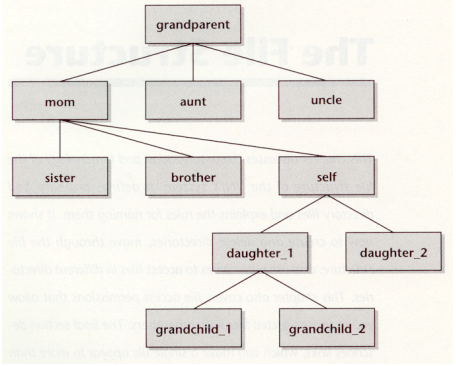
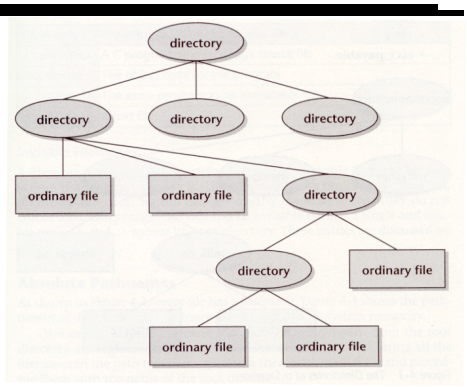
3 - File System

- 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 containers 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 – **m**a**k**e, r**e**move, **c**o**p**y, **m**o**v**e
- In reality, files are simply a stream of bytes
  - Critical design philosophy of Unix
  - Everything in Unix is a string of bytes: files, devices,.etc

Tree Structure



Files and Directories



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 (**.**) can be used as any other character,
    - but 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
  - Willie*Weasil - displays as Willie*Weasil
  - 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…best to use only alphanumeric and underscores in filenames.

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)

Access Controls Provide Security

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

Unix File Hierarchy

/ – root. The source of everything

/bin – Where command executables (programs) are run

/sbin – executables generally of interest only to the super user

/home – where your "home directories" are

/tmp – same as c:\temp in windows. Just for temporary files

/var – system logs and other stuff (probably not interesting)

/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)

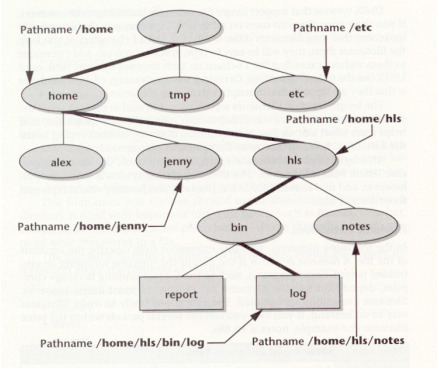
Print Working Directory - pwd

- 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
urmac%more jerry        shell types rest of name
```

- 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 : displays list of filenames starting with string so far
  - Bash : beeps
- 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

```
urmac%ls
jerry    john  joe   jimmy   wesley   tom
bill     willie ken   ted     shirley  joan
urmac%cat joTAB

cat johTAB   … bash shell just beeps and you finish
cat john

urmac%cat joTAB   … C shell lists options and you finish
john   joe   joan
```


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, files beginning with a **.** are not listed
  - c - Sort on time of last edit
  - l - List in "long" form, giving mode, number of links, owner, size, and time of last modification
  - R -Recursively list subdirectories encountered
  - 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
- Syntax: *mkdir [-p] dirname*
  - *p* - allows "missing" parent directories to be created as needed
- Requires write permission in the parent directory

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

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

  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 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, 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
  - *v* - display not printing characters
    - ^X for Control-X
    - M-X for non-ASCII characters with high bit set

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

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
- 'pg' for page,
  - 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 more
  - pages back with arrow and page buttons

more

- syntax:

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

- *c* - clear screen before displaying
- *d* - display error message rather than ringing bell if an illegal command is used
- *f* - do not fold long lines
- *l* - do not treat formfeed 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 ith 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
- q or Q - exit from *more*

PaGe - pg

- Page through a file
- Unlike more, *pg* allows you to back up in the file
- Syntax:*/usr/bin/pg*
- After each screen is displayed, it pauses, displays a **:** prompt, and awaits a command
- Perusal commands
  - *Return* or *Enter* - display next screen
  - (+|-)*number* - simulate scrolling forward or backwards *number* of screens
  - **.** or *CTRL-L* - redisplay current screen

Searching with pg

- *[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

toy	boy	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]

UNIX File Management

·Six types of files

- Regular, or ordinary
- Directory
- Special
- Named pipes
- Links
- Symbolic links

Inodes

·Index node

·Control structure 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

·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 <u>not</u> contain path or file name information

Free BSD Inodes 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

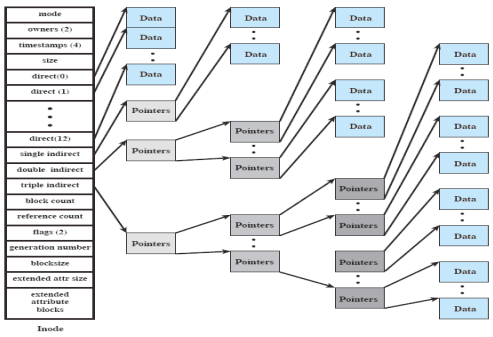
·Kernel and user setable flags

·Generation number for the file

·Size of Extended attribute information

·Zero or more extended attribute entries

FreeBSD Inode and File Structure

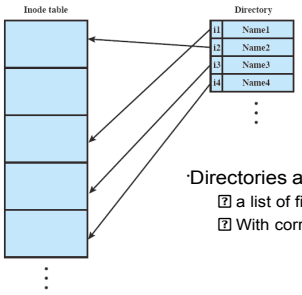


File Allocation

- 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
- 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
- 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 area(s) of partitioned disk is free space
  - Some OS'es only support fixed partitions
    - So disk is stuck with original partition choices
  - Other OS'es support dynamic partitioning tools
    - Partition boundaries can be changed anytime.
    - Changes may be 'virtual' and introduce performance penalties, as they are implemented by software over unmodified hard 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 thereore 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 for the top level
- 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 filesystems as duplicate inode numbers are possible
  - Soft – to the absolute file pathname - can cross filesystems

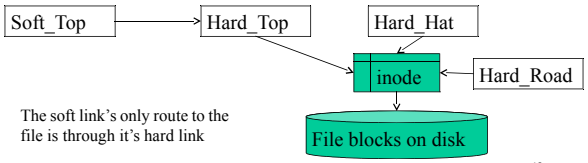
LiNk - ln

- Create a pseudonym for an existing file
- Syntax: ln [-fs] filename [linkname]
  - f - force a hard link to a directory, only available to supersuser (and often even forbidden to superuser!)
  - s - create a symbolic (soft) link
- Hard links are default, they create a pointer to the file (actually to the file's inode 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-restircted 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 to files

- Hard… is to the hard data on disk…
- 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 to the filename (or filepath)
- 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?



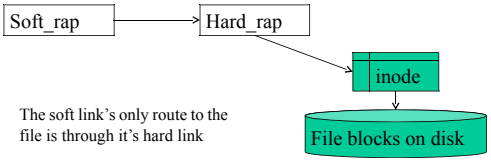
The soft link's only route to the file is through it's hard link

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 all 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?

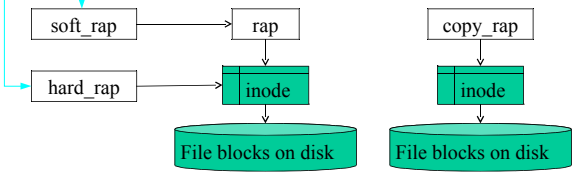
- *ln – link* creates a pseudonym for the given file
  - No new inode is created
  - Only one copy of the actual file exists
  - Modifications via either filename affect the file
- *cp – copy* creates a new copy of the given file
  - A new inode is created
  - Twice as much disk space is used
  - Changes to any copy are not reflected in the other(s)



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

blinking copy moves

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



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>
```

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

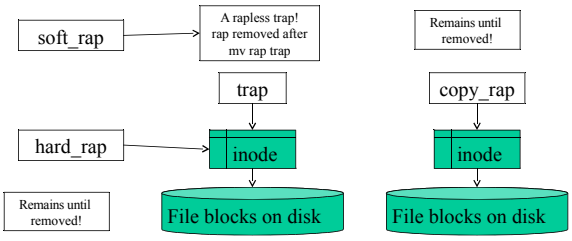
6.Move file 'rap' to 'trap' - i.e. rename
7.Check it all out again ! Look at the directory

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

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

blinking copy moves

- mv – name **mo**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



History (hereafter – mostly outdated!)

- Display the history list in the C or Korn shells
- Syntax: *history [-hr] [n]*
  - *h* - display history list without leading numbers
    - used to produce files for input as a script
  - *r* - display the history list in reverse order with most recent command first
  - *n* - number of previous commands to display
    - if n is omitted, the entire history list is displayed where the entire list length is determined by the value of the shell variable *history*

    Example:

    set history = 40

Using history

- History event specifiers
  - Always preceded by a **!** (bang); this identifies the command as a history command
    - *!!* - Repeat previous command
    - *!n* - Repeat command *n*
    - *!-n* - Repeat the *n*th-to-last command
    - !str - Repeat the last command beginning with *str*
    - *!?str?* - Repeat the last command containing *str*
    - *!#* - Repeat the current command line typed so far
      - Normally used with word designators to select portions of the current command line

history Examples

- Recalling the last command

```
urmac%lpq letter_to_Mom
urmac%!!
lpq letter_to_Mom
```

- Referring to commands by number

```
urmac%history
1  ls
2  cat letter_to_Mom
3  lpr letter_to_Mom
urmac%!2
cat letter_to_Mom
```

- Or just use up-arrow repeatedly to get to the right line.
- **lpr** – line printer command… line printers are fast 'bulk run off' printers which printed a line at a time…pages / second, but with fixed fonts dependent on print drum!

Referring by Relative Number

```
urmac%history
1  ls
2  vi letter_to_Mom
3  nroff -ms letter_to_Mom | more
4  nroff -ms letter_to_Mom | lpr
urmac%!-3
vi letter_to_Mom
urmac%!-2
nroff -ms letter_to_Mom | lpr
```

- **vi** = visual editor… not visual by today's standards but is universal with Unix and quite powerful once learned
- **nroff** = new run off – text formatting for system printer
- **|** - pipe – takes output file from one command and runs it into the next, in this example, **nroff** formats the textfile to print on line printer

!-n Tip

- Since !-n is relative, it can be used to easily execute a set of events repeatedly

```
urmac%vi letter                    edit letter
urmac%nroff -ms letter | more      see if it formats right
urmac%!-2                          repeat edit
vi letter
urmac%!-2                          repeat format
nroff -ms letter|more
urmac%!-2                          repeat edit
vi letter
urmac%!-2                          repeat format
nroff -ms letter|more
```

  - Here **nroff** is piped to **more**, to display a screen at a time.

Referring to Commands by Strings

```
urmac%history
347 who
348 wc -l data
349 cal 1776
350 emacs calendar
351 emacs datebook
urmac%!w
wc -l data
urmac%!wh
who
urmac%!?datebook?
emacs datebook
```

- **emacs** is a powerful and highly customisable editor, which can be optimised to suit user preferences and application : colour coded language specific keywording, formatting etc…

Command Editing with history

- General form of substitution - !:s/old/new/

```
urmac%history
347     who
348     wc -l data
349     cal 1776
350     emacs calendar
351     emacs datebook
urmac%!350:s/calendar/datebook/
emacs datebook
```

- A shortcut for the previous command

```
urmac%^datebook^calendar^
emacs calendar
```

Saving history Across Logins

- If you set the *savehist* shell variable, the shell saves history lines in ~/.history at logout and reads them at the next login
- If you set savehist without specifying a value, the entire history list is saved
  - set history = 20
  - set savehist
- If you give it a value, only that many events will be saved
  - set history =20
  - set savehist = 10