands which can be useful for removable media:
  - The `sync` command ensures all pending writes to filesystems have been written out
  - The `eject` command can eject CDROM drives and similar

- It would be tedious if we had to use a lot of `mount` commands every time a system booted up
- To avoid this, the `/etc/fstab` file lists details for filesystems which can be mounted automatically at boot
- A `mount -a` command mounts these on bootup
- The format is space-separated fields:
  - Filesystem source (device, server/name, label, UUID)
  - Filesystem mount point (`none` for swap)
  - Filesystem type (or `swap` for swap space)
  - Options (comma-separated)
  - Number indicating whether `dump` should backup
  - Number representing order of filesystem check at boot
- The `/etc/mtab` file lists mounted filesystems (and `mount` on its own lists these too)

Checking Filesystems with `fsck`

- Most filesystems should be checked for integrity on a regular basis
- Some types of filesystems (eg journaling filesystems) require this less than others — but all need it
- The `fsck` (file system check) utility can be run to do this
- Normally during bootup after a fixed number of mounts or amount of time
- May be run to automatically fix problems, or manually, where the operator is asked before any changes
- `fsck` is often a front end to various filesystem-type specific utilities (eg `e2fsck`, `dosfsck`, etc
- `fsck` should only be run on *unmounted* filesystems, giving the device file as a parameter: `fsck /dev/sda3`

Backing up and Restoring Filesystems

- There are many ways of backing up filesystems, including:
  - Using RAID to provide mirrored or redundant disks
  - Coping elsewhere using tools such as `cpio`, `tar`, `dd`, `rsync`
  - Using cloud-based backup services
- However the 'traditional' Unix method is backing up to removable media such as tape or CDROM, using `dump`
- `dump` operates at the filesystem level
- Should be used ideally on unmounted, readonly, or at least quiescent filesystems
- Supports ten backup levels, allowing a range of backup strategies including incremental and differential
- The `restore` utility is used to retrieve data

2.5 Reading and Writing Files

Creating Files

- Easiest way to create an empty file: `touch` *`filename`* (more generally, `touch` updates timestamps on files)
- Creating a file with contents: use `cat >`*`filename`* and then Ctrl-D (EndOfFile) to finish:

```
cat >myfile
This is the file contents
and we end it by pressing Control-D
EOF
```

- Or edit file using a text editor
- Many GUI systemshave text editors (eg `gedit` in Gnome)
- But the ubiquitous Unix text editor is `vi`
- Others exist: `pico`, `emacs`...

Basics of `vi`

- *Modeful* — in *command mode* each key is a command
- Movement commands (can precede with a number):
  - Arrow keys (or `h j k l`)
  - `^` for start of line, `$` for end of line
  - `w` goes forward a word; `b` goes back a word
  - `G` goes to a line number (or end of file)
- Commands to enter *insert mode*, to enter text into the file:
  - `a` to append after cursor; `i` to insert before it
  - `c` with movement to change; (`cc` to change a whole line)
  - `o` to open a new line
  - Exit insert mode with ESC key
- `d` with movement to delete (or `dd` for a whole line)
- Enter `:w` to save file; `:q` to quit; or `:wq` to save and quit

Reading File Contents

- `cat` command (short for conCATenate) can string together a number of files and display them
- `more` (and `less`) show files one page at a time:
  - SPACE to move forward a page
  - `/` to search
  - `q` to quit
- To identify what a file is, can use the `file` command

```
file /etc/passwd
more /etc/passwd
```

Summary

- Files, directories and inodes
- The Virtual Filesystem (mounting / unmounting)
- FHS: standard locations for things
- File management: `cd`, `pwd`, `ls`, `cp`, `mv`, `rm`, `mkdir`, `rmdir`
- Links: hard and symbolic; `ln` and `ln -s`
- Permissions, user classes, octal and symbolic permissions
- `chmod`, `chown`, `chgrp`, `umask`
- Filesystem types and devices
- `mount`, `umount` and `/etc/fstab`
- Filesystem maintenance: `fsck`, `dump` and `restore`
- Creating files (
  t touch, `cat`, `vi` and other editors)
- Reading files (`cat`, `more`, `less`, and `file`)