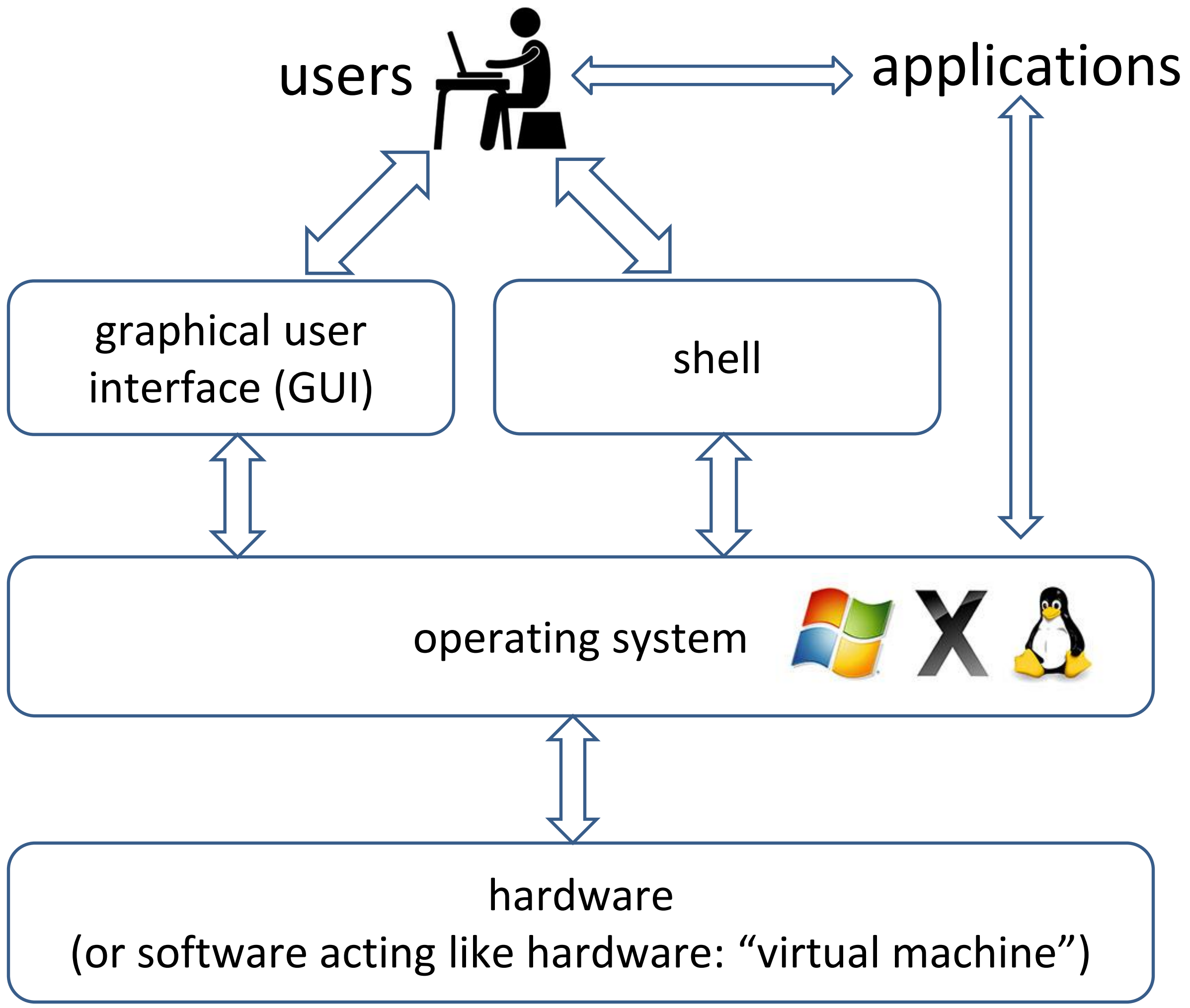
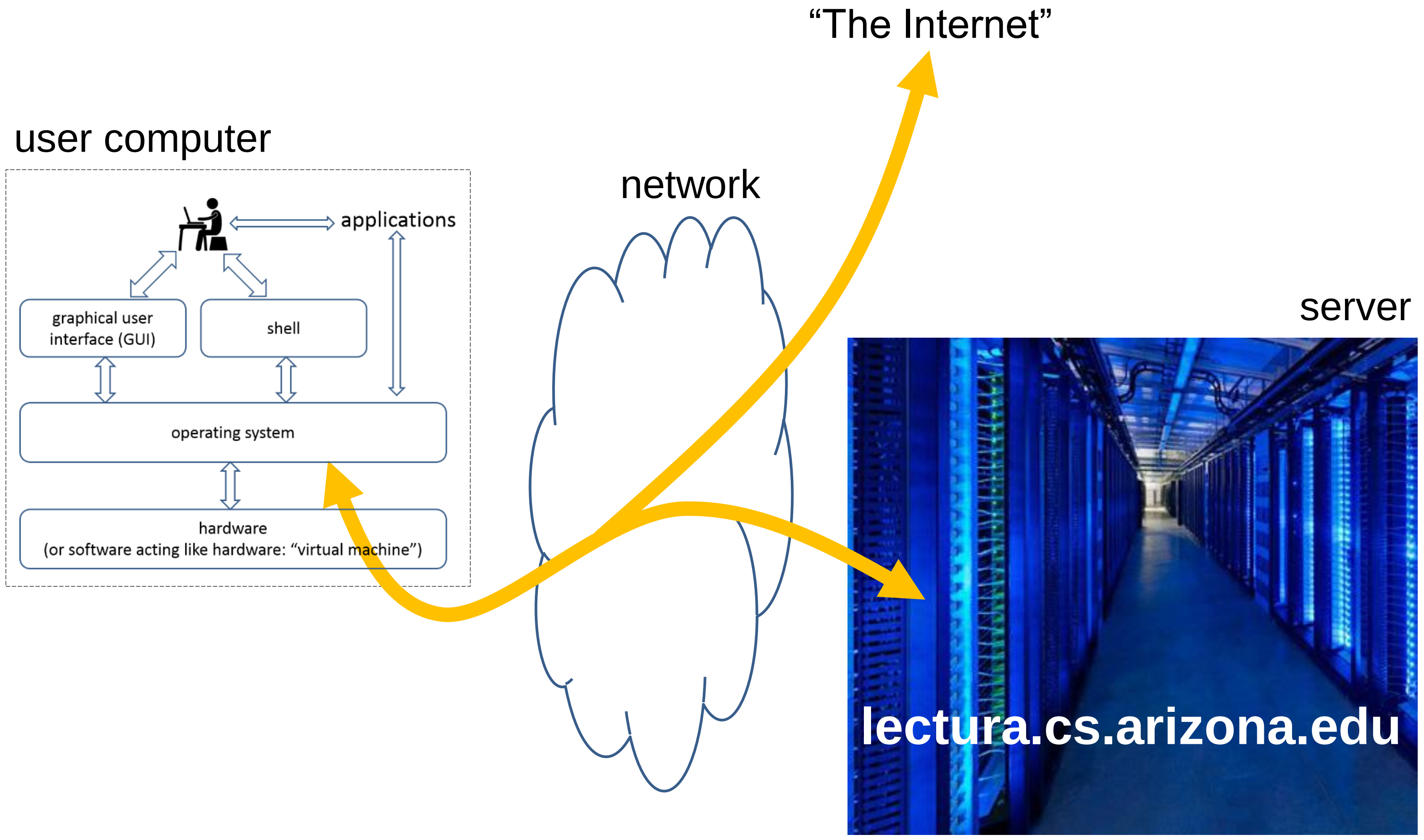
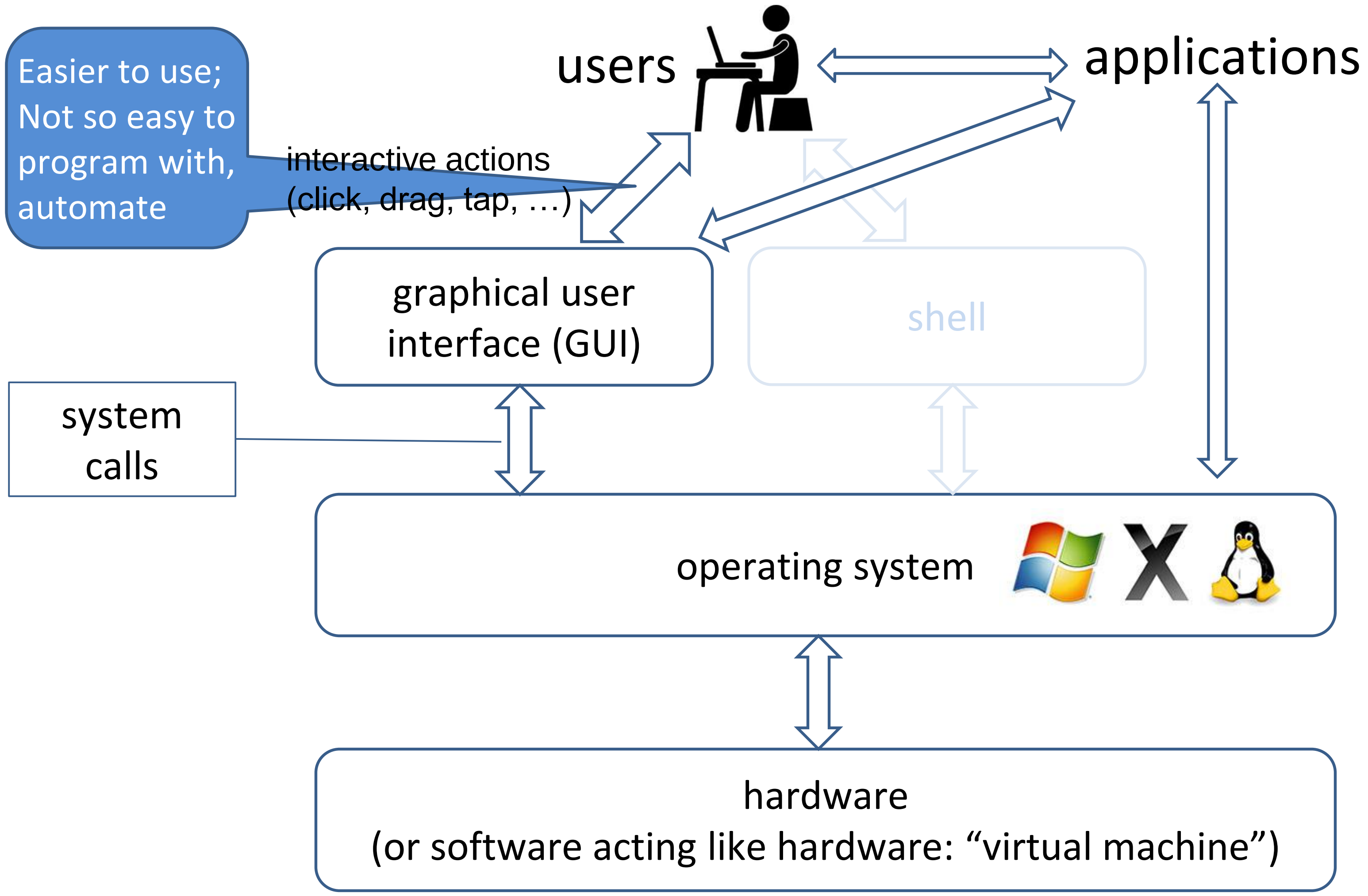
Organization of a computer system


users
applications
graphical user interface (GUI)
shell
operating system
hardware
(or software acting like hardware: “virtual machine”)

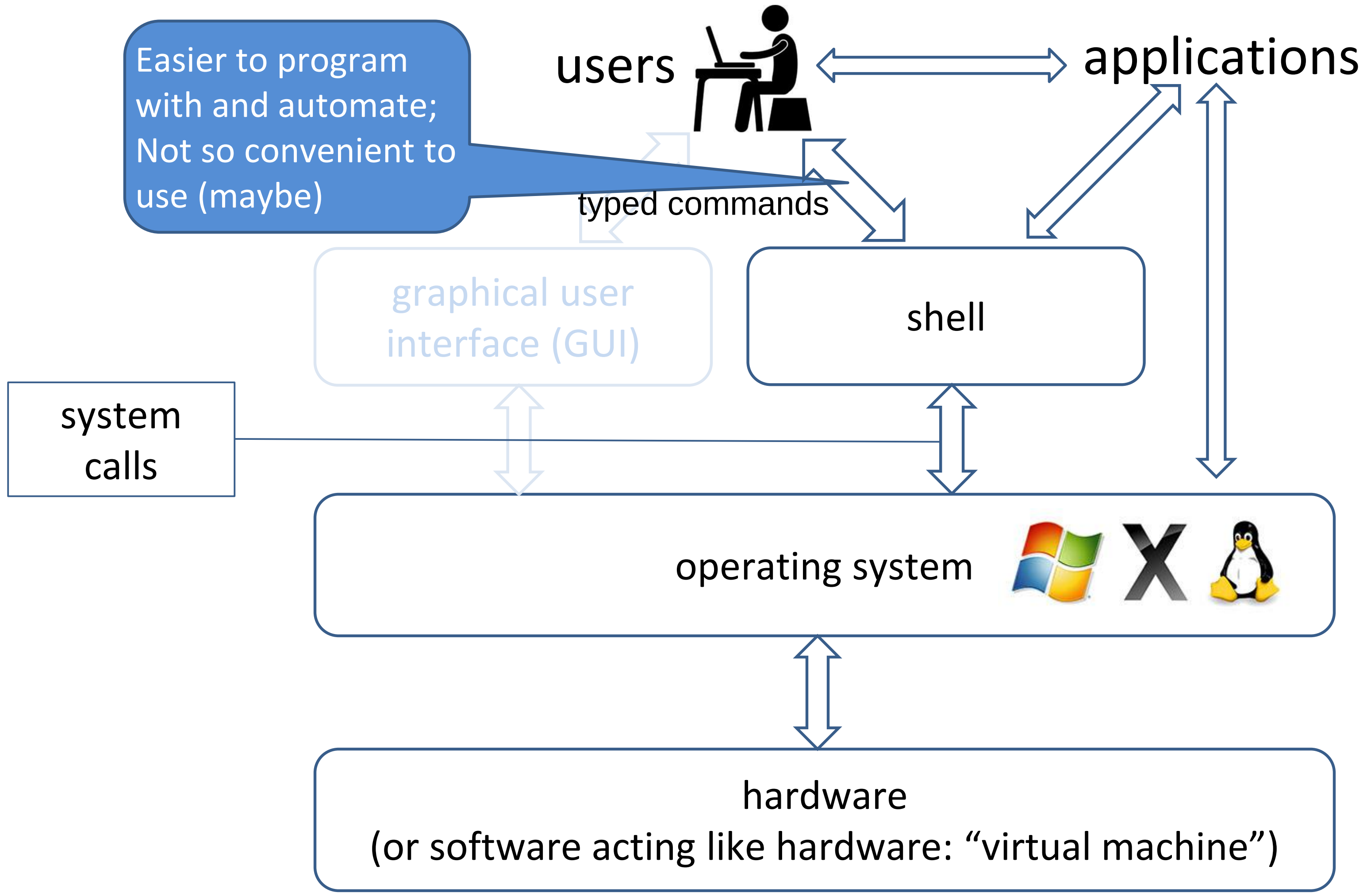
Organization of a computer system



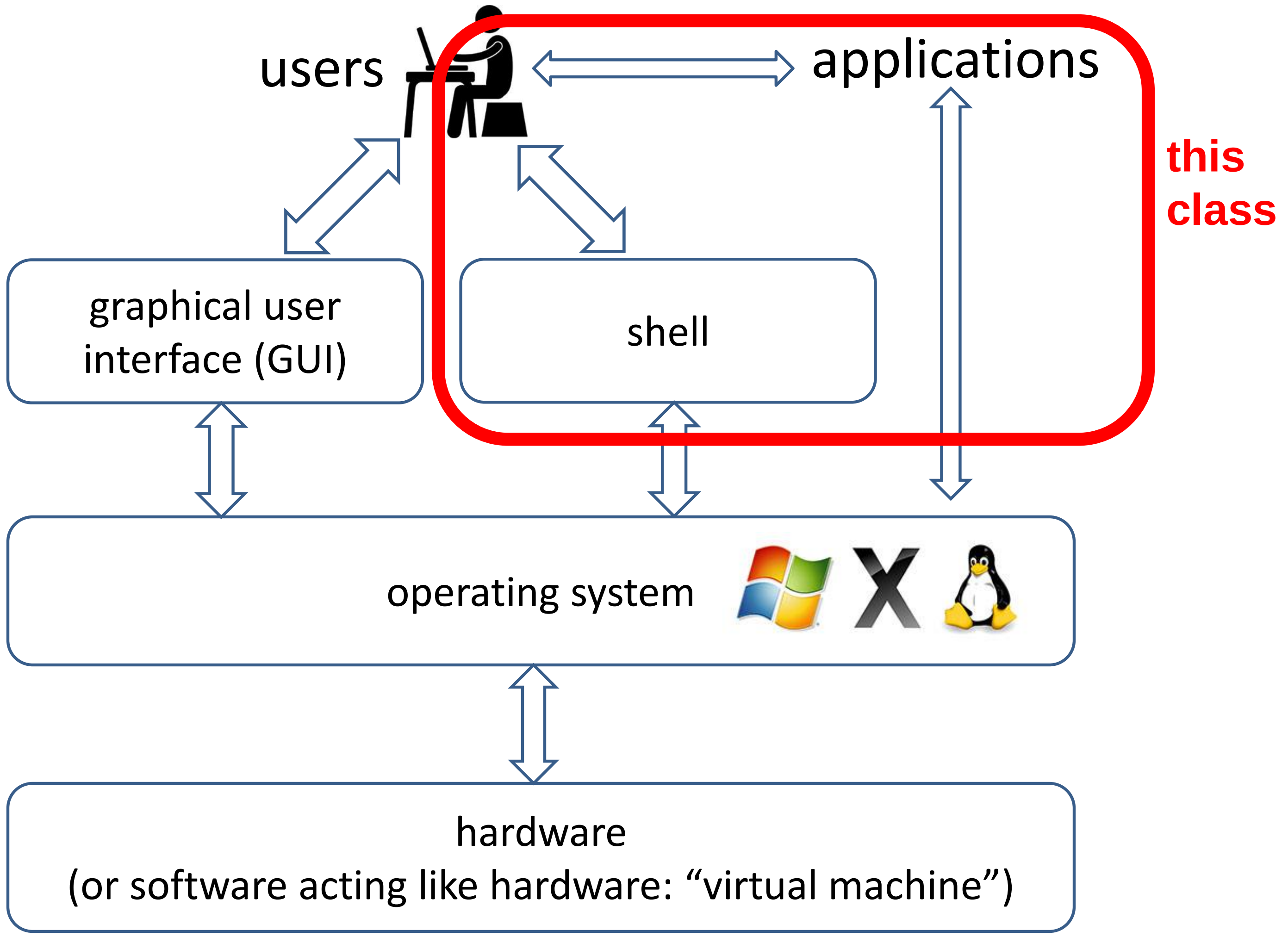
Organization of a computer system


Easier to use;
Not so easy to program with, automate
users
applications
interactive actions
(click, drag, tap, …)
graphical user interface (GUI)
shell
system calls
operating system
hardware
(or software acting like hardware: “virtual machine”)

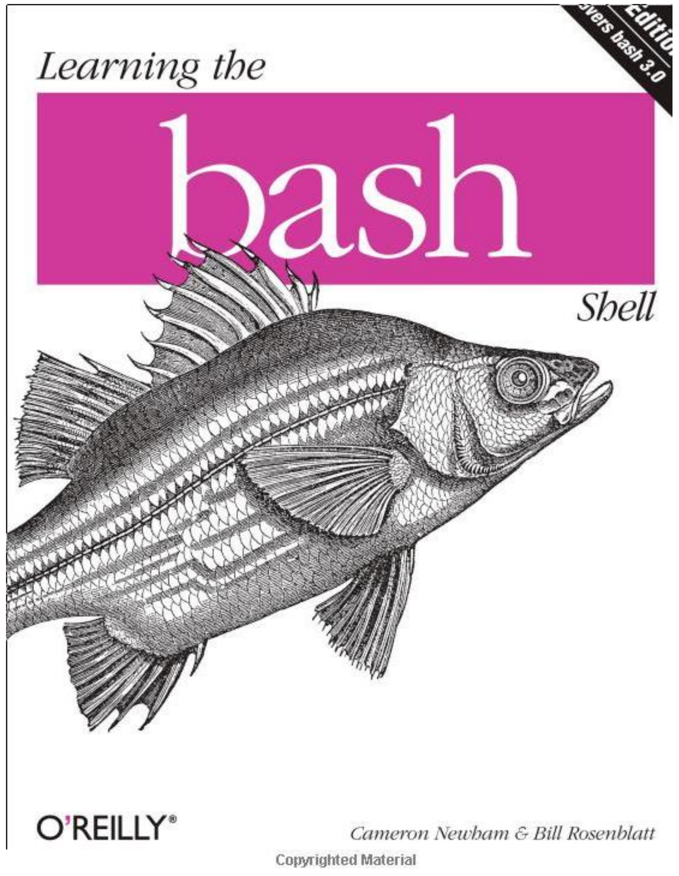
Organization of a computer system


Easier to program with and automate; Not so convenient to use (maybe)
users
applications
typed commands
graphical user interface (GUI)
shell
system calls
operating system
hardware
(or software acting like hardware: “virtual machine”)

Organization of a computer system


users
applications
this class
graphical user interface (GUI)
shell
operating system
hardware
(or software acting like hardware: “virtual machine”)

Reading



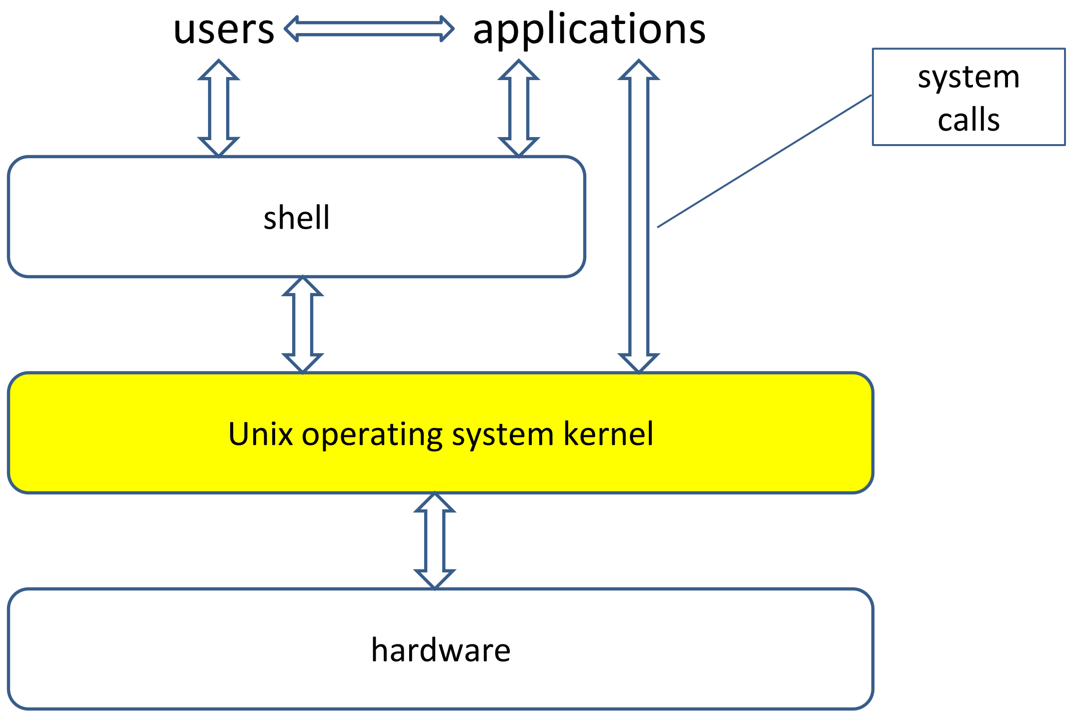
- Chapter 1:
  - Upto "Background Jobs" (page 17)

What is Unix?

- Unix is an operating system
  - sits between the hardware and the user/applications
  - provides high-level abstractions (e.g., files) and services (e.g., multiprogramming)

- Linux:
  - a “Unix-like” operating system: user-level interface very similar to Unix
  - code base is different from original Unix code

Layers of a Unix system



Unix Commands

- Each command performs [variations of] a single task
  - "options" can be used to modify what a command does
  - different commands can be "glued together" to perform more complex tasks
- Syntax:

  command options arguments

  Examples:

Options can (usually) be combined together: these are equivalent

Command	Options	Arguments
pwd		
cd		/home/debray
ls	-a -l	
ls	-al	/usr/local

Unix Commands

- Each command performs [variations of] a single task
  - "options" can be used to modify what a command does
  - different commands can be "glued together" to perform more complex tasks
- Syntax:

    command *options* *arguments*

    Not always required: may have default values

typical defaults:
- input: stdin
- output: stdout
- directory: current

Examples:

Command	Options	Arguments
pwd		
cd		/home/debray
ls	-a -l	
ls	-al	/usr/local

defaults to current directory

Examples of Unix commands I

- Figuring out one's current directory: **pwd**

- Moving to another directory: **cd** *targetdir*

  Examples:

cd /	move to the root of the file system
cd ~ (also: just "**cd**" by itself)	move to one's home directory
cd /usr/local/src	move to /usr/local/src
cd ../..	move up two levels

Examples of Unix commands II

- Command: **ls** — *lists the contents of a directory*
  - Examples:

ls	list the files in the current directory ⚠ *won't show files whose names start with '.'*
ls /usr/bin	list the files in the directory /usr/bin
ls -l	give a "long format" listing (provides additional info about files)
ls -a	list all files in the current directory, including those that start with '.'
ls -al /usr/local	give a "long format" listing of all the files (incl. those starting with '.') in /usr/local

Executing commands

Typing a command name at the **bash** prompt and pressing the **ENTER** key causes the command to be executed.

The command's output, if any, is displayed on the screen. Examples:

```
% hostname
lectura.cs.arizona.edu
% whoami
eanson
% true
% date
Sat Aug 15 18:54:39 MST 2015
% ps
  PID TTY          TIME CMD
22758 pts/18   00:00:00 bash
30245 pts/18   00:00:00 ps
```

Command-line arguments

Most commands accept one or more *arguments*:

```
% cal 9 2015
   September 2015
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

% echo Hello, world!
Hello, world!

% factor 223092870
223092870: 2 3 5 7 11 13 17 19 23
```

Command-line options

Many commands accept *options* that adjust the behavior of the command.

Options almost always begin with a `'-'` (minus sign). Options are usually specified immediately following the command. For most programs the ordering of options is not significant but that is a convention, not a rule.

Examples:

```
% date
Thu Jan 13 02:19:20 MST 2005

% date -u
Thu Jan 13 09:19:22 UTC 2005

% wc Hello.java
      5      14     127 Hello.java

% wc -l -w Hello.java
      5      14 Hello.java
```

We can say that **`wc -l -w Hello.java`** has two options and one *operand*.

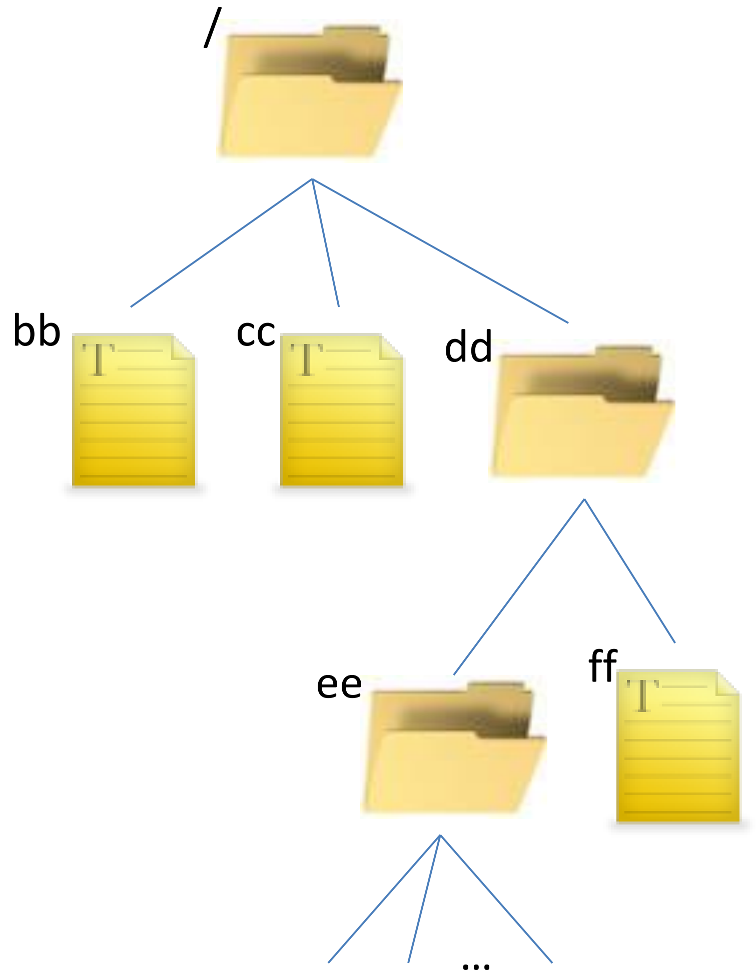
Options, continued

Whitespace is often significant in command lines. For example, the following commands are all invalid: (Try them!)

```
% date-u

% wc -l-w Hello.java

% wc -- notes Hello.java
```

The file system

- A *file* is basically a sequence of bytes
- Collections of files are grouped into *directories* (≈ folders)
- A directory is itself a file
  - ➡ file system has a hierarchical structure (i.e., like a tree)
    - the root is referred to as "/"

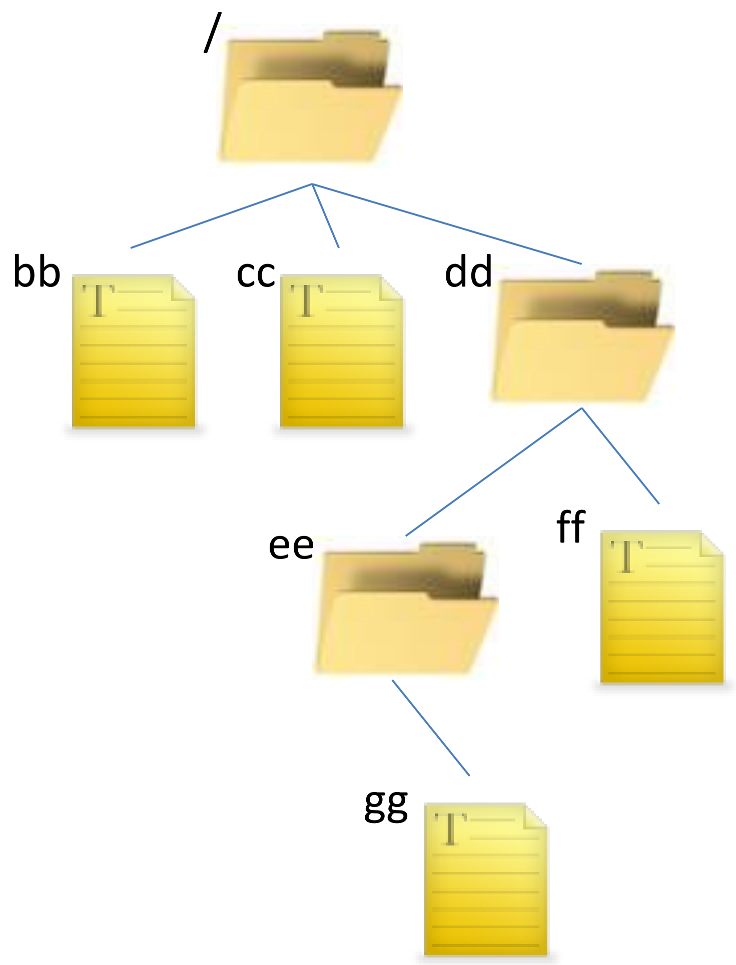


“Everything is a file”

- In Unix, everything looks like a file:
  - documents stored on disk
  - directories
  - inter-process communication
  - network connections
  - devices (printers, graphics cards, interactive terminals, …)
- They are accessed in a uniform way:
  - consistent API (e.g., read, write, open, close, …)
  - consistent naming scheme (e.g., /home/debray, /dev/cdrom)

Referring to files: Absolute Paths

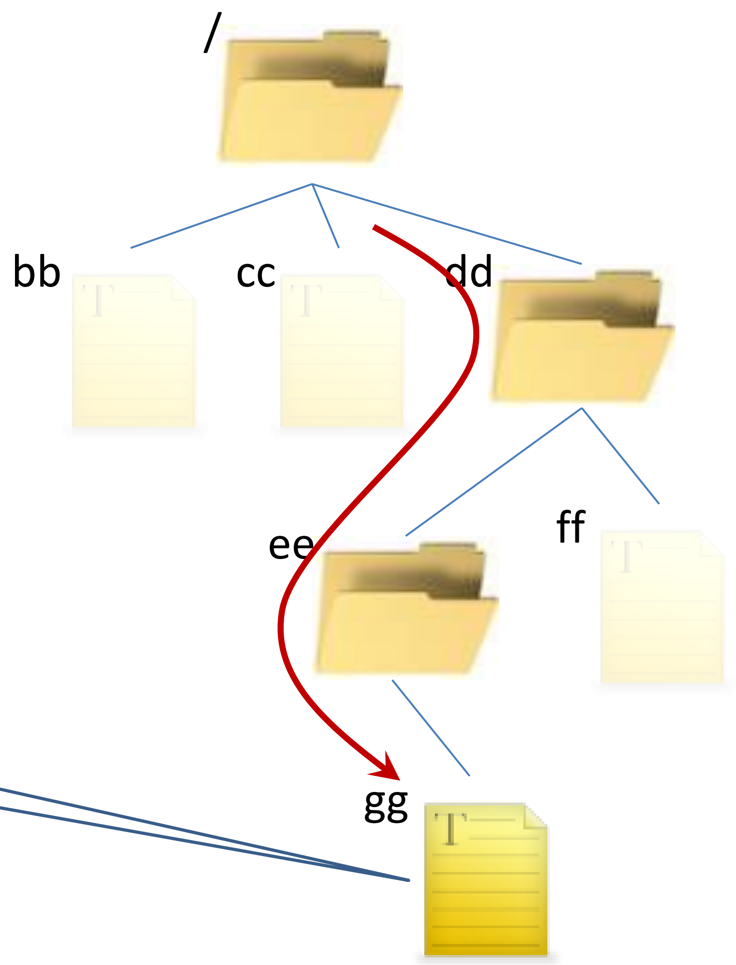
- An *absolute path* specifies how to get to a file starting at the file system root
  - list the directories on the path from the root (“/”), separated by “/”



Referring to files: Absolute Paths

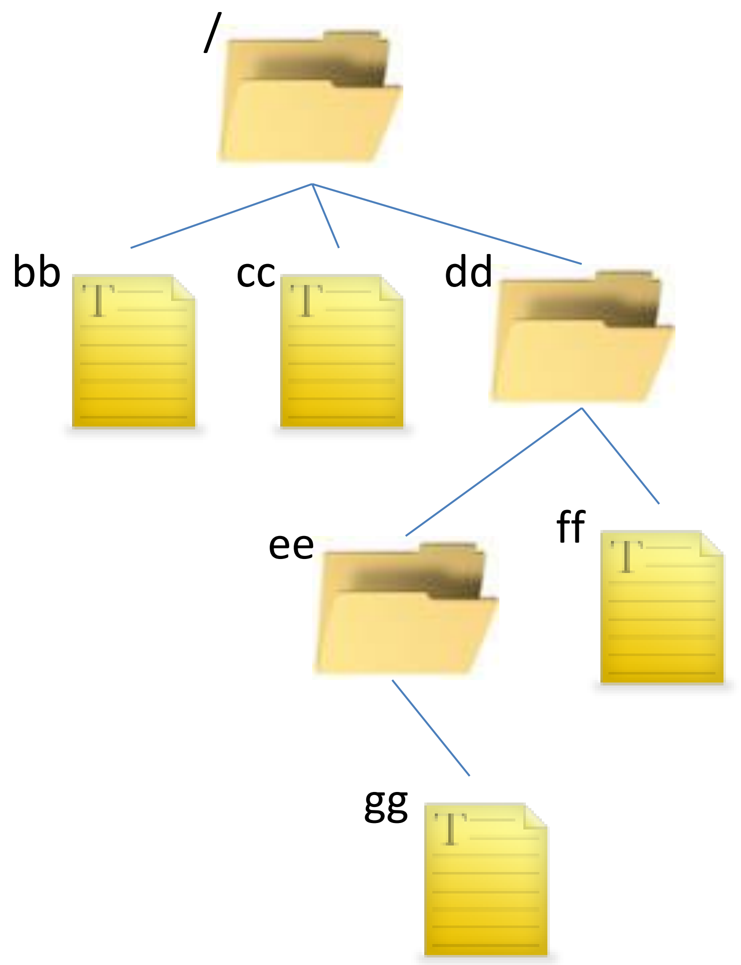
- An *absolute path* specifies how to get to a file starting at the file system root
  - list the directories on the path from the root (“/”), separated by “/”

absolute path: **/dd/ee/gg**



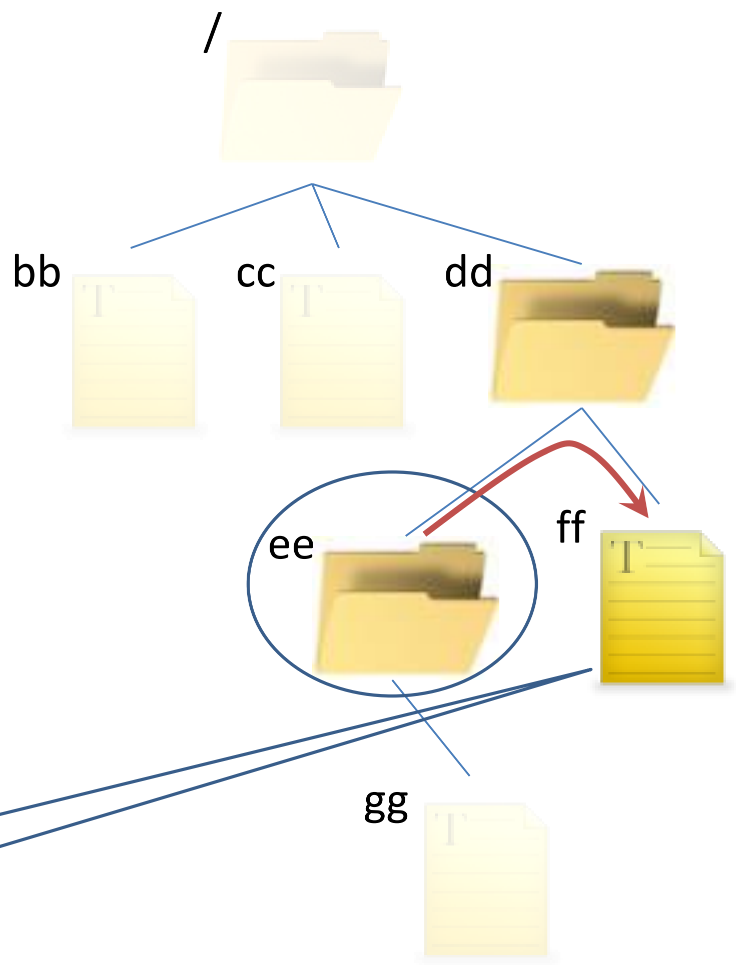
Referring to Files: Relative Paths

- Typically we have a notion of a "*current directory*"
- A *relative path* specifies how to get to a file starting from the current directory
  - '..' means "move up one level"
  - '.' means current directory
  - list the directories on the path separated by "/"



Referring to files: Relative Paths

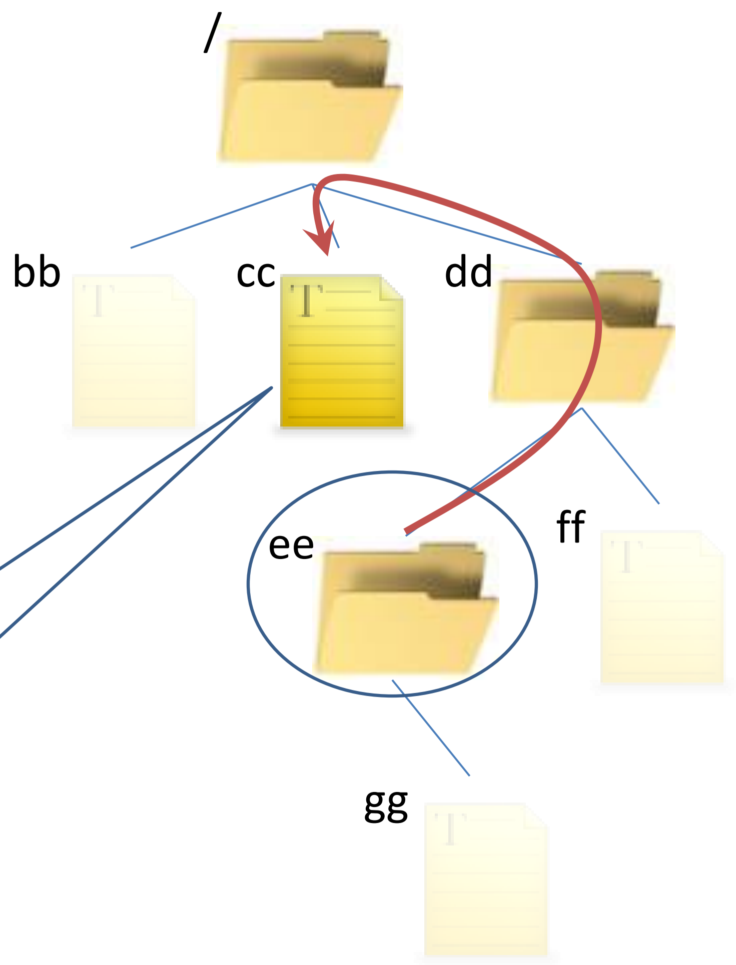
- Typically we have a notion of a “*current directory*”
- A *relative path* specifies how to get to a file starting from the current directory
  - ‘**..**’ means “move up one level”
  - ‘**.**’ means current directory
  - list the directories on the path separated by “/”



Example:
ff relative to **ee** is: **../ff**

Referring to files: Relative Paths

- Typically we have a notion of a "*current directory*"
- A *relative path* specifies how to get to a file starting from the current directory
  - '..' means "move up one level"
  - '.' means current directory
  - list the directories on the path separated by "/"



Example:
cc relative to **ee** is: **../../cc**

Home directories

- Each user has a “home directory”
  - specified when the account is created
  - given in the file **/etc/passwd**

- When you log in, your current directory is your home directory

- Notational shorthand:
  - one’s own home directory: ~
  - some other user **joe**’s home directory: **~joe**

Some commands for dealing with files

- **pwd**
  - *print the name of the current/working directory*
- **ls** [*file*]
  - list a directory contents
- **cd** [*dir*]
  - change the current/working directory
- **cp** *file*$_1$ *file*$_2$
  - copy *file*$_1$ to *file*$_2$
- **vi** [file]
  - the vi editor
- **vimtutor**
  - a tutorial for using vi

Input and output

- Data are read from and written to i/o *streams*
- There are three predefined streams:

    stdin : "standard input" – usually, keyboard input

    stdout : "standard output" – usually, the screen

    stderr : "standard error" – for error messages (usually, the screen)

Other streams can be created using system calls (e.g., to read or write a specific file)

I/O Redirection

- Default input/output behavior for commands:
  - **stdin**: keyboard; **stdout**: screen; **stderr**: screen
- We can change this using I/O redirection:

cmd < *file*	redirect **cmd**'s stdin to read from *file*
cmd > *file*	redirect **cmd**'s stdout to *file*
cmd >> *file*	append **cmd**'s stdout to *file*
cmd &> *file*	redirect **cmd**'s stdout and stderr to *file*
$\mathbf{cmd}_1$ | $\mathbf{cmd}_2$	redirect $\mathbf{cmd}_1$'s stdout to $\mathbf{cmd}_2$'s stdin

Combining commands

- The output of one command can be fed to another command as input.
  - Syntax: $\textit{command}_1 \; \underbrace{|}_{\text{"pipe"}} \; \textit{command}_2$

Example:

ls	lists the files in a directory
more *foo*	shows the file *foo* one screenful at a time
ls | more	lists the files in a directory one screenful at a time

How this works:
- **ls** writes its output to its **stdout**
- **more**'s input stream defaults to its **stdin**
- the pipe connects **ls**'s stdout to **more**'s stdin
- the piped commands run "in parallel"

Finding out about commands I

Figuring out which command to use

apropos *keyword*

man –k *keyword*

"searches a set of database files containing short descriptions of system commands for keywords"

- Helpful, but not a panacea:
  - depends on appropriate choice of keywords
    - may require trial and error
  - may return a lot of results to sift through
    - pipe through **more**

Finding out about commands II

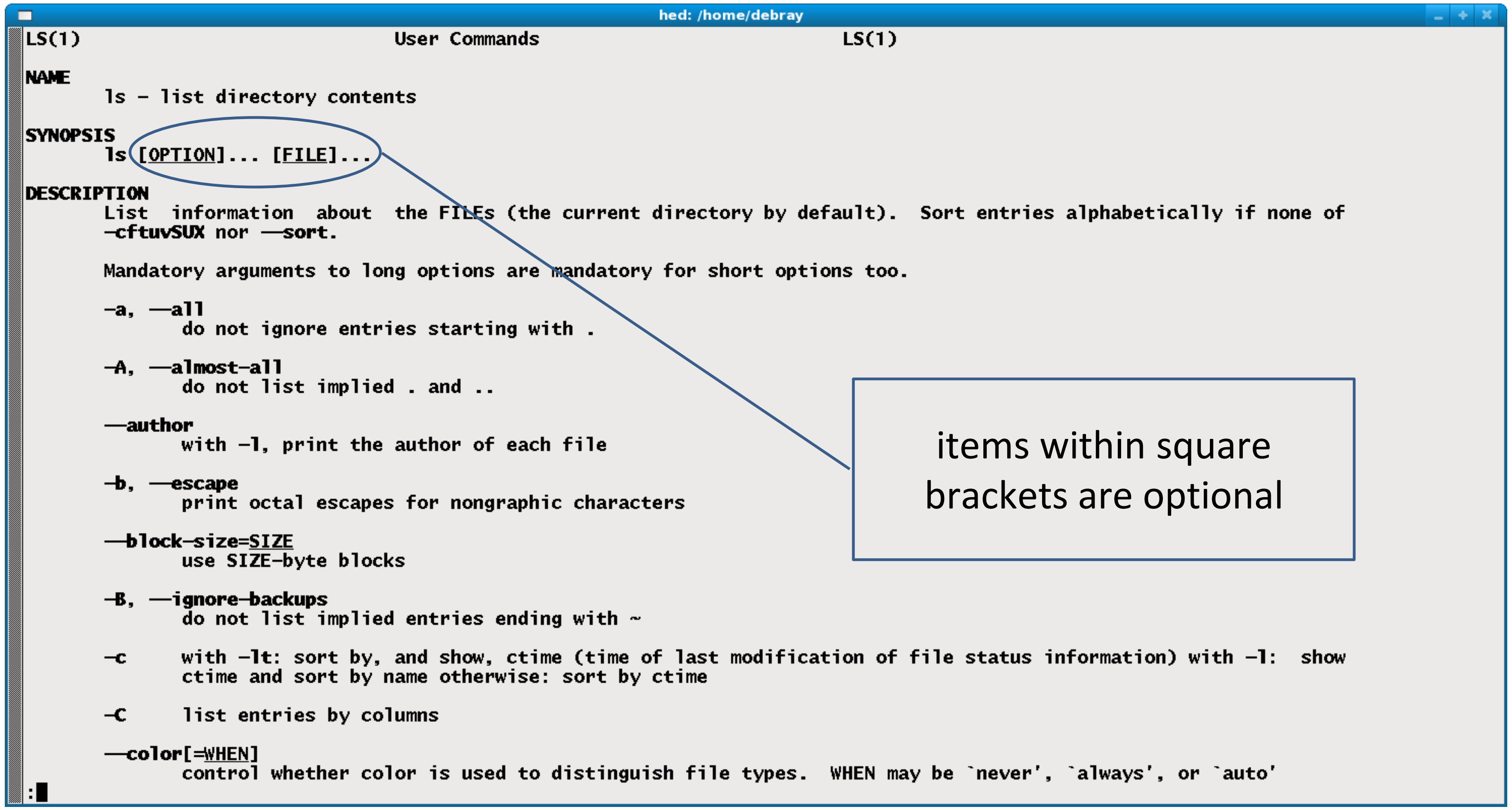
Figuring out how to use a command

man *command*

“displays the on-line manual pages”

- Provides information about command options, arguments, return values, bugs, etc.

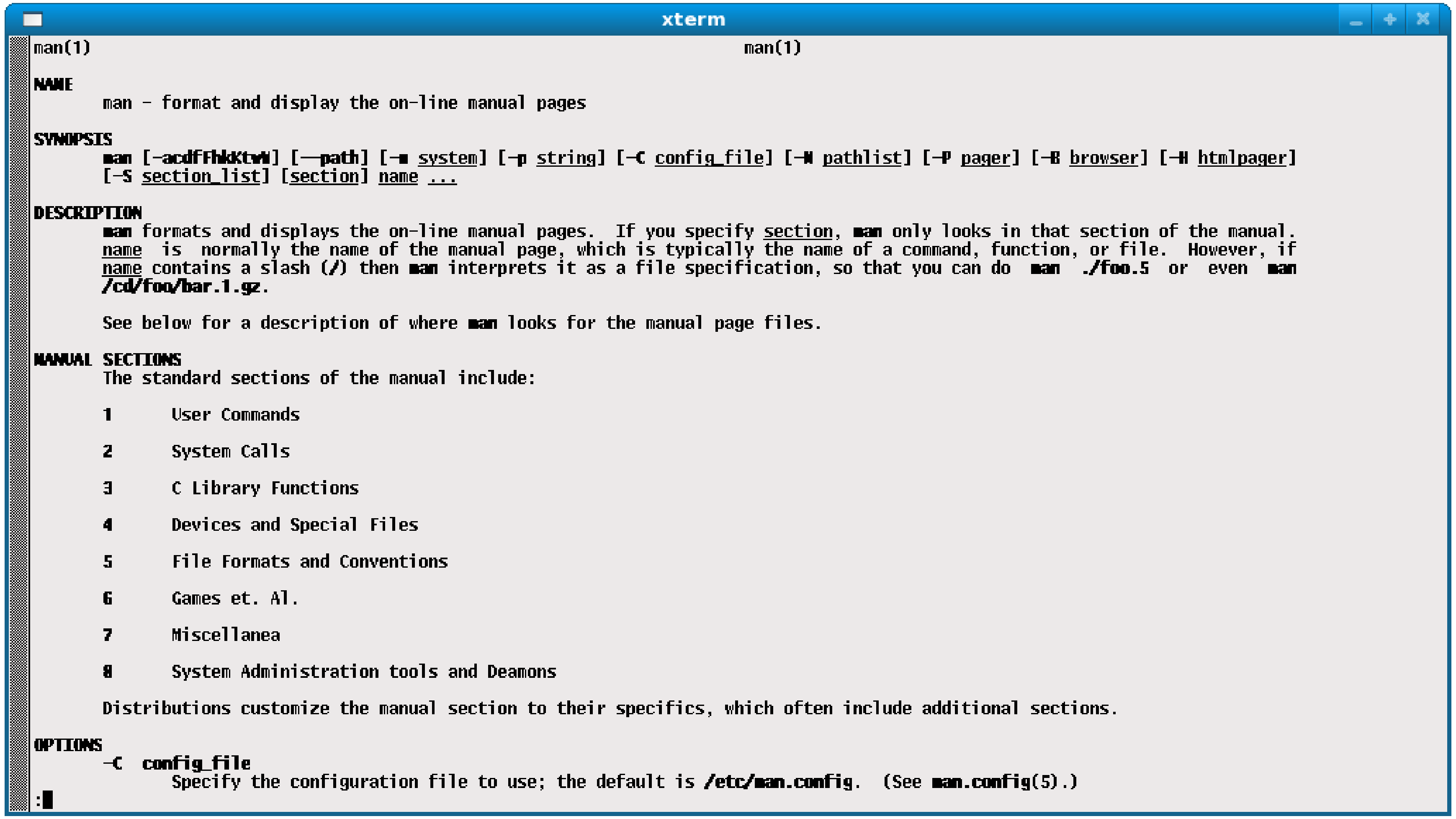
Example: “man ls”



```
LS(1)                         User Commands                        LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).  Sort entries alphabetically if none of
       -cftuvSUX nor --sort.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print octal escapes for nongraphic characters

       --block-size=SIZE
              use SIZE-byte blocks

       -B, --ignore-backups
              do not list implied entries ending with ~

       -c     with -lt: sort by, and show, ctime (time of last modification of file status information) with -l:  show
              ctime and sort by name otherwise: sort by ctime

       -C     list entries by columns

       --color[=WHEN]
              control whether color is used to distinguish file types.  WHEN may be `never', `always', or `auto'
:
```

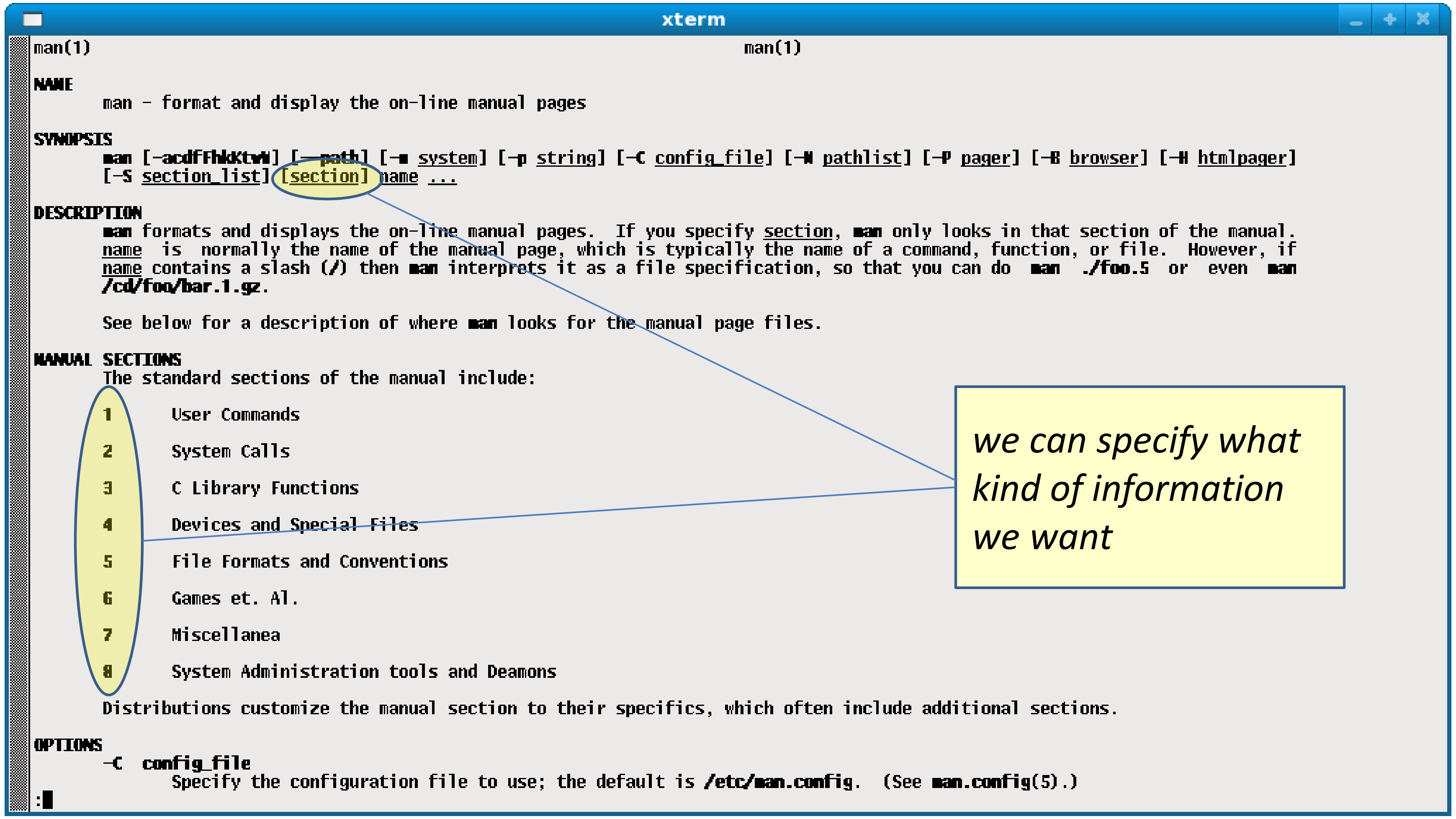
items within square brackets are optional

Example: “man man”



```
man(1)                                                                          man(1)

NAME
       man - format and display the on-line manual pages

SYNOPSIS
       man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file] [-M pathlist] [-P pager] [-B browser] [-H htmlpager]
       [-S section_list] [section] name ...

DESCRIPTION
       man formats and displays the on-line manual pages.  If you specify section, man only looks in that section of the manual.
       name  is  normally the name of the manual page, which is typically the name of a command, function, or file.  However, if
       name contains a slash (/) then man interprets it as a file specification, so that you can do  man  ./foo.5  or  even  man
       /cd/foo/bar.1.gz.

       See below for a description of where man looks for the manual page files.

MANUAL SECTIONS
       The standard sections of the manual include:

       1      User Commands

       2      System Calls

       3      C Library Functions

       4      Devices and Special Files

       5      File Formats and Conventions

       6      Games et. Al.

       7      Miscellanea

       8      System Administration tools and Deamons

       Distributions customize the manual section to their specifics, which often include additional sections.

OPTIONS
       -C  config_file
              Specify the configuration file to use; the default is /etc/man.config.  (See man.config(5).)
:
```

Example: “man man”



```
man(1)                                                                 man(1)

NAME
       man - format and display the on-line manual pages

SYNOPSIS
       man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file] [-M pathlist] [-P pager] [-B browser] [-H htmlpager]
       [-S section_list] [section] name ...

DESCRIPTION
       man formats and displays the on-line manual pages.  If you specify section, man only looks in that section of the manual.
       name  is  normally the name of the manual page, which is typically the name of a command, function, or file.  However, if
       name contains a slash (/) then man interprets it as a file specification, so that you can do  man  ./foo.5  or  even  man
       /cd/foo/bar.1.gz.

       See below for a description of where man looks for the manual page files.

MANUAL SECTIONS
       The standard sections of the manual include:

       1      User Commands

       2      System Calls

       3      C Library Functions

       4      Devices and Special Files

       5      File Formats and Conventions

       6      Games et. Al.

       7      Miscellanea

       8      System Administration tools and Deamons

       Distributions customize the manual section to their specifics, which often include additional sections.

OPTIONS
       -C  config_file
              Specify the configuration file to use; the default is /etc/man.config.  (See man.config(5).)
:
```

we can specify what kind of information we want

Some other useful commands

- **wc** [*file*]
  - *word count*: counts characters, words, and lines in the input
- **grep** *pattern* [*file*]
  - select lines in the input that match *pattern*
- **head** *–n* [*file*]
  - show the first *n* lines of the input
- **tail** *–n* [*file*]
  - show the last *n* lines of the input
- **cp** $file_1$ $file_2$
  - copy $file_1$ to $file_2$
- **mv** $file_1$ $file_2$
  - move $file_1$ to $file_2$

The diff command

- The diff command looks for differences in files
- You will probably want to know this command since it will be used in grading.
- Down the line you will be doing programming homework.
- An correct executable will be given.
- You'll want to use I/O redirection and the diff command to make sure the output of your code matches the output of the given program.

Odds and ends

- Here are some handy options for `ls`:
  - `-t` Sort by modification time instead of alphabetically.
  - `-h` Show sizes with human-readable units like `K`, `M`, and `G`.
  - `-r` Reverse the order of the sort.
  - `-S` Sort by file size
  - `-d` By default, when an argument is a directory, `ls` operates on the entries contained in that directory. `-d` says to operate on the directory itself. Try "`ls -l .`" and "`ls -ld .`".
  - -R Recursively list all the subdirectories.
  - There are many more and you might want to look at the man page and play

Odds and ends

- Two handy options for `cp`:
  - `-R`		Recursively copy an entire directory tree
  - `-p`		Preserve file permissions, ownerships, and timestamps

Odds and Ends

- Many non-alphanumeric characters have special meaning to shells.

- Characters that have special meaning are often called *metacharacters*. Here are the **`bash`** metacharacters:

- `~ ` ! # $ & * ( ) \ | { } [] ; ' " < > ?`

Odds and Ends

- Many non-alphanumeric characters have special meaning to shells.

- Characters that have special meaning are often called *metacharacters*. Here are the **`bash`** metacharacters:

- `~ ` ` ! # $ & * ( ) \ | { } [] ; ' " < > ?`
- If an argument has metacharacters or whitespace we suppress their special meaning by enclosing the argument in quotes.

Odds and Ends

- Many non-alphanumeric characters have special meaning to shells.
- Characters that have special meaning are often called *metacharacters*. Here are the **`bash`** metacharacters:
- `~ ` ` ! # $ & * ( ) \ | { } [] ; ' " < > ?`
- If an argument has metacharacters or whitespace we suppress their special meaning by enclosing the argument in quotes.
- An alternative to wrapping with quotes is to use a backslash to "escape" each metacharacter.

Pattern matching in the shell

- Some metacharacters are used as patterns in shell commands, e.g.:

*****	matches any string
[…]	matches any one of the characters within braces

Example:

ls b*c	list files that begin with **b** and end with **c**
ls a[xyz]*	list files starting with **a** followed by **x**, **y**, or **z**
ls *.pdf	list files ending with "**.pdf**"

Command-line editing and shortcuts

bash supports simple command line recall and editing with the "arrow keys" but many control-key and escape sequences have meaning too. Here are a few:

`^A/^E`	Go to start/end of line.
`^W`	Erase the last "word".
`^U`	Erase whole line. (`^C` works, too.)
`^R`	Do incremental search through previous commands.
`ESC-f/b`	Go forwards/backwards a word. (Two keystrokes: **ESC**, then **f**)
`ESC-.`	Insert last word on from last command line. (Very handy!)

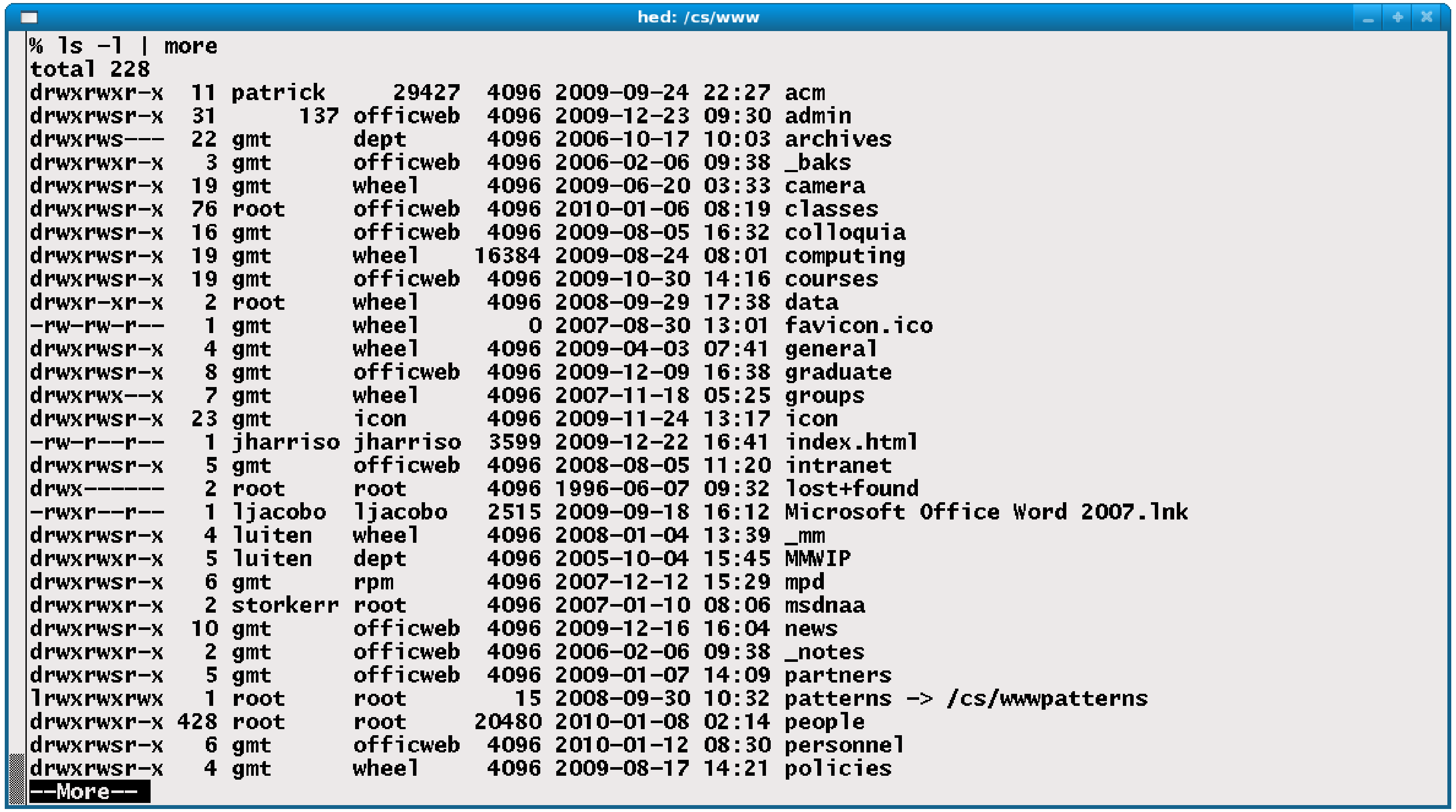
bash also does command and filename completion with **TAB**:

Hit **TAB** to complete to longest unique string.

If a "beep", hit **TAB** a second time to see alternatives.

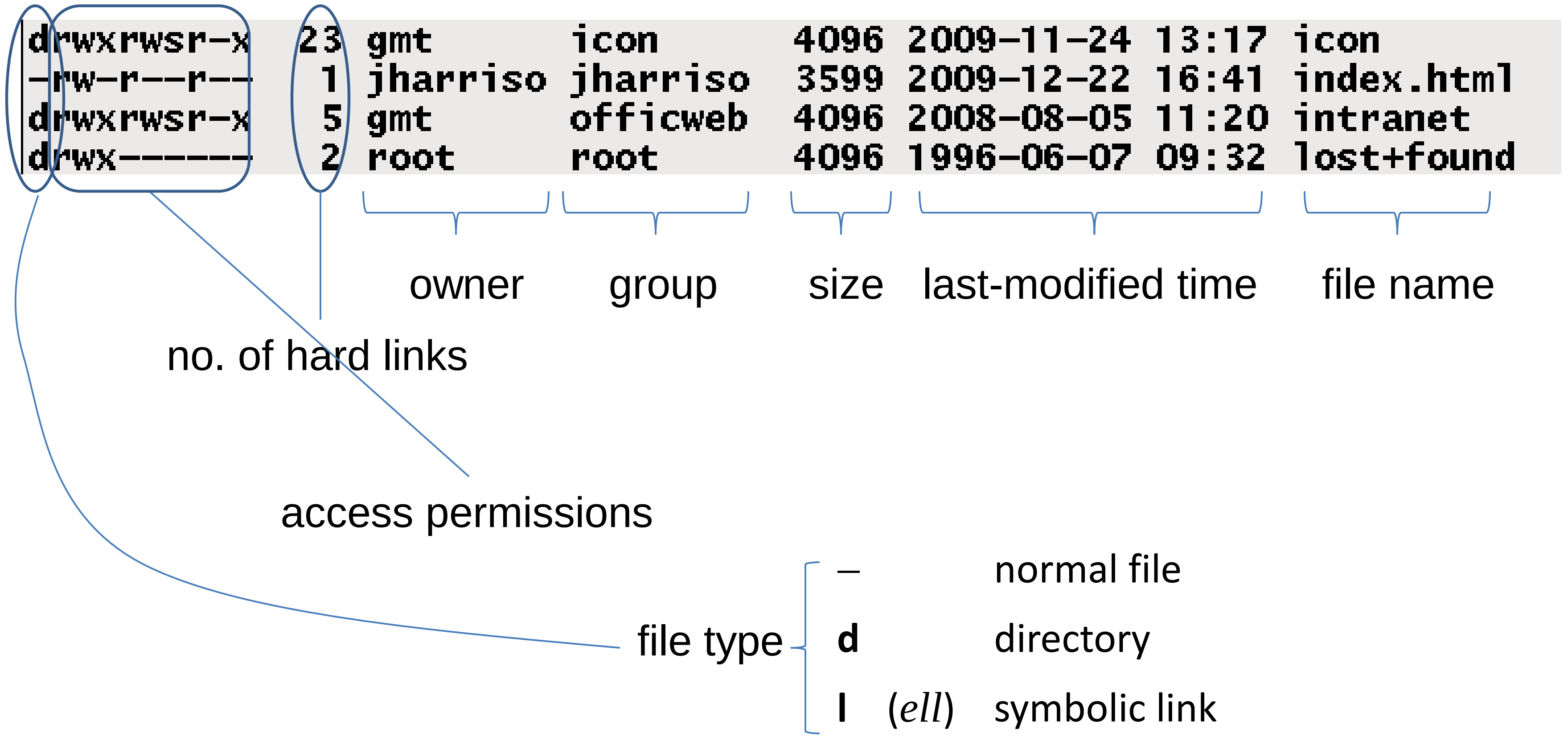
Getting more information about files

- ls –l : provides additional info about files


hed: /cs/www

```
% ls -l | more
total 228
drwxrwxr-x  11 patrick     29427  4096 2009-09-24 22:27 acm
drwxrwsr-x  31        137 officweb  4096 2009-12-23 09:30 admin
drwxrws---  22 gmt      dept      4096 2006-10-17 10:03 archives
drwxrwxr-x   3 gmt      officweb  4096 2006-02-06 09:38 _baks
drwxrwsr-x  19 gmt      wheel     4096 2009-06-20 03:33 camera
drwxrwsr-x  76 root     officweb  4096 2010-01-06 08:19 classes
drwxrwsr-x  16 gmt      officweb  4096 2009-08-05 16:32 colloquia
drwxrwsr-x  19 gmt      wheel    16384 2009-08-24 08:01 computing
drwxrwsr-x  19 gmt      officweb  4096 2009-10-30 14:16 courses
drwxr-xr-x   2 root     wheel     4096 2008-09-29 17:38 data
-rw-rw-r--   1 gmt      wheel        0 2007-08-30 13:01 favicon.ico
drwxrwsr-x   4 gmt      wheel     4096 2009-04-03 07:41 general
drwxrwsr-x   8 gmt      officweb  4096 2009-12-09 16:38 graduate
drwxrwx--x   7 gmt      wheel     4096 2007-11-18 05:25 groups
drwxrwsr-x  23 gmt      icon      4096 2009-11-24 13:17 icon
-rw-r--r--   1 jharriso jharriso  3599 2009-12-22 16:41 index.html
drwxrwsr-x   5 gmt      officweb  4096 2008-08-05 11:20 intranet
drwx------   2 root     root      4096 1996-06-07 09:32 lost+found
-rwxr--r--   1 ljacobo  ljacobo   2515 2009-09-18 16:12 Microsoft Office Word 2007.lnk
drwxrwsr-x   4 luiten   wheel     4096 2008-01-04 13:39 _mm
drwxrwxr-x   5 luiten   dept      4096 2005-10-04 15:45 MMWIP
drwxrwsr-x   6 gmt      rpm       4096 2007-12-12 15:29 mpd
drwxrwxr-x   2 storkerr root      4096 2007-01-10 08:06 msdnaa
drwxrwsr-x  10 gmt      officweb  4096 2009-12-16 16:04 news
drwxrwxr-x   2 gmt      officweb  4096 2006-02-06 09:38 _notes
drwxrwsr-x   5 gmt      officweb  4096 2009-01-07 14:09 partners
lrwxrwxrwx   1 root     root        15 2008-09-30 10:32 patterns -> /cs/wwwpatterns
drwxrwxr-x 428 root     root     20480 2010-01-08 02:14 people
drwxrwsr-x   6 gmt      officweb  4096 2010-01-12 08:30 personnel
drwxrwsr-x   4 gmt      wheel     4096 2009-08-17 14:21 policies
--More--
```

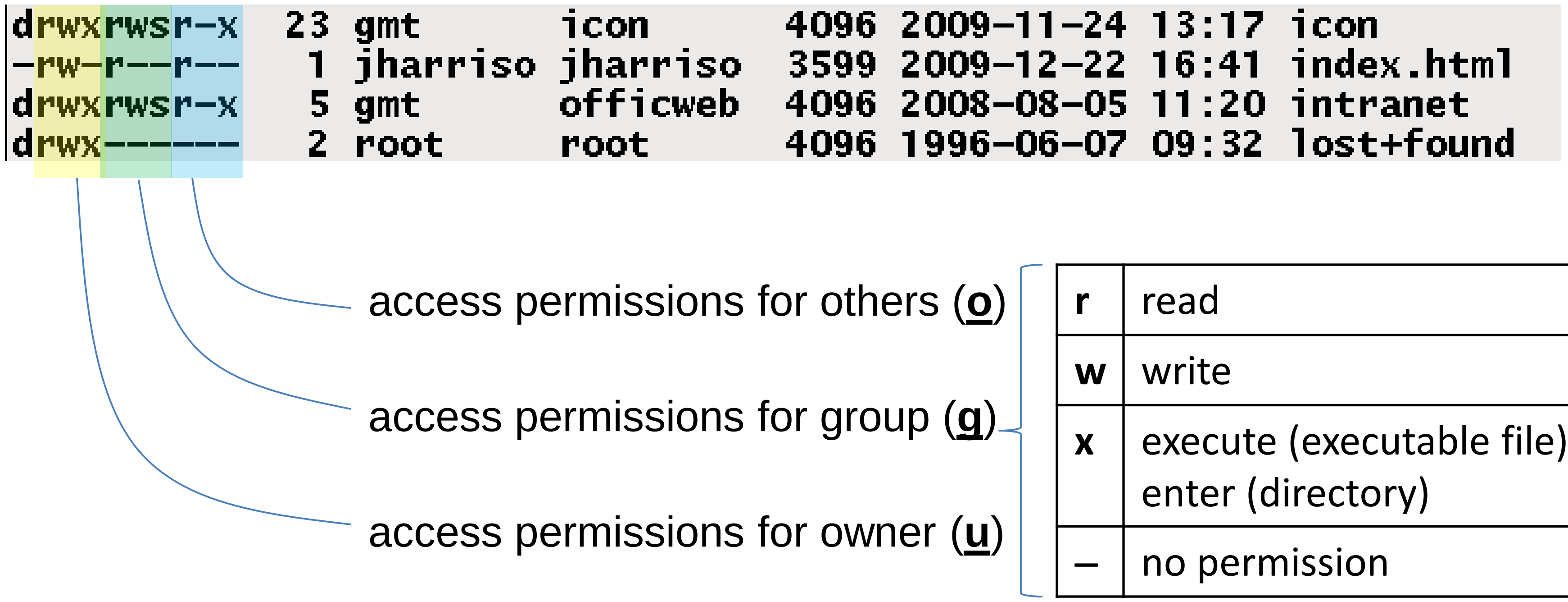
Getting more information about files… (1)



```
drwxrwsr-x  23 gmt      icon      4096 2009-11-24 13:17 icon
-rw-r--r--   1 jharriso jharriso  3599 2009-12-22 16:41 index.html
drwxrwsr-x   5 gmt      officweb  4096 2008-08-05 11:20 intranet
drwx------   2 root     root      4096 1996-06-07 09:32 lost+found
```

–		normal file
d		directory
l	(*ell*)	symbolic link

File access permissions



```
drwxrwsr-x   23 gmt      icon      4096 2009-11-24 13:17 icon
-rw-r--r--    1 jharriso jharriso  3599 2009-12-22 16:41 index.html
drwxrwsr-x    5 gmt      officweb  4096 2008-08-05 11:20 intranet
drwx------    2 root     root      4096 1996-06-07 09:32 lost+found
```

access permissions for others (**o**)
access permissions for group (**g**)
access permissions for owner (**u**)

r	read
w	write
x	execute (executable file) enter (directory)
–	no permission

Changing file access permissions

Command:

chmod $\boxed{\mathit{who}}\pm\boxed{\mathit{what}}\quad \mathit{file}_1\quad \mathit{file}_2 \ldots \mathit{file}_n$

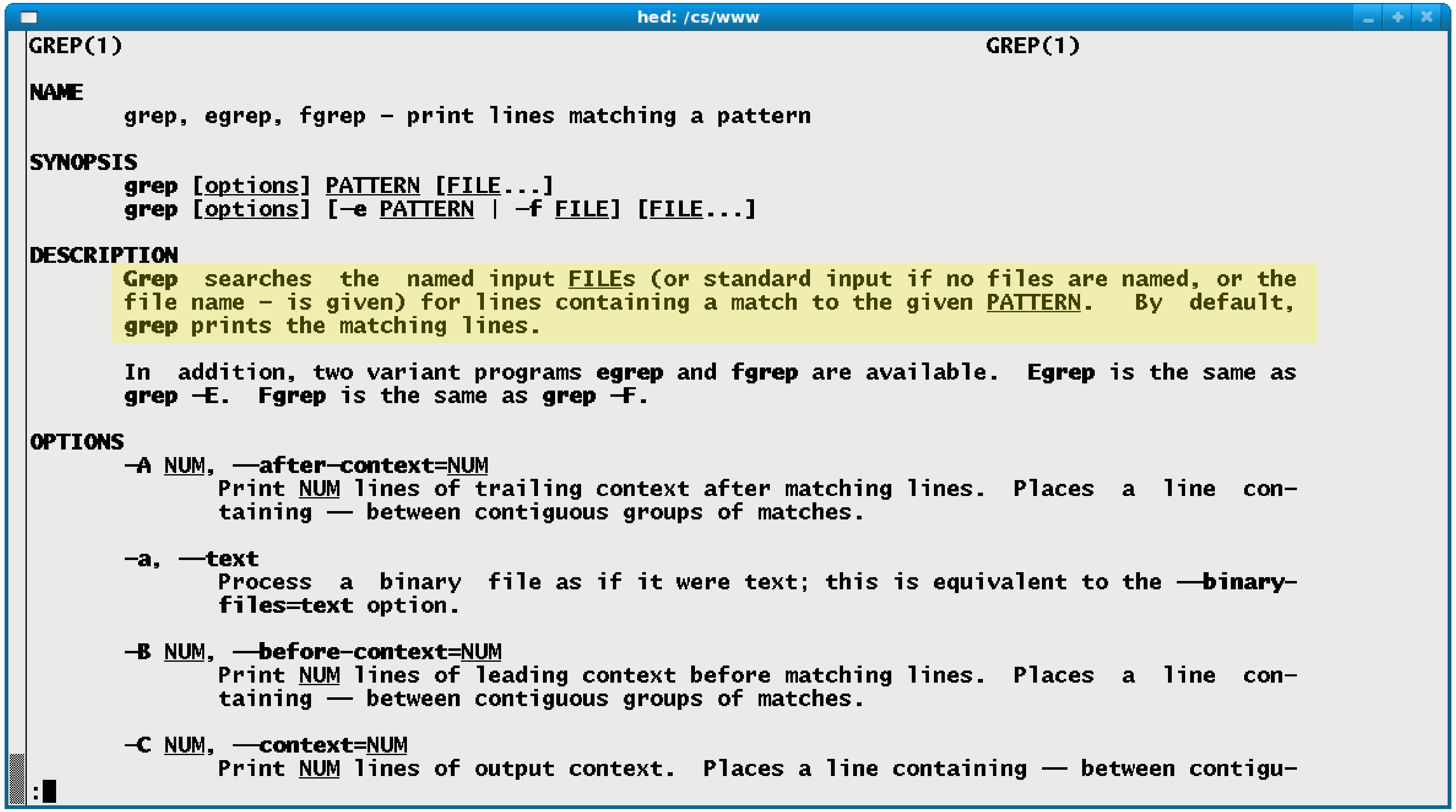
$\in \{\mathrm{r}, \mathrm{w}, \mathrm{x}\}$

$\in \{\mathrm{a}, \mathrm{u}, \mathrm{g}, \mathrm{o}\}$

Example:

chmod u-w foo	remove write permission for user on file foo
chmod g+rx bar	give read and execute permission to group for bar
chmod o-rwx *.doc	remove all access permissions for “other users” (i.e., not owner or group members) for *.doc files
chmod a+rw p*	give read and write permission to everyone for all files starting with p

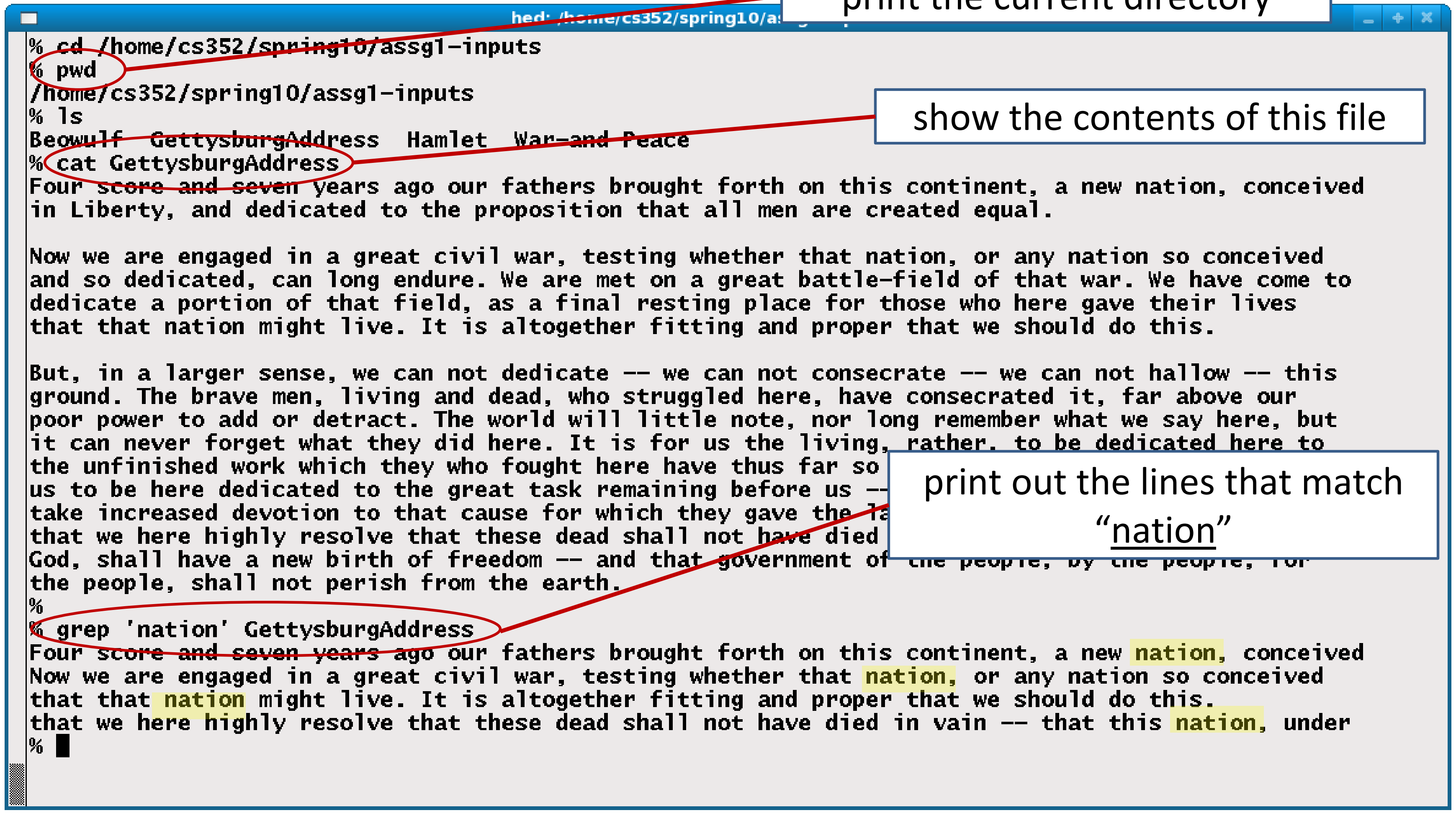
Pattern matching: grep



hed: /cs/www

```
GREP(1)                                                               GREP(1)

NAME
       grep, egrep, fgrep - print lines matching a pattern

SYNOPSIS
       grep [options] PATTERN [FILE...]
       grep [options] [-e PATTERN | -f FILE] [FILE...]

DESCRIPTION
       Grep  searches  the  named input FILEs (or standard input if no files are named, or the
       file name - is given) for lines containing a match to the given PATTERN.   By  default,
       grep prints the matching lines.

       In  addition, two variant programs egrep and fgrep are available.  Egrep is the same as
       grep -E.  Fgrep is the same as grep -F.

OPTIONS
       -A NUM, --after-context=NUM
              Print NUM lines of trailing context after matching lines.  Places  a  line  con-
              taining -- between contiguous groups of matches.

       -a, --text
              Process  a  binary  file as if it were text; this is equivalent to the --binary-
              files=text option.

       -B NUM, --before-context=NUM
              Print NUM lines of leading context before matching lines.  Places  a  line  con-
              taining -- between contiguous groups of matches.

       -C NUM, --context=NUM
              Print NUM lines of output context.  Places a line containing -- between contigu-
:
```

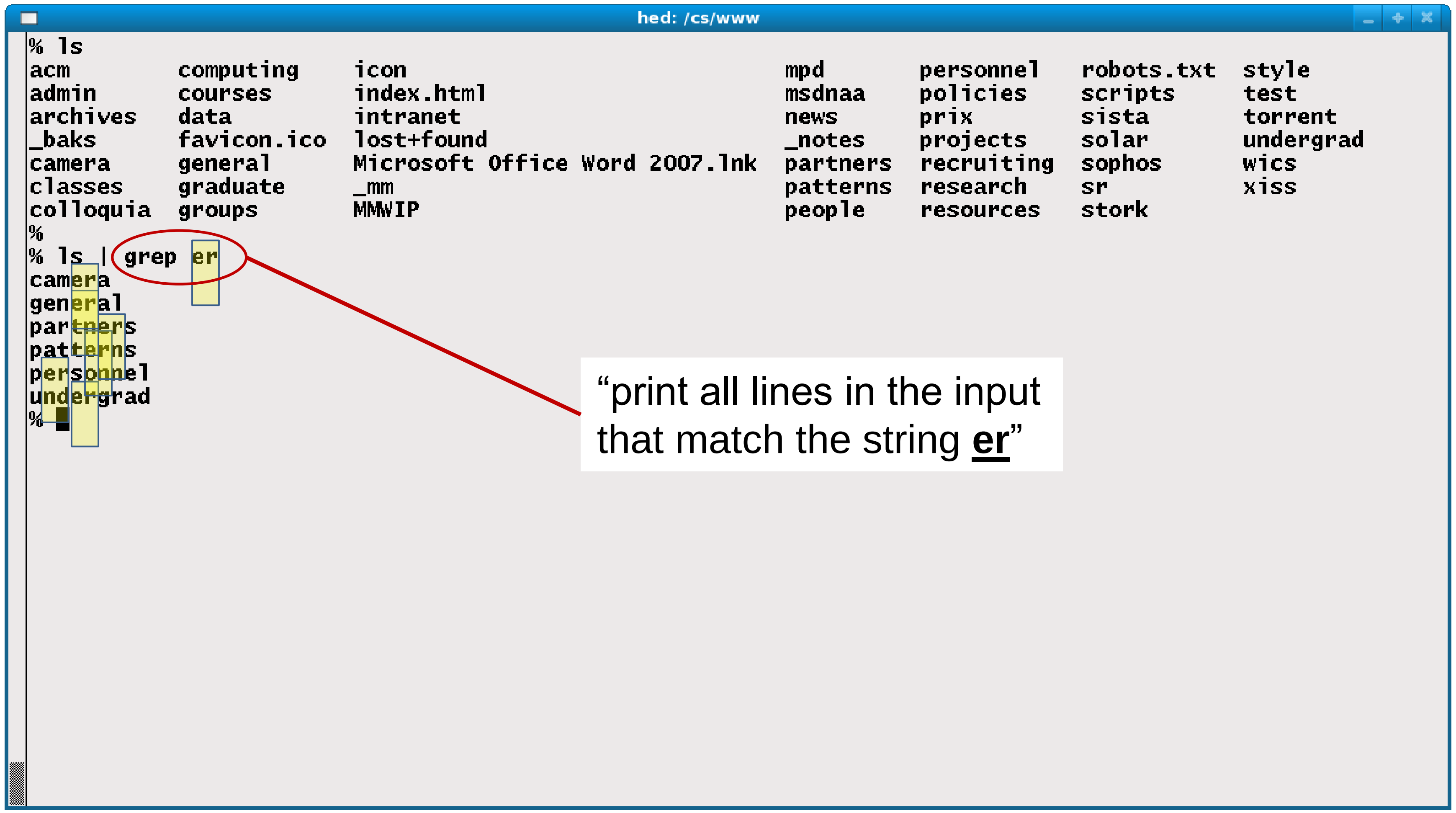
Pattern matching: grep… (1)



```
hed: /home/cs352/spring10/as
% cd /home/cs352/spring10/assg1-inputs
% pwd
/home/cs352/spring10/assg1-inputs
% ls
Beowulf  GettysburgAddress  Hamlet  War-and-Peace
% cat GettysburgAddress
Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived
in Liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived
and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to
dedicate a portion of that field, as a final resting place for those who here gave their lives
that that nation might live. It is altogether fitting and proper that we should do this.

But, in a larger sense, we can not dedicate -- we can not consecrate -- we can not hallow -- this
ground. The brave men, living and dead, who struggled here, have consecrated it, far above our
poor power to add or detract. The world will little note, nor long remember what we say here, but
it can never forget what they did here. It is for us the living, rather, to be dedicated here to
the unfinished work which they who fought here have thus far so
us to be here dedicated to the great task remaining before us --
take increased devotion to that cause for which they gave the la
that we here highly resolve that these dead shall not have died
God, shall have a new birth of freedom -- and that government of the people, by the people, for
the people, shall not perish from the earth.
%
% grep 'nation' GettysburgAddress
Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived
Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived
that that nation might live. It is altogether fitting and proper that we should do this.
that we here highly resolve that these dead shall not have died in vain -- that this nation, under
% 
```

print the current directory
show the contents of this file
print out the lines that match “nation”

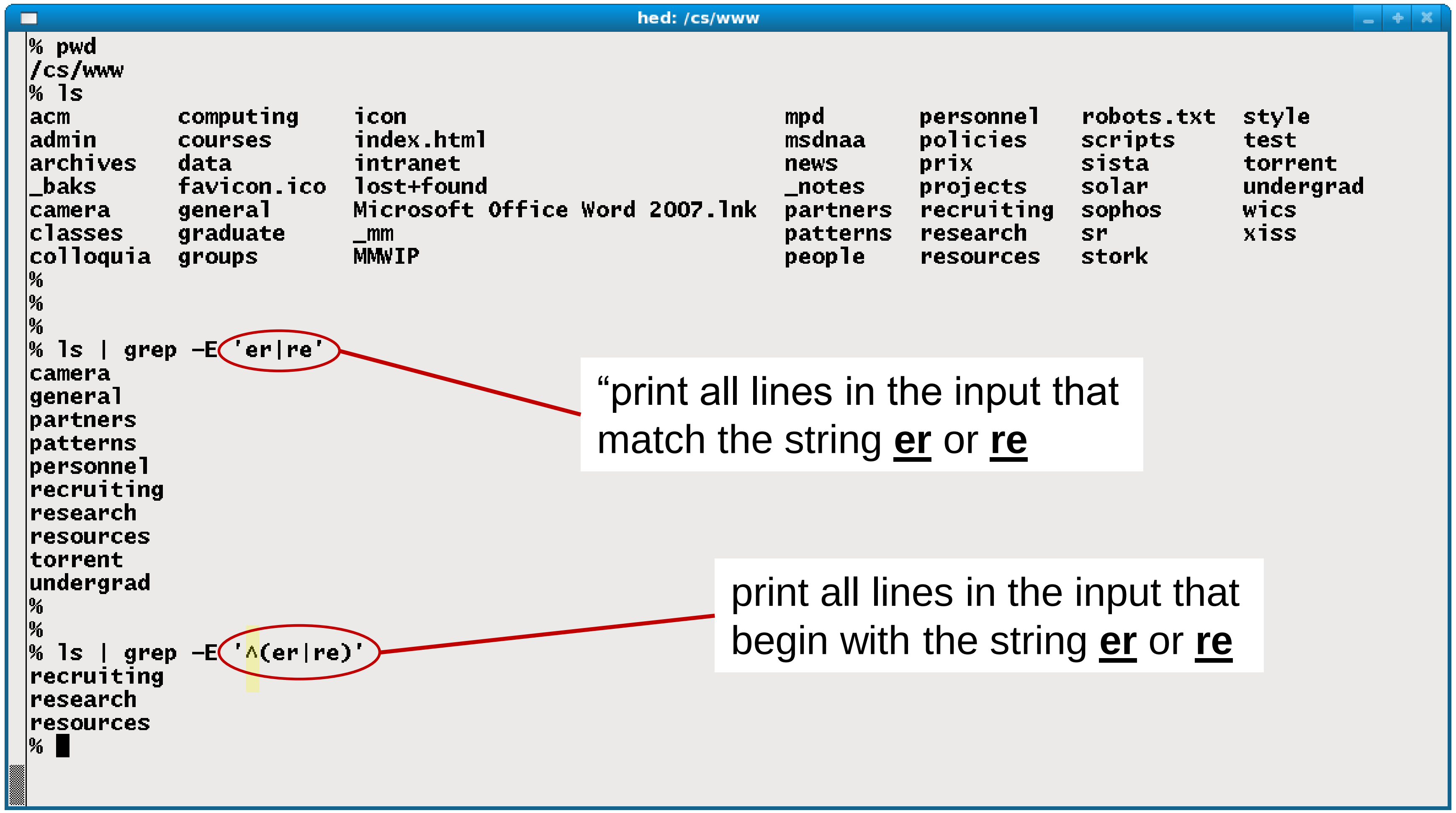
Pattern matching: grep… (2)



```
% ls
acm         computing   icon                           mpd       personnel   robots.txt  style
admin       courses     index.html                     msdnaa    policies    scripts     test
archives    data        intranet                       news      prix        sista       torrent
_baks       favicon.ico lost+found                     _notes    projects    solar       undergrad
camera      general     Microsoft Office Word 2007.lnk partners  recruiting  sophos      wics
classes     graduate    _mm                            patterns  research    sr          xiss
colloquia   groups      MMWIP                          people    resources   stork
%
% ls | grep er
camera
general
partners
patterns
personnel
undergrad
%
```

"print all lines in the input that match the string **er**"

Pattern matching: grep… (3)



```
% pwd
/cs/www
% ls
acm         computing   icon                            mpd       personnel   robots.txt  style
admin       courses     index.html                      msdnaa    policies    scripts     test
archives    data        intranet                        news      prix        sista       torrent
_baks       favicon.ico lost+found                      _notes    projects    solar       undergrad
camera      general     Microsoft Office Word 2007.lnk  partners  recruiting  sophos      wics
classes     graduate    _mm                             patterns  research    sr          xiss
colloquia   groups      MMWIP                           people    resources   stork
%
%
%
% ls | grep -E 'er|re'
camera
general
partners
patterns
personnel
recruiting
research
resources
torrent
undergrad
%
%
% ls | grep -E '^(er|re)'
recruiting
research
resources
%
```

“print all lines in the input that match the string **er** or **re**

print all lines in the input that begin with the string **er** or **re**

Foreground and Background Processes

- Multiple processes can run concurrently
  - at any point, there is exactly one process that you can interact with through the keyboard ("foreground process")
  - remaining processes execute "in the background"
- A process can be started in the background:

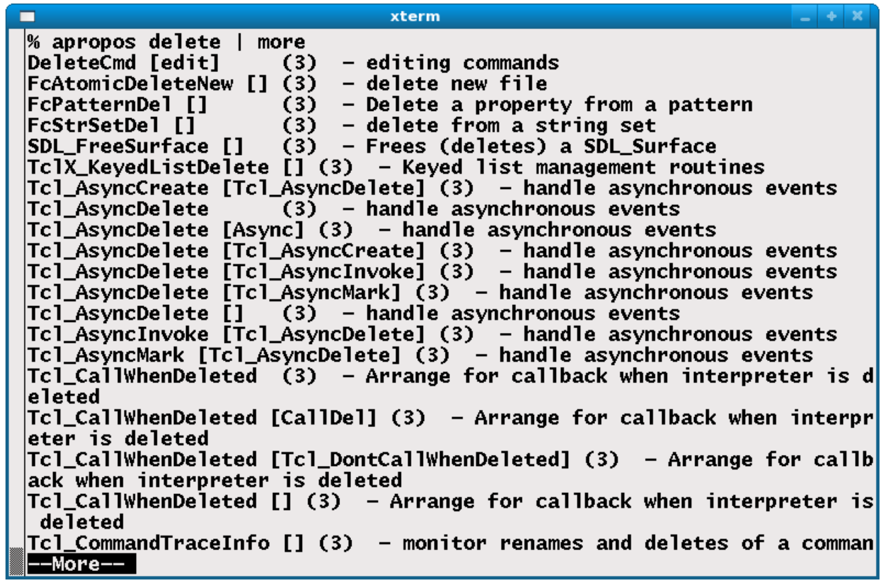
  processName **&**

- The execution of the current foreground process can be paused via ctrl-z
  - "**bg**" will then start it executing in the background
  - "**fg**" will bring it to the foreground

Example: Deleting a file

Figuring out which command to use:

- **apropos delete**
  - produces many screenfuls of output that go by too quickly
- **apropos delete | more**
  - many screenfuls of output, but shown one screenful at a time
  - most of the commands shown aren't relevant



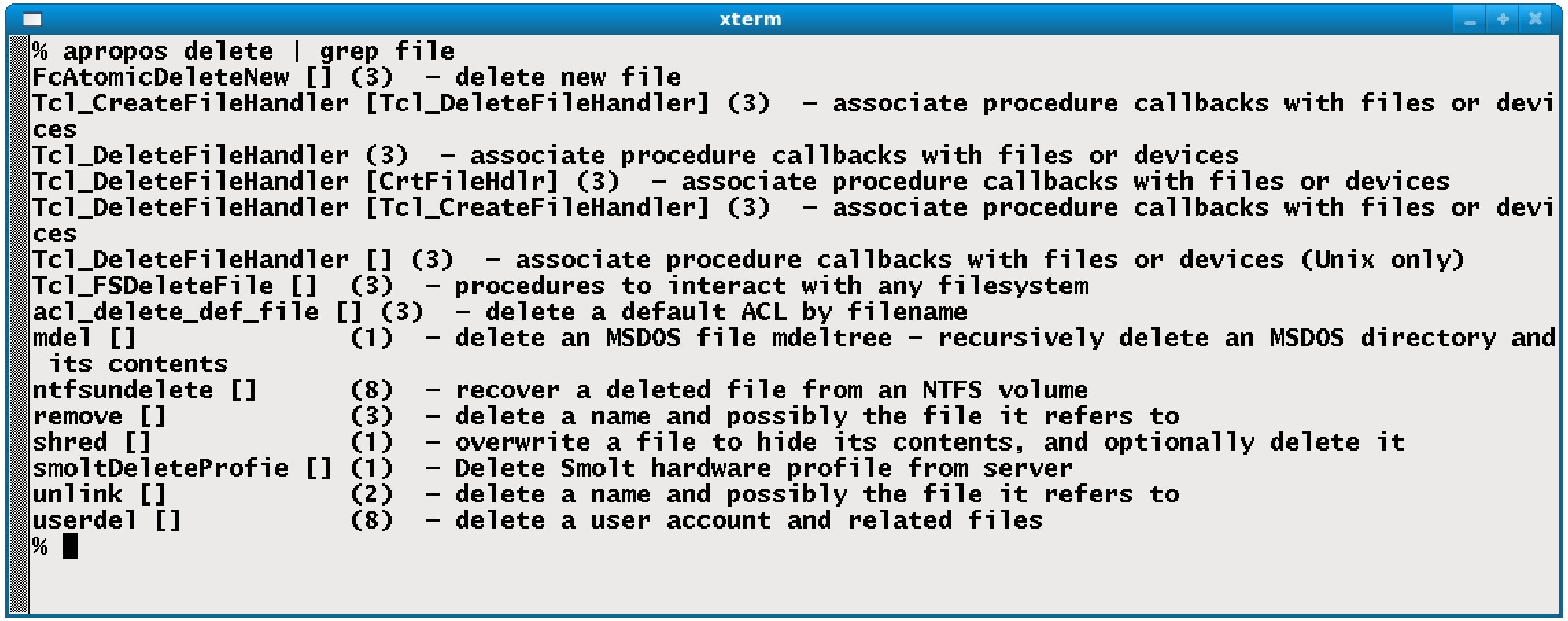
```
% apropos delete | more
DeleteCmd [edit]     (3)  - editing commands
FcAtomicDeleteNew [] (3)  - delete new file
FcPatternDel []      (3)  - Delete a property from a pattern
FcStrSetDel []       (3)  - delete from a string set
SDL_FreeSurface []   (3)  - Frees (deletes) a SDL_Surface
TclX_KeyedListDelete [] (3)  - Keyed list management routines
Tcl_AsyncCreate [Tcl_AsyncDelete] (3)  - handle asynchronous events
Tcl_AsyncDelete      (3)  - handle asynchronous events
Tcl_AsyncDelete [Async] (3)  - handle asynchronous events
Tcl_AsyncDelete [Tcl_AsyncCreate] (3)  - handle asynchronous events
Tcl_AsyncDelete [Tcl_AsyncInvoke] (3)  - handle asynchronous events
Tcl_AsyncDelete [Tcl_AsyncMark] (3)  - handle asynchronous events
Tcl_AsyncDelete []   (3)  - handle asynchronous events
Tcl_AsyncInvoke [Tcl_AsyncDelete] (3)  - handle asynchronous events
Tcl_AsyncMark [Tcl_AsyncDelete] (3)  - handle asynchronous events
Tcl_CallWhenDeleted  (3)  - Arrange for callback when interpreter is d
eleted
Tcl_CallWhenDeleted [CallDel] (3)  - Arrange for callback when interpr
eter is deleted
Tcl_CallWhenDeleted [Tcl_DontCallWhenDeleted] (3)  - Arrange for callb
ack when interpreter is deleted
Tcl_CallWhenDeleted [] (3)  - Arrange for callback when interpreter is
 deleted
Tcl_CommandTraceInfo [] (3)  - monitor renames and deletes of a comman
--More--
```

Example: Deleting a file… (1)

Idea 1: filter out irrelevant stuff

man –k delete | grep file

a lot fewer results;
nothing relevant

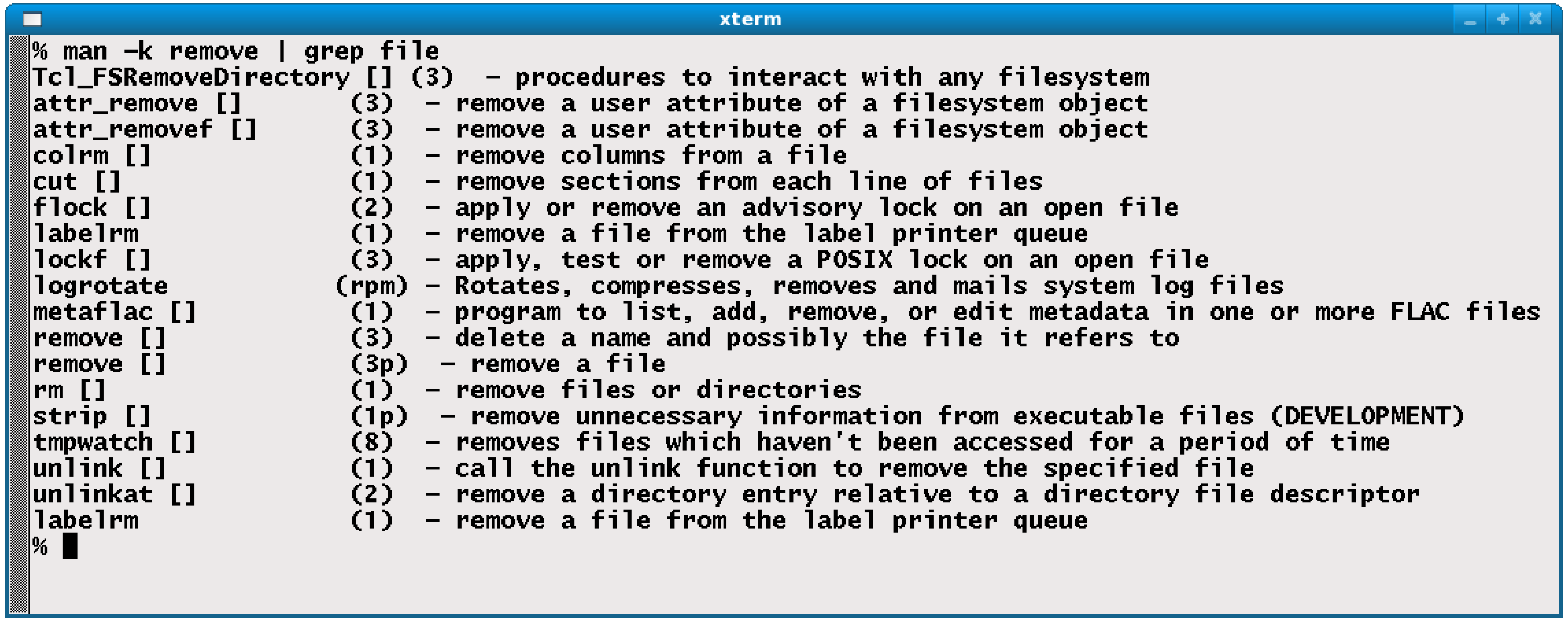


```
% apropos delete | grep file
FcAtomicDeleteNew [] (3)  - delete new file
Tcl_CreateFileHandler [Tcl_DeleteFileHandler] (3)  - associate procedure callbacks with files or devi
ces
Tcl_DeleteFileHandler (3)  - associate procedure callbacks with files or devices
Tcl_DeleteFileHandler [CrtFileHdlr] (3)  - associate procedure callbacks with files or devices
Tcl_DeleteFileHandler [Tcl_CreateFileHandler] (3)  - associate procedure callbacks with files or devi
ces
Tcl_DeleteFileHandler [] (3)  - associate procedure callbacks with files or devices (Unix only)
Tcl_FSDeleteFile []  (3)  - procedures to interact with any filesystem
acl_delete_def_file [] (3)  - delete a default ACL by filename
mdel []              (1)  - delete an MSDOS file mdeltree - recursively delete an MSDOS directory and
 its contents
ntfsundelete []      (8)  - recover a deleted file from an NTFS volume
remove []            (3)  - delete a name and possibly the file it refers to
shred []             (1)  - overwrite a file to hide its contents, and optionally delete it
smoltDeleteProfie [] (1)  - Delete Smolt hardware profile from server
unlink []            (2)  - delete a name and possibly the file it refers to
userdel []           (8)  - delete a user account and related files
% 
```

Example: Deleting a file… (2)

Idea 2: try a different keyword

man –k remove | grep file



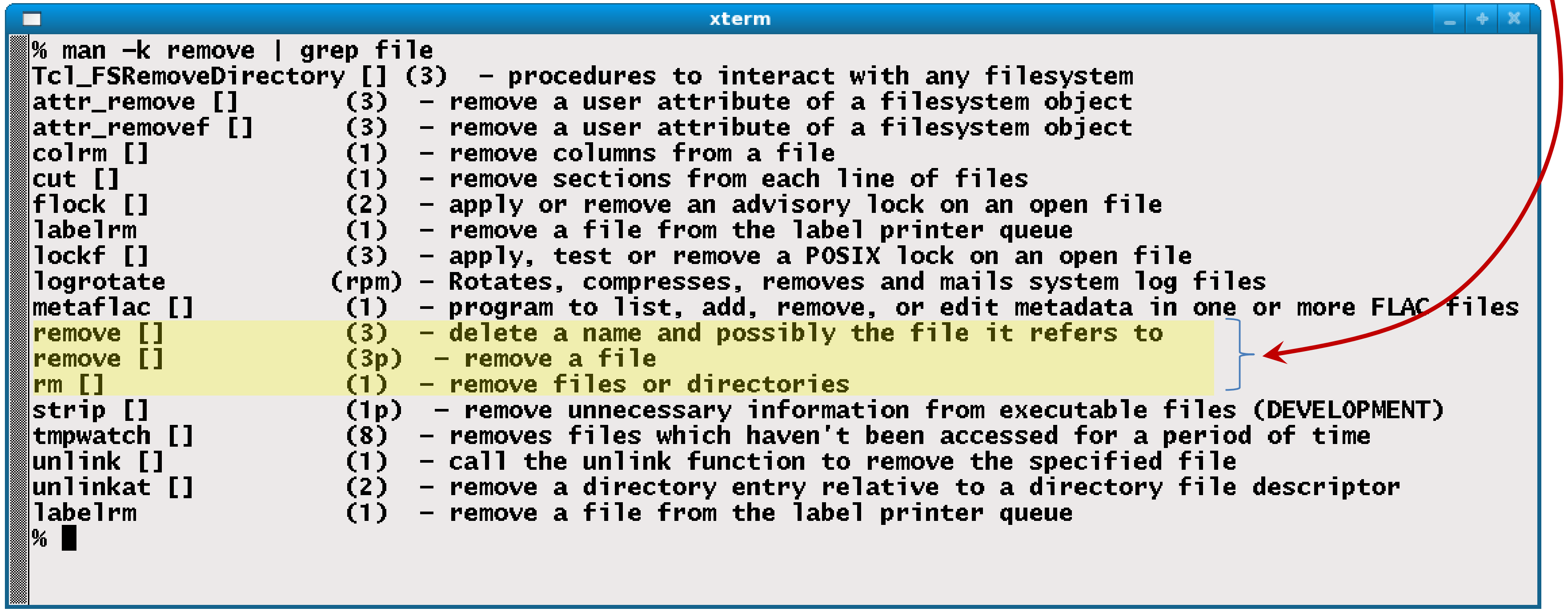
```
% man -k remove | grep file
Tcl_FSRemoveDirectory [] (3)  - procedures to interact with any filesystem
attr_remove []        (3)  - remove a user attribute of a filesystem object
attr_removef []       (3)  - remove a user attribute of a filesystem object
colrm []              (1)  - remove columns from a file
cut []                (1)  - remove sections from each line of files
flock []              (2)  - apply or remove an advisory lock on an open file
labelrm               (1)  - remove a file from the label printer queue
lockf []              (3)  - apply, test or remove a POSIX lock on an open file
logrotate            (rpm) - Rotates, compresses, removes and mails system log files
metaflac []           (1)  - program to list, add, remove, or edit metadata in one or more FLAC files
remove []             (3)  - delete a name and possibly the file it refers to
remove []             (3p)  - remove a file
rm []                 (1)  - remove files or directories
strip []              (1p)  - remove unnecessary information from executable files (DEVELOPMENT)
tmpwatch []           (8)  - removes files which haven't been accessed for a period of time
unlink []             (1)  - call the unlink function to remove the specified file
unlinkat []           (2)  - remove a directory entry relative to a directory file descriptor
labelrm               (1)  - remove a file from the label printer queue
%
```

Example: Deleting a file… (3)

Idea 2: try a different keyword

man –k remove | grep file

these are the only commands that refer to removing files



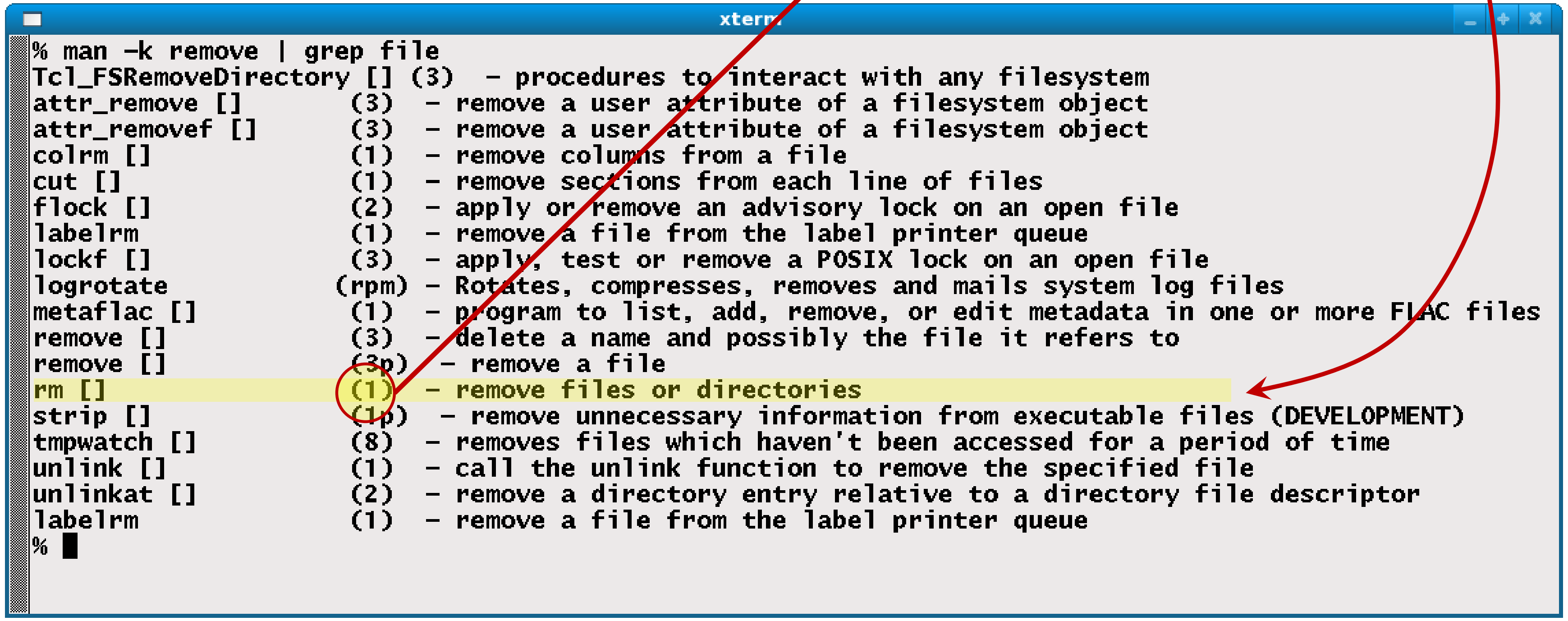
```
% man -k remove | grep file
Tcl_FSRemoveDirectory [] (3)  - procedures to interact with any filesystem
attr_remove []        (3)  - remove a user attribute of a filesystem object
attr_removef []       (3)  - remove a user attribute of a filesystem object
colrm []              (1)  - remove columns from a file
cut []                (1)  - remove sections from each line of files
flock []              (2)  - apply or remove an advisory lock on an open file
labelrm               (1)  - remove a file from the label printer queue
lockf []              (3)  - apply, test or remove a POSIX lock on an open file
logrotate            (rpm) - Rotates, compresses, removes and mails system log files
metaflac []           (1)  - program to list, add, remove, or edit metadata in one or more FLAC files
remove []             (3)  - delete a name and possibly the file it refers to
remove []             (3p)  - remove a file
rm []                 (1)  - remove files or directories
strip []              (1p)  - remove unnecessary information from executable files (DEVELOPMENT)
tmpwatch []           (8)  - removes files which haven't been accessed for a period of time
unlink []             (1)  - call the unlink function to remove the specified file
unlinkat []           (2)  - remove a directory entry relative to a directory file descriptor
labelrm               (1)  - remove a file from the label printer queue
%
```

Example: Deleting a file… (4)

Idea 2: try a different keyword

man –k remove | grep file

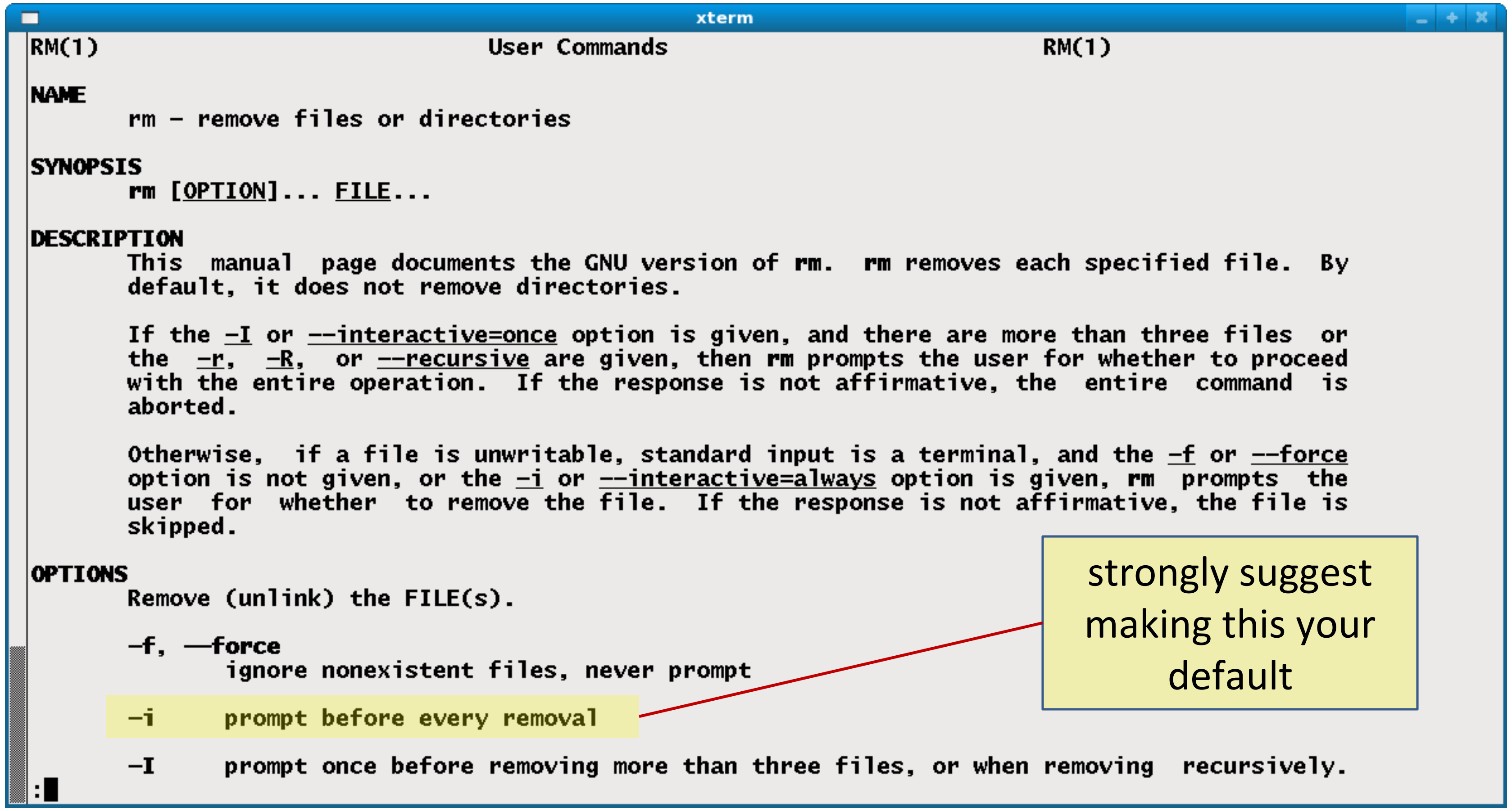
this is the only <u>user command</u> that refers to removing files



```
% man -k remove | grep file
Tcl_FSRemoveDirectory [] (3)  - procedures to interact with any filesystem
attr_remove []         (3)  - remove a user attribute of a filesystem object
attr_removef []        (3)  - remove a user attribute of a filesystem object
colrm []               (1)  - remove columns from a file
cut []                 (1)  - remove sections from each line of files
flock []               (2)  - apply or remove an advisory lock on an open file
labelrm                (1)  - remove a file from the label printer queue
lockf []               (3)  - apply, test or remove a POSIX lock on an open file
logrotate             (rpm) - Rotates, compresses, removes and mails system log files
metaflac []            (1)  - program to list, add, remove, or edit metadata in one or more FLAC files
remove []              (3)  - delete a name and possibly the file it refers to
remove []              (3p)  - remove a file
rm []                  (1)  - remove files or directories
strip []               (1p)  - remove unnecessary information from executable files (DEVELOPMENT)
tmpwatch []            (8)  - removes files which haven't been accessed for a period of time
unlink []              (1)  - call the unlink function to remove the specified file
unlinkat []            (2)  - remove a directory entry relative to a directory file descriptor
labelrm                (1)  - remove a file from the label printer queue
%
```

Example: Deleting a file… (5)

Confirm that this is the appropriate command: **“man rm”**



```
RM(1)                        User Commands                        RM(1)

NAME
       rm - remove files or directories

SYNOPSIS
       rm [OPTION]... FILE...

DESCRIPTION
       This  manual  page documents the GNU version of rm.  rm removes each specified file.  By
       default, it does not remove directories.

       If the -I or --interactive=once option is given, and there are more than three files  or
       the  -r,  -R,  or --recursive are given, then rm prompts the user for whether to proceed
       with the entire operation.  If the response is not affirmative, the  entire  command  is
       aborted.

       Otherwise,  if a file is unwritable, standard input is a terminal, and the -f or --force
       option is not given, or the -i or --interactive=always option is given, rm  prompts  the
       user  for  whether  to remove the file.  If the response is not affirmative, the file is
       skipped.

OPTIONS
       Remove (unlink) the FILE(s).

       -f, --force
              ignore nonexistent files, never prompt

       -i     prompt before every removal

       -I     prompt once before removing more than three files, or when removing  recursively.
:
```

strongly suggest making this your default

Setting defaults for your commands

- Create an “alias” for your command
  - syntax different for different shells
  - bash: **alias** *aliasName=“cmdName”*

    e.g.: alias rm=“rm –i”
  - see “**man alias**” for details
- To have this alias in force whenever you log in, add this line to the file

  ~/.bashrc   // assuming your login shell is “bash”
- To find out your login shell, run the command

  echo $0

Setting defaults for your commands

- Create an "alias" for your command
  - syntax different for different shells
  - bash: **alias** *aliasName="cmdName"*

    e.g.: alias rm="rm –i"
  - see "**man alias**" for details
- To have this alias in force whenever you log in, add this line to the file

  ~/.bashrc // assuming your login shell is "bash"

  Why didn't this work!!!
- To find out your login shell, run the command

  echo $0

Customization

- If bash is specified as your shell in `/etc/passwd` and you login, the instance of bash that's started is said to be a *login shell*.

- When bash is started as a login shell it first reads `/etc/profile`. It then looks for three files in turn: `~/.bash_profile`, `~/.bash_login`, and `~/.profile`. Upon finding one, it executes the commands in that file and doesn't look any further.

- Sometimes you'll want to start another instance of bash from the bash prompt:
  - `%` **`bash`**
  - `%`

- Such an instance of bash is an "interactive non-login shell". It reads `/etc/bash.bashrc` and `~/.bashrc`.

Customization

- So how do I get around this?

Customization

- So how do I get around this?

1. I could add the line to the file that runs when I log in (in this case .profile)

Customization

- So how do I get around this?

1. I could add the line to the file that runs when I log in (in this case .profile)
    1. The problem with this is it won't be defined if I invoke another copy of bash

Customization

- So how do I get around this?

1. I could add the line to the file that runs when I log in (in this case .profile)
    1. The problem with this is it won't be defined if I invoke another copy of bash
2. I could add the line to both .profile and .bashrc

Customization

- So how do I get around this?

1. I could add the line to the file that runs when I log in (in this case .profile)
    1. The problem with this is it won't be defined if I invoke another copy of bash
2. I could add the line to both .profile and .bashrc
    1. This is OK, but I need to remember to change 2 files every time I change things

Customization

- So how do I get around this?

1. I could add the line to the file that runs when I log in (in this case .profile)
    1. The problem with this is it won't be defined if I invoke another copy of bash
2. I could add the line to both .profile and .bashrc
    1. This is OK, but I need to remember to change 2 files every time I change things
3. I could make a file run in both situations.
    1. This is what most programmers do.

Making .bashrc run always

- **Note:**. Please use caution when altering hidden files.
- Use cd with no arguments to go to your home directory.
- Confirm that you've got .profile but not .bash_profile or .bash_login
- Make a directory bashoriginals and copy (cp) .profile and .bashrc into it.
- Edit .profile and add the line:
  source ~/.bashrc
- Now any lines you add to .bashrc will run every time a new shell is created.

Computing with pipes

A key element of the UNIX philosophy is to use *pipelines* to combine programs to solve a problem, rather than writing a new program.

Problem: How many unique users are on lectura?

v1: Get login names

```
% who| cut -f1 -d " "
ken
dmr
ken
francis
rob
walt24
dmr
rob
wnj
dmr
ken
```

v2: Sort login names

```
% who|cut -f1 -d" "|sort
dmr
dmr
dmr
francis
ken
ken
ken
rob
rob
walt24
wnj
```

v3: Get unique login names

```
% who|cut -f1 -d" "|sort|uniq
dmr
francis
ken
rob
walt24
wnj
```

v4: Get the count

```
% who|cut -f1 -d"
"|sort|uniq| wc -l
6
```