CSC 275 – Operating Systems Practicum

Lecture 3 – File System Commands

Part 2 – UNIX Directory Fundamentals

The File System

- A file in UNIX is a sequence of bytes. UNIX imposes no structure in a file and there is no extension implicit in a file name.
- UNIX views peripheral devices as files:

```
[SIEGFRIE@panther ~]$ ed
a
now is the time
for all good people
.
w junk
36
q
[SIEGFRIE@panther ~]$ ls -l junk
-rw-r--r--  1 SIEGFRIE users 36 Sep 13 12:12 junk
```

Files and Their Structure

- **`junk`** is a file with 36 bytes.

```
[SIEGFRIE@panther ~]$ ls -l junk
-rw-r--r--  1 SIEGFRIE users 36 Sep 13 12:12 junk
[SIEGFRIE@panther ~]$ cat junk
now is the time
for all good people
[SIEGFRIE@panther ~]$ od -c junk
0000000   n   o   w       i   s       t   h   e       t   i   m   e  \n
0000020   f   o   r       a   l   l       g   o   o   d       p   e   o
0000040   p   l   e  \n
0000044
```

od

- **`od`** – Octal dump – provides a byte by byte listing of a file.
  - **`-c`** option means interpret as characters.
  - **`-o`** option means display in octal (holdover from PDP-11)
  - **`-x`** option means display in hexadecimal.
- Special characters shown
  - \012   \n – newline (borrows C notation)
  - \010   \b backspace
  - \015   \r carriage return

Tabs and Record Length

- Tabs
  - Tab stops are normally 1, 9, 17, 25, etc..
  - **`stty -tabs`** causes tabs to be replaced by spaces on older systems.
- Record length
  - There is no fixed record length in UNIX; Cr/Lf combination ends a line.
  - There is no special character indicating the end of file. The system keeps track of the number of bytes in the file.

Buffering Input

The shell buffers input, keeping track of what you type until it is exhausted

```
SIEGFRIE@panther ~$ cat
123
123
456
456
789
789
SIEGFRIE@panther ~$ cat
123123^d456^d456SIEGFRIE@panther ~$
```

- The reason that **`^d`** logs you out is that you're telling the system that there's no more input.

What’s In A File?

- A file's format is determined by the program that uses it.
- The file command tries to make a guess.

```
SIEGFRIE@panther:~$ file /bin /bin/ed
/bin:    directory
/bin/ed: ELF 64-bit LSB executable, x86-64, version
  1 (SYSV), dynamically linked (uses shared libs),
  for GNU/Linux 2.6.15, BuildID[sha1]=0x8cb1c74b71
  7b04972bff20e3005bcb8ca4bdc6d8, stripped
SIEGFRIE@panther:~$ file args.c args
args.c: ASCII C program text
args:   ELF 32-bit LSB executable, Intel 80386,
  version 1 (SYSV), dynamically linked (uses shared
  libs), for GNU/Linux 2.2.5, not stripped
```

More About **`file`**

- **`file`** makes an educated guess based on several clues, including C's use of braces and the extension ".c"/
- **`file`** reads the first couple hundred bytes looking for clues:

```
SIEGFRIE@panther:/bin$ od /bin/chmod | more
0000000 042577 043114 000402 000001 000000 000000
    000000 000000… … …

[SIEGFRIE@panther ~]$ od/bin/ed | more
0000000 042577 043114 000402 000001 000000 000000
    000000 000000… …
```

magic numbers indicating executable file

More About Files

- You can even do the following:

```
SIEGFRIE@panther ~$ od -c junk > temp
SIEGFRIE@panther ~$ ed ch2.1
36
r temp ←——reads temp into the file being edited
176
…… …
```

Why No File Structure?

- The ***advantage*** is that there are very few restrictions in terms of working with files. The ***disadvantage*** is that you can do many things that a user never wants to do by accident.
- This places file format responsibilities squarely in the hands of an applications programmer.

Directories and Filenames

- Each file's full name is unique. My file's name might be **`/home/siegfried/c/echo.c`** but when I do a directory listing, I get

```
SIEGFRIE@panther:~/c$ ls echo*
echo2.c  echo.c
```

- Every running program (or ***process***) has a working directory, inherited from its parent process. This can be changed but it leaves the parent process's directory unchanged.

Why Subdirectories?

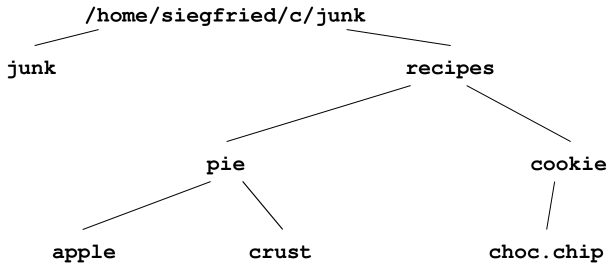
- Setting up subdirectories for files related to a common project is an extremely good idea.

```
SIEGFRIE@panther:~/c$ pwd
/home/siegfried/c
SIEGFRIE@panther:~/c$ mkdir recipes
SIEGFRIE@panther:~/c$ cd recipes
SIEGFRIE@panther:~/recipe$ pwd
/home/siegfried/c/recipes
SIEGFRIE@panther:~/recipes$ mkdir pie cookie
SIEGFRIE@panther:~/recipes$ ed pie/apple
…
wq
SIEGFRIE@panther:~/recipes$
```

What if you forgot where you put a file?

```
SIEGFRIE@panther:~/junk]$ ls
junk  recipes
SIEGFRIE@panther:~/junk$ file *
junk:    ASCII English text
recipes: directory
SIEGFRIE@panther:~/junk$ ls recipes
cookie  pie
SIEGFRIE@panther:~/junk$ ls recipes/pie
apple   crust
SIEGFRIE@panther:~/junk$
```

Directory Structure



du

- **du** – *d*isk ***u***sage – shows how many disk blocks are used by the various files.

```
SIEGFRIE@panther:~/junk$ du .
4       ./recipes/pie
4       ./recipes/cookie
12      ./recipes
20      .
SIEGFRIE@panther:~/junk$ du -a
4       ./recipes/pie
4       ./recipes/cookie
12      ./recipes
4       ./junk
20      .
SIEGFRIE@panther:~/junk$
```

du - An Example

- Combined with grep, we can look for a specific file:

```
SIEGFRIE@panther:~$ du -a | grep ch2
4       ./bbb/ch2.1
4       ./bbb/ch2.2
4       ./ch2.1
```

Directories

- Directories contain the ***inode*** number (where administrative “stuff” is located) and the directory name in the next 14 bytes. (This may not be true if the system uses long names.)
- Directories cannot be written the way other files are.

  e.g., **`who > .`**

  will not work.

Traversing Directories

```
SIEGFRIE@panther:~$ cd
SIEGFRIE@panther:~$ cd recipes
SIEGFRIE@panther:~/recipes$ cd ..; pwd      up one level
/home/siegfried
SIEGFRIE@panther:~$ cd ..; pwd         up one level
/home
SIEGFRIE@panther:/home$ cd ..; pwd     up one level
/
SIEGFRIE@panther:/$ cd ..; pwd         can’t go any higher
/
SIEGFRIE@panther:/$
```

Permissions

- You can deny access to other users (even to yourself) by specifying the permissions granted to the file’s owner, owner’s group, and eveyone else.
- If this isn’t good enough, you can encrypt files with the **`crypt`** command (and a key known only to the user).

Users

- Users are identified for the system by **`uid`** (user identification number) and **`gid`** (group identification number).

```
SIEGFRIE@panther:/$ grep oracle /etc/passwd
oracle:x:500:500:Oracle software owner:/home/oracle:
  /bin/bash

[SIEGFRIE@panther:/$ ls -l /etc/passwd
-rw-r--r--  1 root root 2056 Apr  8 20:14
  /etc/passwd
```

More About Directories

- Directories cannot be written the way other files are.

  e.g., **`who > .`**

  will not work
- ls –ld . Prints the directory listing for the current directory, not its files.

```
SIEGFRIE@panther:~$ ls -ld
drwx--x--x  16 SIEGFRIE users 4096 Sep 17 14:12 .
```

`chmod` – Change (Permission) Mode

- **`chmod -x junk`**

  takes away everyone’s execute permission
- **`chmod +r junk`**

  gives everyone’s read permission
- **`chmod u+rgo-w junk`**

  gives owner (or ***user***) read permission

  takes away group and other’s everyone’s write permission

More on **`chmod`**

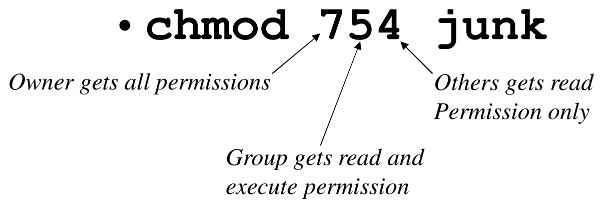
- **`u`** user or owner
- **`g`** group
- **`o`** other

- **`r`** read permission
- **`w`** write permission
- **`x`** execute permission

- **`+`** add a permission
- **`–`** takes away a permission

`chmod` Permission Codes

- We can set all of these at once using
  - **`chmod 754 junk`**



Owner gets all permissions

Group gets read and execute permission

Others gets read Permission only

`4`	read
`2`	write
`1`	execute

chmod Permission Codes

- **7** read, write, execute
- **6** read, write
- **5** read, execute
- **4** read only
- **3** write and execute
- **2** write only
- **1** execute only
- **0** no permission

- What will these do?

```
chmod 725 junk
chmod 640 junk
chmod 632 junk
chmod 751 junk
```

- Rewrite these in **ugo ±rwx** format

User Masks

- When UNIX creates a new file, it assumes the mode (permission scheme) is:

  666 (for non-executable ordinary files)

  777 (for executable files and directories)

- From this, it subtracts the user mask. To set the user mask, use the **umask** command.
- The format is **umask binaryMask**

*Contains the permissions that you are turning **off**.*

`umask` – Some Examples

- `umask 122` – turns execute off for owner, turns write off for group and others.
- `umask 023` – turns write off for group and write and execute off for others.
- `umask 065` – turns read and write off for group and read and execute off for others.
- Typing `umask` without a parameter will result in getting the mask displayed.

Relevant Stuff on Directories and `ls`

- ls –l will display a long listing.
  - If followed by a file name, it will display this information about the file.
  - If followed by a directory name, it will display information about the files in that directory.
- Examples

```
SIEGFRIE@panther:~$ ls -l bin/wc
-rwxr-xr-x  1 SIEGFRIE users 6386 Aug 19 09:26
  bin/wc
SIEGFRIE@panther:~$ ls -ld .
drwx--x--x  16 SIEGFRIE users 4096 Oct  5 15:17 .
SIEGFRIE@panther:~$ who > .
-bash: .: Is a directory
```

Inodes

- Inode is short for *information node*.
- Inodes contain most of the important information about a file including:
  - length of the file (in bytes).
  - device id (identifying the device in which the file is located).
  - user id of the file's owner.
  - group of the file (usually that of the file's owner)
  - file mode (the type of the file – regular, directory, link, device, and the permission codes.
  - times and dates of last modification and access
  - link count - the number of (hard) links pointing to the inode.
  - pointers to the disk blocks containing the file's contents.

`ls` – A Few More Examples

```
SIEGFRIE@panther:~$ date
Mon Aug  5 14:08:43 EDT 2013
SIEGFRIE@panther:~$ date >junk
SIEGFRIE@panther:~$ ls -l junk
-rw-r--r--  1 SIEGFRIE users 29 Oct  6 11:36 junk
SIEGFRIE@panther:~$ ls -lu junk
-rw-r--r--  1 SIEGFRIE users 29 Sep 17 14:11 junk
SIEGFRIE@panther:~$ ls -lc junk
-rw-r--r--  1 SIEGFRIE users 29 Oct  6 11:36 junk
SIEGFRIE@panther:~$ more junk
Tue Oct  6 11:36:25 EDT 2009
SIEGFRIE@panther:~$ ls -lu junk
-rw-r--r--  1 SIEGFRIE users 29 Oct  6 11:36 junk
```

`ls` – A Few More Examples

```
SIEGFRIE@panther:~$ chmod 444 junk
SIEGFRIE@panther:~$ ls -lu junk
-r--r--r--  1 SIEGFRIE users 29 Oct  6 11:36 junk
SIEGFRIE@panther:~$ ls -lc junk
-r--r--r--  1 SIEGFRIE users 29 Oct  6 11:37 junk
SIEGFRIE@panther:~$
```

More Examples of `ls`

```
SIEGFRIE@panther:~$ ls recipes
cookie  pie
SIEGFRIE@panther:~$ ls -lut recipes
total 8
-rw-r--r--  1 SIEGFRIE users 63 Oct  6 12:13 pie
-rw-r--r--  1 SIEGFRIE users 81 Oct  6 12:12 cookie
SIEGFRIE@panther:~$ date >x
SIEGFRIE@panther:~$ ls -i recipes
116293764 cookie  116293769 pie
SIEGFRIE@panther:~$
```

Files and Inodes

- The only connection between the file name and file's contents is the inode.
- Links are established by having the inode number and the name appearing in the same directory listing.

`ln`

- **`ln`** *`file newname`*

  establishes *`newname`* as an alternate way of accessing *`file`*.
- Example

```
SIEGFRIE@panther:~/recipes$ ln cookie oreo
SIEGFRIE@panther:~/recipes$ ls -l
total 12
-rw-r--r--  2 SIEGFRIE users 81 Oct  6 12:12 cookie
-rw-r--r--  2 SIEGFRIE users 81 Oct  6 12:12 oreo
-rw-r--r--  1 SIEGFRIE users 63 Oct  6 12:13 pie
```

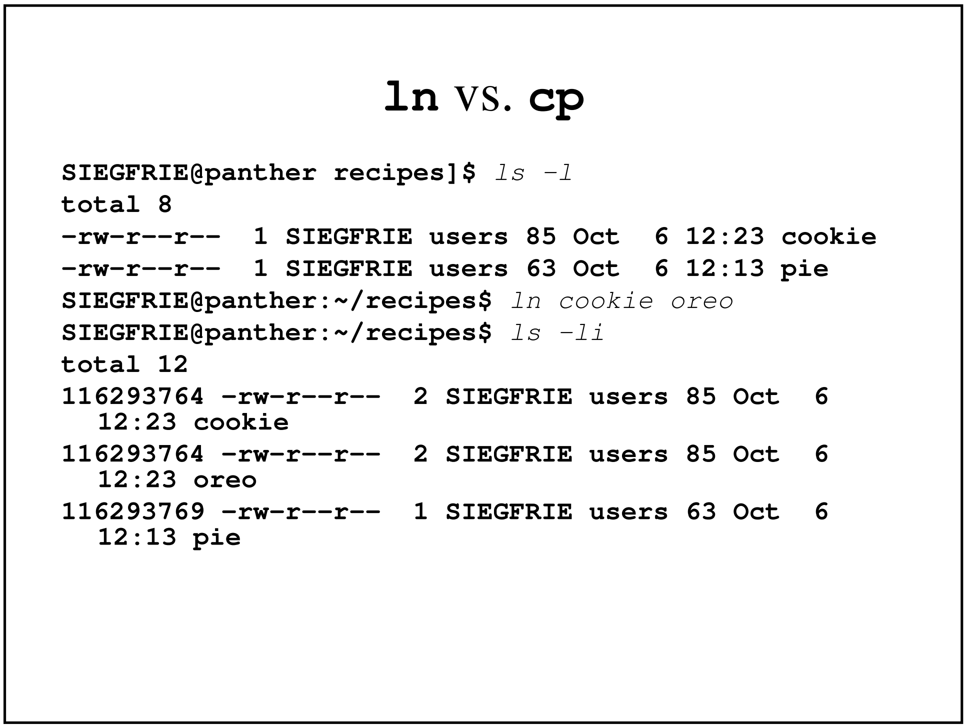
More About **ln**

- Changing the link will also change the file.

```
SIEGFRIE@panther:~/recipes$ echo pie >> oreo
SIEGFRIE@panther:~/recipes$ ls -l
total 12
-rw-r--r--  2 SIEGFRIE users 85 Oct  6 12:23 cookie
-rw-r--r--  2 SIEGFRIE users 85 Oct  6 12:23 oreo
-rw-r--r--  1 SIEGFRIE users 63 Oct  6 12:13 pie
SIEGFRIE@panther:~/recipes]$ rm oreo
SIEGFRIE@panther:~/recipes]$ ls -l
total 8
-rw-r--r--  1 SIEGFRIE users 85 Oct  6 12:23 cookie
-rw-r--r--  1 SIEGFRIE users 63 Oct  6 12:13 pie
SIEGFRIE@panther recipes]$
```

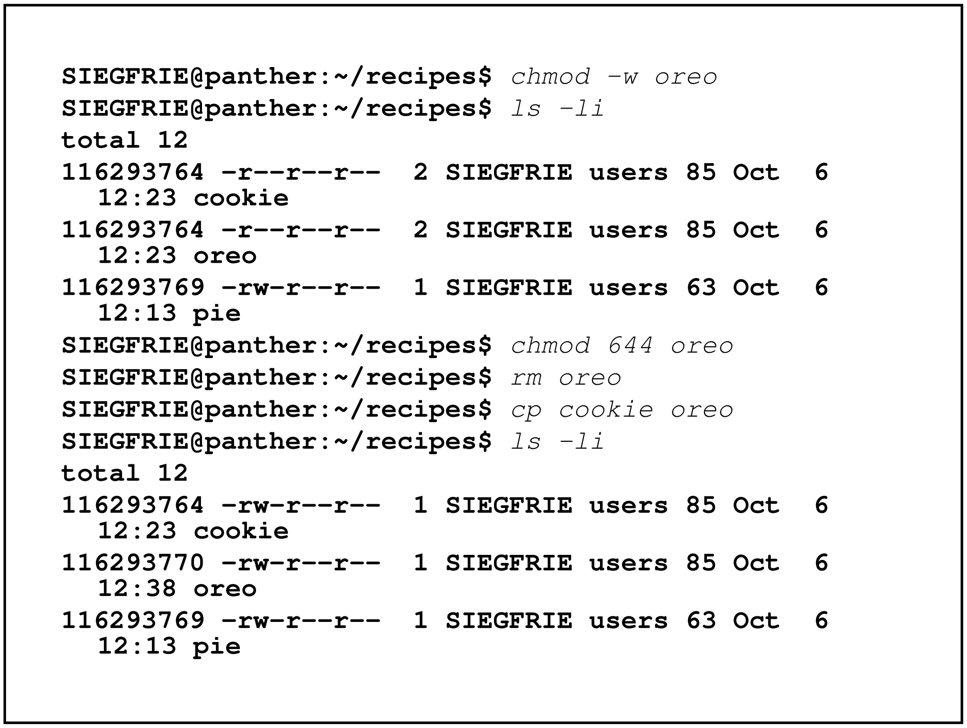
How the Basic File Commands Work

cp	UNIX creates a brand new file with its own inode number.
mv	The filename changes, it remains in its current directory and has its entry moved to the new directory, but inode number remains the same.
ln	A new directory entry is created with the existing file's inode number.
rm or **rmdir**	The directory entry is deleted. If there are no other links, the inode is deleted, too.



ln vs. **cp**

```
SIEGFRIE@panther recipes]$ ls -l
total 8
-rw-r--r--  1 SIEGFRIE users 85 Oct  6 12:23 cookie
-rw-r--r--  1 SIEGFRIE users 63 Oct  6 12:13 pie
SIEGFRIE@panther:~/recipes$ ln cookie oreo
SIEGFRIE@panther:~/recipes$ ls -li
total 12
116293764 -rw-r--r--  2 SIEGFRIE users 85 Oct  6
  12:23 cookie
116293764 -rw-r--r--  2 SIEGFRIE users 85 Oct  6
  12:23 oreo
116293769 -rw-r--r--  1 SIEGFRIE users 63 Oct  6
  12:13 pie
```



```
SIEGFRIE@panther:~/recipes$ chmod -w oreo
SIEGFRIE@panther:~/recipes$ ls -li
total 12
116293764 -r--r--r--  2 SIEGFRIE users 85 Oct  6
  12:23 cookie
116293764 -r--r--r--  2 SIEGFRIE users 85 Oct  6
  12:23 oreo
116293769 -rw-r--r--  1 SIEGFRIE users 63 Oct  6
  12:13 pie
SIEGFRIE@panther:~/recipes$ chmod 644 oreo
SIEGFRIE@panther:~/recipes$ rm oreo
SIEGFRIE@panther:~/recipes$ cp cookie oreo
SIEGFRIE@panther:~/recipes$ ls -li
total 12
116293764 -rw-r--r--  1 SIEGFRIE users 85 Oct  6
  12:23 cookie
116293770 -rw-r--r--  1 SIEGFRIE users 85 Oct  6
  12:38 oreo
116293769 -rw-r--r--  1 SIEGFRIE users 63 Oct  6
  12:13 pie
```

```
SIEGFRIE@panther:~/recipes$ chmod -w oreo
SIEGFRIE@panther:~/recipes$ ls -li
total 12
116293764 -rw-r--r--  1 SIEGFRIE users 85 Oct  6
  12:23 cookie
116293770 -r--r--r--  1 SIEGFRIE users 85 Oct  6
  12:38 oreo
116293769 -rw-r--r--  1 SIEGFRIE users 63 Oct  6
  12:13 pie
SIEGFRIE@panther:~/recipes$ rm oreo
rm: remove write-protected regular file `oreo'? u
SIEGFRIE@panther:~/recipes$ rm oreo
rm: remove write-protected regular file `oreo'? y
SIEGFRIE@panther:~/recipes$
```

The Directory Hierarchy

- UNIX (and Linux) uses a hierarchical (tree- structure) directory system, the top level being **/**.

```
SIEGFRIE@panther:~$ ls /
bin   dev   initrd   media  opt  sbin  sys  tmp  usr
boot  etc   lib      misc   proc selinux   test u01  var
delete_this home  lost+found mnt root srv  tftpboot users
[SIEGFRIE@panther:~$
```

Interesting Directories

`/`	Root of the file system
`/bin`	Essential programs in binary or executable form
`/dev`	Device files
`/etc`	Administative files and miscellany
`/etc/motd`	Login message of the day
`/etc/passwd`	Password file (*no longer visible*)
`/lib`	Essential libraries for compilers, etc.
`/tmp`	Temporary files
`/unix`	Kernel

More Interesting Directories

`/usr`	Users' stuff
`/usr/adm`	System administrator's (sysadmin) stuff
`/usr/bin`	User binaries (executable files)
`/usr/games`	Games
`/usr/include`	Include header files for C compiler
`/usr/man`	Online manual
`/usr/src`	Source code for utilities
`/usr/spool`	Spool file directories
`/usr/spool/mail`	spool directories for mail

Devices

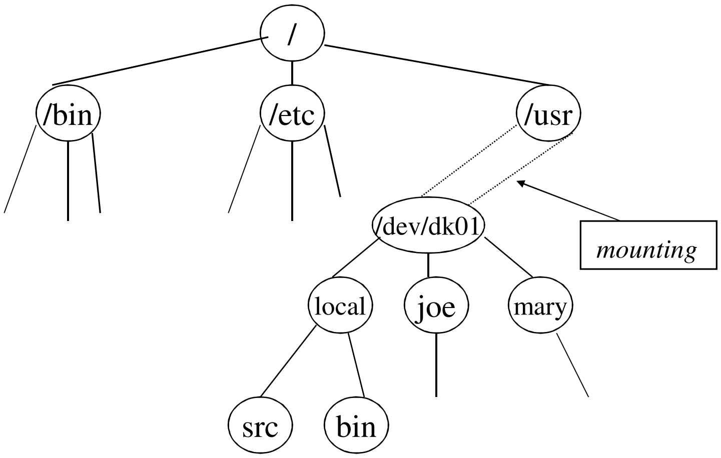
- Devices are treated like other files to a large extent.
- Peripheral will have file names like **`/dev/mt0`** or **`/dev/tty01`** and can be used in the manner of other files on many systems.
- You can write **`cp /dev/mt01 junk`** because a file is just a pattern of bytes.

```
[SIEGFRIE@panther:~$ ls -l /dev
total 0
crw-------  1 root root    36,   8 Apr  8 16:23 arpd
lrwxrwxrwx  1 root root          3 Apr  8 20:24 cdrom -> hda
crw-------  1 root root     5,   1 Apr  9 04:04 console
... ...
brw-r-----  1 root root   253,   8 Apr  8 16:24 dm-8
crw-------  1 root root    36,  14 Apr  8 16:23 dnrtmsg
crw-------  1 root root    13,  64 Apr  8 16:23 event0
brw-rw----  1 root floppy   2,  44 Apr  8 20:24 fd0u1680
brw-rw----  1 root floppy   2,  60 Apr  8 20:24 fd0u1722
... ...
lrwxrwxrwx  1 root root         15 Apr  8 20:24 stderr ->
   /proc/self/fd/2
lrwxrwxrwx  1 root root         15 Apr  8 20:24 stdin ->
   /proc/self/fd/0
lrwxrwxrwx  1 root root         15 Apr  8 20:24 stdout ->
   /proc/self/fd/1
lrwxrwxrwx  1 root root          4 Apr  8 20:24 systty ->
   tty0
crw-rw----  1 root tty      4,  10 Apr  8 16:23 tty10
crw-rw----  1 root tty      4,  11 Apr  8 16:23 tty11
crw-rw----  1 root tty      4,  12 Apr  8 16:23 tty12
... ...
```

Major and Minor Device Codes

- Each directory listing for a device contains two numbers:
  - – Major device code – indicates the type of device.
  - – Minor device code – indicates which device of that type.

File-System Mounting



Some Useful Things

```
SIEGFRIE@panther:~$ who am i
SIEGFRIE pts/4  Oct  6 13:23 (pool-… … .verizon.net)
SIEGFRIE@panther:~$ tty
/dev/pts/4
SIEGFRIE@panther:~$ ls -l /dev/pts/4
crw--w----  1 SIEGFRIE tty 136, 4 Oct  6 13:47 /dev/pts/4
SIEGFRIE@panther:~$ date > /dev/pts/4
Tue Oct  6 13:48:03 EDT 2009
SIEGFRIE@panther:~$ mesg n
SIEGFRIE@panther:~$ ls -l /dev/pts/4
crw-------  1 SIEGFRIE tty 136, 4 Oct  6 13:48 /dev/pts/4
SIEGFRIE@panther ~$ mesg y
SIEGFRIE@panther:~$ ls -l /dev/pts/4
crw--w----  1 SIEGFRIE tty 136, 4 Oct  6 13:48 /dev/pts/4
SIEGFRIE@panther:~$
```

- Many programs uses this when `stdin` and `stdout` are redirected.

`chown` and `chgrp`

- `chown` changes the owner of the file as well as the group(if the option is chosen.
  - Only the superuser (`root`) can do this in Linux.
- `chgrp` can change group to which the file belongs.

`find`

- **`find`** allows the user to search through a list of directories for files that meet a particular criterion.
- The general form is

```
find directory -criteria
```

`find` - criteria

- Criteria include:

`-inum` *`N`*	with inode number *`N`*
`-links` *`N`*	with *`N`* links
`-name` *`pattern`*	with name matching *`pattern`*
`-newer` *`file`*	newer than *`file`*
`-user` *`name`*	owned by *`name`*

find - Examples

```
SIEGFRIE@panther:~$ find -name alloc.c
./c/alloc.c
SIEGFRIE@panther:~/junk$ find -user SIEGFRIE
.
./morejunk
…….
./cookie
SIEGFRIE@panther:~/junk$ ls -i
64881438 411     64905148 CSC 390.7z  64881441 file1
64881443 ls.out  … 64881446 phone-book
SIEGFRIE@panther:~/junk$ find -inum 64881438
./411
```

whereis

- `whereis` searches for a command and will list where its executable or source file is or where its online manual page is located, searching in the standard places that Linux places these files.
- It lists the absolute path for the binary, source or manual files with names given as command parameters.
- Options
  - **-b** lists only binary (or executable) files
  - **-m** lists only manual pages
  - **-s** lists source files only.

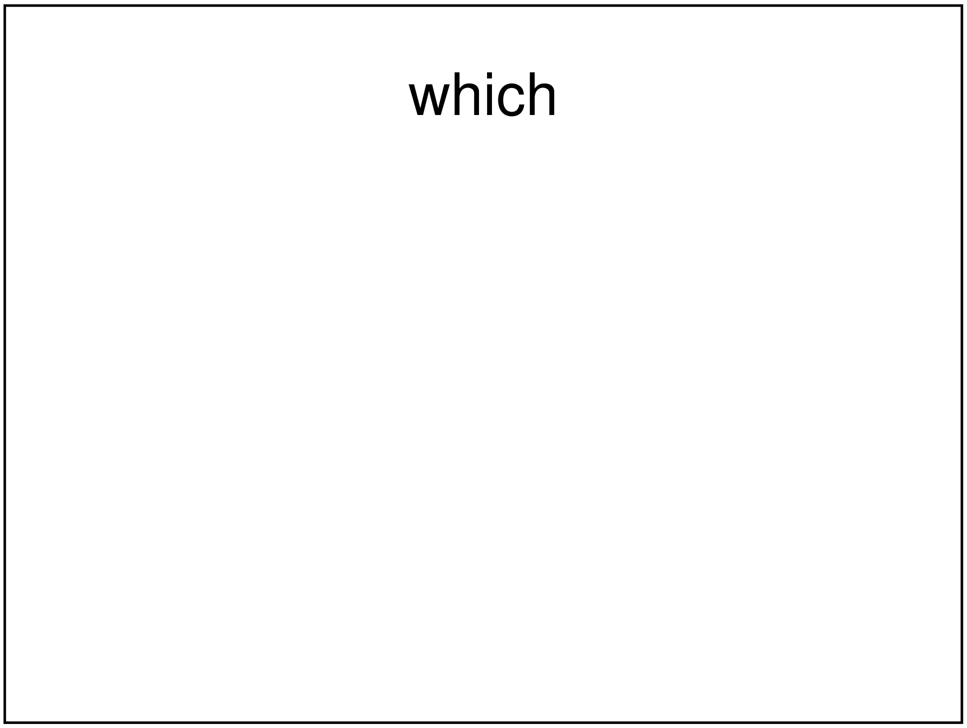
`whereis` - Examples

```
SIEGFRIE@panther:~$ whereis ftp
ftp: /usr/bin/ftp /usr/bin/X11/ftp
/usr/share/man/man1/ftp.1.gz
SIEGFRIE@panther:~$ whereis -b ftp
ftp: /usr/bin/ftp /usr/bin/X11/ftp
SIEGFRIE@panther:~$ whereis cat
cat: /bin/cat /usr/share/man/man1/cat.1.gz
SIEGFRIE@panther:~$ whereis -s cat
cat:
SIEGFRIE@panther:~$ whereis -m cat
cat: /usr/share/man/man1/cat.1.gz
SIEGFRIE@panther:~$ whereis ls ln
ls: /bin/ls /usr/share/man/man1/ls.1.gz
ln: /bin/ln /usr/share/man/man1/ln.1.gz
SIEGFRIE@panther:~$
```

`which`

- **`which`** returns the path of the commands that appear on the command line.
- The files that appear are in the current path of the user.
- Example

```
SIEGFRIE@panther:~/bin$ which cat
/bin/cat
SIEGFRIE@panther:~/bin$ which ls
/bin/ls
SIEGFRIE@panther:~/bin$ which f77
/usr/bin/f77
SIEGFRIE@panther:~/bin$
```



which