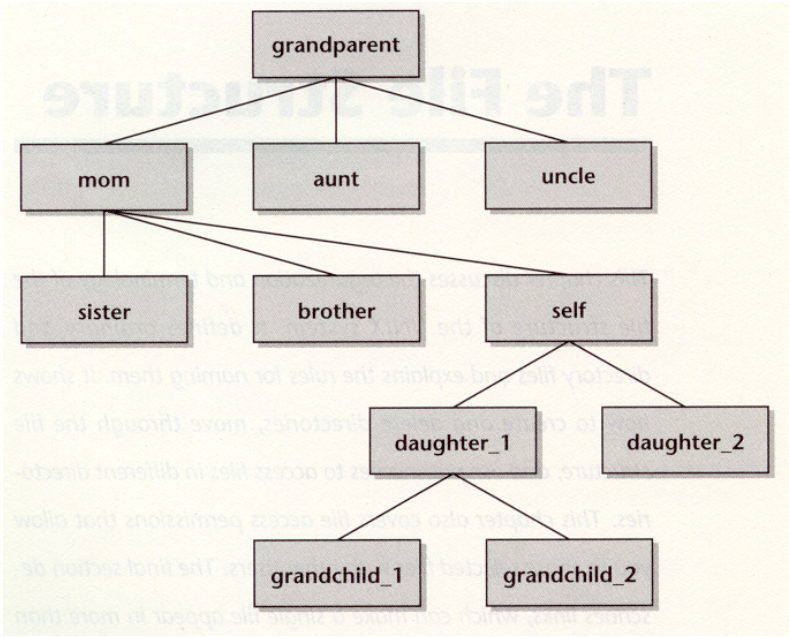
# 3 - File System – mostly old Unix notes

- *no harm to know a bit about history & Unix, as traditions persist*
- Unix provides a *hierarchical, or tree-like,* file system
  - Supports two main objects
    - Files
    - Directory files, or directories
  - From Unix's view, there is no difference between the two
- From the user's view, files are data that you want, directories are folders of files
  - A directory is just a file, with a list of files with indexes to their locations (via inodes (hard link) or alias (soft links)
  - Directories cannot be edited directly, but only by filesystem cmds e.g. mkdir, rmdir, rm, cp, mv etc – **ma**k**e**, r**e**move, c**o**py, **m**o**v**e
- In reality, files are simply a stream of bytes
  - Critical design philosophy of Unix
  - Everything in Unix is a file = string of bytes: files, devices,.etc

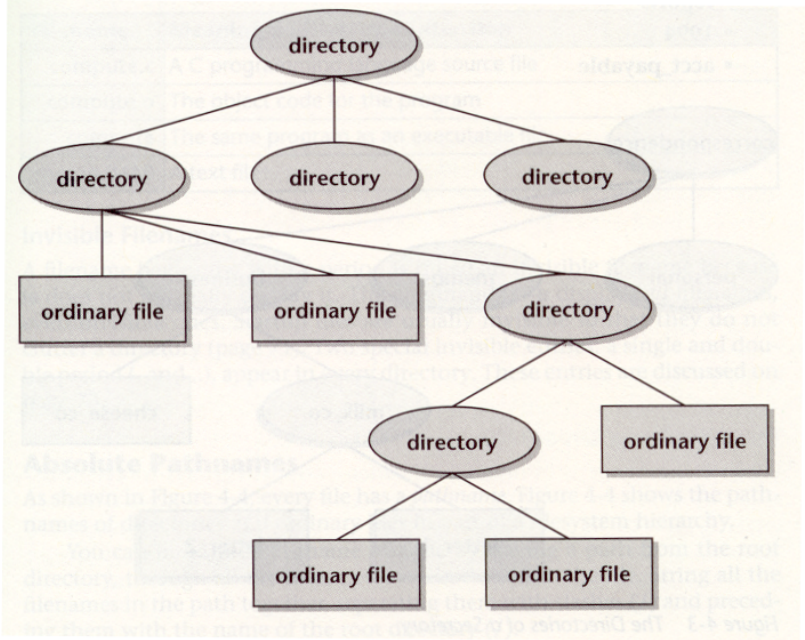


# Tree Structure






# Files and Directories






# Linux File Hierarchy – some directories for example not retention!

/ – root. The source of everything, so only one root!
- Don't confuse with /root which is root user's home directory

/bin – where binary commands and executables are for all
/sbin – executables generally of interest only to the super user
- Some variants combine both above for ease, but pros & cons depending

/home – where your "home directories" are
/tmp – for temporary files used by system, frequently wiped
/var – system logs and other stuff – frequently updated
/usr – laid out like /, but with less system-critical stuff
(don't confuse with /users, which has user directories)
/dev – has files that represent different devices on the system
/proc – has runtime system info (try: cat /proc/cpuinfo)
/opt - stores optional apps, not installed by default e.g. google
/snap – modern distro-agnostic apps, packaged with all it needs!

# File Names

- Every file has a *filename*
- Current versions of Unix allow up to 255 characters, older versions limit to 14
- Almost any character can be used in a filename
  - Sticking with the following will save many problems
    - Uppercase letters (A-Z)
    - Lowercase letters (a-z)
    - Numbers (0-9)
    - Underscore (_)
  - Period ( **.** ) **-** but
    - can be confusing as also used as any other character,
    - may denote special files,
      - such as hidden system (e.g. profile) files which begin with .
      - only displayed if the *'-a'* option is used with *ls*
  - Reserved Naming Exception is the *root* directory, which is always named /
    - No other file can use this name
    - '/' are also used to denote successive subdirectories in a pathname. (Perversely, Microsoft uses a '\' in DOS pathnames! )



# Other File Name Rules

- Avoid punctuation marks & SPACE (underscore is OK)
  - they complicate parsing & presentation of filenames**
- Other characters may be used by "escaping" them with a "\" character
  - Back*Slash - displays as Back*Slash
  - Much confusion can result from this **
- Unix/Linux File names are case sensitive
  - my_file, My_File, and MY_FILE are all unique names

** Of course other characters such as comma ',' and dash '-' can be used, but they introduce unintentional non-standard parsing issues when processing filenames…in that they are treated as a separator, effectively ending the word, and require overriding by 'escaping' them with a backslash…

**BOTTOM LINE** best to use only alphanumeric and underscores in filenames.



# Access Controls Provide Security

- Generally accepted that there are files users cannot access or modify
- This provides security for:
  - System
  - Work groups
  - Individual users



# File Access Controls

- Unix provides file security with *access levels* and *permissions*
- *Access levels* define who specific *permissions* belong to
  - user (u)
  - group (g)
  - other (o)
- *Permissions* define what the members of an *access level* are allowed to do
  - read (r)
  - write (w)
  - execute (x)

# Groups and Permissions - 1

**Groups**

In Unix, all users belong to at least one group.
Each group member has similar access to the filesystem.

There are 3 categories of user:-
the **u**ser,
the **g**roup to which the user belongs
and **w**orld/**o**thers – everyone else on the system.

**Permissions**

Files and directories all have associated "permissions" .
Permissions tell the OS who can do what with your files and directories.
The permissions are:

**r**ead, **w**rite, e**x**ecute

# Groups and Permissions - 2

To see the permissions on a file, do a 'ls –l'

```
urmac:/urcwd urid$ ls -l
-r--r--r--  1 urid  urgroup  17375 Apr 26 2000  rgb.txt
-rw-r--r--  1 urid  urgroup  17375 Apr  5 02:57 set10.csv
drwxr-xr--  1 urid  urgroup   1024 Jan 19 19:39 tests

mode      #links urid  urgroup  blocks last modified name
```

Changing Permissions
chmod [ugo]+[rxw] <filename>
e.g. chmod u+w rgb.txt - gives me permission to change rgb.txt

Changing Ownership - can only be done by admin/superuser
chown <user>.<group> <filename>
e.g. chown urid.iuns rgb.txt - makes rgb.txt owned by iuns group

# Groups and Permissions - 3

```
-r--r--r--  1 urid  urgroup  17375 Apr 26 2000  rgb.txt
-rw-r--r--  1 urid  urgroup  17375 Apr  5 02:57 set10.csv
drwxr-xr--  1 urid  urgroup   1024 Jan 19 19:39 tests
```

user group other

What they do:
files: read – Allows you to read the file
write – Allows you to modify the file
execute – Allows you run the file as a script or binary program
(***!!!Omission is common reason code won't run!!!!***)

directories:
read – lets you get a directory listing of files
write – lets you add or remove files from directory
execute – lets you access files in the directory

# Print Working Directory - pwd

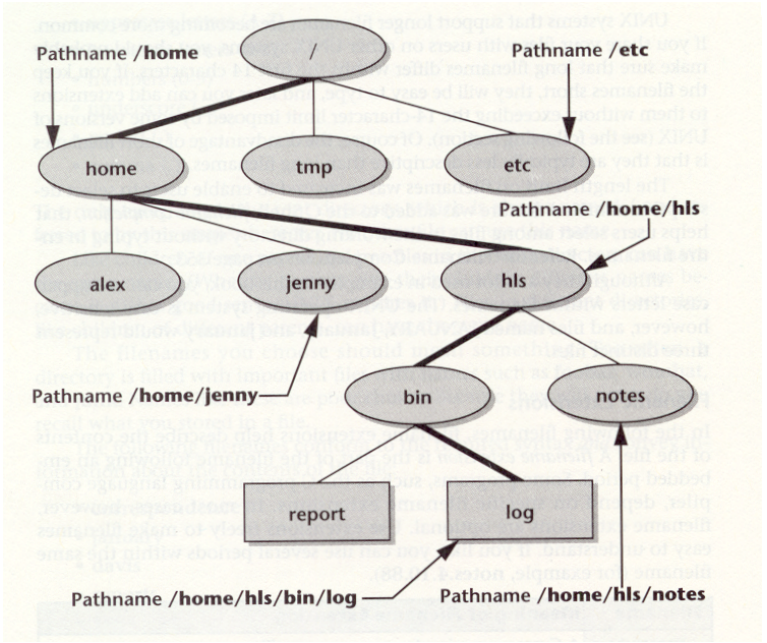
- Display pathname of current working directory
- Syntax: *pwd*

# Absolute Pathnames

- Every file has a pathname
- A pathname is built by tracing a path from the root directory, through all intermediate directories, to the file
- String all the directory filenames in the path together, separating them with slashes ( / ) and preceeding them with the root directory ( / )
- Example: /usr/src/cmd/date.c



# Path Names






# Relative Pathnames

- Shortcuts
  - **.** (this directory)
  - **..** (the parent directory)
  - ~ (your home user directory)
- These make using *relative pathnames* easy



# Filename Completion in the C & bash Shells

- Only works within current directory!
  - extends to sub-directories with use of '/' subdirectory separator
- Just type first part of name and press TAB, (in some older C shells, ESC is used instead of TAB)
- Example:

  urmac%*ls*

  jerry john joe jimmy wesley tom

  bill willie ken ted shirley joan

  urmac%more jeTAB *shell types rest of name: all if unique; else*

  urmac%more jerry *common stem; TAB again name list of stem*
- shell variables may need to be set to effect autocompletion
  - e.g. *csh* will complete file names for you, if the *filec* *set filec*

# Filename Completion in bash shell

- If the what is typed before TAB is not unique to any file in the current directory, the shell doesn't know which file you mean, so it completes as much as it can and then
  - C shell: shows list of filenames starting with string stem
  - Bash : beeps, & variants do as C shell above

- Either way continue typing more characters to indicate what file you want, hit TAB again and
  - Either the filename will be completed
  - Or it will go as far as it can and repeat first stage above



# Filename Completion Examples

Assume % is the command line prompt here:

urmac%*ls*

| | | | | | |
|---|---|---|---|---|---|
| jerry | john | joe | jimmy | wesley | tom |
| bill | willie | ken | ted | shirley | joan |

urmac%*cat jo*TAB

cat johTAB … bash shell just beeps and you finish

cat john … but on extra TAB behaves as below

urmac%*cat jo*TAB … C shell lists options and you finish

john joe joan



# Review of some common commands

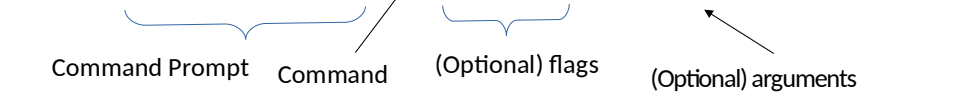
Aims

- To give a little more depth to commands after first pass intro
- An appreciation of the range and complexity of options available
  - Not to confuse
  - Nor to expect you to remember
    - – I don't, so don't expect it from you
      - That's why there's a manual/info utilities
      - Clearly use it or lose it; practice helps, if needed!
  - But to avoid you reinventing the wheel …
    - No need to recode, when most common needs are already developed over years
    - So check the flags before flogging yourself!

# The Command Prompt

- Commands are the way to "do things" in Unix consisting of a command name with options called "flags" and arguments
- Commands are typed at the *command prompt*
- In Unix, *everything* (including commands) is case-sensitive

```
[prompt]$ <command> <flags> <args>
Cs1>:/abc123/~$ ls -l -a directory1
```




**Note**:
1) In Linux/Unix, system trusts you know what you're doing, so no checks!
2) Many commands will alert only if there was an error in it's execution; but could almost destroy your system without telling you! Careful!
3) Optional flags can be combined into a single contiguous continuous string preceeded by a single dash '-' rather than being listed separately… So ls -l -a … can be replaced by ls -la … or ls -al … order is unimportant

# LiSt directory – ls List the contents of a directory

NB everything within square brackets is optional, but must be preceded with a -, indicating flag field!

Syntax: ls [-aAcCdfFgilLqrRstu1] filename

- a - List all entries, without this, hidden system files beginning with a '**.**' are not listed
- c - Sort on time of last edit (c for changed)
- l - List in "long" form, giving mode, number of links, owner, size, and time of last modification
- 1 – as 'l', with no header line, a line per file – best for scripts
- R (UPPER CASE)- Recursively list subdirectories encountered
- r (lower case) – reverse the default sort order from other flags
- s - Give size of file in kilobytes
- t - Sort by time modified, latest first



# Change Directory - cd

- Change current working directory
- Syntax: *cd [directory]*
  - Without argument, changes to your login directory
  - Otherwise changes to *directory* specified



# MaKe DIRectory - mkdir

- Creates new directories
- Requires write permission in the parent directory
- Syntax: *mkdir [ -p ] pathname*
  - *p* - allows "missing" parent directories to be created as needed in the pathname
    - e.g. if currently in ~/rubb
      and issue mkdir -p ./bags/morebags
      it will generate bags within rubb as well as morebags
    - Useful in scripts, which may fail if parent directories missing!!!!
- BUT if in directory ~/rubb and issue
  - **mkdir .bags**
    - will make a hidden directory .bags
      which will be unseen unless ls -a is used
  - **mkdir ./bags/morebags**
    - will fail with no such file or directory message
      - *Which is the reason for the -p flag*



# ReMove file(s) - rm

- Removes (deletes) files
- Syntax: *rm [ - ] [ -fir] filename ...*
  - - Treat following arguments a filenames, so you can remove files starting with a minus sign
  - f - Force files to be removed without displaying permissions, asking questions, or reporting errors
  - i - Interactive, ask before removing each file
  - r - Recursively delete the contents of a directory, its subdirectories, and the directory itself
    - Unless inside it,
      - in which case you can't refer to it,
      - Use wildcard to delete all contents including subdirs : rm -r *cd

Variations may be implementation dependent… so check if needed, before use:

Older Unix To remove an entire filetree starting at or within the current directory (use ./ for current dir (does not apply to Linux:) , else name the dir) use the rm -r filename option as specified in previous slide, taking care not to do from root. (usually blocked by default anyway)

# ReMove DIRectory - rmdir

- Removes *empty* directories
- Syntax: *rmdir directory...*
- An error will be reported if the directory is *not* empty
- This is a safety issue, to minimise risk of deleting files within a directory.

- Clearly to delete a filetree would be incredibly tedious if all files within all directories at all levels had to be deleted first, from the bottom up…
- So use the rm -r option as specified in previous slide, taking care not to do from root. (usually blocked anyway)



# MoVe file - mv

- Move or rename files
- Syntax: *mv [ - ] [ -fi ] filename1 filename2*
  *mv [ - ] [ -fi ] directory1 directory2*
  *mv [ - ] [ -fi ] filename ... directory*
  - **-** - interpret all following arguments as filenames. This allows filenames that begin with a minus sign
  - *f* - force, overriding mode restrictions and the *-i* option. Also suppress warning messages about modes that would restrict overwriting
  - *i* - interactive, displays the name of the file or directory with a ? if the move would overwrite an existing file or directory



# CoPy file(s) -cp

- Syntax: *cp [ -ip ] filename1 filename2*
  *cp -rR [ -op ] directory1 directory2*
  *cp [ -iprR ] filename ... directory*
  - i - interactive, prompt for confirmation whenever the copy would overwrite an existing file
  - *p* - preserve, duplicate not only the file contents but also modification time and permission modes
  - *r* or *R* - recursive, if any of the source files are directories, copy the directory along with its files, including any subdirectories and their files

NB not all flags are case independent for all commands… e.g. man -k .. searches summary, man -K searches fulltext!



# Forms of cp – can't copy dir into file etc.

- *cp refuses to copy a file on top of itself, or into itself : for editors*
- *cp [ -ip ] filename1 filename2*
  - copies *filename1* ***onto*** *filename2*
- *cp -rR [ -op ] directory1 directory2*
  - recursively copies *directory1*, along with its contents and subdirectories, ***into*** *directory2*
- *cp [ -iprR ] filename ... directory*
  - copies *filename(s)* ***into*** directory
- Beware of a recursive cp like this (inoperable in modern Linux)
  urmac%*cp -r /home/myid/src /home/myid/src/backup*
  - Why? -
    - Well try putting you and yours inside yourself…again & again
      - Either you'll spend forever - infinite loop - no endtime
      - Or bust yourself - no space
  - Variations include mounting a backup drive (so it is now part of the filesystem) and trying to backup the entire filesystem onto the mounted backup drive
  - Easy oversight if idiot proof GUI tools are normally used for backup

# cp Versus mv

- *mv* simply changes the filename or absolute pathname of the affected files, the file's inodes are **not** changed
- *cp* creates new files so new inodes are created, unless the copy is overwriting an existing file
- Both will overwrite (&lose) any existing files with same destination filename
- Interactive flag will check with user…
  - Do you want to overwrite file …..?
- mv moves a directory into a target directory
  - inconsistent with mv on files,
  - but avoids target directory being overwritten…!



# touch

- Update the access and modification times of a file
- Syntax: *touch [-c] [-f] filename ...*
  - *-c* - Do not create *filename* if it doesn't exist,
    ( the default is to create *filename)*
  - *-f* - Attempt to force the update in spite of read/write permissions associated with *filename*
- Why would you want to do this?
  - This is particularly useful in software development, e.g. to update file timestamp, fooling the system into thinking it is updated, thus ensuring it is included in latest build…etc,



# conCATenate file(s) - cat

- Concatenate files to standard output
- Syntax: *cat [ -benstuv ] [filename...]*
  - *b* - number all lines except blanks
  - *e* - display non-printing characters and $ at end-of-line
  - *n* - number all lines
  - *s* - substitute a single blank line for multiple adjacent blank lines
  - *t* - display non-printing and tab characters
  - *u* – unbuffered – to ensure output is immediately available as to
    - stdout – output seen so user knows program is running
    - Or in FIFO pipe so next process can proceed

    Buffering results from blocking for efficient transfers to disk/ network
    and is unspecified in POSIX for 'cat', so -u ensures unbuffered.
    Unix supports it, Linux ignores it as the default is unbuffered.
    (Main points of this : POSIX not comprehensive; buffering blocks I/O until block full)
  - *v* - display non printing characters as follows:
    - ^X for Control-X
    - M-X (Meta: normally ALT key but may need setting to Option Key in OS X) for non-ASCII characters with high bit set, where X is the corresponding ASCII character



# Neat cat Tips

- *cat filename1 filename2 > filename3*
  - creates a new file, *filename3* that consists of
    - the contents of *filename1*
    - followed by the contents of *filename2*
- Cat filename1
  - displays the contents of filename1 on std. out - screen
- *cat > filename1*
  - creates a new file named filename1 whose contents are whatever you type on the keyboard

*urmachine%cat > my_file*
*The cat in the hat*
*smiled back at me.*
*^d (end-of-file character)*
*urmachine%*

# head

- Display the first few lines of a specified file
- Syntax: *head [-n] [filename...]*
  - *-n* - number of lines to display, default is 10
  - *filename...* - list of filenames to display
- When more than one filename is specified, the start of each files listing displays
  ==>filename<==



# tail

- Displays the last part of a file
- Syntax: *tail +|-number [lbc] [f] [filename]*
  or: *tail +|-number [l] [rf] [filename]*
  - *+number* - begins copying at distance *number* from beginning of file, if *number* isn'y given, defaults to 10
  - *-number* - begins from end of file
  - *l - number* is in units of lines
  - *b* - units are blocks
  - *c* - units are characters
  - *r* - print in reverse order
  - f - if input is not a pipe, do not terminate after end of file has been copied but loop. This is useful to monitor a file being written by another process…
  - In general (not for tail)
    - data loss problems if writer overtakes reader – standard file buffer issue!
    - Possible answers – but foolproof answer involves process synchronisation
      - pause writer – can also lose data…
      - bigger buffer...faster reader



# Word Count - wc

- Display a count of lines, words, and characters
- Syntax: *wc [ -lwc ] [ filename ... ]*
  - *l* - count lines
  - *w* - count words
  - *c* - count characters

- Example:

```
urmac%wc testfile
   7       43      168     testfile
urmac%
```



# More or less paging through a file

- 'More'
  - pages through a text file, but can't page back…
  - Historical .. have you ever seen and read an overprinted page…!?

- 'pg' for page, - even more older than more!
  - command no longer exists in 18.04, or in Mac BSD Unix
  - Replaced in bash by a new command 'pager'
    - which defaults to and invokes 'less'
  - Historical info:
    - pages through a text file, and can page back,
    - but awkward syntax – next page

- 'Less' is 'more' (Linux humour on Unix, on Multics!)
  - more modern & flexible than more, but good ideas copied so less difference*
  - pages back with arrow and page buttons
  - *as everywhere, imitation is the sincerest form of flattery

# more

- syntax:

*more [-cdflsu] [-lines] [+linenumber] [+/pattern] [filename...]*

  - *c* - clear screen before displaying
  - *D* – prompt with "[..space to continue ..q to quit …h for help] rather than ringing bell if an illegal command is used
    - Current Linux -

```
cs1>:/abc123/~$ ls -l -a directory1
```

  - *f* - do not fold long lines
  - *l* - do not treat formfeed ( ^L ) characters as page breaks
  - *s* - squeeze multiple adjacent blank lines into one
  - *u* - suppress underlining escape sequence



# Basic search commands for more

- I SPACE - display another screen or i more lines
- I RETURN - display another line or i more lines
- v - drop into *vi* at the current line
- i/*pattern* - search for the i$^{th}$ occurrence of *pattern*
- h - help, gives list of commands
- !*command* - invokes a shell to execute *command*
- **.** - repeat last command
- ^\ - halts display of text
  - but tends to lose some output while doing it;
  - try ^s & ^q (stop & quit stop to restart) back in teletype days
    - But modern systems spew so fast to terminals, that such commands are ineffective… to late, all gone! use more or less
- q or Q - exit from *more*



# Searching with pg – on way out…!

*Available on 16.04, but not on 20.04 – vi(m) based*

*[ optional ] - within square brackets below*

- *[i]/pattern/* - search forward for the *ith* occurrence (default i=1) of *pattern*
- *[i]^pattern^ or [i]?pattern?* - search backwards for the *ith* occurrence (default i = 1) of *pattern*
- After searching, *pg* will normally display the line containing *pattern* at the top of the screen
- *h* displays summary of commands
- *q* or *Q* exits



# Metacharacters - use based on Regular Expressions

Characters with special meaning to the shell

*, ?, [...],

* matches any grouping of zero or more characters

*- but in RegEx zero or more of preceding char only*

? matches any single character

[...] allows matching a range of characters

[0-9] matches any digit

[A-Z] matches any capital letter

[a-d] matches a, b, c, or d

[aeiou] matches any vowel

# Metacharacter Examples

- Suppose a directory listing of your files shows the following files:

urmac%ls

Boy toy coy cow soy1 soy2 soy.1 soy1.1
bow mow1 mow2 mow3 say1.1 say1.2 say1.3
hay shay tray fray flay chow slay bay buy

- How do we select groups of these files using metacharacters?
  - Will see more later when we do regular espressions but basic idea is a few characters followed by o/a and then w/y; or in RegExp expressed as… *[oa][wy]

  - e.g. ls *[oa][wy]



# Metacharacter Examples

- Suppose a directory listing of your files shows the following files:

urmac%ls

Boy toy coy cow soy1 soy2 soy.1 soy1.1
bow mow1 mow2 mow3 say1.1 say1.2 say1.3
hay shay tray fray flay chow slay bay buy

- How do we select groups of these files using metacharacters?
  - Will see more later when we do regular espressions but basic idea is a few characters followed by o/a and then w/y; or in RegExp expressed as… *[oa][wy]

  - e.g. ls *[oa][wy]



# Preliminary Outline - For perusal.

Liable to changes
Before and After
Lecture

Likewise for
Previous Recordings

Links, inodes
&
underlying file structure

# Links & underlying file structure

- Links
  - Save space in having a copy, by merely copying a link
  - Only one copy, easier to edit, & manage
    - No need to propagate changes
  - Need some basic understanding to use filesystem better
- Two link types
  - Hard – to the actual file blocks on disk – or abstraction*
    - Hard links Are independent of each other
      - Delete one, others remain to access file
      - And file remains until all hard links are removed
        - Can do **find / -inum 12345** to find all files with given inode
  - Soft – to the name in the directory entry
    - Change/move/delete the name/file
      - and soft link is lost to the file

# Links & inodes

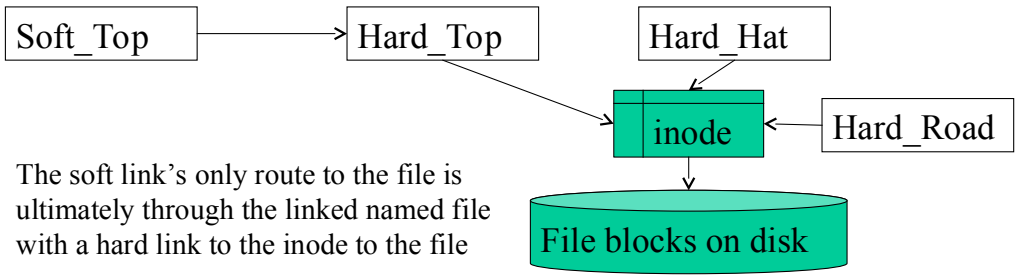
- Inode – index node on disk
  - Is pointed to by directory entry for file
  - In turn points to file blocks on disk
  - Limited number (of blocks & inodes) on any device
    - These limits can block new file creation, showing disk full
      - the disk may not be full, just the inodes are all used
      - Ubuntu default is one per 16Kbytes
      - But can be changed… not a general issue
    - Although superceded by memory cards,
    - DVD+RW only support about 1,000 writes,
      - so if a concern, (lots of reuse) mount with
        - noatime option to avoid inode time update writes for access
        - Read only if no need to write

  less of an issue now that cloud/servers replacing fixed storage!

# Hard & Soft Links (directory entries) to files

- Hard link… is to the hard inode data on disk…to identify file block
  - hard (unbreakable)
  - Hardy links… keep any one, and it will keep the file
  - Or … need to remove all hard links to remove the file..
- Soft … is merely a link to the filename (or filepath) in a directory entry
  - Soft … breakable
  - Softie … remove the softie's hard link and the softie is lost..

  ..it's only way to the file is through it's link to a hard link…
- Can you do a soft link to a soft link? Try and see!? Why?




The soft link's only route to the file is ultimately through the linked named file with a hard link to the inode to the file

# Links

Secure erase of a file will affect all linked ones

Links

- Hard
  - A link to the inode
  - Like another (path) name for a file
  - File isn't gone until all hard links are deleted
    - Link count shows up in ls -l listing
  - Can use to avoid accidental deletes by a safe-list
  - can't use (hard links)
    - for directories, to avoid cycles in filetree and infinite filesearch loops in filetree
    - Across devices (systems, partitions etc.) as inodes internal to device
- Soft
  - A link to another filename
    - Which ultimately links to an inode
  - If target filename is removed, no access to file via link
  - Can be used across filessystems … just another path...name

# UNIX File Management

- Six types of files
  - Regular, or ordinary
  - Directory
  - Special
  - Named pipes
  - Links
  - Symbolic links

# Inodes – index node… originators unsure

- Display inode numbers for files using '-i' flag...ls -i ~
  - e.g. to show inodes in increasing sequence (sort on key 1) for files

  $ls -il | sort -k 1

  14553700 -rw-r--r-- 1 james james 0 Sep 21 11:58 a
  14553701 -rw-r--r-- 1 james james 0 Sep 21 11:58 b
  14553702 -rw-r--r-- 1 james james 0 Sep 21 11:58 c
- Inodes are a control structure/(c-talk for ~ data record) that contains key information for a particular file
  - Admin tracking – who, when …
  - Where it is on the disk…
- Several filenames may be associated with a single inode
  - But an active inode is associated with only one file, and
  - Each file is controlled by only one inode

# Inode – Unix originals...other derivatives similar

- inodes (or index nodes) contain information about files

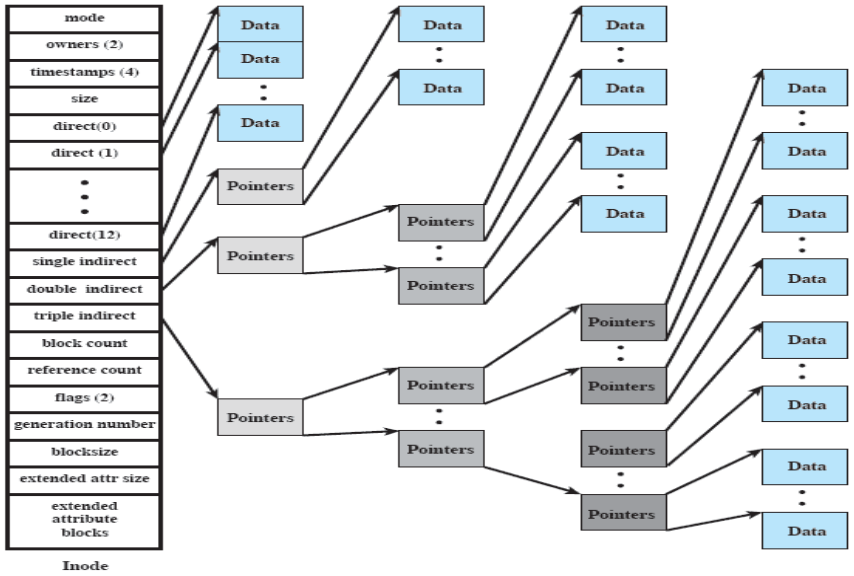
owner myid
group 300
type regular file
permissions rwxr-xr-x
last accessed Aug 23 1999 1:45 PM
last modified Jul 4 1999 9:17 AM
size 6030 bytes
number of links 2
disk addresses

- inodes do not contain path/file name information,
- directories hold filenames with links to their inodes

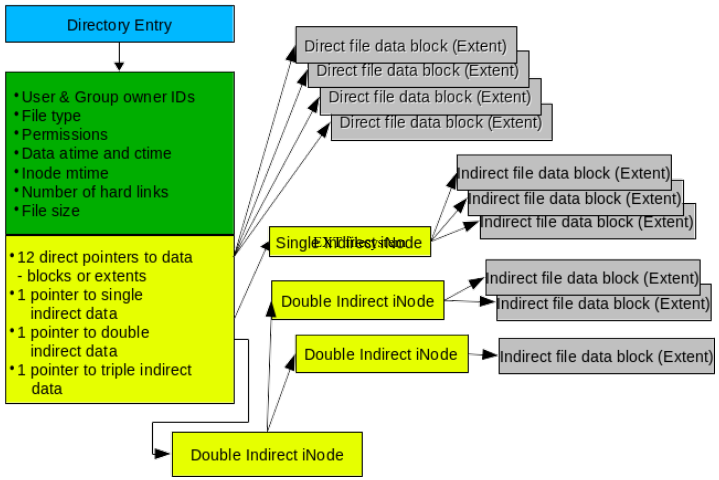
# Free BSD (OS X also based on BSD) include:

- The type and access mode of the file
- The file's owner and group-access identifiers
- Creation time, last read/write time
- File size
- Sequence of block pointers
- Number of blocks and Number of directory entries
- Blocksize of the data blocks
- Kernel and user settable flags
- Generation number for the file
- Size of Extended attribute information
- Zero or more extended attribute entries

# FreeBSD Inode and File Structure




# Linux EXT filesytem – note filename is in directory, not inode.




# Some (web) links on filesystems

https://opensource.com/article/17/5/introduction-ext4-filesystem

https://metebalci.com/blog/a-minimum-complete-tutorial-of-linux-ext4-file-system/

https://www.sans.org/blog/understanding-ext4-part-1-extents/

NB Ambiguity…

- Filesystem at user level refers to directory structure
- Filesystem at storage level refers to block to device mapping
  - Only for system installation & configuration
    - We can ignore in this course

This background is so you know some background,

In case you go reading and wonder what's going on!

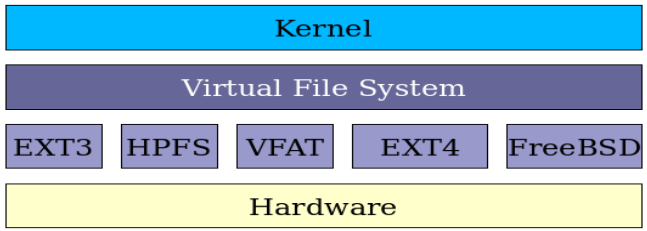
# Example 4k blocks on 1GB disk... too small but simple!

- How many blocks?
  - 4K ~ $2^{12}$ ~ 4 x $10^3$; 1 GB ~ $2^{30}$ ~ $10^9$; so $2^{30-12} = 2^{18}$ = 1/4 million 256K blocks = how many indices needed
- Need min of 18 bits to index each block on disk as there are $2^{18}$ blocks
- + extra bits for keys! Round up to binary power of bytes = 32 bits = 4 bytes
  - Binary powers interface better & faster with binary address decoders
- So 1K indices per 4K block, need 256K blocks indexed, so 256 index blocks
- Needs 256 x 4KB storage = 1MB out of 1 GB ~ 1/1000$^{th}$ (or $2^{-10}$ to be exact)
- Inode (but not every index block) also typically holds...
  - The file's owner and group-access identifiers
  - Creation time, last read/write time
  - File size
  - Sequence of block pointers
  - Number of blocks and Number of directory entries
  - Blocksize of the data blocks

# Example 4k blocks on 1TB disk... more realistic size!

- How many blocks?
  - 4K ~ $2^{12}$ ~ 4 x $10^3$; 1 TB ~ $2^{40}$ ~ $10^{12}$; so $2^{40-12} = 2^{28}$ = 256 million blocks
  - = how many indices needed
- Need min of 28 bits to index each block on disk as there are $2^{28}$ blocks
- + extra bits for keys! Round up to binary power of bytes = 64 bits = 8 bytes
  - Binary powers interface better & faster with binary address decoders
- So 512 indices per 4K block, need 256M blocks indexed, so 512K index blocks
- Needs 512K x 4KB storage = 1MB out of 1 TB ~ 1/1,000,000th (or $2^{-20}$ to be exact)
- Inode (but not every index block) also typically holds...
  - The file's owner and group-access identifiers
  - Creation time, last read/write time
  - File size
  - Sequence of block pointers
  - Number of blocks and Number of directory entries
  - Blocksize of the data blocks

# Linux has a complex Virtual Filesystem (VFS)




can interface to many different types of underlying filesystems…
- for use with all OS'es, incl.OS X : FAT (File Allocation Table)
- other Apples : older HFS+, newer APFS may need extra sware,
- can also read Win : NTFS (New Technology File System);

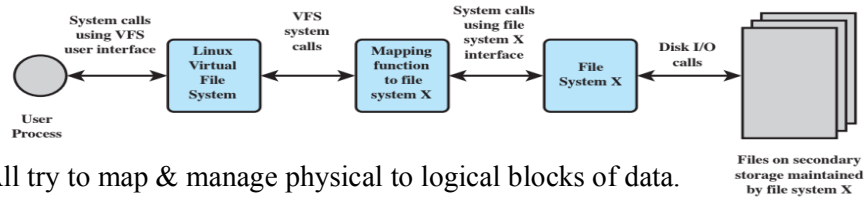
NB Ambiguity… filesystems above mean how the physical storage device is physically mapped (like formatting), prior to use of the device for installation of OS and or data filesystem, and will ignore more for now. Filesytems above do NOT refer to 'logical' data files and their organisation, as looked at recently.

# Linux Virtual File System concept

For bewildering overview / reference:-

https://en.wikipedia.org/wiki/Comparison_of_file_systems

https://en.wikipedia.org/wiki/List_of_file_systems




All try to map & manage physical to logical blocks of data.
Various filesystems and abstractions attempt to resolve issues such as backup, reliability, flexibility and filesystem mapped to disk partitions expandability with minimal admin effort; e.g. Logical Volume Manager (LVM), but generally introduce performance penalties due to extra mapping processes.

Some try to maintain data integrity by error checksums CoW (Copy on Write) and although devised decades ago, have licensing stability & recovery* issues
- now Linux options: ZFS (Zettabyte FileSystem); btrfs – B-tree (better) FS
If corrupted, harder to recover, error correction inbuilt like encryption – wait?

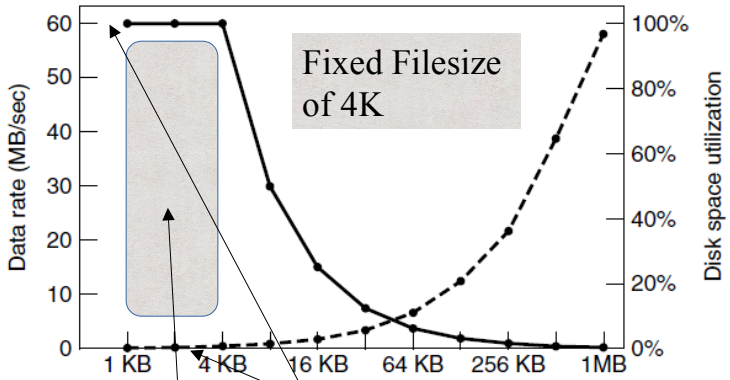
# LVM : basic purpose : dynamic resizing without copy & move

- Create single logical volumes of multiple physical volumes or entire hard disks (somewhat similar to RAID 0, but more similar to JBOD),
- Manage large hard disk farms by adding/replacing disks without downtime or service disruption, in combination with hot swapping.
- filesystems can be easily resized as needed, instead of estimating a fixed partition size at the start, then resizing needing awkward backup & move.
- Make consistent backups by taking snapshots of the logical volumes.
- Encrypt multiple physical partitions with one password.
- Basically LVM is a thin software layer on top of the hard disks and partitions, which creates an abstraction of continuity and ease-of-use for managing hard drive replacement, repartitioning and backup.
- Introduces processing overhead delays both reading & calculating maps
- May look at later after more familiarity with CLI, GUI interfaces coming

# Blocksize - tradeoffs

- Disks are normally buffered to 'amortize' latency (US textbook speak)
  - i.e. to lower the average access time for disk data (like mortgage repaying)
  - If you're going to waste time and effort going to a slow SSD / slower disk, then make it worth your while, and bring a lot into a RAM buffer
- Block is the minimum allocation to a file on disk, but has knock-on effects
  - large blocks – risks space waste, & too few indices... inodes & filecounts
    - too large for most files : most of block wasted with small files
    - fewer blocks for a fixed disk size
    - Fewer indices-> less index blocks
    - small index field & large blocks > less index blocks - run out of indices?
  - small blocks – needs more index blocks, more levels and processing.
    - too small for most files : need more blocks, links or index to next blocks
    - more blocks for a fixed disk size
    - more indices -> more index blocks
    - larger index field & small blocks -> more index blocks

Extreme(ly simple) example of Space-Time tradeoff!




Time vs Space metrics for 4K files vs various 'disk-use' sizes (X-axis)
Bigger disk → more files → better use of disk space;
- but more indexing & searching, so slower transfer
Note with only one 4K file, great speed, but waste of space!
More files => better use of space, but need space & time to process

# Blocksize – tradeoffs - space

Principles also applies to SSD's as they are still slower than RAM

Basic tradeoffs : try to minimise wasted space

- Smaller blocksize to filesize to minimise wasted space
  - Small files don't need a whole block => large blocks wasted!
  - Even for small blocks On average 0.5 block wasted AV(0.0 – 1.0)
- Definitely for a filesystem of mostly small files,
  - use smaller blocksize to avoid half or block wasted on average
  - But smaller blocks => more blocks
  - => more blocks => more indices => more address bits
    - => fewer indices per block => more index blocks
    - With smaller blocks, mean more index blocks
    - So waste
      - more space on index blocks!
      - more time going through levels of index blocks

# Blocksize – tradeoffs – also time

Still applies to SSD's as they are still slower than RAM

Basic tradeoffs : also try to minimise wasted time

- Big blocks
  - Slower to transfer and process ... memory management
- Small blocks
  - More to index and search through

Overall – it's a statistical distribution with associated time tradeoffs

Mostly set by default values, for most needs, but for specialist setups

- But simple approach : match blocksize to average small filesize
  - Larger blocks
    - For filesystems for databases / video streams with large files
  - Small blocks
    - For lots of small files, such as typical office/document work
- Try to get entire index in RAM / buffer for fast access

Just to appreciate some of the issues, find details if you ever need them.

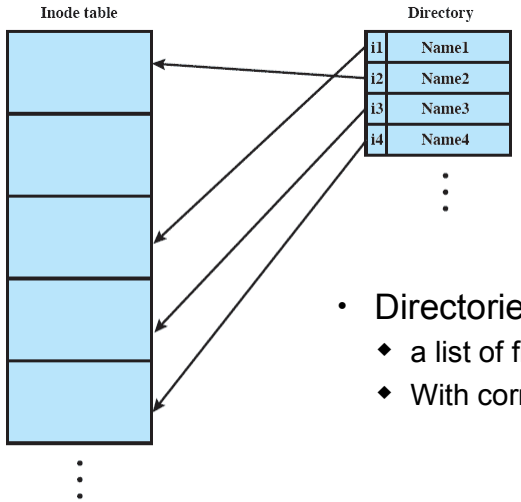
# File Allocation

- File allocation is done on a block basis.
- Allocation is dynamic
  - Blocks may not be contiguous
- Index method keeps track of files
  - Part of index stored in the file inode.
- Inode includes a number of direct pointers
  - And three indirect pointers
    - With recursively deeper blocks of pointers
      - To even deeper blocks of pointers
        - » Which eventually point to file data blocks
- The file-tree index in the inode combines
  - max. flexibility in file size

with

  - Min. indexing overhead (both space and time)

# UNIX Directories and Inodes




- Directories are files containing:
  - a list of filenames
  - With corresponding pointers to inodes

Figure 12.15 UNIX Directories and Inodes

# Disks, Partitions, Mounts & Links

- Disks – a bare empty disk needs to be formatted
  - Basically mapping out 'sites' on the disk…
  - Normally done at the factory…
  - May need reformatting to operate with other OS'es
- Partition table - to create partitions on a disk
  - There are tools to do this, but need to know what you are doing to avoid a mess.
- Partitions or *slice* –
  - effectively independent devices with distinct names
    - e.g. /dev_name/subdir_name
  - There must be at least one partition on a disk
  - Remaining areas of unamed/undefined partitions are free space,
  - Some OS'es only support fixed partitions
    - So disk is stuck with original partition choices
    - Quick to work, but slow to change, as partitions need backup and restore
  - Other OS'es support dynamic partitioning tools (e.g. Logical Volume Manager)
    - Partition boundaries can be changed anytime.
    - Changes may be 'virtual' and introduce performance penalties, as they are implemented by software over unmodified hard partitions.

# Filesystems on partitions

- Filesystem –
  - data structure for each partition containing inodes for files
    - Inodes are local to each filesystem, and therefore are not unique across filesystems
    - (so hard links to inodes cannot be made across filesystems)
    - But softlinks, link to the pathname for the file and can therefore link across filesystems … but they have other problems.
  - Top of the data structure has an unnamed directory, which is the
    - mount point when grafted into a bigger directory tree
    - Root of directory tree, (but not called root) within the device/partition filesystem
- Mounts – where the filesystem is grafted into the bigger filesystem tree
- Links
  - Hard –
    - to the physical disk inodes – each with a unique number in the filesystem
    - can't cross device/partitions as duplicate inode numbers are possible
  - Soft – to the absolute file pathname - can cross filesystems – like windows shortcut



# LiNk - ln

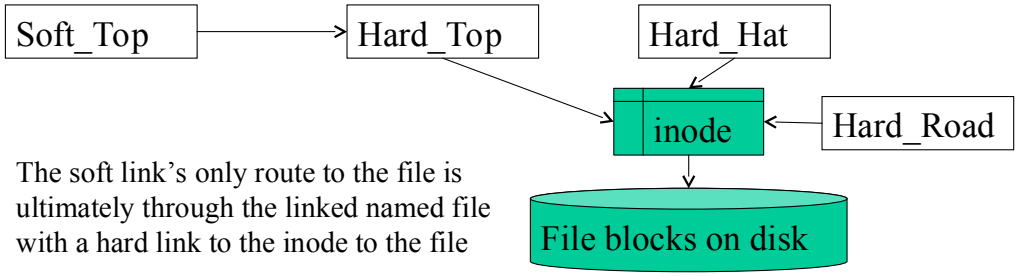
- Create a pseudonym (alias) for an existing file
- Syntax: ln [-fs] filename [linkname]
  - f - force a hard link to a directory, only available to superuser (and often even forbidden to superuser- version dependent!)
  - s - create a symbolic (soft) link
- Hard links are default, a pointer from the directory entry for the filename to the file's inode on disk which points to file)
- Symbolic, or soft links, create an indirect pointer to the file via the pathname (i.e. filename)
  - Although only the superuser can create a hard link to a directory, in a non-restricted system, any user can create a soft link
  - Soft links also work across file systems, hard links don't
    - Across => different mounts or volumes or
    - e.g. backup system makes extensive use of soft links



# Hard & Soft Links (directory entries) to files

- Hard link… is to the hard inode data on disk…to identify file block
  - hard (unbreakable)
  - Hardy links… keep any one, and it will keep the file
  - Or … need to remove all hard links to remove the file..
- Soft … is merely a link to the filename (or filepath) in a directory entry
  - Soft … breakable
  - Softie … remove the softie's hard link and the softie is lost..

  ..it's only way to the file is through it's link to a hard link…
- Can you do a soft link to a soft link? Try and see!? Why?




The soft link's only route to the file is ultimately through the linked named file with a hard link to the inode to the file



# Why Hard and Soft Links?

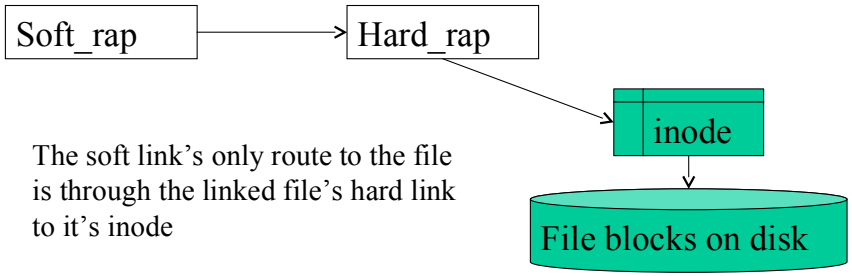
- Hard links all have equal status
  - When a file with hard links is deleted,

    the actual file (+ inode) remains

    until <u>all</u> of the hard links have been deleted
- Soft links can have the original file deleted while soft links still remain … can lead to confused corrupt filesystems
- Soft links can also be confusing when used to change directories
  - Suppose you have a soft link called My_Dir that points to your home directory, /home/myid and you *cd My_Dir*
  - Then, if you execute *pwd*, it shows /home/myid!
  - *pwd* shows the name of the linked-to directory rather than the name of the link

# Why Not Just cp?

- *ln – link* creates a pseudonym for the given file with inodes
  - Not created for hardlinks – same file on disk, so point to same inode
  - but oddly are notionally for softlinks (ln -s ) as links to the directory link
  - In both cases only one copy of the actual file exists
  - Modifications via any filename/pathname will affect the file
- *cp – copy* creates a new copy of the given file
  - A new inode is created
  - Twice as much disk space is used
  - Changes to any copy are not reflected in the other(s) – next slide

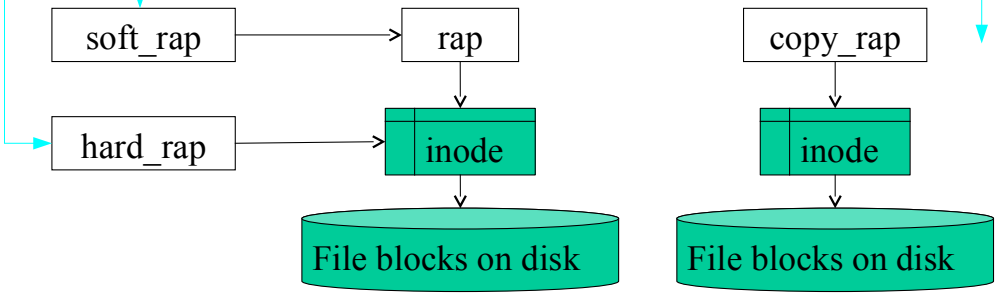



The soft link's only route to the file is through the linked file's hard link to it's inode



# blinking copy moves – see next slide

- cp – complete **co**p**y** – total copy of file and inode
- ln – **li**n**k** name
  - Hard – to file…the file remains, while any hard link remains.
  - Soft – to existing name, useless link if existing name removed or changed






# Easy to lose loose ends with soft links!

```
cs1> cat >rap
Wrapping Up!
cs1> cp rap copy_rap
cs1> ln rap hard_rap
cs1> ln -s rap soft_rap
cs1> ls -il
total 0
32742103 -rw-------   1 jsad1 csdipact2012 13 2011-10-12 12:54 copy_rap
32742102 -rw-------   2 jsad1 csdipact2012 13 2011-10-12 12:54 hard_rap
32742102 -rw-------   2 jsad1 csdipact2012 13 2011-10-12 12:54 rap
32742105 lrwxrwxrwx   1 jsad1 csdipact2012  3 2011-10-12 12:55 soft_rap -> rap
cs1> mv rap trap
cs1> ls -il
total 0
32742103 -rw-------   1 jsad1 csdipact2012 13 2011-10-12 12:54 copy_rap
32742102 -rw-------   2 jsad1 csdipact2012 13 2011-10-12 12:54 hard_rap
32742105 lrwxrwxrwx   1 jsad1 csdipact2012  3 2011-10-12 12:55 soft_rap -> rap
32742102 -rw-------   2 jsad1 csdipact2012 13 2011-10-12 12:54 trap
cs1> cat soft-rap
cat: soft-rap: No such file or directory
cs1> cat trap
Wrapping Up!
cs1>
```

Check inode numbers below :-
- hard → same
- soft → different

1.Create file 'rap'
2.Make a copy 'copy_rap'
3.Make a hard link 'hard_rap' to file 'rap'
4.Make a soft link 'soft_rap' to file 'rap'
5.Check it all out! Look at the directory

NB the default link permissions leave it open to all!

6.Move file 'rap' to 'trap' - i.e. rename
7.Check it all out again ! Look at the directory

Original soft-rap is still linked to rap, but now moved trap.

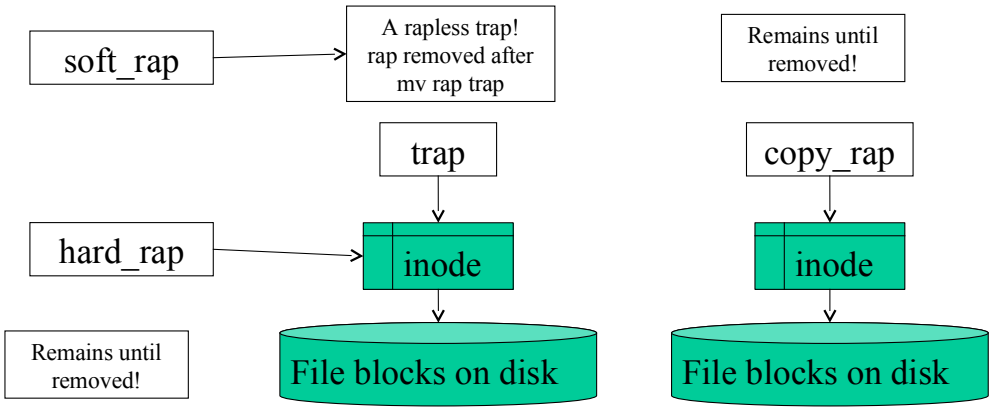
8.Try the soft link to the original name 'rap'
9. it's gone!
10.But the new filename trap is fine

Bottom line : soft links can leave you lost...best avoided... unless you can't!



# blinking copy moves

- mv – na**m**e **m**o**v**e only
  - but old name is overwritten in directory entry and lost
    - *trap was rap in previous slide / directory before mv rap trap*
  - Any softlinks to the old name go nowhere



# blinking catches

In operating systems, particularly Open Source, there are often tradeoffs (often political between development groups or companies) between things not specified in POSIX, or things not implemented in a given distribution….e.g.

**Hard links**

- Undesirable to have a hard link to a directory - to avoid infinite loops in 'tree' search (although shouldn't, doesn't mean it's not done – most modern systems block it!)
- Can't have hard links across filesystems
  - Each filesystem has a different set of unique inodes, with unique id's, within the relevant filesystem, but the id's will not be unique across filesystems, so hard links only work within a single filesystem
- Are falling out of favour for these reasons with increasingly large filesystems consisting of several smaller ones on devices, whether local or distributed, each with separate inode set and structure. (Abstractions to help: LVM & ZFS etc...later)
- can't delete file until all hardlinks are gone… but can use find inode option...
- **Soft / symbolic links**
  - Are just links to a name, i.e. a directory entry, so can link to all of the above: directories, across devices and filesystems, both local and distributed.
  - Always point to the latest version of the file, where a file is repeatedly rewritten, as in software development with repeated regeneration of executables (each would have a different inode so hard links would point to the original version) e.g. as in C => *.o files (compiler output/object files)
  - Suffer from dead end missing links..



# Links – best news is can mostly ignore

- Most of material in this lecture is background information so you can be aware of the minefield so you
  - can understand the relevant manual entries
    - i.e. whether hard or soft links are followed in recursive directory searches, copies or backups etc.
  - might have some idea what is going on if weird things are happening your filesystem
  - Understand why systems need maintenance, and ultimately a good cleaning
  - Can adopt good filesystem management practices…
    - e.g. try to avoid dead end links – and to minimise their use
    - Understand the use of tools which tidy them up
  - Can answer basic questions if asked somewhere



# Script - make typescript of terminal session

- handy for recording for reference or assignments

script [options] [file]

- Terminated by ^d - standard EOF
- Default for omitted file is typescript
- NB : it will fail with I/O file redirection operators >,>>
- Instead it uses
- - a for append, to add to and avoid overwriting file
- Designed to be used in interactive mode, although options to capture command output exist
- don't use with commands which
  - hop over the screen, i.e. editors.. even vi(m), nano etc.,
  - Or any other graphical output commands.
- don't use in pipes… it may read parts of intermediate files