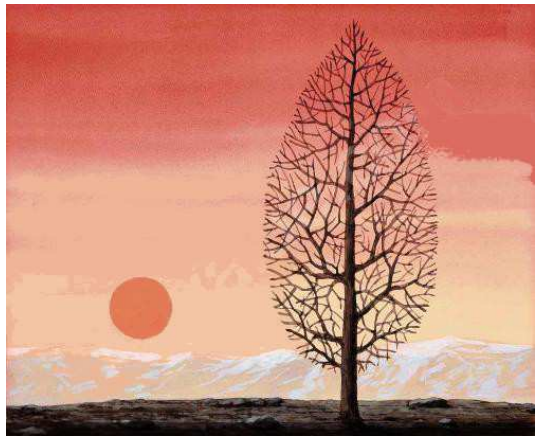
Files, Directories and Inodes



Files (1)

- abstraction from hardware (Device, Head, Cylinder, Sector)
- call file by a *file name*

Files (2)

programs need a *reference* to a file

they cannot use the file name on each write operation

- system level: *file descriptor* = `int`
- C library level: *file pointer* = `FILE *`
- C++ library level: *file stream class* = `fstream`
- JAVA : *file stream class* = `FileReader/Writer`

Text Files

all UNIX tools understand plain text files in ASCII encoding

only contains printable characters, see isprint(3), iswprint(3)

separator is *newline*

usually last line is terminated by a *newline*
otherwise some tools may fail or give unexpected results

the command `wc -l` counts *newline* characters

What *is* a *newline*?

seems like a dull question ...

This is a *newline* (1)

depends on the operating system

- **LF** UNIX–family:
  GNU/Linux, BSD, Solaris,...,Mac OS X, BeOS, Amiga, RISC OS, ...
- **CR+LF** DOS–family:
  DEC RT-11, CP/M, MP/M, DOS, OS/2, Microsoft Windows
- **CR** Apple–family:
  Commodore machines, Apple II family and Mac OS up to version 9

[Wikipedia: Newline]

This is a *newline* (2)

depends on the character set

for example Unicode:

```
LF:    Line Feed, U+000A
CR:    Carriage Return, U+000D
CR+LF: CR followed by LF, U+000D followed by U+000A
NEL:   Next Line, U+0085
FF:    Form Feed, U+000C
LS:    Line Separator, U+2028
PS:    Paragraph Separator, U+2029
```


iconv: Converting between Encodings

```
iconv -f utf-8 -t iso8859-1 file.utf8 >file.latin1
```

- `-f` encoding *from*
- `-t` encoding *to*
- `-c` discard encoding errors

Standard File Descriptors

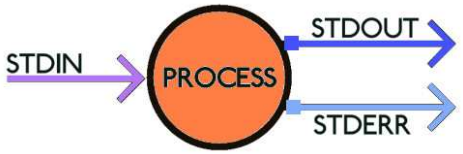
By default, a process has three open file descriptors.

Descriptor	FILE *	meaning	device
0	stdin	standard input	keyboard
1	stdout	standard output	terminal
2	stderr	standard error	terminal

These can be redirected on the shell:

<	redirects the standard input
>	redirects the standard output
>>	redirects the standard output, appends to file
2>	redirects the standard error

Processes and Stdin/Stdout (default)



processes

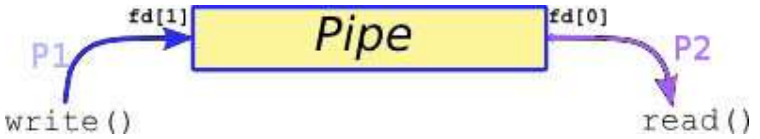
- read from `stdin`
- write to `stdout` *and* `stderr`

Processes and Stdin/Stdout

example: sort reads stdin, writes stdout, EOF is [Strg-d]:

```
$ sort
zhn
rfa
qqm
[press Ctrl-d aka Strg-d]
qqm
rfa
zhn
```


Pipes: Connect Output of Process 1 to Input of Process 2



Syntax:

```
P1 | P2
```

Pipes: Example

```
$ cat german_soccer
Bayer 04 Leverkusen;56
Borussia Dortmund;64
Eintracht Frankfurt;46
FC Bayern München;84
FC Schalke 04;49
Sport-Club Freiburg;45
$ sort -t";" -k2 -n <german_soccer
Sport-Club Freiburg;45
Eintracht Frankfurt;46
FC Schalke 04;49
Bayer 04 Leverkusen;56
Borussia Dortmund;64
FC Bayern München;84
```


Pipes: Example (continued)

```
$ sort -t";" -k2 -n <german_soccer  | cut -d";" -f1
Sport-Club Freiburg
Eintracht Frankfurt
FC Schalke 04
Bayer 04 Leverkusen
Borussia Dortmund
FC Bayern München
```

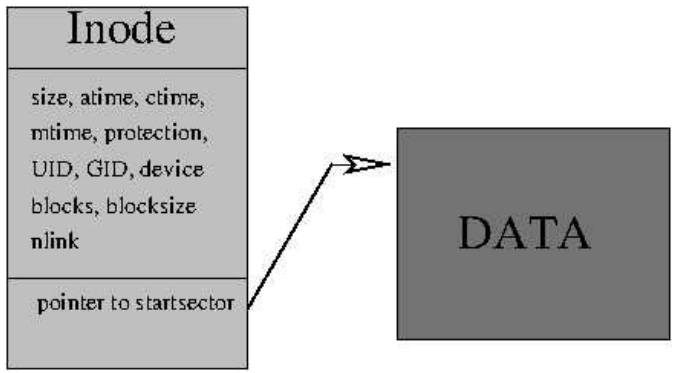

Pipes: xargs transforms stdin to command line args

delete all links (simple version, see manual find(1))

example:

```
find . -type l | xargs rm
```


The Inode



Inode Contents

contains administrative data of a file

can be read by the `stat()` system call

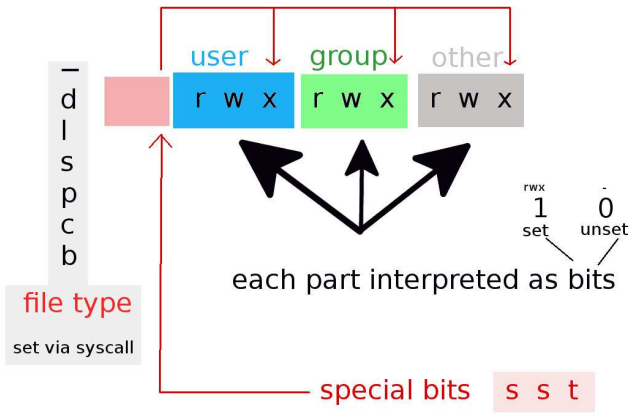

```
struct stat {
    ino_t         st_ino;      /* inode */
    dev_t         st_dev;      /* device of this file */
    mode_t        st_mode;     /* protection + file type */
    nlink_t       st_nlink;    /* number of hard links */
    uid_t         st_uid;      /* user ID of owner */
    gid_t         st_gid;      /* group ID of owner */
    dev_t         st_rdev;     /* device type (if inode device) */
    off_t         st_size;     /* total size, in bytes */
    blksize_t     st_blksize;  /* blocksize for filesystem I/O */
    blkcnt_t      st_blocks;   /* number of blocks allocated */
    time_t        st_atime;    /* time of last access */
    time_t        st_mtime;    /* time of last modification */
    time_t        st_ctime;    /* time of last change */
};
```


The File Type

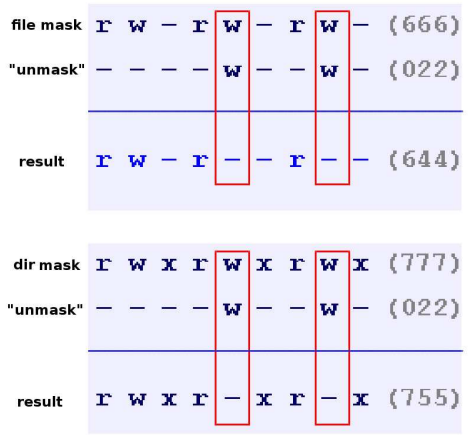
. . . determines which kind of file it is . . .

type	description	
regular file	,,normal" file with user data or a program	-
directory	contains file names pointing to their inodes	d
link	points to another file	l
socket	UNIX domain socket, process communication	s
pipe	process communication	p
block device	device handling data in blocks	b
character device	device handling data char by char	c

The Protection Bits



Default Protection of a New File (umask)



Setting Protection Bits (chmod)

```
$ ls -l mytestfile
-rw-------    1 dw       users          12 Apr  6 10:53 mytestfile


$ chmod 640 mytestfile


$ ls -l mytestfile
-rw-r-----    1 dw       users          12 Apr  6 10:53 mytestfile
```


Meaning on directories (1)

read a directory: read the file names in it

```
$ ls -ld testdir
dr--------    2 dw       users        1024 Apr  6 11:41 testdir
$ ls -l testdir
/bin/ls: testdir/testfile1: Permission denied
/bin/ls: testdir/testfile2: Permission denied
total 0
$ cd testdir
bash: cd: testdir: Permission denied
$ cat >testdir/testfile3
bash: testdir/testfile3: Permission denied
```


Meaning on directories (2)

execute bit to a directory: search path allowed

```
$ ls -ld testdir
d--x------    2 dw       users        1024 Apr  6 11:41 testdir
$ cd testdir
$ cat testfile1
0000000000000000000
1111111111111111111
2222222222222222222
3333333333333333333
```

Meaning on directories (3)

write to a directory: creating/removing files allowed

makes sense only if *execute bit* is set

```
$ ls -ld testdir
drwx------    2 dw       users        1024 Apr  6 11:41 testdir
$ cd testdir
$ ls -l
total 1
-rw-r--r--    1 dw       users          80 Apr  6 11:47 testfile1
-rw-r--r--    1 dw       users           0 Apr  6 11:41 testfile2
$ rm testfile2
$ ls -l
total 1
-rw-r--r--    1 dw       users          80 Apr  6 11:47 testfile1
```


Special Protection Bits Needed

in a world writable directory like `/tmp`

everybody may remove anybody's files

> STICKY DIRECTORIES
> When the sticky bit is set on a directory, files in that directory may be unlinked or renamed only by root or their owner. Without the sticky bit, anyone able to write to the directory can delete or rename files. The sticky bit is commonly found on directories, such as /tmp, that are world-writable.

Special Protection Bit: Sticky Bit

The sticky bit is set by a leading 1 in the `chmod` calls.

Example:

```
$ chmod 1777 testdir
$ ls -ld testdir
drwxrwxrwt    2 dw       users        1024 Apr  6 11:50 testdir
```


Special Protection Bit: Setuid Bit (1)

the setuid mechanism is needed for controlled access to

- parts of a file
- a file by a certain set of users

and must be logically controlled by a process

Special Protection Bit: Setuid Bit (2)

example: the file `/etc/master.passwd` on BSD contains encrypted user passwords

```
-rw-------  root  wheel  /etc/master.passwd
```

(`/etc/shadow` on Linux)

```
root:$1$TeLs7PIX$ebgD6bh573GWHN12Aaut5/:
      0:0::0:0:root:/root:/bin/csh
sysi40:$1$dIQvGCrn$f46M9fNfWTmOVsyfQEwdu0:2040:1000:
      :0:0:........:/home/sysi40:/usr/local/bin/bash
```

`sysi40` wants to change his password

⇝needs write access to `/etc/master.passwd`

but if he had write access he could change root's password too

Special Protection Bit: Setuid Bit (3)

Solution: processes have two User–IDs

- RUID: real User–ID – who starts the process
- EUID: effective User–ID – decides about file access

normally RUID==EUID

> If the Setuid Bit is set on a program, the EUID is the UID of the program owner.

With the `ps` command, we see the EUID.

Special Protection Bit: Setuid Bit (4)

```
-r-sr-xr-x  root  wheel /usr/bin/passwd

-rw-------  root  wheel /etc/master.passwd
```

- `/etc/master.passwd` owner root, protection `rw-------`
- program `passwd` can change one line in `/etc/master.passwd`
- program `passwd` is owned by root
- program `passwd` has the setuid bit
- sysi40 starts program `passwd`
- process `passwd` has RUID=sysi40, EUID=root

⇝ for this process the `rw-` part counts

Special Protection Bit: Setuid Bit (5)

The setuid bit is set by a leading 4 in the `chmod` calls.

Example:

```
$ chmod 755 my_program
$ ls -l my_program
-rwxr-xr-x    1 dw       users      255996 Jan 17 18:38 my_program
$ chmod 4755 my_program
$ ls -l my_program
-rwsr-xr-x    1 dw       users      255996 Jan 17 18:38 my_program
```

Special Protection Bit: Setgid Bit

The same mechanism holds for the GID field of the protection.

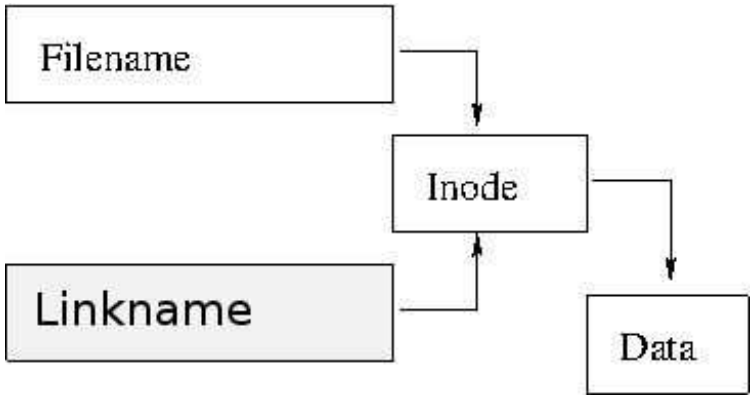
We have

- RGID the real GID
- EGID the effective GID

The setgid bit is set by a leading 2 in the chmod calls.

```
$ chmod 2755 my_other_program
$ ls -l my_other_program
-rwxr-sr-x    1 dw       users     1723139 Feb 15 15:11 my_other_program
```

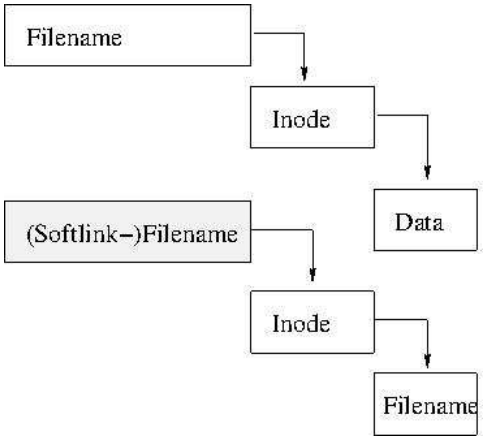

Hardlink Count: Hardlinks



Hardlink count is a reference counter.

The inode gets removed if the hardlink count becomes zero.

Symbolic Links: Softlinks



The file has no idea about whether a softlink points to it.

Hardlinks and Softlinks

	Hardlinks	Softlinks
command	ln	ln –s
performance	faster	slower
have a file type	no	yes
reach across filesystems	no	yes
to directories	no	yes
distinguish original/link	no	yes
traceable	no	yes
may be broken	no	yes

In general, commands and system calls follow softlinks, but there are exceptions: lstat(), tar,...

Hardlinks on Directories

.	points to current directory
..	points to parent directory

created via mkdir / removed via rmdir

there is no other way to create hard links (even for root)

reasons:

- would break acyclic structure (disk usage utilities etc)
- would create two or more parent directories

Inode Timestamps

Inode Timestamps:

- atime (access time): time of last access, e.g.
  `functions: read(), execve(), mknod(), pipe(), utime()`
- ctime (change time): time of last change of inode info
  `functions: chown(), chgrp(), chmod(), ...`
- mtime (modification time): time of last change of files's data
  `write(), mknod(), truncate(), utime()`

command that affects timestamps: `touch`