Ch. 1 - Introduction to System Administration

- use automation
- not everything in this book is relevant to everyone
    - take what is needed and leave the rest

Thinking About System Administration

- system management
    - tension between authority and responsibility on the one hand and service and cooperation on the other
- trick is to find a balance between being accessible to users and their needs while still maintaining authority and sticking to the policies put in place for the overall system welfare
- goal of effective system administration is to provide an environment where users can get done what they need to
    - easy and efficient
    - demands of security
    - other users' needs
    - inherent capabilities of the system
    - realities and constraints of the human community in which they all are located
- kill all of 's process

```
# kill -9 `ps aux | awk '$1=="<username>" {print $2}'
```

- also this

```
$ write chavez
```

- a few basic strategies that can be applied to virtually any component tasks:
    - know how things work
        - omnipresent simple-to-use tools attempt to make system administration simple for an uninformed novice
        - someone has to understand the nuances and details of how things really work
    - plan before doing
    - make it reversible (backups)
    - make changes incrementally
    - test, test, test, before releasing

Becoming Superuser

- superuser (on Unix) refers to a privileged account with unrestricted access to all files and commands
- username of this account is root
  - log in as root directly
  - execute the command `su`
- the `su` command may be used to change one's current account to that of a different user after entering the proper password
  - takes the username
  - root is the default when no argument is provided After you enter the su command (without arguments), the
- system prompts you for the root password
  - if correct, will get the normal root account prompt (by default, a number sign: #)
  - otherwise get an error message and return to the normal command prompt
- can exit from the superuser account with exit or Ctrl-D
- can suspend the shell and place it in the background with the `suspend` command
  - can return to it later using `fg`
- new shell from `su` inherits the environment from the current shell environment
  - different from environment that root would get after logging in
- can simulate an actual root login session with the following command form

```
$ su –
```

- Unix superuser has all privileges all the time
  - access to all files, commands, etc.
- easy for a superuser to crash the system, destroy important files, etc.
- do not do routine work as superuser
- root account should always have a password
  - change periodically
- set or change the superuser password (as superuser)

```
# passwd

// Solaris and FreeBSD systems when su'd to root
# passwd root
```

- do not leave a logged-in session unattended - especially root
- can use `xlock` and other screen locking programs

Controlling Access to the Superuser Account

- any user with root password may become root at any time with `su`
  - can be configured on Solaris with /etc/default/su
- BSD systems limit access to `su` to members of group 0 (usually named wheel)
  - any user may su to root on FreeBSD if the wheel group has a null user list in the group file (/etc/group)
  - default configuration is a wheel group consisting of just root.

Running a Single Command as root

- `su root -c "<command>"` - best used for scripts
  - wrap in quotations if command contains any special shell characters, etc.
  - prompts for password (if necessary)
  - can bg command with & inside quotations

sudo: Selective Access to Superuser Commands

- sudo freely available facility that is not universal (e.g. POSIX or Unix specified)
  - installed by default on Red Hat Enterprise Linux and its derivatives
    - only installs ready-to-use in RHEL 7 and newer
    - RHEL 7 has a new option on installation to make user administrator which adds sudo access
      - does so by adding that user to the wheel group
    - RHEL 3 through 6 also installs sudo by default
      - configured such that only root could run commands through it
      - easiest way to fix is to add one or more users to the wheel group
      - run visudo as root and uncomment the %wheel ALL=(ALL)... line
  - minimal installation of Debian 7 (Wheezy) does not include sudo
    - instaled with "Standard system utilities" package set during installation
    - will automatically adds the non-administrative user to the sudo group
    - not true of Debian 6 and earlier
      - install via apt-get after installation and configure it by hand
  - not installed by default in FreeBSD or NetBSD
  - OpenBSD ships with sudo installed by default
    - configured so that only root can run commands through it
- su is closer to a universal "get me superuser privileges" command than sudo
  - systems like Ubuntu and OS X have the root account locked by default specifically to force you to use sudo instead of su
  - so su is not universal either

- requires only the user's own password to run the command
  - may run additional commands for a limited period of time without having to enter a password again
    - defaults to five minutes
    - can extend the time period by an equal amount by running `sudo -v` before it expires
    - can terminate the grace period by running `sudo -K`
- uses a configuration file
  - usually /etc/sudoers
  - must be set up by the system administrator
  - has sections that can define sudo aliases
    - uppercase symbolic names for groups of computers, users and commands
    - always use full pathnames for security
- users can use `sudo -l` to list available sudo commands
  - commands should be selected for use with sudo with care
    - shell scripts should not be used
    - any utility which provides shell escapes - the ability to execute a shell command from within a running interactive program - should not be used (editors, games, and even output display utilities like more and less
  - command runs as root when a user runs a command with sudo
    - any command run from within a utility will also be run as root
    - most text editors provide shell escapes
      - any command that allows the user to invoke an editor should also be avoided
- sudo package provides the visudo command for editing /etc/sudoers
  - locks the file
  - performs syntax checking when editing is complete

Communicating with Users

Sending a Message

- `write username [tty]`
  - use tty for a user that is logged in more than once
  - ^D to end
  - can sometimes use over a network with `write <username>@<hostname>`
- `rwho` command may be used (when available) to list all users on the local subnet
- `talk` command formats messages as a split chat screen
- can disable messages from users using `mesg n` , which will be overridden by messages from superuser

Sending a Message to All Users

- `wall` - write all
  - end with ^D

The Message of the Day

- written in /etc/motd, guaranteed to have user's attention
  - maintenance schedules
  - news
  - announcements
  - etc
- can often disable with .hushlogin file in home dir

Specifying the Pre-Login Message

- /etc/issue for prompt on unused terminals (HP-UX, Linux, Tru64)

About Menus and GUIs

- often system specific

Ups and Downs

- good points and bad points, see text

Where Does the Time Go?

- `plod` can be used to record and categorize various tasks performed (see Ch. 14)

Ch. 2 - The Unix Way

Files

- central to Unix
- commands are executable files
- device I/O is made to look like file I/O (all the way down to lowest levels)
- most IPC occurs via file-like objects
- filesystem - hierarchical (tree-structured) directory organization
  - / - root directory or base

- access to files controlled via ownership and protection
  - large part of Unix security

File Ownership

- two owners
  - user owner
  - group owner
- independent of each other
- allows for customization of ownership/security based on needs
- use `'ls -l` to display file ownership
- for new files
  - user who creates it is owner
  - for group owners
    - usually current group of user who creates the file
    - for BSD, same as group owner of containing directory
      - FeeBSD & Tru64 by default
      - most allow selective use of this pattern by using setgid on the dir
- use `chown` and `chgrp` to change ownership

```
chown new-owner files
```

- only the superuser can run the chown command on most systems
- use the -R option (R for recursive) to change the ownership of an entire directory tree
- can change both the user and group owner in a single operation

```
chown new-owner:new-group files
```

- use the chgrp command to change a files group ownership
  - also supports -R
  - non-root users calling `chgrp` must be file owner and member of the target group

```
$ chgrp new-group files
```

File Protection

- file mode on Unix
- set with `chmod`
  - r, w, x
  - different meanings for file vs. dir

- dir
    - r - search (`ls`)
    - w - alter contents (`mv` , `rm` , `cp` , etc)
    - x - `cd` into dir
    - to cd to a directory, you need
    - only execute access to `cd` into dir since reading of the directory file itself is not needed
    - need read access to the directory file to expand wildcards in order to run any command that lists or use files in the directory via an explicit or implicit wildcard (`ls` without arguments or `cat *.dat`)
    - `ls -l` also needs execute access because reading file sizes from disk implicitly changes directories
- write access on a file is not needed to delete it
    - write access to the containing dir is sufficient
- summary for dirs
    - r - allows users to list the names of the files in the directory, but does not reveal any of their attributes (i.e., size, ownership, mode, and so on)
    - x - lets users work with programs in the directory specified by full pathname, but hides all other files
    - r-x - lets users work with programs in the directory and list the contents of the directory, but does not allow them to create or delete files in the directory
    - -wx - used for a drop-box directory
        - users can change to the directory and leave files there
        - can't discover the names of files placed there by others
        - sticky bit is also usually set on such directories
    - rwx - lets users work with programs in the directory, look at the contents of the directory, and create or delete files in the directory
- access classes
    - user access (u) - access granted to the owner of the file
    - group access (g) - access granted to members of the same group as the group owner of the file (but does not apply to the owner himself, even if he is a member of this group)
    - other access (o) - access granted to all other normal users
- set file mode with `chmod` and an access string
    - accepts -R
    - access class - u, g, o, a (all)

    - operator - + (add), - (remove), = (set exact)
    - r,w,x
    - also accepts octal values
    - access class defaults to a when ommitted
    - = without an access string removes all access
    - X access type grants execute access to the specified access classes only when execute access is already set for some access class
        - use to grant group or other read and execute access to all the directories and executable files within a subtree while granting only read access to all other types of files

- `umask <octal access`

    - all future files created are given this protection automatically
    - usually put a `umask` command in the system-wide login initialization file and in the individual login initialization files given to users when creating their accounts

- special purpose access modes

    - t - save text mode, sticky bit
        - files: keep executable in memory after exit
        - directories: restrict deletions to each user's own files
        - turns on the sticky bit
    - s setuid bit
        - files: set process user id on execution
    - s setgid bit
        - files: set process group id on execution
        - directories: new files inherit directory group owner
    - l file locking
        - files: set mandatory file locking on reads/writes (solaris and tru64 and sometimes linux)
            - set via the group access type and requires that group execute access is off
            - displayed as s in ls -l listings

- save-text access on directories

- sticky bit has a different meaning when it is set on directories

    - user may only delete files that she owns or for which she has explicit write permission granted
    - even with write access to the directory (overrides the default Unix behavior)
    - designed to be used with directories like /tmp
        - world-writable
        - may not be desirable to allow any user to delete files at will

    - set using the user access class - `# chmod u+t /tmp`

- setgid access on a directory has a special meaning

    - files created in that directory will have the same group ownership as the directory (rather than the user owner's primary group)
    - useful for groups of users who need to share a lot of files
    - `# chmod g+s /pub/chem2`

- numerical equivalents for special access modes

    - `chmod 4755 uid` - setuid access
    - `chmod 2755 gid` - setgid access
    - `chmod 6755 both` - setuid and setgid access - 2 highest bits on
    - `chmod 1777 sticky` - sticky bit
    - `chmod 2745 locking` - file locking with group execute off

How to Recognize a File Access Problem

- problems that come up are usually file ownership or protection problems (from author's experience)
- for the local administrator
    - always test every change before going on to the next one-multiple
    - random changes are almost always bad
    - keep track of your actions as you perform them
- check the permissions on everything
- always check the return value of system calls if you suspect a file protection problem,
- try running the command or program as root
    - if it works fine, it's almost certainly a protection problem.
- common way of creating file ownership problems is by accidentally editing files as root

Mapping Files to Disks

- Unix maps files to disk blocks

- inode

    - data structure on disk that describes and stores a file's attributes
    - physical location on disk
    - user owner and group owner IDs
    - file type (regular, directory, etc., or 0 if the inode is unused)
    - access modes (permissions)

    - most recent inode modification, data access, and data modification times
        - first item will correspond to the file creation time if the file's metadata does not change
    - number of hard links to the file
    - size of the file
    - disk addresses of:
        - disk locations for the data blocks that make up the file, and/or
        - disk locations of disk blocks that hold the disk locations of the file's data blocks (indirect blocks), and/or
        - disk locations of disk blocks that hold the disk locations of indirect blocks (double indirect blocks: two disk addresses removed from the actual data blocks)

- number of inodes are created when a filesystem is initially created

    - usually becomes the maximum number of files of all types that can exist in the filesystem
    - example - one inode for every 8 KB of actual file storage

- inodes are given unique numbers

    - each distinct file has its own inode
    - an unused inode is assigned to a newly created file

- regular files are files containing data

    - ascii text files
    - binary data files
    - executable program binaries
    - program input or output
    - etc.

- directory is a binary file consisting of a list of the other files it contains

    - possibly including other directories (try `od -c`)
    - entries are filename-inode number pairs
        - mechanism by which inodes and directory locations are associated
        - data on disk has no knowledge of its (logical) location within its filesystem

- special files are used for character and block device files (e.g I/O)

    - reside in the directory /dev and its subdirectories
        - /devices under solaris
    - character special files
        - character-based or raw device access
        - used for unbuffered data transfers to and from a device (e.g., a terminal)

        - generally have names beginning with r (for "raw") or reside in subdirectories of /dev whose names begin with r
    - block special files
        - block I/O device access
        - used when data is transferred in fixed-size chunks known as blocks (e.g., most file i/o)
        - have the same name as character special files minus the initial r

- some devices (including disks) have both kinds of special files

- a link allows several filenames (directory entries) to refer to a single file on disk

    - hard link associates two (or more) filenames with the same inode
        - separate directory entries that all share the same disk data blocks
        - the command `$ ln index hlink` creates an entry in the current directory named hlink with the same inode number as index and the link count in the corresponding inode is increased by 1
        - may not span filesystems because inode numbers are unique only within a filesystem
        - should be used only for files and not for directories
    - symbolic links are pointer files that refer to a different file or directory elsewhere in the filesystem
        - may span filesystems because they point to a unix pathname not to a specific inode
        - created with the -s option to `ln`

- the two types of links behave similarly but not identically

    - disk contents for a hard link and soft link to the same file are initially the same
        - for the soft link the disk contents referenced by the address in its inode contain the pathname for the linked file
            - dereferencing then returns the linked file's inode and then ultimately the data in the blocks pointed to by the inode
        - the hard link points directly to the inode
    - directory listings will return the same information for the hard link and the pointed to file
        - moving one, will however, not affect the other because it only alters the directory entry
        - (pathnames are not stored in the inode)
    - deleting the original file will not affect the hard link which still points to the original inode
        - the corresponding disk blocks are only freed when an inode's link count reaches zero
    - a new file created with the same name as the original will not affect the hard link (will be assigned a free inode)
    - all regular files are technically hard links (i.e., inodes with a link count >= 1)

  - symbolic link appears as a separate entry in directory listings
    - marked as a link with an "l" as the first character in the mode string
    - always very small files
      - every hard link to a given file (inode) is exactly the same size
  - changes made by referencing either the real filename or the symbolic link will affect the contents of the original file
  - deleting the original file will break the symbolic link (it not point to anything)
    - if another file with the same name is subsequently created the symbolic link will be linked to it
  - deleting the symbolic link will have no affect on the original file

- NOTE: figure 2-2 is a good illustration

- Tru64 has context-dependent symbolic links

- a socket (officially Unix domain socket) is a special type of file used for communications between processes

  - communications end point
  - tied to a local system port
  - processes may attach to it

- named pipes are pipes opened by applications for interprocess communication

  - named means applications refer to them by their pathname
  - System V feature that has migrated to all versions of Unix
  - often reside in the /dev directory
  - also known as FIFOs

- can use `ls` to identify file types

  - long directory listing `ls -l` identifies the type of each file it lists via the initial character of the permissions string:
  - `-` - plain file (hard link) d directory
  - `l` - symbolic link
  - `b` - block special file
  - `c` - character special file
  - `s` - socket
  - `p` - named pipe
  - -F option will append a special character to each filename indicating its type (on most systems)
  - -o option color-codes filenames in the output based on their file type (on some systems)
  - -i option can determine the equivalent file in the case of hard links
    - tells `ls` to display the inode number associated with each filename

- ls can't distinguish between text and binary files (regular files)

  - can use the `file` command - e.g. `# file *`

Processes

- a single executable program that runs in its own address space (NOTE: processes vs. threads will be addressed later)
  - distinct from a job or a command
    - on Unix systems a job or command may be composed of many processes working together to perform a specific task

Interactive Processes

- initiated from and controlled by a terminal session

- may run either in the foreground or the background

  - foreground processes remain attached to the terminal
  - while running, the foreground process is the only process that can receive direct input from the terminal

- job control allows a process to be moved between the foreground and the background at will

  - when a process is moved from the foreground to the background, the process is temporarily stopped, and terminal control returns to its parent process (usually a shell)
  - the background job may be resumed and continue executing unattached to the terminal session that launched it
  - it may also eventually be brought to the foreground, and once again become the terminal's current process

- processes may also be started initially as background processes

- terminal process control

  - `&` - run command in background
  - ^Z - stop a foreground process
  - `jobs` - list background processes
  - `%n` - reference to a background job number n, e.g. `kill %2`
  - `fg` - bring backround process to foreground
  - `%?str` - reference to a background job command containing the characters in str
  - `bg` - restart stopped background processes
  - ~^Z - suspend `rlogin` session
  - ~~^Z - suspend second-level `rlogin` session

- NOTE:

  - `ssh` , `telnet` and `rlogin` are ways of logging in to a multi-user computer from another computer over a network
    - SSH is a recently designed, high-security protocol
      - uses strong cryptography to protect connection against eavesdropping, hijacking and other attacks
    - telnet and rlogin are both older protocols offering minimal security
  - some other differences with behavior and environment handling
  - prefer SSH in all cases, especially when all systems are not behind the same firewall

Batch Processes

- not associated with any terminal
  - submitted to a queue for sequentially execution of jobs
- Unix offers a very primitive batch command , but vendors whose customers require queuing have generally imple- mented something more substantial. Some of the best known are the
- best known vendor implemented versions
  - Network Queuing System (NQS)
    - developed by NASA and used on many high-performance computers including Crays, as well as several network-based process-scheduling sys- tems from various vendors. These facilities
- usually support heterogeneous as well as homogeneous networks
- attempt to distribute the aggregate CPU load evenly among the workstations in the network (load balancing)

Daemons

- daemons are server processes
  - often initiated at boot time
  - run continuously while the system is up
  - wait in the background until a process requires their service

Important Unix Daemons

Facility	Description	Daemon Names
`init`	first created process	`init`
`syslog`	system status/error message logging	`syslogd`
email	mail message transport	`sendmail`

Facility	Description	Daemon Names
printing	print spooler	`lpd` , `lpsched` , `qdaemon` , `rlpdaemon`
cron	periodic process execution	`crond`
tty	terminal support	`getty` (and similar)
sync	disk buffer flushing	`update` , `syncd` , `syncher` , `fsflush` , `bdflush` , `kupdated`
paging and swapping	daemons to support virtual memory management	`pagedaemon` , `vhand` , `kpiod` , `pageout` , `swapper` , `kswapd` , `kreclaimd`
`inetd`	master TCP/IP daemon*	`inted`
name resolution	DNS server process	`named`
routing	routing daemon	`routed` , `gated`
DHCP	dynamic network client configuration	`dhcpd` , `rpcbind`
RPC	RPC facility network port-to-service mapper	`portmap` , `rpcbind`
NFS	native Unix network file sharing	`nfsd` , `rpc.mountd` , `rpc.nfsd` , `rpc.statd` , `rpc.lockd` , `nfsiod`
Samba	file/print sharing with Windows systems	`smbd` , `nmbd`
network time	network time sync	`timed` , `ntpd`

- - `inetd` is responsible for many others
    - `telnetd`
    - `ftpd`
    - `rshd`
    - `imapd`
    - `pop3d`
    - `fingerd`
    - `rwhod`
    - see /etc/inted.conf for the full list

Process Attributes

- process ID (PID) - unique identifying number used to refer to the process.
- parent process ID (PPID) - PID of the process's parent process (the process that created it)
- nice number - process's scheduling priority
    - a number indicating its importance relative to other processes
    - distinguished from its actual execution priority, which is dynamically changed based on both the process's nice number and its recent CPU usage
- TTY - terminal (or pseudo-terminal) device associated with the process
- real and effective user ID (RUID, EUID)
    - real UID is the UID of the user who started it
    - effective UID is the UID that is used to determine the process's access to system resources (such as files and devices)
    - they are usually the same
    - when the setuid access mode is set on an executable image
        - the EUIDs of processes executing it are set to the UID of the file's user owner and they are accorded corresponding access rights
- real and effective group ID (RGID, EGID)
    - real GID is the user's primary or current group
    - effective GID is used to determine the process's access rights
        - same as the real GID except when the setgid access mode is set on an executable image
    - EGIDs of processes executing such files are set to the GID of the file's group owner and they are given corresponding access to system resources

The lifecycle of a process

- existing process makes an exact copy of itself (forking)
- new process (child process) has the same environment as its parent process
    - assigned a different PID
- the image in the child process's address space is overwritten by the one the child will run
    - done via the exec system call
    - fork-and-exec
- the new program (or command) completely replaces the one duplicated from the parent
    - the environment of the parent still remains
        - values of environment variables
        - assignments of standard input, standard output, and standard error
        - execution priority
- command `grep`
    - user's shell process forks, creating a new shell process to run the command

    - new shell process execs grep, which overlays the shell's executable image in memory with grep's
    - begins executing
    - process dies when finished
- ultimate ancestor for every process on a Unix system is the process with PID 1, init
    - created during the boot process
    - creates many other processes (fork-and-exec)
    - usually one or more executing the getty program
        - each assigned to a different serial line
        - display the login prompt and wait for someone to respond to it
        - when someone does, the getty process execs the login program

            - validates user logins, among other activities

            - if verified, login execs the user's shell

            - login does not fork in this case (forking is not always required to run a new program)

        - after login, the user's shell is the same process as the getty that was watching the unused serial line
        - getty process changed programs twice by execing a new executable
            - will create new processes to execute the commands that the user types
- on exit, process sends a signal to inform its parent process that is has completed
    - on logout, login shell sends a signal to its parent, init, as it dies, letting init know that it's time to create a new getty process for the terminal
    - init forks again and starts the getty, and the whole cycle repeats itself again and again as different users use that terminal

Setuid and setgid file access and process execution

- setuid and setgid access modes allow ordinary users to perform tasks requiring privileges and access rights that are ordinarily denied to them
    - `write` command is owned by the tty group which also owns all of the terminal and pseudo terminal device files
    - `write` command has setgid access
        - allows any user to use it to write a message to another user's terminal or window (do not normally have any access)
        - user's effective GID is set to that of the group owner of the executable file when executing `write`
- setuid and/or setgid access are also used by
    - the printing subsystem
    - programs like mailers
    - other system facilities

- setuid programs are notorious security risks
- setuid access should be avoided since it involves greater security risks than setgid

The relationship between commands and files

- Unix does not distinguish between commands and files in the ways that some systems do
    - few commands are built into each Unix shell
    - Unix commands are executable files stored in one of several standard locations within the filesystem
- access to commands == access to command files
    - no other privilege mechanism by default
- Unix shells use search paths to locate the executable's images for commands
    - a search path is simply an ordered list of directories in which to look for command executables
    - typically set in an initialization file ($HOME/.profile or $HOME/.login)
- faulty (incomplete) search path is the most common cause for "Command not found"
- stored in the PATH environment variable
- search path is used whenever a command name is entered without an explicit directory location
    - locate command the OS first looks for a file named after the command in the first dir in colon separate PATH variable, then second, and so on
    - the order of the directories in the search path is important when more than one version of a command exists
- be aware of working on systems that have both the BSD and System V versions of commands
- most of the Unix administrative utilities are located in the directories /sbin and /usr/sbin
    - locations of administrative commands can vary widely between Unix versions
- /sbin and /usr/sbin typically aren't in the search path unless explicitly added
    - either add these directories to the search path or provide the full pathname for the command when executing administrative commands

Devices

- device access through files simplifies user device access and I/O operations
    - rarely a need to worry about the specific characteristics of devices and device I/O
- Unix special file mechanism allows many device I/O operations to look just like file I/O
- administrator does need to know the details
- device files are characterized by their major and minor numbers
    - allows the kernel to determine which device driver to use to access the device (via the major number) and specific method of access (via the minor number)
- major and minor numbers appear in place of the file size in long directory listings (using `ls -l`)

- device files are created with the `mknod` command
  - takes the desired device name and major and minor numbers as its arguments
  - many systems provide a script named MAKEDEV (located in /dev)
    - easy-to-use interface to mknod

An In-Depth Device Example: Disks

- disk drives as example

- Unix organizes all user-accessible files into a single hierarchical directory structure

  - files and directories it contains may be spread across several different disk drives.
  - disks are divided into one or more fixed-size partitions on most Unix systems
    - physical subsets of the disk drive that are separately accessed by the operating system
  - may be several partitions or just one on
  - each physical disk may have one or more partitions

- disk partition containing the root filesystem is called the root partition

  - doesn't need to comprise the entire disk drive

- disk containing the root partition is generally called the system disk

- root filesystem is the first one mounted

  - early boot process
  - remaining filesystems are mounted afterwards

- mounting a disk means more for Unix than other OSes

  - refers to the process of making the device's contents available on many OSes
  - files and directories physically located on each disk partition are arranged in a tree structure
  - integral part of the process of mounting a disk partition involves grafting its local directory structure into the overall Unix directory tree
  - files physically residing on that device may be accessed via the usual Unix pathname syntax once this is finished
  - OS handles mapping pathnames to the correct physical device and data blocks

- there are a few times when the disk partition must be accessed directly for admins

  - `mount` operation is the most common
  - disk partitions may be accessed in two modes, block mode and raw (or character) mode
    - different special files are used from each mode

  - character access mode does unbuffered I/O
    - generally causes data transfer to or from the device with every read or write system call
  - block devices do buffered I/O on a block basis
    - buffers data until the operating system can transfer an entire block of data at one time

- most disk partition-related commands require a specific type of special file and won't accept the other kind

- NOTE: most Linux versions and newer versions of BSD do not distinguish between the two types of special files for IDE disks and provide only one special file per disk partition

  - IDE (Integrated Drive Electronics) is a standard electronic interface used between a computer motherboard's data paths or bus and the computer's disk storage devices
  - IDE interface is based on the IBM PC Industry Standard Architecture (ISA) 16-bit bus standard
  - also used in computers that use other bus standards
  - adopted as a standard by the American National Standards Institute (ANSI)
    - ANSI name for IDE is Advanced Technology Attachment (ATA)
    - ATA standard is one of several related standards maintained by the T10 Committee
  - IDE controller is often built into the motherboard
  - controllers were separate external devices prior to the IDE drive
    - IDE reduced problems associated with storage devices and integrated controllers

- command to mount a disk partition needs to specify the physical disk partition to be mounted (mount's first argument) and the location to place it in the filesystem, its mount point (the second argument)

```
# mount /dev/disk0a /
```

Fixed-disk special files

- special file names for disk partitions are highly implementation-dependent
  - common logic underlies all of the various naming schemes
- disk special file names can encode
  - type of disk
  - disk controller
  - disk location on its controller
  - disk partition within the physical disk (as well as the access mode)
- see table 2-8 for examples from different platforms

Special Files for Other Devices

- other device types have special files named differently
  - use the same basic conventions

Device/use	Special file naming
floppy	/dev/[r]fd*
	/dev/floppy
tape devices	/dev/rmt
	/dev/rmt/
nonrewinding	/dev/nrmt
SCSI	/dev/rst
default tape drive	/dev/tape
CD-ROM devices	/dev/cd
	/dev/cdrom
serial lines	/dev/tty
	/dev/term/
slave virtual terminal windows	/dev/tty[p-s]
	/dev/pts/
master/control virtual terminal devices	/dev/pty[p-s]
console device	/dev/console
some System V	/dev/syscon
AIX	/dev/lft0
process controlling TTY	/dev/tty
memory maps - physical	/dev/mem
memory maps - virtual	/dev/kmem
mouse interface	/dev/mouse
null devices	/dev/null
null devices	/dev/zero
pseudorandom number generation - blocking	/dev/random

Device/use	Special file naming
pseudorandom number generation - non-blocking	/dev/urandom
always full device (Linux)	/dev/full

- the allways full device is a special file
  - returns the error code ENOSPC ("No space left on device") on writing
  - provides an infinite number of zero bytes to any process that reads from it

Commands for listing the devices on a system

Unix Version	Command	Description
AIX	`lscfg`	list all devices
	`lscfg -v -l device`	device config detail
	`lsdev -C -s scsi`	list all SCSI IDs
	`lsattr -E -H -l device`	display device attributes
FreeBSD	`pciconf -l -v`	list PCI devices
	`camcontrol devlist`	list SCSI devices
HP-UX	`ioscan -f -n`	detailed device listing
	`ioscan -f -n -C disk`	limit to device class
Linux	`lsdev`	list major devices
	`scsiinfo -l`	list SCSI devices
	`lspci`	list PCI devices
Solaris	`dmesg`	boot messages identify all devices
	`getdev`	list devices
	`getdev type=disk`	limit to device class
	`devattr -v device`	device detail
Tru64	`dsfmgr -s`	list devices

The Unix Filesystem Layout

The Root Directory

This is the

- base of the filesystem's tree structure
    - all other files and directories are logically contained underneath the root directory
- /bin
    - traditional location for executable (binary) files for the various Unix user commands and utilities
    - some files within /bin are merely symbolic links to files in /usr/bin
    - /bin is sometimes a link to /usr/bin . Other directories that hold Unix commands are
    - /usr/bin and /usr/ucb also hold Unix commands
- /dev
    - device directory
    - contains special files
    - divided into subdirectories in most System V-based versions of Unix
        - each holding special files of a given type
        - names indicate the type of devices it contains
            - dsk and rdsk for disks accessed in block and raw mode
            - mt and rmt for tape drives
            - term for terminals (serial lines)
            - pts and ptc for pseudo-terminals
    - Solaris introduced a new device directory tree
        - /devices
        - many files under /dev are links to files in subdirectories of /devices
- /etc and /sbin
    - system configuration files and executables
    - contain many administrative files and configuration files
    - System V-style boot script subdirectories
        - rcn.d and init.d
        - located under one of these two locations on systems using this style of booting
    - /etc traditionally contained the executable binaries for most administrative commands In recent Unix versions, these files have
    - moved to /sbin and /usr/sbin in recent Unix versions
        - /sbin used for files required to boot the system
        - /usr/sbin contains all other administrative commands

    - /etc also contains a subdirectory default
        - holds files containing default parameter values for various commands
    - sysconfig subdirectory on Linux holds network configuration and other package-specific, boot-related configuration files
    - /etc contains two additional notable directories on AIX
        - /etc/objrepos stores the device configuration databases
        - /etc/security stores most security-related configuration files
- /home
    - conventional location for users' home directories
    - may also be a sepa- rate filesystem.
- /lib
    - location of shared libraries required for booting the system (i.e., before /usr is mounted)
- /lost+found
    - lost files directory
    - disk errors or incorrect system shutdown may cause files to become lost
        - lost files refer to disk locations that are marked as in use in the data structures on the disk, but that are not listed in any directory (i.e., an inode with a link count greater than zero that isn't listed in any directory)
    - `fsck` runs on boot and, among other things, finds these files
    - usually a lost+found directory on every disk partition
    - some Unix systems do not create the directory until it is needed
- /mnt
    - temporary mount directory
        - empty directory conventionally designed for temporarily mounting filesystems
- /opt
    - directory tree into which optional software is often installed
    - optional software products are installed under /var/opt on some systems
        - /usr/lpp on AIX
- /proc
    - process directory
    - enable processes to be manipulated using Unix file access system calls
    - files in this directory correspond to active processes
    - also additional files containing various information about the system configuration on Linux
        - interrupt usage
        - I/O port use
        - DMA channel allocation
        - CPU type
        - etc
    - HP-UX operating system does not use /proc.

- /stand
  - boot-related files, including the kernel executable
  - Solaris uses /kernel
  - Linux systems use /boot
  - FreeBSD systems use /stand for installation and system configuration-related programs and use /boot for kernels and related files used for booting
- /tcb
  - directory tree for security-related database files on some systems offering enhanced security features (HP-UX and Tru64) (trusted computing base)
  - configuration files related to the TCB are also stored under /etc/auth
    - /usr/tcb may also be used for this purpose
- /tmp
  - temporary directory
  - available to all users as a scratch directory
  - system administrator should see that all the files in this directory are deleted occasionally
  - one of the Unix startup scripts will usually clear /tmp
- /usr
  - contains subdirectories for
    - locally generated programs
    - executables for user and administrative commands
    - shared libraries
    - other parts of the Unix operating system
  - sometimes contains applica- tion programs.
- /var
  - spooling and other volatile directories (varying data)

The /usr Directory

- /usr/bin

  - command binary files and shell scripts
  - contains public executable programs that are part of the Unix system
  - many executables for the X Window System are stored in /usr/bin/X11 or /usr/X11R6/bin

- /usr/include

  - include files
  - contains C-language header files that define the C programmer's interface to standard system features and program libraries
  - /usr/include/sys contains operating system include files

- /usr/lib
  - library directory for public library files
  - contains the standard C libraries for mathematics and I/O
  - generally have names of the form libx.a or libx.so
  - extensions specify a regular (statically linked) and shared library, respectively
- /usr/local
  - local files
  - directory /usr/local/bin holds executable programs that were developed locally or retrieved from the Internet and any sources other than the operating-system vendor (by convention)
  - may be other subdirectories here to hold related files
    - man - manual pages
    - lib - libraries
    - src - source code
    - doc - documentation
- /usr/sbin
  - administrative commands (except ones required for booting, which are in /sbin)
- /usr/share
  - shared data
  - certain CPU architecture-independent static data files are stored in subdirectories under /usr/share on some recent systems
  - files could be shared among a group of networked systems
- /usr/share/man
  - one location for the manual pages directory tree
  - contains the online version of the Unix reference manuals
  - divided into subdirectories for the various sections of the manual.
- /usr/src
  - source code for locally built software packages (FreeBSD and Linux)
  - FreeBSD also uses the /usr/ports directory tree for retrieving and building additional software packages
- /usr/ucb
  - contains standard Unix commands originally developed under BSD

    - recent System V-based systems also provide BSD versions of commands so that users may use the form that they prefer
    - some BSD-based versions have similar directories for System V versions of commands

The /var Directory

- holds data that changes over time
- /var/adm
    - administrative directory (home directory of the special adm user)
    - traditionally contains the Unix accounting files
    - many Unix versions have moved them
- /var/cron, /var/news
    - /var contains subdirectories used by many system facilities
    - /var/cron and /var/news are used by the cron and Usenet news facilities, respectively
- /var/log
    - location for log files maintained by many system facilities
- /var/mail
    - user mailbox location
- /var/run
    - contains files holding the current process IDs of various system daemons and other server and/or execution instance-specific data
- /var/spool
    - contains subdirectories for Unix subsystems that provide different kinds of spooling services
        - print spooling system
        - the mail system
        - cron facility

Ch. 3 - Essential Administrative Tools and Techniques

Getting the Most from Common Commands

Getting Help

- manual page facility
- getting help for a command
- specifying a specific section
- using -k (or apropos) to search for entries for a specific topic

- useful, lesser known features
  - can request multiple manual pages within a single man command - `$ man umount` fsck `newfs`
  - has a -a option on FreeBSD, Linux, and Solaris systems
    - retrieves the specified manual page(s) from every section of the manual
- manual pages are generally located in a predictable location within the filesystem
  - often /usr/share/man
  - can configure the man command to search multiple man directory trees by setting the MANPATH environment variable to the colon-separated list of desired directories

Changing the search order

- searches the various manual page sections in a predefined order
  - commands
  - system calls and library functions
  - then the other sections (i.e., 1, 6, 8, 2, 3, 4, 5, and 7 for BSD-based schemes)
- many operating systems allow this ordering scheme to be customized via the MANSECTS entry within a configuration file
  - FreeBSD - MANSECT environment variable (colon-separated)
  - Linux (Red Hat) - MANSECT in /etc/man.config (colon-separated)
  - Linux (SuSE) - SECTION in /etc/manpath.config (space-separated)
  - Solaris - MANSECTS in /usr/share/man/man.cf and/or the top level directory of any manual page tree (comma-separated)

Setting up man -k

- `apropos` is an alias to `man -k`
- `man -k` may need setup if the system claims to support it
- command uses a data file indexing all available manual pages
  - file often must be initially created by the system administrator
  - may need to be updated from time to time

```
# makewhatis Most systems
# makewhat /usr/share/man (for Solaris)
```

- `catman -w` command is used on AIX, HP-UX, and Tru64

Piping into grep and awk

- can use `grep` for searching files or piping output into
- also `egrep`
- `awk` is also useful in pipes
  - can also be used for summing and mapping over specific fields/columns

    - can be used to generate strings from the `date` command and others

```
% ps -aux | egrep 'chavez|PID'
% alias pu "ps -aux | egrep '\!:1|PID'" % pu chavez
$ ps -ef | grep "[q]uake" | awk '{print $1}'
$ (date ; ps -ef | grep "[q]uake" | awk '{print $1 " [" $7 "]"}' \ | sort | uniq)
# find / -user chavez -fstype 4.2 ! -name /dev/\* -ls | \
    awk '{sum+=$7}; END {print "User chavez total disk use = " sum}'
$ sys_doc > `date | awk '{print $3 $2 $6}'`.`hostname`.sysdoc
$ date +junk_%d%b%y%H%M%S.junk
```

Finding Files

- locates files with common, specified characteristics
    - searches anywhere on the system specified
- general syntax

```
# find starting-dir(s) matching-criteria-and-actions
```

- starting-dir(s) is the set of directories where find should start looking for files
    - searches all directories underneath the listed directories by default
- matching-criteria tell find what types of files to look for
    - does not distinguish between file-selection options and action-related options
    - often helpful to think of them as separate types

Option	Meaning
`-atime <n>`	last access exactly n days ago
`-mtime <n>`	last modified exactly n days ago
`-newer <file>`	modified more recently than file was
`-size <n>`	n 512-byte blocks long (rounded up to next block
`-type <c>`	specifies file type - f = plain file, d = directory, and more
`fstype <typ>`	specifies filesystem type
`name <nam>`	filename is nam
`-perm <p>`	access mode is p
`-user <usr>`	owner is usr
`-group <grp>`	group owner is grp
`-nouser`	file's owner is not listed in the password file
`-nogroup`	file's group owner is not listed in the group file

- may precede time periods, sizes, and other numeric quantities with a plus sign
- can include wildcards with the -name option
  - must quote them
- multiple conditions are joined with AND by default
  - may also be joined with -o for OR combination
  - grouping is allowed using escaped parentheses
  - an exclamation point may be used for NOT

```
-atime +60 -mtime +120
\( -atime +7 -o -mtime +30 \)
! -name gold.dat -name \*.dat
```

- -perm option allows searching for files with a specific access mode (numeric form)
  - unsigned value specifies files with exactly that permission setting
  - preceding the value with a minus sign searches for files with at least the specified access
    - specified permission mode is XORed with the file's permission setting
- actions options tell find what to do with each file it locates that matches all the specified criteria

Option	Meaning
`-print`	display pathname of matching file

Option	Meaning
`-ls`	display long directory listing for matching file
`-exec <cmd>`	execute cmd on file
`-ok <cmd>`	prompt before executing command on file
`xdev`	restrict the search to the filesystem of the starting directory
`-prune`	don't descend into encountered directories (e.g. non-recursive)

- default on newer systems is -print
  - called -exec on older systerms
- -exec and -ok commands must end with an escaped semicolon (;)
  - `{}` may be used in commands as a placeholder for the pathname of each found file

```
-exec rm -f {} \;
```

- NOTE: no spaces between the opening and closing curly braces
  - may only appear once within the command.
- lists the pathname of all C source files under the current directory

```
$ find . -name \*.c -print
```

- administrative uses of `find`
  - monitoring disk use
  - locating files that pose potential security problems
  - performing recursive file operations
- locate large disk files

```
$ find /chem -size +2048 -mtime +30 -exec ls -l {} \;
```

- can use -ls instead of -exec
  - more efficient because the directory listing is handled by find internally (rather than having to spawn a subshell for every file)
- search for files not modified in a month or not accessed in three months

```
$ find /chem -size +2048 \( -mtime +30 -o -atime +120 \) -ls
```

- can delete files automatically as it finds them

```
# find / \( -name a.out -o -name core -o -name '*~' \
    -o -name '.*~' -o -name '#*#' \) -type f -atime +14 \
    -exec rm -f {} \; -o -fstype nfs -prune
```

- matching criteria and actions may be placed in any order
    - evaluated from left to right
- lists all regular files under the directories /home and /aux1 that are larger than 500K and were last accessed over 30 days ago, removes those named core

```
# find /home /aux1 -type f -atime +30 -size +1000 -print \
    -name core -exec rm {} \;
```

- find has security uses
- lists all files that have setuid or setgid access set

```
# find / -type f \( -perm -2000 -o -perm -4000 \) -print
# find / \( -perm -2000 -o -perm -4000 \) -print | \ diff - files.secure
```

- can use as a simple method for tracking changes that have been made to a system in the course of a certain time period or as the result of a certain action

```
# touch /tmp/starting_time
# perform some operation
# find / -newer /tmp/starting_time
```

Repeating Commands

- `xargs` command is a way of automating similar commands on a group of objects
    - more flexible than `find` because it can operate on any set of objects
    - most often used as the final component of a pipe
- defaults to sending as a single string list of arguments when piping it through
- can send its incoming arguments to the specified command in groups by using -n option
    - takes the number of items to use at a time as its argument

So far, all of the xargs commands we've look at have

- by default places the incoming items at the end of the specified command
- can place each incoming line of input at a specified position within the command to be executed
    - use -i option and use the form {} as placeholder for each incoming line within the command
    - can specify a different pair of placeholder characters as the argument to -i if curly braces are needed elsewhere in the command

```
# ps -ef | grep "[q]uake" | awk '{print $1}' | \
    xargs -i chargefee {} 10000
```

- -t option displays each constructed command before executing
- -p option allows you to selectively execute commands by prompting you before each one
- -t and -p together provides the safest execution mode and also enables nondestructively debug a command or script by answering no for every offered command
- -i and -n don't interact as expected
  - -i and -n conflict with one another
  - the one appearing last wins
- to execute commands containing pipes, I/O redirection, compound commands joined with semicolons, etc
  - use the -c option to a shell to execute the desired command
- to lookat the final lines of a group of files and then view all of them a screen at a time

```
$ ls -1 test00* | xargs -i /usr/bin/sh -c \
       'echo "****** {}:"; tail -15 {}; echo ""' | more
```

Creating Several Directory Levels at Once

Many people are unaware of the

- options offered by the mkdir command
- -m to set the file mode when creating
  - can use either a numeric mode or a symbolic mode as the argument to the -m option
  - can also use a relative symbolic mode
- -p option tells it to create any missing parents required for the subdirectories specified as its arguments
  - same command without -p will give an error if all of the parent subdirectories are not already present

Duplicating an Entire Directory Tree

- common to need to move or duplicate an entire directory tree while preserving the directory structure, file contents, and ownership and mode settings for every file
  - can accomplish using
    - tar
    - cpio
    - sometimes even cp

- tar command `# tar -cf - -C /<working dir> <dir to archive>` creates an archive consisting of // and all of the files and directories underneath it and writes it to standard output
  - the `-` argument to the -f option means to write the archive to stdout
  - -C option sets the current directory for the first tar command to /
- the tar command `tar -xvpf -` extracts files from standard input (indicated by -f -), retaining their previous ownership and protection
  - the -v option gives detailed output
- piping the first tar command to the second (when operating in a different dir) will require a final mv command to make the copied dir identical to the original dir name

```
# cd /<parent dir>
# tar -cf - -C /<working dir> <dir to archive> | tar -xvpf -
# mv <dir to archive> <new dir>
```

- need to vary to copy only a subset of the files and directories under the original dir

```
# mkdir /<parent dir>/<new dir>
set ownership and protection for newdir if necessary
# cd /<parent dir>/<old dir>
# tar -cvf - <subdir1> ... <glob> ... <file> ... | \
       tar -xvpf - -C /<parent dir>/<new dir>
```

- first two commands are necessary only if does not already exist
- copy only a single branch of the subtree

```
# mkdir /chem1/newdir
set ownership and protection for newdir if necessary
# cd /chem1/newdir
# tar -cvf - -C /chem/olddir src/viewers/rasmol | tar -xvpf -
```

- `cpio` can perform similar functions

```
# mkdir /chem1/newdir
set ownership and protection for newdir if necessary # cd /chem1/olddir
# find . -print | cpio -pdvm /chem1/newdir
```

- `cp` command has a -p option on all considered systems

```
# cp -pr /chem/olddir /chem1
# mv /chem1/olddir /chem1/newdir
```

- -r option stands for recursive and causes `cp` to duplicate the source directory structure in the new location

- tar works differently than cp does in the case of symbolic links
    - tar recreates links in the new location
    - cp converts symbolic links to regular files

Comparing Directories

- two identical directories may diverge over time
- use `dircmp` to determine the differences between two directories
- `diff -r` provides the equivalent functionality on FreeBSD and Linux systems
- default output from dircmp indicates only whether the corresponding files are the same or not
- add - d to determine exactly what the file differences are
    - runs diff for each pair of differing files (works only for text files)
- add -s to decrease the amount of output by limiting the second section of the report to files that differ

Deleting Pesky Files

- occasionally a user winds up with a file that cannot be removed with `rm`
    - possible for a buggy application program to put a file into an inconclusive state
    - can also be achieved with certain filesystem manipulation tools
- can be solved with the directory editor feature of the GNU emacs text editor
    - also useful to teach quoting of stange filenames

1. invoke emacs on the directory in question
    - include its path on the command line
    - or by entering its name at the prompt produced by Ctrl-X Ctrl-F
2. emacs enters its directory editing mode
    - move to the file in question using the usual emacs commands
3. enter d
    - directory editing mode subcommand to mark a file for deletion
    - use u to unmark a file
    - # to mark all auto-save files
    - ~ to mark all backup files
4. enter x
    - delete all marked files
    - answer the confirmation prompt in the affirmative

- emacs can also be useful for viewing directory contents when they include files with bizarre characters embedded within them

Putting a Command in a Cage

- system security involves tradeoffs between convenience and risk
- one way to mitigate the risks arising from certain inherently dangerous commands and subsystems is to isolate them from the rest of the system
- accomplished with the chroot command
  - runs another command from an alternate location within the filesystem
  - tricks the command into thinking that that the location is actually the root directory of the filesystem
- takes the alternate top-level directory
- FreeBSD also has a facility called jail
  - stronger version of chroot
  - allows specifying access restrictions for the isolated command

Starting at the End

- `tail` is principally used to display the last 10 lines of a file (or standard input)
  - -f option displays new lines as they are added to the end of a file
    - useful for monitoring the progress of a command that writes periodic status information to a file
- advantage over the `tee` command is that the `tail` command may be killed and restarted as many times as you like without affecting the tar command
- some versions of tail also include a -r option
  - displays the lines in a file in reverse order
  - HP-UX does not support this option
- Linux provides this feature in the `tac` command

Be Creative

- to see all the files in the current directory - `$ echo *`
  - tells the shell to display the value of "*"
  - expands to all files not beginning with a period in the current directory
- `echo` and `cd` are shell builtins

Essential Administrative Techniques

Periodic Program Execution: The cron Facility

- allows scheduling of programs for periodic execution

- administrative functions are performed without any explicit action by the system administrator (or any other user)

- `cron` is useful for running commands and scripts according to a preset schedule
    - can send the resulting output
        - to a log file
        - as a mail or terminal message
        - to a different host for centralized logging
- the `cron` command starts the crond daemon
    - normally started automatically by one of the system initialization scripts
- crontab files
    - usual: /var/spool/cron/crontabs
    - FreeBSD: /var/cron/tabs, /etc/crontab
    - Linux: /var/spool/cron (Red Hat) /var/spool/cron/tabs (SuSE), /etc/crontab (both)
- crontab format
    - usual: System V (no username field)
    - BSD: /etc/crontab (requires username as sixth field)
- cron.allow and cron.deny files
    - usual: /var/adm/cron
    - FreeBSD: /var/cron
    - Linux: /etc (Red Hat), /var/spool/cron (SuSE) Solaris: /etc/cron.d
- related facilities
    - usual: none
    - FreeBSD: periodic utility
    - Linux: /etc/cron.* (hourly,daily,weekly,monthly)
    - Red Hat: anacron utilitya
- cron log file
    - usual: /var/adm/cron/log
    - FreeBSD: /var/log/cron
    - Linux: /var/log/cron (Red Hat), not configured (SuSE)
    - Solaris: /var/cron/log
- file containing PID of crond
    - usual: not provided
    - FreeBSD: /var/run/cron.pid
    - Linux: /var/run/crond.pid (Red Hat), /var/run/cron.pid (SuSE)

- boot script that starts cron
    - AIX: /etc/inittab
    - FreeBSD: /etc/rc
    - HP-UX: /sbin/init.d/cron
    - Linux: /etc/init.d/cron
    - Solaris: /etc/init.d/cron
    - Tru64: /sbin/init.d/cron
- boot script configuration file: cron-related entries
    - AIX: none used
    - FreeBSD: /etc/rc.conf: cron_enable="YES" and cron_flags="args-to-cron"
    - HP-UX: /etc/rc.config.d/cron: CRON=1
    - Linux: none used (Red Hat, SuSE 8), /etc/rc.config: CRON="YES" (SuSE 7)
    - Solaris: /etc/default/cron: CRONLOG=yes
    - Tru64: none used

crontab files

- what to run and when to run it are specified by crontab entries
    - comprise the system's cron schedule
- any user may add entries to the cron schedule by default
    - entries are stored in separate files for each user
        - usually /var/spool/cron/crontabs
    - files are named after their username
        - /var/spool/cron/crontabs/root
- crontab files are not ordinarily edited directly
    - created and modified with the crontab command
- crontab entries direct cron to run commands at regular intervals
    - each one-line entry in the crontab file has the format

```
minutes  hours  day-of-month  month  weekday  command
```

- final field, `command`, can contain spaces within it
    - it consists of everything after the space following weekday
    - other fields must not contain embedded spaces

Field	Meaning	Range
minutes	minutes after the hour	0-59
hours	hour of the day	0-23 (0 = midnight)

Field	Meaning	Range
day of the month	numeric day within a month	1-31
month	month of the year	1-12
weekday	day of the week	0-6 (0 = Sunday)

- an entry in any of these fields can be
  - a single number
  - a pair of numbers separated by a dash (indicating a range of numbers)
  - a comma-separated list of numbers and/or ranges
  - an asterisk (a wildcard that represents all valid values for that field)
    - NOTE: the common case for this if for all increasingly sized divisions from a value (e.g. minutes -> hours, so specify minutes and run for these minutes on every hour)
- use # for commented lines
- the command field can be any Unix command or group of commands (properly separated with semicolons)
- entire crontab entry can be arbitrarily long
  - must be a single physical line in the file
- cron will use any text following a % sign as standard input for command
  - subsequent percent signs can be used to subdivide this text into lines
- cron daemon reads the crontab files when it starts up and whenever there have been changes to any of the crontab files in most implementations
  - read once every minute in some (older) versions
- BSD crontab file /etc/crontab uses a slightly different entry format
  - adds an additional field between the weekday and command fields
- see book for FreeBSD and Linux crontab entry format enhancements

Adding crontab entries

- generally create crontab entries with the `crontab` command
  - installs the text file specified as its argument into the cron spool area by default
- any previously installed crontab entries will be replaced
- the -l option to crontab lists the current crontab entries
  - can redirect output to a file to edit
- the BSD-style /etc/crontab file must be edited manually
- -r option removes all current crontab entries
- -e option allows for directly modifying and reinstalling current crontab entries in a single step
- most crontab commands also accept a username as their final argument
  - allows root to list or install a crontab file for a different user
- -u on FreeBSD and Linux

- need to carefully consider which user should execute each command run by cron
  - root
    - general system functions
    - security monitoring
    - filesystem cleanup
  - lp
    - cleanup and accounting related to print spooling
  - sys
    - performance monitoring
  - uucp
    - running tasks in the UUCP file exchange facility

cron log files

- almost all versions of cron provide some mechanism for recording its activities to a log file
  - automatic on some systems
  - messages are routed through the syslog facility on others
  - usually set up at installation time
  - occasionally necessary to configure syslog . For example, on SuSE Linux sys- tems, you'll need to add an entry for cron to the syslog configuration file /etc/syslog. conf (discussed later in this chapter).

Using cron to automate system administration

- see list for examples and durations
- useful to group long commands into custom script
- can be used to run a command a set number of times for a certain period and then stop
- FreeBSD provides the periodic command for the same reason
  - see text for more info

Linux: The /etc/cron.* directories

- Linux systems provides the /etc/cron.* subdirectories
  - scripts are run via these crontab entries on Red Hat Systems

```
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

cron security issues

- two main types
  - making sure the system crontab files are secure
    - addressed by setting (if necessary) and checking the ownership and protection on the crontab files appropriately (should not be world-writeable)
    - should be included in any filesystem security monitoring that you do
  - making sure unauthorized users don't run commands using cron
    - addressed by the files cron.allow and cron.deny
      - control access to the crontab command
    - both files contain lists of usernames, one per line
    - controlled in the following way
      - if cron.allow exists, a username must be listed within it in order to run crontab
      - if cron.allow does not exist but cron.deny does exist, any user not listed in cron.deny may use the crontab command
        - cron.deny may be empty to allow unlimited access to cron
      - only root can use crontab if neither exists
        - except FreeBSD and Linux
        - default build configuration of cron allows everyone to use it
- cron.allow and cron.deny files control only whether a user can use the crontab command or not
  - do not affect whether any existing crontab entries will be executed which will be executed until they are removed

System Messages

- system facilities generate status messages
- error messages are generated whenever there are hardware or software problems
- system administrator's most important job is monitoring these messages

The syslog facility

- syslog is a message-logging facility which provides a more general way to specify where and how some types of system messages are saved

- syslogd option to reject nonlocal messages

  - AIX: -r
  - FreeBSD: -s
  - HP-UX: -N
  - Linux: -r to allow remote messages
  - Solaris: -t
  - Tru64: List allowed hosts in /etc/syslog.auth (all hosts are allowed if it doesn't exist)

- file containing PID of syslogd
  - usual: /var/run/syslog.pid
  - AIX: /etc/syslog.pid
- current general message log file
  - usual: /var/log/messages
  - HP-UX: /var/adm/syslog/syslog.log
  - Solaris: /var/adm/messages
  - Tru64: /var/adm/syslog.dated/current/*.log
- boot script that starts syslogd
  - AIX: /etc/rc.tcpip FreeBSD: /etc/rc
  - HP-UX: /sbin/init.d/syslogd
  - Linux: /etc/init.d/syslog
  - Solaris: /etc/init.d/syslog
  - Tru64: /sbin/init.d/syslog
- boot script configuration file: syslog-related entries
  - usual: none used
  - FreeBSD: /etc/rc.conf: syslogd_enable="YES" and syslogd_flags="opts"
  - SuSE Linux: /etc/rc.config (SuSE 7), /etc/sysconfig/syslog (SuSE 8); SYSLOGD_PARAMS="opts" and KERNEL_LOGLEVEL=n

Configuring syslog

- syslogd, the system message logging daemon, writes messages to the locations specified by admin
  - collects messages sent by various system processes and routes them to their final destination based on instructions given in /etc/syslog.conf
  - organizes system messages in two ways
    - part of the system that generated them
    - importance
- format

```
facility.level  destination
```

- facility - name of the subsystem sending the message
- level - severity level of the message
- destination - file, device, computer or username to send the message to

- two fields must be separated by tab characters on most systems
    - spaces are allowed under Linux and FreeBSD
- most important defined facilities
    - kern - kernel
    - user - user processes
    - mail - mail subsystem
    - lpr - printing subsystem
    - daemon - system server processes
    - auth - user authentication system (nonsensitive information)
    - authpriv - user authentication system (security sensitive information)
        - some systems have only one of auth and authpriv
    - ftp
    - cron
    - syslog - internal messages
    - mark - timestamps produced at regular intervals (e.g., every 15 minutes)
    - local* - eight local message facilities (0-7)
        - some operating systems use one or more of them
- severity levels (decreasing seriousness)
    - emerg - system panic
    - alert - serious error requiring immediate attention
    - crit - critical errors like hard device errors
    - err - other errors
    - warning - warnings
    - notice - noncritical messages
    - info - informative messages
    - debug - extra information helpful for tracking down problems
    - none - ignore messages from this facility
    - mark - selects timestamp messages (generated every 20 minutes by default)
        - not included by the asterisk wildcard (and you wouldn't really want it to be)
- multiple facility-level pairs may be included on one line by separating them with semi-colons
- multiple facilities may be specified with the same severity level by separating them with commas
- an asterisk may be used as a wildcard throughout an entry
- see text for different types of destinations and a sample syslog.conf

- all messages are appended to log files
  - must monitor size and truncate them periodically
  - log file must already exist when the syslogd process reads the configuration file entry referring to it in order for it to be recognized on some systems
    - need to create an empty log file, add a new entry to syslog.conf, and signal (`kill -HUP`) or restart the daemon in order to add a new log file

Enhancements to syslog.conf

- several operating systems offer enhanced versions of the syslog configuration file
- see text for examples

The logger utility

- can be used to send messages to the syslog facility from a shell script

```
# logger -p auth.alert -t DOT_FILE_CHK \ "$user's $file is world-writeable"
```

- -i option, which includes the process ID within the syslog log message

Hardware Error Messages

- error messages related to hardware problems often appear within system log files
- some Unix versions also provide a separate facility for hardware-related error messages
- `dmesg` command is found on FreeBSD, HP-UX, Linux, and Solaris systems
  - primarily used to examine or save messages from the most recent system boot
  - some hardware informational and error messages also go to this facility

```
$ dmesg | egrep 'down|up'
```

The AIX error log

Viewing errors under HP-UX

The Tru64 binary error logger

Administering Log Files

- must consider
  - limiting the amount of disk space log files consume while simultaneously retaining sufficient data for projected future requirements
  - monitoring the contents of log files in order to identify and act upon important entries

Managing log file disk requirements

- log files grow without bounds and can quickly consume quite a lot of disk space
  - common solution is to keep only a fraction of the historical data on disk
- one approach involves periodically renaming the current log file and keeping only a few recent versions on the system
  - periodically delete the oldest one
  - rename the current one
  - recreate it
- see script in text
- can be run periodically via `cron`
- back up log files on a regular basis so that older ones can be retrieved from backup media in the event that their information is needed
- disk space won't actually be released until you send a HUP signal to the associated daemon process holding the file open (usually syslogd) when removing active log files
- AIX has related functionality built into syslog
- FreeBSD provides the `newsyslog` facility
- Red Hat Linux systems provide a similar facility via `logrotate` and logrotated
  - run daily by default via a script in /etc/cron.daily
  - operations are controlled by the configuration file /etc/logrotate.conf
- logrotate is open source and can be built on other Linux and Unix systems as well

Monitoring log file contents

- easy to generate huge amounts of logging information very quickly
- often need to sift through log files
- `swatch` facility runs in a variety of modes
  - examines new entries as they are added to a system log file
  - monitor an output stream in real time
  - check through a file on a one-time basis
- on recognizing a pattern specified in its input it can perform some specified action
- swatch can be used to monitor any output, not just syslog events
- -t continuously examines the tail of the file (similar to `tail -f`)
- -f scans a file once for matching entries (useful when running swatch via cron)
- -p monitors the output from a running program
- another free tool for this purpose is logcheck

Managing Software Packages

- see text for table of options for various platforms

Building Software Packages from Source Code

mtools: Using configure and accepting imperfections

- mtools is a set of utilities for directly accessing DOS-format floppy disks on Unix systems
- outlines use of `configure` , `make` , etc

bzip2: Converting Linux-based make procedures

jove: Configuration via make file settings

Internet software archives

- General
  - http://sourceforge.net
  - http://www.gnu.org
  - http://freshmeat.net
  - http://www.xfree86.org
  - http://rtfm.mit.edu
- AIX
  - http://freeware.bull.net
  - http://aixpdslib.seas.ucla.edu/aixpdslib.html
- FreeBSD
  - http://www.freebsd.org/ports/
  - http://www.freshports.org
- HP-UX
  - http://hpux.cs.utah.edu
  - http://www.software.hp.com (drivers and commercial packages)
- Linux
  - http://www.redhat.com
  - http://www.suse.com
  - http://www.ibiblio.org/Linux
  - http://linux.davecentral.com
- Solaris
  - http://www.sun.com/bigadmin/downloads/
  - http://www.sun.com/download/
  - ftp://ftp.sunfreeware.com/pub/freeware/
  - http://www.ibiblio.org/pub/packages/solaris/
- Tru64
  - http://www.unix.digital.com/tools.html
  - ftp://ftp.digital.com

- http://gatekeeper.dec.com
- http://www.tru64.compaq.com

Ch. 4 - Startup and Shutdown

About the Unix Boot Process

- bootstrapping - process of bringing a computer system to life and making it ready for use
  - computer needs its operating system to function
  - must also get the operating system started on its own without having any of the services normally provided by the operating system to do this
- basic boot process is very similar for all Unix systems
  - mechanisms used to accomplish it vary quite a bit from system to system
- depend on both the physical hardware and the operating system type (System V or BSD)
- boot process can be initiated automatically or manually
- can begin when the computer is powered on (a cold boot) or as a result of a reboot command from a running system (a warm boot or restart)
- phases of normal Unix boot
  - basic hardware detection (memory, disk, keyboard, mouse, etc.)
  - executing the firmware system initialization program (happens automatically)
  - locating and running the initial boot program (by the firmware boot program) usually from a predetermined location on disk
    - may perform additional hardware checks prior to loading the kernel
  - locating and starting the Unix kernel (by the first-stage boot program)
    - kernel image file to execute may be determined automatically or via input to the boot program
  - the kernel initializes itself and then performs final, high-level hardware checks, loading device drivers and/or kernel modules as required
  - the kernel starts the init process, which in turn starts system processes (daemons) and initializes all active subsystems
    - system begins accepting user logins when ready

From Power On to Loading the Kernel

- boot process begins when the instructions stored in the computer's permanent, nonvolatile memory (BIOS, ROM, NVRAM, etc) are executed
- storage location for the initial boot instructions is generically referred to as firmware (in contrast to "software")
- executed automatically when the power is turned on or the system is reset
- exact sequence of events may vary according to the values of stored parameters

- firmware instructions may also begin executing in response to a command entered on the system console (as we'll see in a bit). How- ever they are initiated, these instructions are used to locate and start up the system's boot program, which in turn starts the Unix operating system.
- boot program is stored in a standard location on a bootable device . For a nor- mal boot from disk, for example, the
- boot program might be located in block 0 of the root disk or, less commonly, in a special partition on the root disk for a normal boot disk
- boot program may be the second file on a bootable tape or in a designated location on a remote file server in the case of a network boot of a diskless workstation
- usually more than one bootable device on a system
  - firmware program may include logic for selecting the device to boot from
    - may be a list of potential devices to examine
- first bootable device that is found is usually the one that is used in the absence of other instructions
- boot program is responsible for loading the Unix kernel into memory and passing control of the system to it
- some systems have two or more levels of intermediate boot programs between the firmware instructions and the independently-executing Unix kernel
- some systems use different boot programs depending on the type of boot
- PC systems follow this same basic procedure
- firmware is basically just smart enough to figure out if the hardware devices it needs are accessible
- kernel executable image is named unix (System V-based systems), vmunix (BSD-based system) or something similar
  - stored in or linked to the root directory
  - AIX - /unix (actually a link to a file in /usr/lib/boot)
  - FreeBSD - /kernel
  - HP-UX - /stand/vmunix
  - Linux - /boot/vmlinuz
  - Tru64 - /vmunix
  - Solaris - /kernel/genunix
- kernel prepares itself to run the system by initializing its internal tables once control is passed to it
  - creates in-memory data structures at sizes appropriate to current system resources and kernel parameter values
- kernel may also complete the hardware diagnostics that are part of the boot process and install loadable drivers for the various hardware devices present on the system

- when prep is done, kernel creates init program with PID 1
    - process 0, if it exists, is part of the kernel itself
        - often the scheduler or the swapper (which moves process memory pages to and from swap space under System V)

Booting to Multiuser Mode

- init is the ancestor of all subsequent Unix processes and the direct parent of user login shells
- init prepares system for users during the remainder of the boot process
- has to verify the integrity of the local filesystems beginning with root
- to get around kernel and init program being on root filesystem
    - there is sometimes copy of the kernel in the boot partition of the root disk as well as in the root filesystem
    - alternatively if the executable from the root filesystem successfully begins executing it is probably safe to assume that the file is OK
    - on System V the root filesystem is mounted read-only until after it has been checked
        - init remounts it read- write
    - kernel handles checking and mounting the root filesystem itself in the traditional BSD approach
- other activities performed by init
    - checking the integrity of the filesystems, traditionally using the `fsck` utility
    - mounting local disks
    - designating and initializing paging areas
    - performing filesystem cleanup activities
        - checking disk quotas
        - preserving editor recovery files
        - deleting temporary files in /tmp and elsewhere
    - starting system server processes (daemons) for subsystems
        - printing
        - electronic mail
        - accounting
        - error logging
        - cron
    - starting networking daemons and mounting remote disks
    - enabling user logins, usually by starting getty processes and/or the graphical login interface on the system console (e.g., xdm), and removing the file /etc/nologin, if present
- specified and carried out by means of the system initialization scripts
    - traditionally stored in /etc or /sbin or their subdirectories
    - executed by init at boot time

- users may log in to the system once described activities are complete
    - boot process is complete
    - system is in multiuser mode

Booting to Single-User Mode

- booting process can place the system in single-user mode
    - doesn't have to complete all the initialization tasks required for multiuser mode
- single-user mode is a system state designed for administrative and maintenance activities
    - needs unshared control of the system
- init forks to create a new process
    - then executes the default shell (usually /bin/sh) as user root
- occasionally called maintenance mode
- system might enter single-user mode automatically occurs if there are any problems in the boot process that the system cannot handle
    - filesystem problems that `fsck` cannot fix in its default mode
    - errors in one of the system initialization files
- single-user mode represents a minimal system startup
    - root access to the system
    - many of the normal system services are not available at all or are not set up
    - search path and terminal type not set correctly
    - no daemons are running
    - system is not connected to the network
    - available filesystems may be mounted read-only
    - modifying files is initially disabled
    - only commands that physically reside on these filesystems are available initially

Password protection for single-user mode

- single-user mode does not require a password on older Unix systems
    - significant security problem
- most systems now require that the root password be entered before granting system access in single-user mode
    - accomplished via the sulogin program that is invoked automatically by init once the system reaches single-user mode on some System V-bases systems
- see text for description of various single-user login requirements

Firmware passwords

- some systems also allow assigning a separate password to the firmware initialization program

- for Linux, commonly used boot-loader programs (lilo and grub) have configuration settings that accomplish this purpose
    - grub package provides the grub-md5-crypt utility for generating the MD5 encoding for a password

Starting a Manual Boot

- almost all modern computers can be configured to boot automatically when power comes on or after a crash
- booting is initiated by entering a simple command in response to a prompt when autoboot is not enabled
- see text for low-level boot commands for our supported operating systems

Linux

- lilo - traditional Linux boot loader
    - kernels available for booting are predefined
    - can specify kernel and parameters
- grub boot loader is newer
    - can enter boot commands manually
    - root option on the kernel command locates the partition where the root directory is located (we are using separate / and /boot partitions here)
    - can send `kernel single`

Boot Activities in Detail

Boot messages

Saved boot log files

- most Unix versions automatically save some or all of the boot messages from the kernel initialization phase to a log file
    - system message facility (syslogd daemon) and the related System V `dmesg` utility are often used to capture messages from the kernel during a boot
    - must execute the `dmesg` command to view the messages from the most recent boot
    - can also view them in the /var/run/ dmesg.boot file on FreeBSD systems
- common for syslogd to maintain only a single message log file
    - conventional message file is /var/log/messages.

General considerations

- init controls the multiuser mode boot process
    - runs whatever initialization scripts it has been designed to run

    - structure of the init program determines the fundamental design of the set of initialization scripts for Unix version
        - script naming
        - location
        - sequence in which they are run
        - constraints placed upon the scripts' programmers
        - assumptions under which they operate
- differences in the System V and BSD versions of init that determines the differences in the boot process for the two types of systems

Preliminaries

- system initialization scripts usually perform a few preliminary actions before getting down to the work of booting the system
    - define any functions and local variables that may be used in the script and setting up the script's execution environment
    - path is deliberately set to be as short as possible
    - only system directories appear in it to ensure that only authorized, unmodified versions of commands get executed
    - other scripts are careful always to use full pathnames for every command that they use

Preparing filesystems

Preparing the filesystem for use is the

- first and most important aspect of the multiuser boot process is to prep filesystem

    - mount the root filesystem and other vital system filesystems (such as /usr)
    - handle the remainder of the local filesystems

- task of filesystem checking is the responsibility of the fsck* utility

- considering traditional, non-journaled Unix filesystems

    - modern filesystem types use journaling techniques adapted from transaction processing to record and, if necessary, replay filesystem changes
    - avoids the need for a traditional `fsck` command and its slow verification and repair procedures

- fsck's job for traditional Unix filesystem types (ufs - FreeBSD and ext2 - Linux)

    - ensure that the data structures in the disk partition's superblock and inode tables are consistent with the filesystem's directory entries and actual disk block consumption
    - designed to detect and correct inconsistencies between them
        - disk blocks marked as in use that are not claimed by any file
        - files existing on disk that are not contained in any directory

    - deals with filesystem structure
    - doesn't deal with the internal structure or contents of any particular file
    - ensures filesystem-level integrity, not data-level integrity
    - most cases can be repaired automatically at boot time and are completely benign
    - sometimes finds more serious problems
        - requires administrator intervention

- normal practice is to check all filesystems on every boot for traditional BSD

    - filesystem inconsistencies do on occasion crop up at times other than system crashes

- System V-style filesystems are not checked if they were unmounted normally when the system last went down

    - results in much faster boots

Checking and mounting the root filesystem

- root filesystem is the first filesystem that the boot process accesses as it prepares the system for use
- see text for sample script to check and mount root filesystem
- `fsstat` command determines whether a filesystem is clean
- root filesystem is mounted read-only until after it is known to be in a viable state as a result of running fsstat and `fsck` as needed on many systems
- remounted read-write - `# mount -o rw,remount /`
- corresponding command is `# mount -u -o rw /` on FreeBSD systems

Preparing other local filesystems

- traditional BSD approach to checking the filesystems is to check all of them via a single invocation of `fsck`
    - some System V systems have adopted this method as well
    - see text for script with long switch case to do this
    - executes the command `fsck` -p to check the filesystem's consistency
    - -p option stands for preen and says that any needed repairs that will cause no loss of data should be made automatically
        - almost all repairs are of this type, this is a very efficient way to invoke `fsck` - need to run `fsck` it manually when it boots to single-user mode if it cannot fix a disk on its own
    - rare, most common after crash from electrical storm, etc.
    - most vulnerable disks are those with continuous disk activity

- local filesystems can be mounted with the `mount` command (BSD - `mount -a -t ufs`) once all the local filesystems have been checked (or it has been determined that they don't need to be)
  - -a option says to mount all filesystems listed in the system's filesystem configuration file
  - -t option restricts the command to filesystems of the type specified as its argument

Saving a crash dump

- most Unix versions automatically write the current contents of kernel memory when a system crashes due to an operating system-level problem
  - crash dump
  - usually written to the primary swap partition
  - AIX - `sysdumpdev`
  - FreeBSD sets it via the dumpdev parameter in /etc/rc.conf
- crash dump is just a core dump of the Unix kernel
  - can be analyzed to figure out what caused the kernel program/system to crash
- need to take action to to save swap partition contents after a crash
  - will be overwritten when the system is booted and paging is restarted
  - `savecore` command copies the contents of the crash dump location to a file within the filesystem
    - exits without doing anything if there is no crash dump present
    - usually executed automatically as part of the boot process - `savecore /var/adm/crash`
- crash dumps themselves are conventionally a pair of files named something like vmcore.n (the memory dump) and kernel.n, unix.n, or vmunix.n (the running kernel)
  - extension is an integer that is increased each time a crash dump is made
  - additional files holding other system status information may be created as well
- `savecore` often disabled in the delivered versions of system initialization files since crash dumps are not needed by most sites

Starting paging

- paging can be started once the filesystem is ready and any crash dump has been saved
  - normally happens before the major subsystems are initialized
  - ordering of the remaining multiuser mode boot activities varies tremendously
- paging is started by `swapon -a`
  - activates all the paging areas listed in the filesystem configuration file

Security-related activities

- must ensure that available security measures are in place and operational
- enhanced security levels systems include
  - utilities to verify the integrity of system files and executables

    - run at boot time and must complete successfully before users are allowed access to the system
- initialization scripts on many systems often try to ensure that there is a valid password file (containing the system's user accounts)
    - `vipw` utility for editing the password file
        - makes sure that only one person edits the password file at a time
- see text for command bash pattern to detect and correct issues with editing a sensitive file

Checking disk quotas

- most Unix systems offer an optional disk quota facility
    - allows the available disk space to be apportioned among users as desired
    - depends on database files that need to be checked and possibly updated at boot time
        - `quotacheck -a` , `quotaon -a`
    - `quotacheck` checks the internal structure of all disk quota databases
        - enables disk quotas with `quotaon`

Starting servers and initializing local subsystems

- important subsystems can be started when all the prerequisite system devices are ready
    - electronic mail
    - printing
    - accounting
- most rely on daemons (server processes) started automatically by one of the boot scripts
    - purely local subsystems that do not depend on the network are usually started before networking is initialized
    - subsystems that do need network facilities are started afterwards
- sample script checks if daemon is already running via a lock file before starting
- update
    - process that periodically forces all filesystem buffers (accumulated changes to inodes and data blocks) to disk. It does so by
    - runs the sync command
        - makes sure that the disks are fairly up-to-date should the system crash
    - name varies
        - bdflush
        - syncd
        - syncer
        - fsflush
    - Linux runs both update and bdflush
    - this daemon should not be disabled
        - will seriously compromise filesystem integrity
- syslogd