A
Northern Illinois University
Academic Computing Services
Workshop

UNIX Basics for Superusers

Michael G. Prais
Swen Parson 120
753-1057

Examining the Filesystem(s)

All the pieces of UNIX are stored in a hierarchical filesytem
that appears as the roots of a tree extending below ground.
The hierarchy is developed by organizing files into directories
which can have subdirectories.
The directory structure provides a *path* to a file.

pwd	Prints the working (current) directory.

The directory names along the path are separated by a slash (/).

ls	Lists the contents of the current directory.

Nothing is displayed before the next prompt when the directory is empty
of visible filenames.

Filenames that begin with a period (.) are not displayed by *ls*.
Listing the current directory using the *-a* option
provides information about all files in a directory
including the directory (.) and its parent directory (..).

ls -a	Displays all files in the current directory.

Filenames can be made invisible by preceding them with a period.

alias l "ls -aCF \!* | more"	Defines an easily typed version of a command that useful for examining the filesystem.

l	Lists all files in the current directory in columns with their filetype.

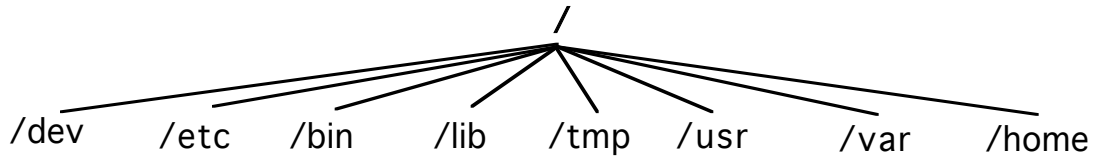
Filenames are displayed with the following suffices:

*	Indicates an executable file.
/	Indicates a subdirectory.
@	Indicates an indirect link to another file.

Watch the first line of the listing; it displays the total number of files.

q	Stops *more* from listing additional filenames.
cd ..	Changes the current directory to the parent directory.
l	Lists the files in the now current directory.
cd /	Changes the current directory to the root directory.
l	Lists the files in the current (root) directory.

The *root directory* (/) contains several important subdirectories as well as important files.



The file */boot* is the first file executed as the system starts; the system then executes the file */vmunix*.

The file */vmunix* is the *kernel* of the operating system. It has all the basic functions and all the connections to the hardware through sets of instructions called *device drivers*.

l /sys — Displays files needed to reconfigure and rebuild the kernel.

l /etc — Displays the administrative commands and the system configuration files that are required for minimal operation.

There are many links in */etc* to maintain compatibility while reorganizing.

The directory */dev* contains all the special files that provide access to the device drivers in the kernel and the devices attached to the system.

l /dev/*mem — Displays the processor memory drivers for kernel and available memory.

l /dev/tty* — Displays the tty (terminal) drivers.

There are two types of device drivers:
character-oriented and block-oriented drivers.
Character-oriented drivers send a stream of
(raw) unbuffered characters to the device
so that a user supplied routine can flexibly buffer the characters.
These drivers provide direct calls to the devices.
Character-oriented device drivers are identified
with the letter r (for raw) at the start of their names.

l /dev/r* — Displays character-oriented device drivers.

l /dev/rsd* — Displays character-oriented SCSI disk drivers.

Under SunOS, the number in the name is the SCSI address
and the letter is the drive partition.

l /dev/rst* — Displays character-oriented SCSI cartridge tape drivers.

l /dev/rmt* — Displays character-oriented

non-SCSI reel-to-reel tape drivers.

l /dev/rfd* Displays character-oriented floppy diskette drivers.

Block-oriented drivers use intrinsic routines to buffer character blocks before sending them to the device. Block-oriented drivers index each block in a file with a number (address) and are capable of random as well as sequential access. Unlike character-oriented devices, block-oriented devices can be rewound. These drivers provide calls to strategies that access devices directly. Block-oriented drivers are not available for tape drives.

l /dev/sd* Displays block-oriented SCSI disk drivers.

l /dev/fd* Displays block-oriented floppy diskette drivers.

l /tmp Displays temporary files in use or forgotten by various processes or users.

The directory */tmp* is usually erased whenever the system is rebooted.

The directory */var* is used to contain directories and files that are variable in size, such as adm, spool, tmp, and preserve. Files in the directory */var* are often changing. The existence of */var* and the links from */usr* allow the size of */usr* to remain constant.

l /var Displays directories which often vary in size.

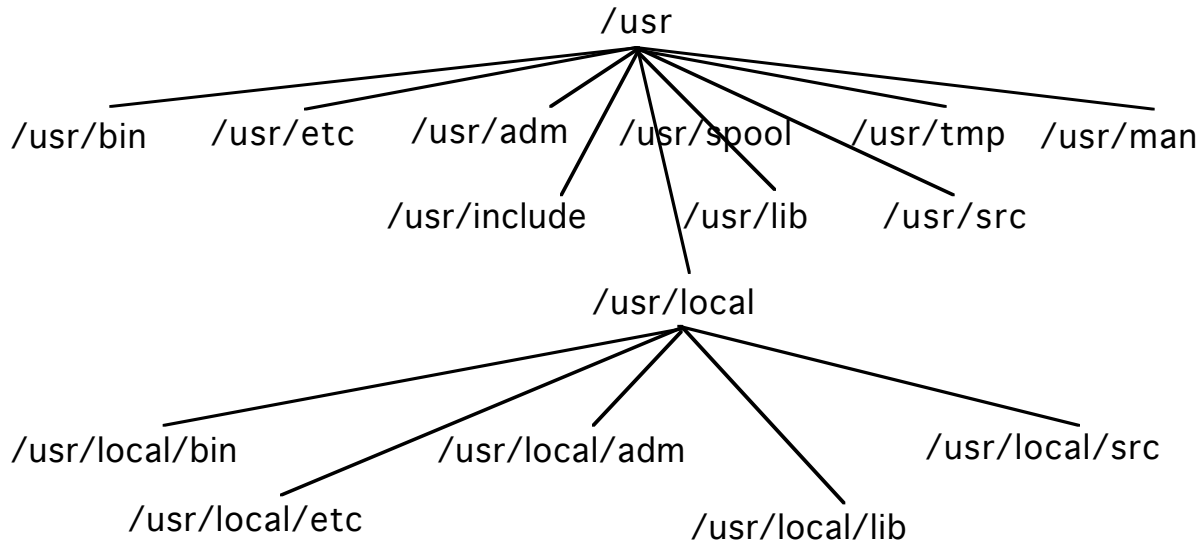
l /var/tmp Displays temporary files in use or forgotten by various processes or users.

The directory */var/tmp* is not erased when the system is rebooted.

l /var/preserve Displays a directory used by editors to save interrupted editing sessions.

l /usr Displays the files in /usr.

The directory */usr* contains a variety of files and directories.
Originally, */usr* contained "everything else"
including user home directories.
It is often wise to separate files
so that similar files with differing lifetimes
can be duplicated (backed up) on separate tapes.
The links to other directories is the result of this reorganization.



l /usr/etc	Displays administrative utilities and configuration files not necessary for minimal operation.
l /usr/bin	Displays executable commands that are usually supplied with UNIX and that may not be needed for minimal operation.
l /usr/ucb	Displays directories that are used to store files used to build BSD utilities. Files in */usr/lib* are linked into */usr/bin*.
l /usr/src	Displays directories that contain the source code for UNIX commands.

l /usr/include	Displays directories that contain standard declarations (headers) of programing objects used in UNIX commands.
l /usr/lib	Displays directories that contain libraries of object code for building UNIX commands.
l /usr/man	Displays directories containing manual pages describing the various files and commands.
l /usr/local	Displays subdirectories, such as *src*, include, *lib*, and *bin*, for building local utilities.

The directory */usr/local* provides a place where you can build and store utilities that can be used by any user of your workstation. This directory is basically a place to publish utilities (with the blessing of the superuser). To limit the size of */usr*, */usr/local* is often linked to */local*, or something similar, on a separate partition which is used to hold directories for application packages that are purchased to add value to your system.

l /usr/adm	Displays accounting utilities and log files. */usr/adm* is linked to */var/adm*.
l /usr/spool	Displays directories for queuing files by utilities like *mail*, *lpr*, *uucp*, and so on. */usr/spool* is linked to */var/spool*.
l /usr/tmp	Displays a directory for temporary storage. */usr/tmp* is linked to */var/tmp*.
l /sbin	Displays files needed to mount */usr*.
l /bin	Displays the basic executable commands that are required for minimal operation. */bin* is linked to */usr/bin*.

l /lib	Displays object modules used to build the compilers and preprocessors. */lib* is linked to */usr/lib*.

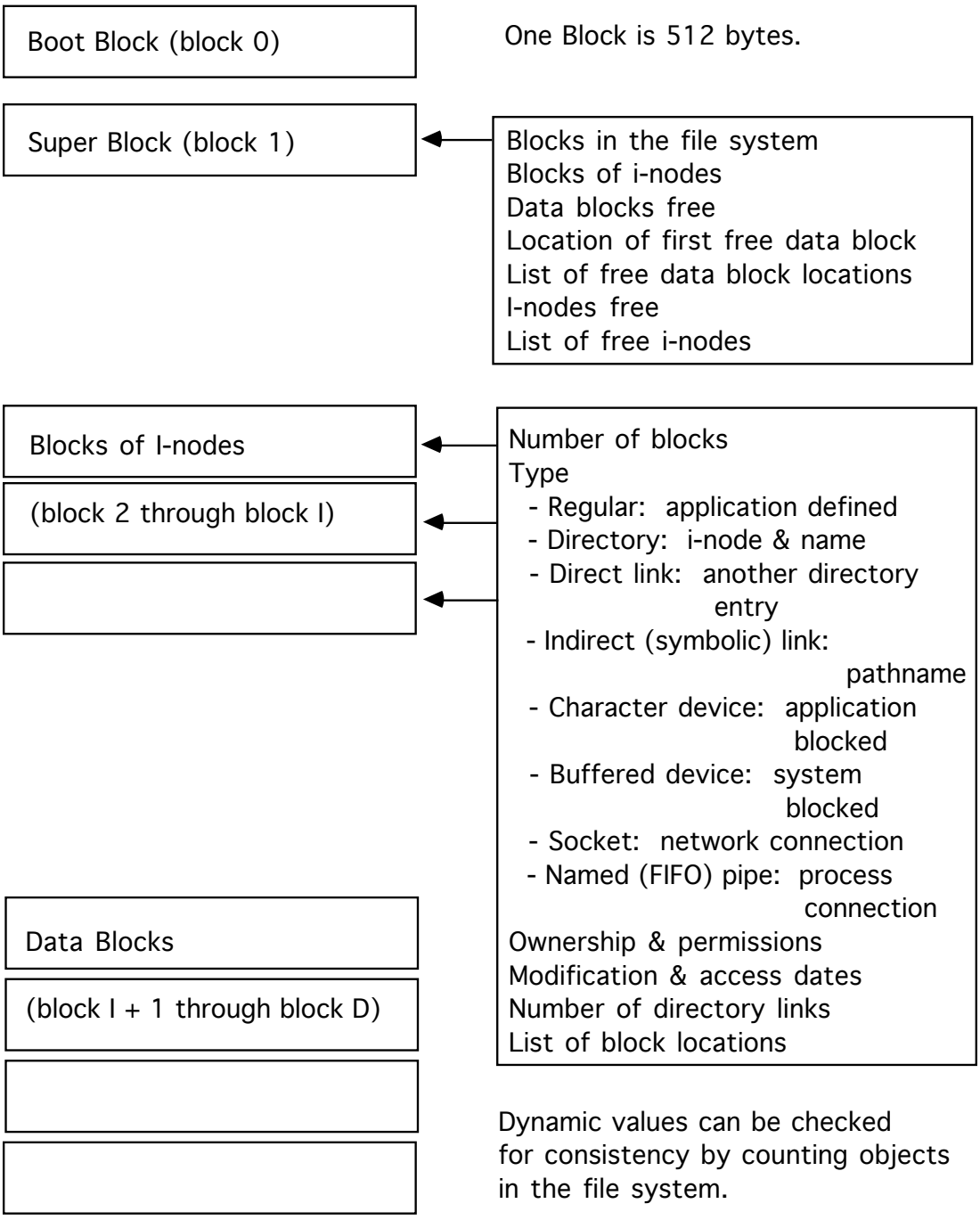
The directory */home*, */users*, */usr1*, or something similar, should be used to contain user home directories whose contents change often.

l /home	Displays the user home directories.

The Underlying Structure of a File System

A *file system* is a structure on a storage medium such as a disk drive that holds information in files.
The structure of a file system is not just a container for objects.
A file system consists of a *boot block*, a *super block*, blocks of file information (*i-nodes*), and blocks of data and pointers to other blocks of data.

The Unix File System



The *boot block* contains the instructions
that are read by the system ROM monitor and
that describe how to get the */boot* file from disk to processor memory.

The *super block* contains information that defines
the extent of the file system
and tracks the available i-nodes and data blocks.

Each *i-node* describes a file in the file system *except for its name*
and points to the contents of the file in data blocks
or to the driver that accesses information from outside of the file system.
Filenames are held in directory files (directories):
a directory connects a filename to an i-node.
Each filename is a direct link to an i-node.
This allows several directories to address the same file (i-node)!
An i-node remains until its last link is removed.
The i-node structure allows devices to be addressed as files.

od -s .	Displays the contents of the current directory as a file.

The operation of a command on arguments, that is, an action on objects,
typically requires the system to move blocks of information
between processor memory and disk memory.
What a process does with a file depends on the type of file.
The *filetype* for each file (object) in the file system
is stored in its i-node.
The system checks the filetype whenever instructed to access a file.
The long listing for a file or directory displays the filetype
as the first character in the description.

ls -l

Displays the long listing of file information including type, permissions, ownership, size, and modification date for files in the current directory.

-	Indicates a direct link to a regular file.
d	Indicates a direct link to a directory.
l	Indicates an indirect (soft or symbolic) link to a file.
c	Indicates a character-oriented device driver.
b	Indicates a block-oriented device driver.
p	Indicates a named FIFO pipe, that is, a process that acts as a file.
s	Indicates a socket, that is, a network connection that acts like a file.

When the object is a *direct link, hard link, regular file, or directory file*,
the kernel identifies the i-node that contains the information requested
and handles the request for a block (buffered) device.
When the object is an *indirect, soft, or symbolic link file*,
the kernel reads its contents to get another filename or pathname.
When the object is a *special file*,
the kernel identifies a device driver by its *major device number*
and requests that the driver handle the exchange of information
for that device using the *minor device number* to configure it.
When the object is a *pathname*,
the kernel identifies the i-node
that contains the current directory or that contains the root directory
and handles the request for the contents of the directory inode
from a block (buffered) device.
The kernel then identifies the next piece of the pathname,
looks for the next piece of the pathname in the contents of the directory,
and so on, until it is dealing with one of the above filetypes.

While not special files themselves,
accessing direct links to regular files, indirect links, and directories
result in requests to use a block-oriented device driver
after the kernel determines the location on a disk drive
that is to be accessed.

ls -l /etc | more	Displays the long listing of file information including type, permissions, ownership, size, and modification date for files in the /etc directory.

All devices known to the system have files (i-nodes) listed in */dev*.
The directory */dev* is conventional and not special.
The device numbers are listed in the long file listings of device files
in place of the file size.

ls -l /dev | more	Displays the long listing of file information including type, permissions, ownership, size, and modification date for files in the /dev directory.
#mknod /dev/ttyo c 129 0	Creates a special file to access a device driver identified by major device number *129* with the parmeters given by minor device number *0*.

The existence of unusual special files is an indication of
a security problem because only the superuser can create special files.

cd	Sets your home directory as your current directory.
ls -ia	Displays the i-node numbers associated with the current directory.
touch myfile	Creates an empty file in the current directory.
ls -i myfile	Displays the i-node number of *myfile*.

ln myfile myfile2	Creates a second link to *myfile*.
ls -i myfile*	Displays the i-node number associated with two files.

I-nodes are unique within a file system.
The files just created were done so in the */home* directory which is the */dev/sd2g* filesystem.
The *ncheck* command is used to find all the names of an i-node within a file system.

ncheck -i *i-node* /dev/sd2g	Displays the name of *i-node* by checking all directories within a filesystem.
df	Lists the file systems and their capacity.
#ncheck -s *filesystem*	Displays all special files in *filesystem*.

The i-node carries ownership and permission attributes.
Each i-node has individual and group owners.
The individual owner is usually the creator.
Read (examine), write (change), and execute/extend permissions are defined for the individual owner (user), the group owner (group), and outsiders (others).
Extend permission on a directory allows the use of a directory name in a pathname,
that is, pathnames is allowed to extend through the directory.
The permissions are listed by the long listing for files in three groups of three characters *rwx rwx rwx* for user, group, and outsider permissions, respectively.

ls -lg myfile	Displays the ownership and permissions of a file.
ls -lg	Displays the ownership and permissions of the files in the current directory.

ls -lgd	Displays the ownership and permissions of the current directory (as a file).

Files are protected from deletion
(and directories are protected from the creation of unwanted files)
by changing the write permission of the current directory.

chmod u-w .	Removes your ability to create and remove files in the current directory.
ls -lgd	Displays the ownership and permissions of the current directory (as a file).
/bin/rm myfile	Attempts to remove a file, but is denied permission.
chmod u+w .	Allows you to create and delete files in the current directory.
/bin/rm myfile	Removes a file.

It is a security risk to allow users write permission on directories
owned by the superuser.

ls -lgd / /etc /bin /lib /dev ls -lgd /usr /usr/etc /usr/bin /usr/lib ls -lgd /var /var/spool	Displays ownership and permissions on some directories owned by the superuser.

It is a major security risk to allow users write permissions on
the startup files of the superuser in the root directory (/).

ls -lg /.[a-s]* ls -lg /vmunix /boot /rc* ls -lg /etc /usr/etc | more ls -lg /var/spool/crontabs/cron/root	Displays ownership and permissions on some files owned by the superuser.

It is a security risk to allow users write permission on executable files available to all users.

ls -lg /bin /usr/bin /usr/ucb | more ls -lg /usr/local | more	Displays ownership and permissions on some executable files owned by the superuser and used by all.

It is a security risk to allow users write permission on special files in the /dev directory.

ls -lg /dev | more	Displays ownership and permissions on special files owned by the superuser or other system users.

Terminal drivers /dev/tty* are owned by root when not in use, but are given to a user with read and write permissions for all when the user calls in or out on them.
The directories /tmp, /var/tmp, /var/preserve, /var/spool/mail, and /var/spool/uucp/uucppublic have permissions for all to use. However, the *sticky bit* is set so that only the individual owner of a file in one of these directories can remove that file.

ls -lg /tmp	Displays the permissions on /tmp. Notice the *t* in the execute field for the individual owner.

The individual owner is originally the creator of the i-node and the group owner is originally the current group of creator. When the *set group id bit* of the current directory is set, the group owner is originally the group owner of the current directory. The set group id bit appears as an *s* in the group execute/extend field.

On creation, certain permissions are removed from files as specified by the individual owner.
Permissions on regular files are removed from all permissions granted (777) and permission on directories are removed from all read and write permissions granted (666).
The permissions that are removes are set by the *umask* command.

umask	Displays the permissions that are removed on creation.

The digits of the umask refer to the permissions of the individual owner, the group owner, and the outsiders in that order.
Each digit is a sum of permission values:

4	Indicates read access.
2	Indicates write access.
1	Indicates execute or extend access.

A umask of 22, that is, 022, signifies that
write access is removed at creation from group owners and outsiders.

ls -l myfile2	Displays the permissions of myfile2.

A umask of 267 signifies for regular files
that write access is removed for the individual owner and
that execute access is removed for the group owners and
all access is removed for outsiders.

umask 267	Specifies which permissions are removed when files are created.
touch myfile3	Creates an empty file in the current directory.
mkdir mydir	Creates a directory within the current directory.
ls -l my*	Displays the permissions of files and directories that begin with my.

Extend permission for directories allows
that directory to be used in a pathname,
that is, to extend the pathname to a file with this directory name.

Because the umask is subtracted from (XORed with) 666 for directories,
extend permission is granted for subdirectories
only when execute permission is not granted.

```
6 = 110      6 = 110
    XX0          XX1
-------      -------
    YY0          YY1
```

It is possible to allow outsiders to examine a series of directories
without allowing them to use the files found there.

chmod u+x myfile2	Adds execute permission to a file for a user.
ls -l myfile2	Displays the permissions of myfile2.

Files marked for execution should contain instructions
that have been generated by a compiler or assembler
or that have been created for an interpreter.
Typing the filename at a prompt
(just like you have done for ls, chmod, umask, and so on)
executes the file.

myfile2	Executes a null instruction.
myfile3	Attemps to execute the file, but is denied permission.
ls -l /usr/bin	Displays a variety of executable commands.

It is a security risk to allow files created by the superuser
to have any permissions automatically granted to outsiders.
The umask for the superuser should be no freer than 027.
A superuser must be aware of the permissions set on all files
used or created in order to protect the configuration and operation
of the system.
However, creating a file as the superuser leaves the superuser
as the individual owner and prevents others form using this file.

The superuser, and only the superuser, can change individual file ownership.

#chown *another_username* myfile2 Changes individual ownership.

Any user can change group ownership of a file from one group to another if the user is a member of both groups.

groups Displays the groups in which you belong.

#chgrp *another_group* myfile2 Changes group ownership.

#chown -R *username.groupname directoryname*

Changes the individual and group ownership of all the files in a directory.

Setting the *set userid bit* allows the user who executes a file to assume the identity of the individual owner of the file during its execution.

chmod u+s myfile2 Sets the set userid bit.

ls -l myfile2 Lists information about a file.
Notice the *s* in the user execution bit.

It is a major security risk for the superuser to create files with the userid bit set.

Setting the *set groupid bit* allows the user who executes a file to assume the identity of the group owner of the file during its execution.

chmod +x myfile3 Makes a file executable.

chmod g+s myfile3 Sets the set groupid bit.

ls -l myfile3 Lists information about a file.
Notice the *s* in the group execution bit.

The permissions on indirect links are read and write by all.
Changing the permissions on indirect links only effects the permissions on the target of the link.

umask 022	Returns the user mask to the value recommended for typical users.

Manipulating the File System

File systems are placed in partitions on a disk drive.
A *partition* is a section of disk drive that is isolated from other sections and which can be duplicated (backed up to tape) as a whole.
A partition is a set of contiguous *cylinders* on a disk drive.

df	Displays the disk partitions, their capacities, and their availability.
dkinfo sd0	Displays the characteristics of a drive.
more /etc/format.dat	Displays disk and controller types and characteristics.

The *format* command formats and repartitions a drive.
Formating destroys any information on disk.
Format is not available in the workshop.
Do not format you drive without instructions from an experienced superuser.

#format *drive*	Displays and changes the structure of a drive.
p	Displays and changes the partition table of a drive.
p	Prints the partition table.

sd0a	starting cyl	0, # blocks	324000	(400/0/0)
sd0b	starting cyl	400, # blocks	183060	(226/0/0)
sd0c	starting cyl	0, # blocks	1317060	(1626/0/0)
sd0d	starting cyl	626, # blocks	324000	(400/0/0)
sd0e	starting cyl	0, # blocks	0	(0/0/0)
sd0f	starting cyl	0, # blocks	0	(0/0/0)
sd0g	starting cyl	1026, # blocks	324000	(400/0/0)
sd0h	starting cyl	1426, # blocks	162000	(200/0/0)

l — Labels the drive with the current partition table.

q — Exits the partition table.

q — Exits format.

Note that partition *sd0c* is the whole disk.
You can repartition your drive,
but backup your drive to tape before attempting to do this.
The letters *a* through *h* select a partition
whose starting cylinder and size are to be changed.
The program prompts for the starting cylinder and then the size.
The size can be given as a simple number of blocks
or as the number of cylinders in the format (*number* /0/0).

To repartition a drive efficiently,
you need to know the cylinders/drive, the heads/cylinder,
the sectors/track, the blocks/sector, and the bytes/block for the drive.
The tracks/cylinder is the same as the heads/cylinder.
For instance, the calculation of the cylinders required
to store 150 megabytes on a particular drive is shown below.

1 block/sector x 54 sectors/track x 15 heads/cylinder x 1 head/track
= 810 blocks/cylinder

810 blocks/cylinder x 1626 cylinders/drive = 1,317,060 blocks/drive

150 MB/partition x 1024 KB/MB x 2 blocks/KB = 307,200 blocks/partition

(307,200 blocks/partition)/(810 blocks/cylinder)
= 379.26 cylinders/partition

Rather than specify the partition size in megabytes or blocks,
it is more efficient to use cylinders when you repartition.
Starting with the approximate size in megabytes,
calculate an approximate size in cylinders,
choose a even number of cylinders,
and recalculate the size in megabytes.

(384 cylinders/partition x 810 blocks/cylinder)/(2048 blocks/MB)
= 151.875 MB/partition

A super block and i-node blocks must be placed on a partition
for it to be used as a UNIX file system.

#newfs *partition*	Installs a file system on a new partition and places a *lost+found* directory in the top directory of the filesystem for the filesystem checking command (fsck).

When all cylinders are not allocated by the filesystem,
recalculate the partition size and repartition the drive
to pick up the cylinders that were not allocated.

File systems must be *mounted* in order to be accessed.

mount	Displays all the mounted file systems and their *mount points*.
#mount *partition* /mnt	Attaches a partition at /mnt.

File systems are mounted at any available directory,
while any structure below that directory becomes unavailable.
The directory */mnt* is available as a temporary mount point.
The / (root) of the mounted file system takes the name of the mount point
and all files below it are referenced to the mount point.
The directory */usr* is usually a mounted file system.
Mounted file systems should always have a *lost+found* directory
at their root.

ls /usr Displays the files in the /usr directory.

File systems must be unmounted to be checked and changed.
All files must be *closed* before a file system can be unmounted.
Having a directory on the filesystem as your current directory
is an example of a process holding a file (the directory file) open.
The device is considered *busy* until then.

#umount *partition* Unmounts a file system
contained in a partition.

The file */etc/fstab* describes the file systems that are to be mounted.

more /etc/fstab Displays the contents of /etc/fstab.

The following lines describe the information in */etc/fstab*.

#comment
device mount_point type access mount_order

Note the swap partitions and the partitions that are to be ignored
because they do not hold a filesystem.
The file */etc/mtab* describes the file systems that are mounted.
This file should not be edited.

more /etc/mtab Displays the contents of /etc/mtab.

A file system should be regularly checked to insure consistency.
This checking is automatically done when the system is started.

#fsck -p Checks and repairs inconsistencies.

The inconsistencies that are checked for and removed are found by comparing the information in the super block, the i-nodes,
and the directories, and by counting the i-nodes, the links in directories, and the data blocks.

Check the size and format of the elements in each i-node.
Check for and remove data blocks
located outside of the file system.
Check for and isolate data blocks
claimed by an i-node and the free list.
Check for and isolate data blocks
claimed by more than one i-node.
Count data blocks and reset number of blocks in each i-node.

Check the size and format of each directory.
Check for and remove directory entries
that point to out-of-range i-nodes.
Check for and remove directory entries
that reference i-nodes in the free list.
Count directories that reference each i-node
and reset number of links in each i-node.

Check the paths of each directory entry to / (root).
Place unattached blocks and i-nodes
in the lost+found directory
at / (root) of the file system.

Check the size and format of elements in the super block.
Check for unreferenced i-nodes and add to free list.
Check for and remove out-of-range i-nodes in the free list.
Reset the free i-node count.
Check for unclaimed data blocks and add to free list.
Check for and remove out-of-range data blocks in the free list.
Reset the free data block count.

When your disks become full,
you have to find and remove all extraneous files.
This should be done on a regular basis beforehand.
Eventually you may have to add another drive to your system.

du ~	Displays the disk use by directory.
bc	Starts the calculator.
CTRL \	Stops the calculator with a *core* dump.

find ~ -name 'core' -print

Displays all files with the name *core*
found in your home directory.

find ~ -name 'core' -ok ls -l {} \;

Prompts to run the *ls -l* command
on each file found *{}*.
Answer *y* to execute the command.

find ~ -name 'core' -exec ls -l {} \;

Runs the *ls -l* command
on each file found *{}* without prompting.

find ~ -name 'core' -print | xargs -p ls -l

Prompts once (-p) to run the *ls -l* command
on each filename generated by *find*.

These commands can be used to archive and remove files
by replacing the *ls -l* command.

The *find* command has other options that selects files by other attributes found in the i-node.

find / -xdev -type d -user root -perm 2 -ls

Displays information about directories
owned by the superuser
and changeable by outsiders.

The *-xdev* argument is used to limit searches in this workshop as it prevents searching across partitions.

find / -xdev -type f -user root -perm 2 -ls

Displays information about files
owned by the superuser
and changeable by outsiders.

find / -xdev \(-type b -o -type c \) -print | grep -v /dev

Displays special files outside of /dev.

find / -xdev \(-perm -4000 -o -perm -2000 \) -ls

Displays information about files
with *userid* and *groupid* bits set.

The dashes preceding the four digit permissions are necessary.
It results in the comparison (*filemode*&*value*)==*value*.

find / /etc -prune -user \! root -ls — Displays files in / and /etc only not owned by the superuser.

find / -xdev -nouser -print — Displays files owned by a user not included in the *passwd* file.

find / -xdev -nogroup -print — Displays files owned by a group not included in the *groups* file.

find /home/*username* -user \! *username* -ls

Displays files in your directory that are not owned by you.

find /home -type d -perm 2 -ls Displays user directories that can be changed by outsiders.

find /home -perm 777 -ls Displays unprotected user files.

Duplication of groups of files or of the whole file system is insurance that allows you to replace information in the event of hardware, software, administrative, or user failure.

The tape archive command *tar* is most useful to transfer files and directories between systems (or even large directories). Most software is provided on tape in *tar format*. The alternative is *cpio format*.

#tar vcf /dev/rst0 **.** Verbosely creates a tar file from the files in the current directory and places that file on the SCSI tape drive 0.

#tar tf /dev/rst0 | more Lists the table of contents of the first (tar) file on the tape.

#tar vxf /dev/rst0 **.** Verbosely extracts the files from the tar file on SCSI tape drive 0 and places them in the current directory.

The tape archive does not need a tape; it can be demonstrated with files.

cd /usr/bin Changes the current directory.

tar vcf ~/mytar ./ch* Creates a tar file *mytar* in your home directory containing all the commands that begin with *ch*.

Note the use of ./ which allows files to be restored
to the current directory.

The verbose option of *tar* lists each filename as it is processed.

cd	Returns to your home directory.
tar tf mytar	Displays all the files contained in a tar file.
mkdir mybin	Makes a directory *mybin* in the current directory.
cd mybin	Changes the current directory.
tar vxf ../mytar .	Extracts all the files in a tar file and places them in the current directory.
ls	Lists the files in the current directory.

#chdir *dirA*; tar xf - | (chdir *dirB*; tar cf -)

Copies files and directories between directory *dirA* and *dirB*.

The *dump* and *restore* commands are more appropriate
for duplicating file systems for back up purposes.
Anyone in the *operator* group can use these commands.

Dump can archive only those files that have been modified
since the last time files were dumped.
Dump provides a *dump level* which is a numeral from 0 to 9
which specifies selecting only those files that were modified
since the last dump of a lower level.

The last dates for each dump level are kept in /etc/dumpdates.

cat /etc/dumpdates	Displays the dump levels and dates.
dump w	Lists which filesystems that need to be backed up.

Dump is not available in this workshop
because it requires access to a filesystem and updates */etc/dumpdates*.

#dump 9ucsdtf 600 1200 18 /dev/nrst0 /dev/rsd0g

Does a level **9** dump
of the file system in partition sd0g
to a **f**ile /dev/nrst0 which is a **c**artridge tape
with a **s**ize of 600 feet,
a **d**ensity of 1200 bytes per inch,
18 **t**racks, and **u**pdates /etc/dumpdates.

The arguments must be listed in the same order as the options *sdtf*.

The device ***n****rst0* does **n**ot rewind so that its choice allows
a sequence of dump files to be placed on the same tape.
The *mt* command manipulates tapes.

#mt -f /dev/rst0 retension

Exercises a dormant tape prior to use.

#mt -f /dev/**n**rst0 fsf

Moves a non-rewinding tape forward
past one (dump) file.

#mt -f /dev/**n**rst0 bsf

Moves a non-rewinding tape backward
past one (dump) file.

#mt -f /dev/rst0 rewind

Rewinds the tape when finished.

The command */etc/restore* has an interactive component
that allows examination of a dump file
and selection of individual files for extraction.

#restore if /dev/rst0	Displays a prompt for the extraction of files archive in a dump file on tape.
help	Displays available commands.
pwd	Displays the current directory.
ls	List files in current directory.
cd *subdirectory*	Makes a subdirectory the current directory.
add *filename*	Marks a file or directory for extraction.
delete filename	Unmarks a file or subdirectory for extraction.
verbose	Sets restore to list each files as it is extracted.
extract	Begins extraction of all marked files.
quit	Exits restore.
#restore rf /dev/rst0	Restores the whole dump file to the current directory.