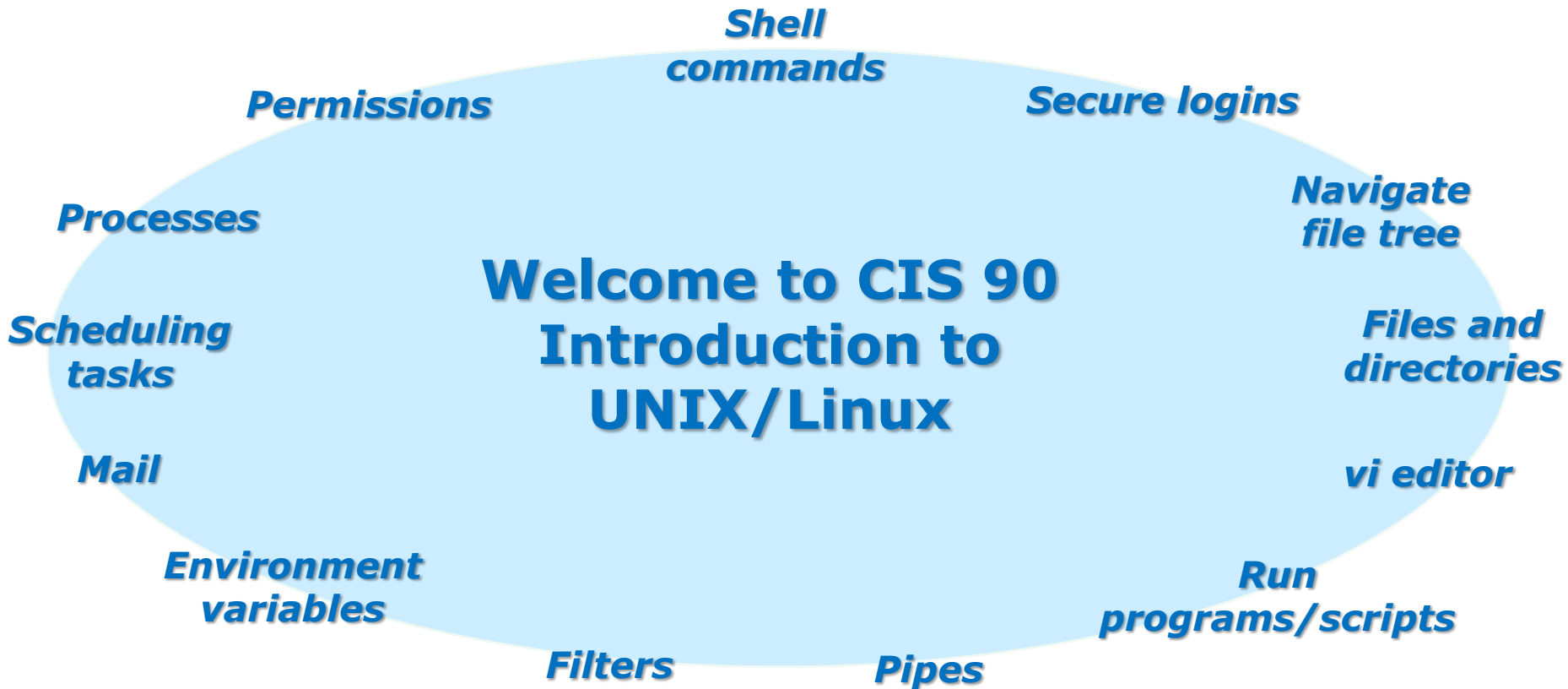


Lesson Module Checklist

- Slides
- Converted WB
- Flash cards
- Page numbers
- 1st minute quiz
- Web Calendar summary
- Web book pages
- Commands
- Lock turnin directory at midnight
- Lab 7 tested
- Lab X1 tested
- 9V backup battery for microphone
- Backup slides, CCC info, handouts on flash drive



Student Learner Outcomes

1. Navigate and manage the UNIX/Linux file system by viewing, copying, moving, renaming, creating, and removing files and directories.
2. Use the UNIX features of file redirection and pipelines to control the flow of data to and from various commands.
3. With the aid of online manual pages, execute UNIX system commands from either a keyboard or a shell script using correct command syntax.

Introductions and Credits



Jim Griffin

- Created this Linux course
- Created Opus and the CIS VLab
- Jim's site: <http://cabrillo.edu/~jgriffin/>



Rich Simms

- HP Alumnus
- Started teaching this course in 2008 when Jim went on sabbatical
- Rich's site: <http://simms-teach.com>

And thanks to:

- John Govsky for many teaching best practices: e.g. the First Minute quizzes, the online forum, and the point grading system (<http://teacherjohn.com/>)



Student checklist (How to attend from home or in the classroom)

- 1) Browse to the CIS 90 website Calendar page
 - <http://simms-teach.com>
 - Click CIS 90 link on left panel
 - Click Calendar link near top of content area
 - Locate today's lesson on the Calendar
- 2) Download the presentation slides for today's lesson for easier viewing
- 3) Click Enter virtual classroom to join CCC Confer session
- 4) Connect to Opus using Putty or ssh command

Student checklist (How to layout your screen when attending class)

CCC Confer

Downloaded PDF of Lesson Slides

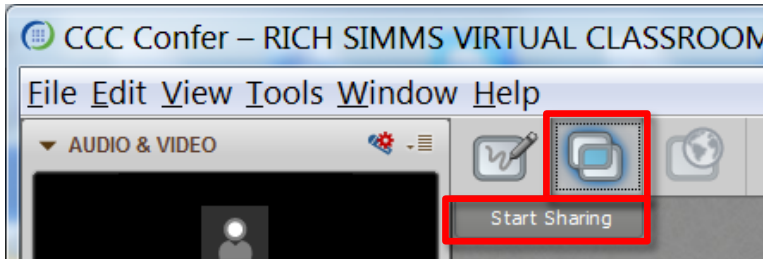
The screenshot displays a virtual classroom environment with several overlapping windows. On the left is a 'Rich's Cabrillo College CIS 90 Calendar' window. In the center is a 'CCC Confer - RICH SIMMS VIRTUAL CLASSROOM' window showing a video feed of a person and a 'Class Activity - Where are you now?' map. To the right is a 'cis90lesson01.pdf - Adobe Acrobat Pro' window showing a slide titled 'The CIS 90 System Playground'. Below the calendar is a 'cis90lesso....pdf' window. At the bottom right are two terminal windows showing login prompts and system information. Blue arrows point from text labels to specific windows: 'CCC Confer' points to the confer window, 'Downloaded PDF of Lesson Slides' points to the Acrobat window, 'CIS 90 website Calendar page' points to the calendar window, and 'One or more login sessions to Opus' points to the terminal windows.

CIS 90 website
Calendar page

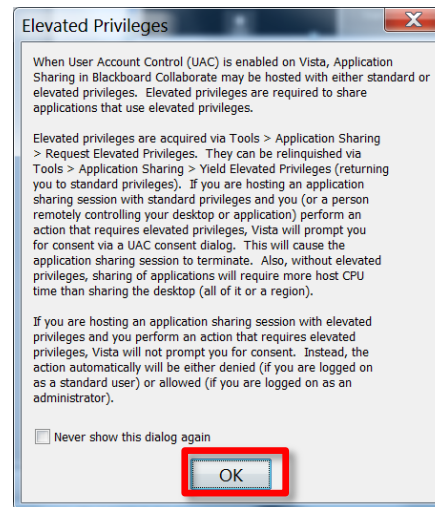
One or more login
sessions to Opus

Student checklist (To share your desktop with the class)

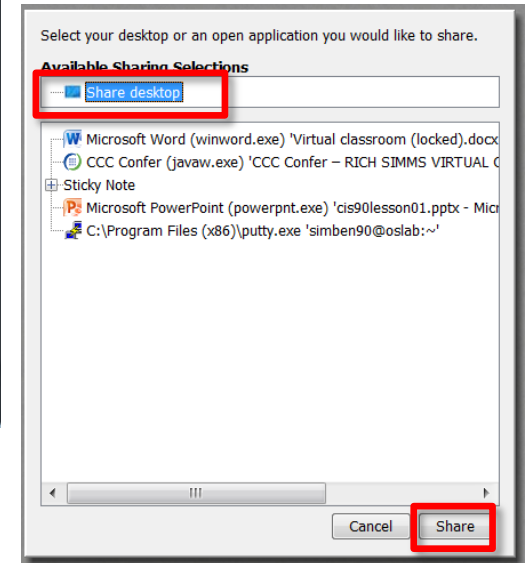
1) Instructor gives you sharing privileges



2) Click overlapping rectangles icon. If white "Start Sharing" text is present then click it as well.



3) Click OK button.



4) Select "Share desktop" and click Share button.



Instructor: **Rich Simms**

Dial-in: **888-886-3951**

Passcode: **136690**



Al



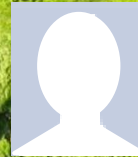
Tim



Emilio



Chris



Eddie



Clara



Ryan



Ethan



Monte



Cameron



Tanner



Tess



Ahmad



Mike F.



Mario



Roberto



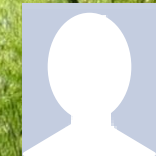
Diego



John



Efrain



Ivers



Django



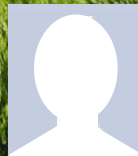
Ian



Abraham C.



Benji



Abraham N.



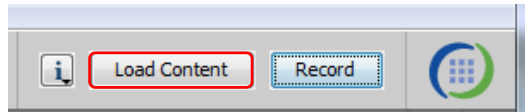
Mateo



Rich's CCC Confer checklist - setup

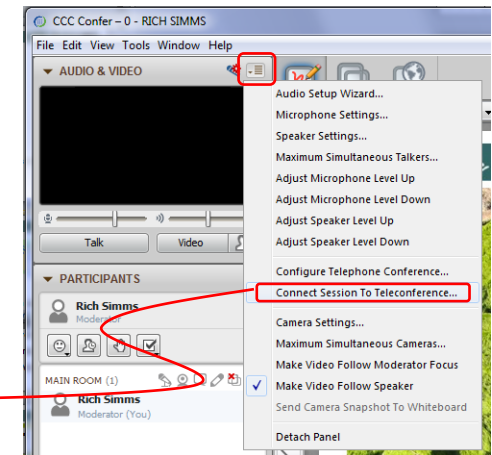
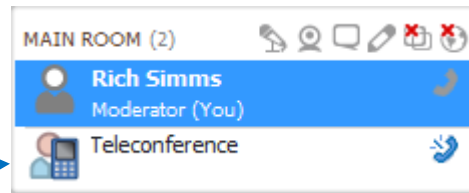


[] Preload White Board



[] Connect session to Teleconference

Session now connected to teleconference



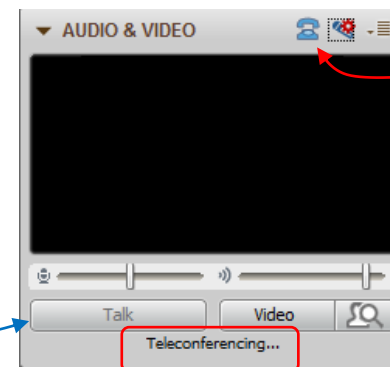
[] Is recording on?



Red dot means recording

[] Use teleconferencing, not mic

Should be greyed out



Should show as this live "off hook" telephone handset icon and the Teleconferencing ... message displayed

Rich's CCC Confer checklist - app layout



The screenshot shows a Windows desktop environment with the following components and annotations:

- CCC Confer -0 - RIC...:** The main application window on the left, showing a video feed of Rich Simms, a list of participants (Rich Simms, Moderator), and a chat window.
- foxit for slides:** A red box with an arrow pointing to the Foxit Reader window displaying 'cis90lesson07.pdf'.
- chrome:** A red box with an arrow pointing to the Google Chrome browser window displaying 'simms-teach.com/docs/cis90/cis-90-TEST-1-Fall-12.pdf'.
- putty:** A red box with an arrow pointing to the PuTTY terminal window showing a login session for 'simben90@oslab:~'.
- vSphere Client:** A red box with an arrow pointing to the vCenter - vSphere Client window showing the 'CIS 192' virtual machine.

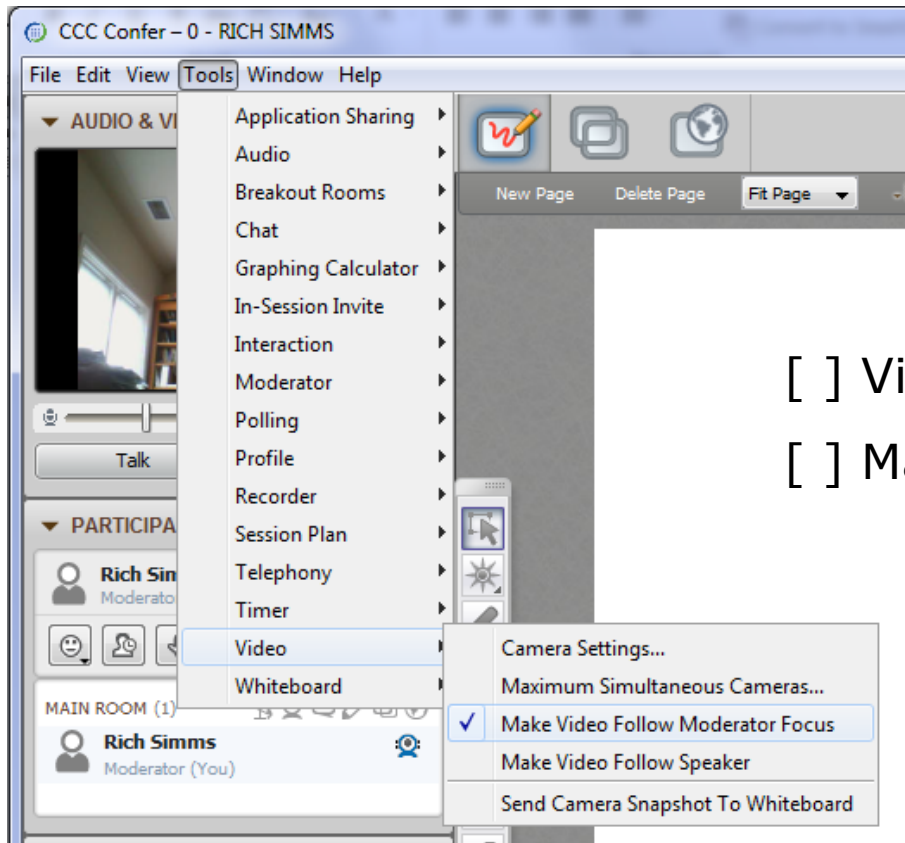
The desktop taskbar at the bottom includes icons for Internet Explorer, File Explorer, Google Chrome, and other applications. The system clock in the bottom right corner shows 6:52 AM on 10/10/2012.

[] layout and share apps





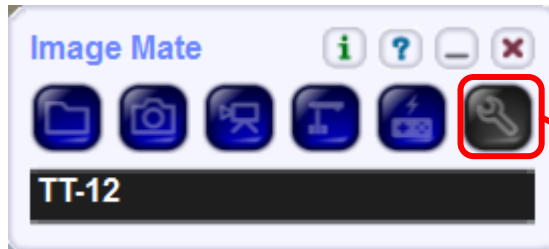
Rich's CCC Confer checklist - video



[] Video (webcam)

[] Make Video Follow Moderator Focus

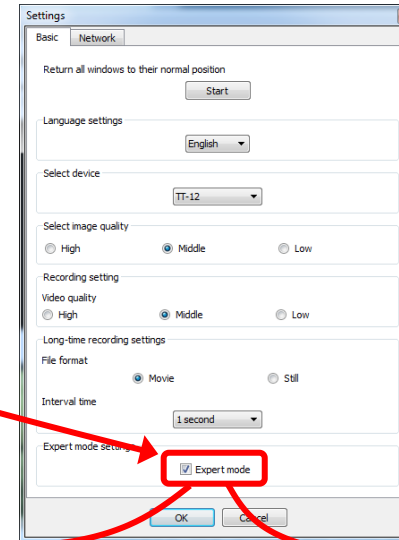
Rich's CCC Confer checklist - Elmo



Elmo rotated down to view side table



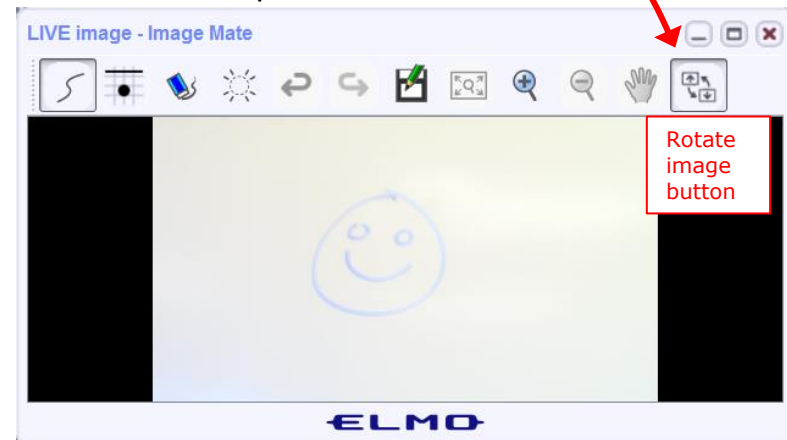
Run and share the Image Mate program just as you would any other app with CCC Confer



The "rotate image" button is necessary if you use both the side table and the white board.

Quite interesting that they consider you to be an "expert" in order to use this button!

Elmo rotated up to view white board



Rich's CCC Confer checklist - universal fix



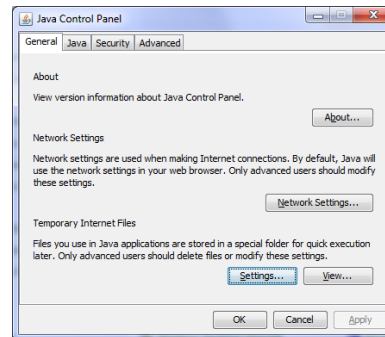
Universal Fix for CCC Confer:

- 1) Shrink (500 MB) and delete Java cache
- 2) Uninstall and reinstall latest Java runtime

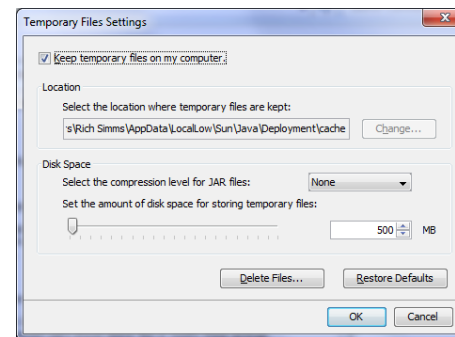
Control Panel (small icons)



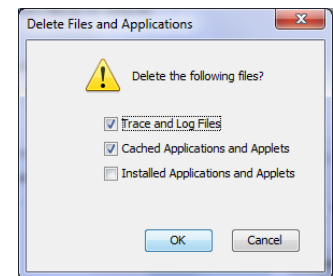
General Tab > Settings...



500MB cache size



Delete these



Google Java download



Quiz

Please answer these questions **in the order** shown:

See electronic white board

email answers to: risimms@cabrillo.edu

(answers must be emailed within the first few minutes of class for credit)

Input/Output Processing

Objectives	Agenda
<ul style="list-style-type: none">• Identify the three open file descriptors an executing program is given when started.• Be able to redirect input from files and output to files• Define the terms pipe, filter, and tee• Use pipes and tees to combine multiple commands• Know how to use the following useful UNIX commands:<ul style="list-style-type: none">o findo grepo wco sorto spell	<ul style="list-style-type: none">• Quiz• Questions• Warmup• Housekeeping• Review• File descriptors• Pipelines• New commands• Tasks using pipelines

Sound Check

*Students that dial-in should mute their line using *6 to prevent unintended noises distracting the web conference.*

*Instructor can use *96 to mute all student lines.*

Questions

Questions?

Lesson material?

Labs? Tests?

How this course works?

- Graded work in home directories
- Answers in /home/cis90/answers

Who questions much, shall learn much, and retain much.

- Francis Bacon

If you don't ask, you don't get.

- Mahatma Gandhi

Chinese
Proverb

他問一個問題，五分鐘是個傻子，他不問一個問題仍然是一個傻瓜永遠。

He who asks a question is a fool for five minutes; he who does not ask a question remains a fool forever.



Lab 5

Post Mortem

Lab 5 – Results

MD - Making Directories

Step 4 - class/tests directory is missing (x)

RF - Renaming Files

Step 2 - Poems not renamed to poems

MF - Moving Files

Steps 1, 2, 3, and 5 - magna_carta not in docs/ (x)

Step 11 - better_town not in edits/ (xxxxxx)

CF - Copying Files

Step 4 - sonnet6 not copied (xx)

RFD - Removing Files and Directories

Step 3 - letter.bak not removed (xx)

Steps 4, 5, 6 - Lab2.0 not removed (xx)

LF - Linking Files

Step 2 - no greeting hard link to motd (xxx)

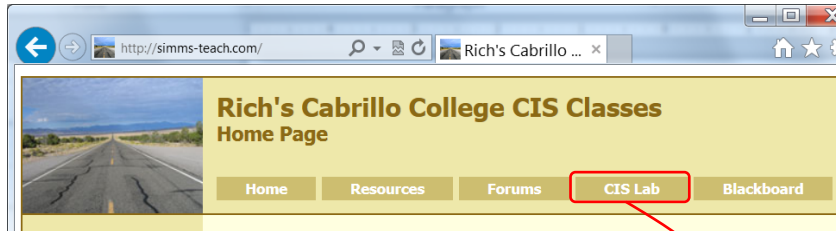
22 labs
submitted



4 labs not
submitted



Want some help working the labs?



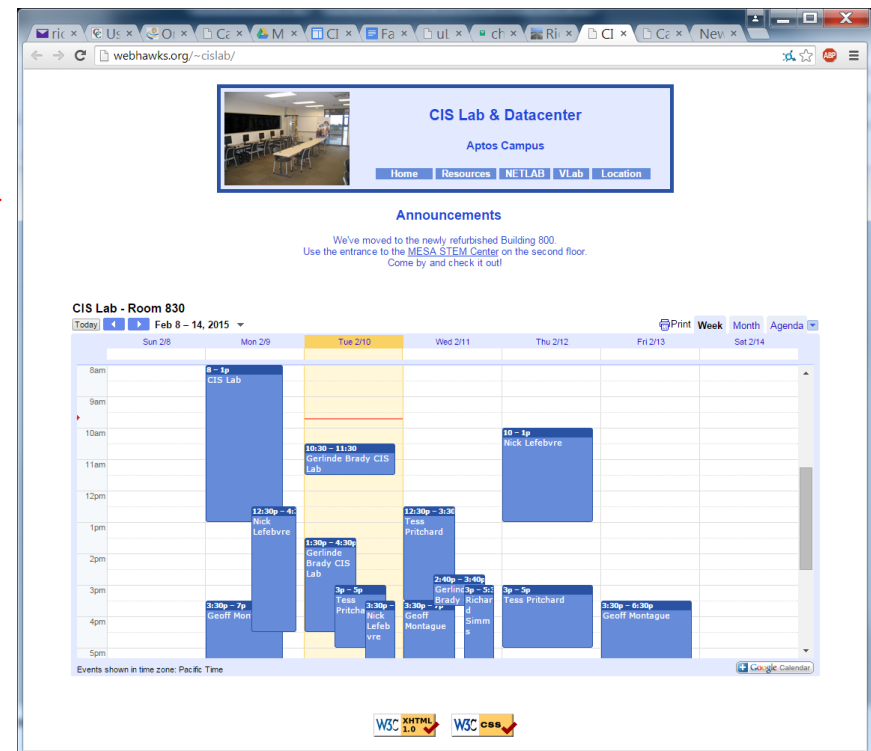
If you would like some additional come over to the CIS Lab. There are student lab assistants and instructors there to help you.

*Nick, Sean, and Geoff are
CIS 90 Alumni.*

*Tess and Mike F. are in our
class!*

*Michael M. is the other
Linux instructor.*

*I'm in there Wednesdays
3:00-5:30pm.*



CIS 90 Tutoring Available

<http://www.cabrillo.edu/services/tutorials/>

TUTORIALS

ANNOUNCEMENTS & DEADLINES

- New subjects for Spring 2014:
- American Sign Language
- Computer Applications/Business Technology (CABT)
- Computer and Information Systems (CIS)
- History 17A

Welcome to the Tutorials Center!

We offer **FREE** peer tutoring to Cabrillo students who are enrolled in the course/s for which they need help.

- Tutoring is by appointment. The days and times of tutoring sessions are established by the office.
- Sessions are weekly and for the duration of the semester.
- Tutoring sessions are scheduled in small groups. Sessions last 1-2 hours depending on the class. Occasionally, sessions may be one to one but that is not guaranteed.
- Come directly to the TC office to schedule (second floor of library).

The following classes are being tutored for Spring 2014:

- Accounting 1A, 1B, 6, 54A, 151A, 159, 163
- American Sign Language (ASL) 1, 2
- Biology 4, 5, 6
- Computer Applications/Business Technology (CABT) 31, 38, 41, 101, 157, 160
- Computer and Information Systems (CIS) 81, 90, 172**
- Chemistry 1A, 1B, 2, 30A, 30B, 32

CONTACT INFORMATION

Tutorials Center

Location Room 1080A - Learning Resource Center

Phone 831.479.6470

Email tutorialscenter@cabrillo.edu

Coordinator Lori Chavez

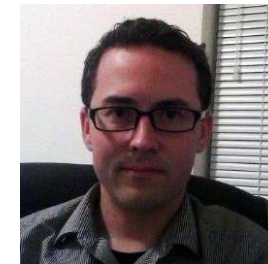
Phone 831.479.6126

Email lochavez@cabrillo.edu

Hours Monday - Thursday: 9am - 5pm
Friday: 9am - 1pm

MAP, DIRECTIONS, & PARKING

DEPARTMENT STAFF & FACULTY DIRECTORY

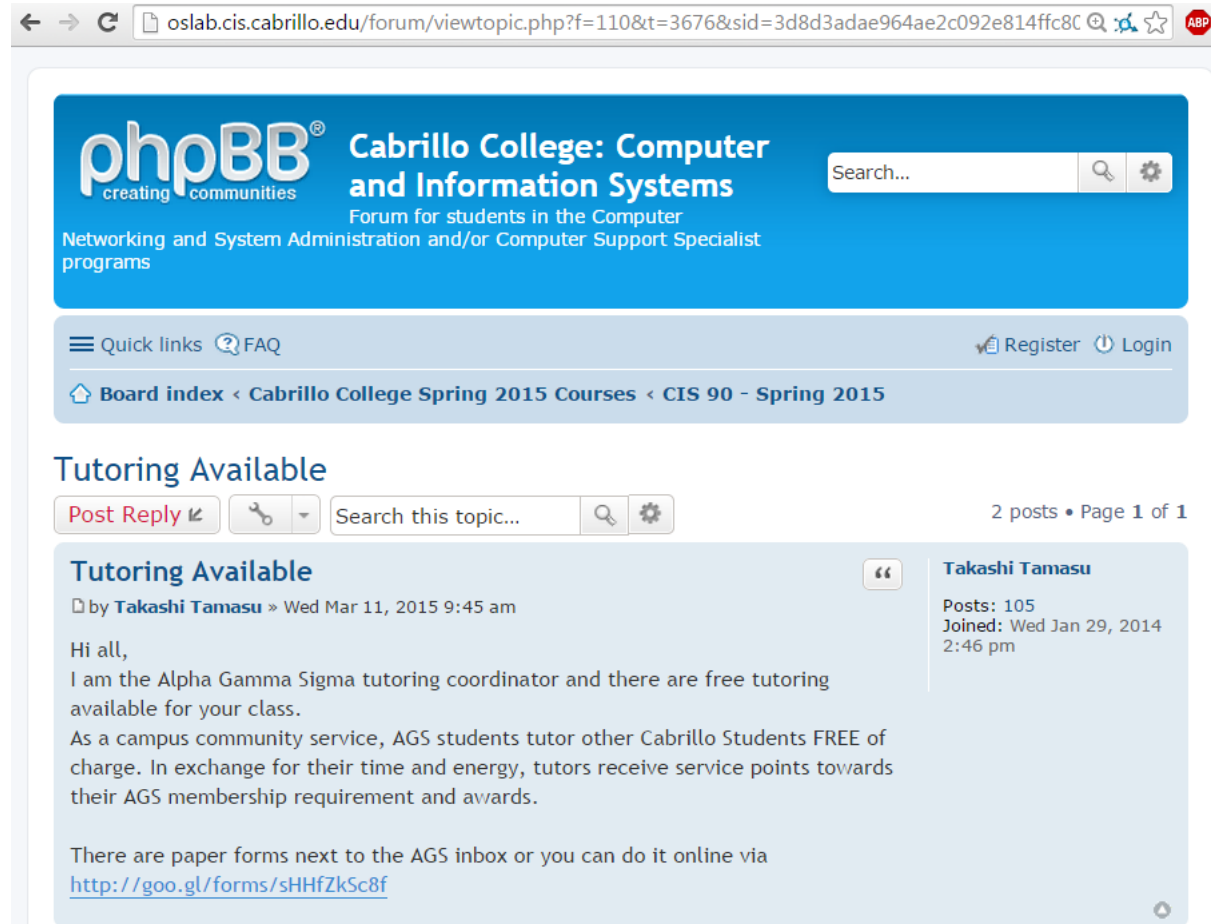


Geoffrey Montague

All students interested in tutoring need to come directly to the Tutorials Center to schedule, register and fill out some paperwork. This is just a one-time visit.

The tutoring will take place at the STEM center.

More CIS 90 Tutoring Available



The screenshot shows a web browser window displaying a phpBB forum page. The browser's address bar shows the URL: `oslab.cis.cabrillo.edu/forum/viewtopic.php?f=110&t=3676&sid=3d8d3adae964ae2c092e814ffc8c`. The forum header is blue and contains the phpBB logo, the forum title "Cabrillo College: Computer and Information Systems", and a search bar. Below the header, there are navigation links for "Quick links", "FAQ", "Register", and "Login". The breadcrumb trail indicates the current location: "Board index < Cabrillo College Spring 2015 Courses < CIS 90 - Spring 2015". The main topic is "Tutoring Available", with 2 posts and 1 page. The post is by "Takashi Tamasu" and was made on "Wed Mar 11, 2015 9:45 am". The post content states that the user is the Alpha Gamma Sigma tutoring coordinator and offers free tutoring to AGS students in exchange for service points. A link to a Google Form is provided at the bottom of the post.

oslab.cis.cabrillo.edu/forum/viewtopic.php?f=110&t=3676&sid=3d8d3adae964ae2c092e814ffc8c

phpBB® creating communities
Cabrillo College: Computer and Information Systems
Forum for students in the Computer Networking and System Administration and/or Computer Support Specialist programs

Search...

Quick links FAQ Register Login

Board index < Cabrillo College Spring 2015 Courses < CIS 90 - Spring 2015

Tutoring Available

Post Reply Search this topic... 2 posts • Page 1 of 1

by **Takashi Tamasu** » Wed Mar 11, 2015 9:45 am

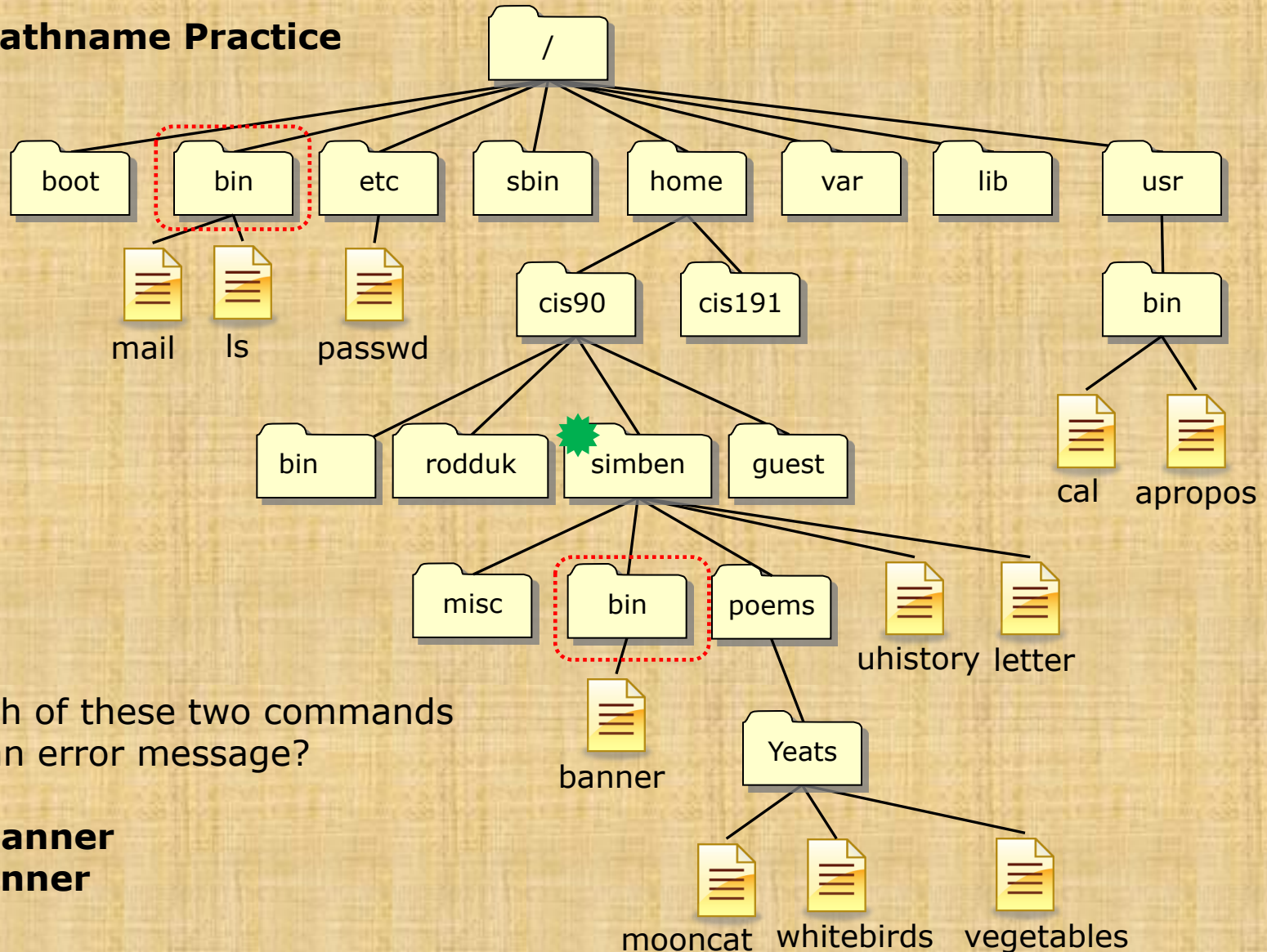
Hi all,
I am the Alpha Gamma Sigma tutoring coordinator and there are free tutoring available for your class.
As a campus community service, AGS students tutor other Cabrillo Students FREE of charge. In exchange for their time and energy, tutors receive service points towards their AGS membership requirement and awards.


There are paper forms next to the AGS inbox or you can do it online via <http://goo.gl/forms/SHHfZkSc8f>

Takashi Tamasu
Posts: 105
Joined: Wed Jan 29, 2014 2:46 pm

Warmup

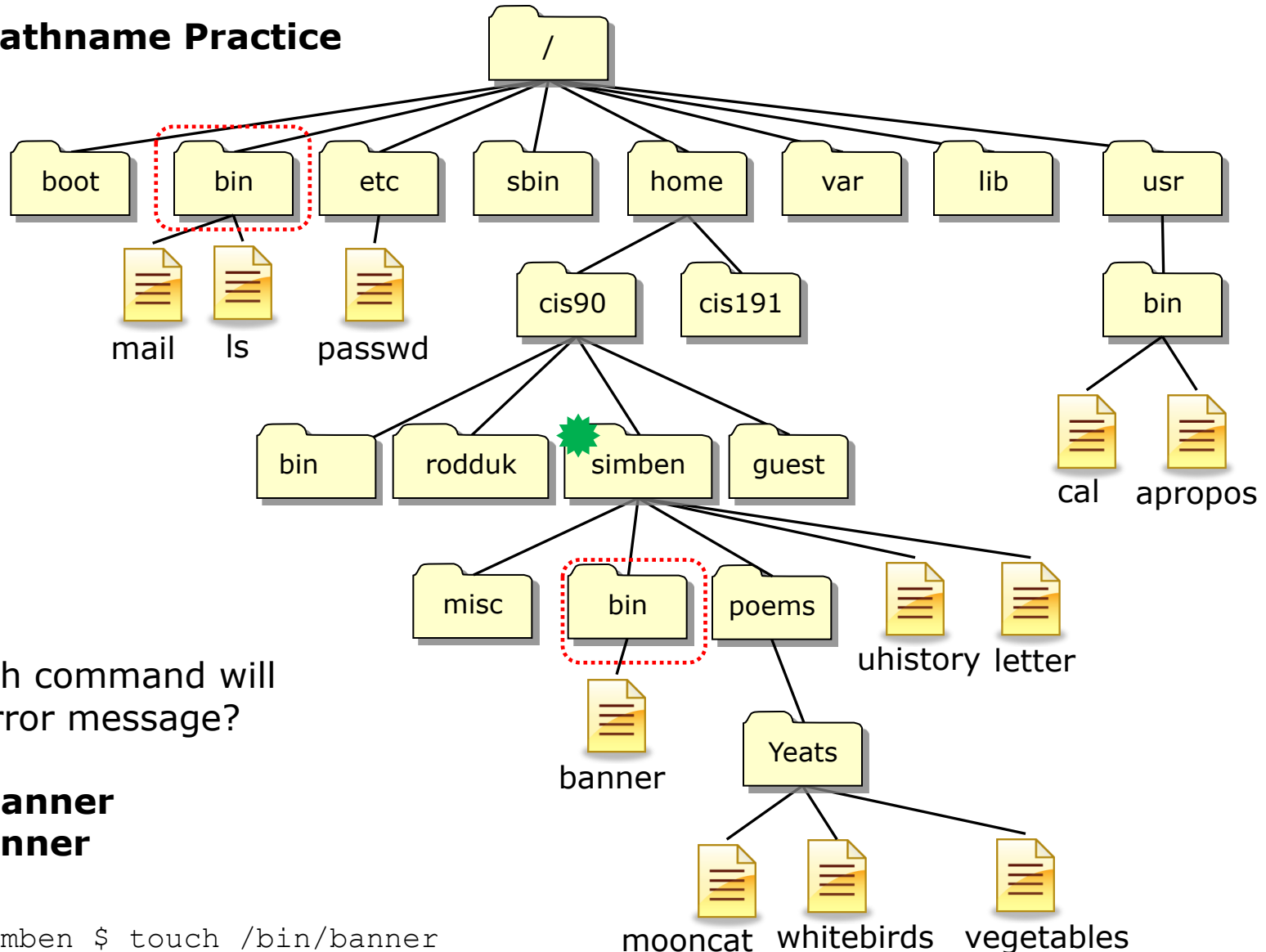
File Tree Pathname Practice




From  which of these two commands will generate an error message?

touch /bin/banner
touch bin/banner

File Tree Pathname Practice



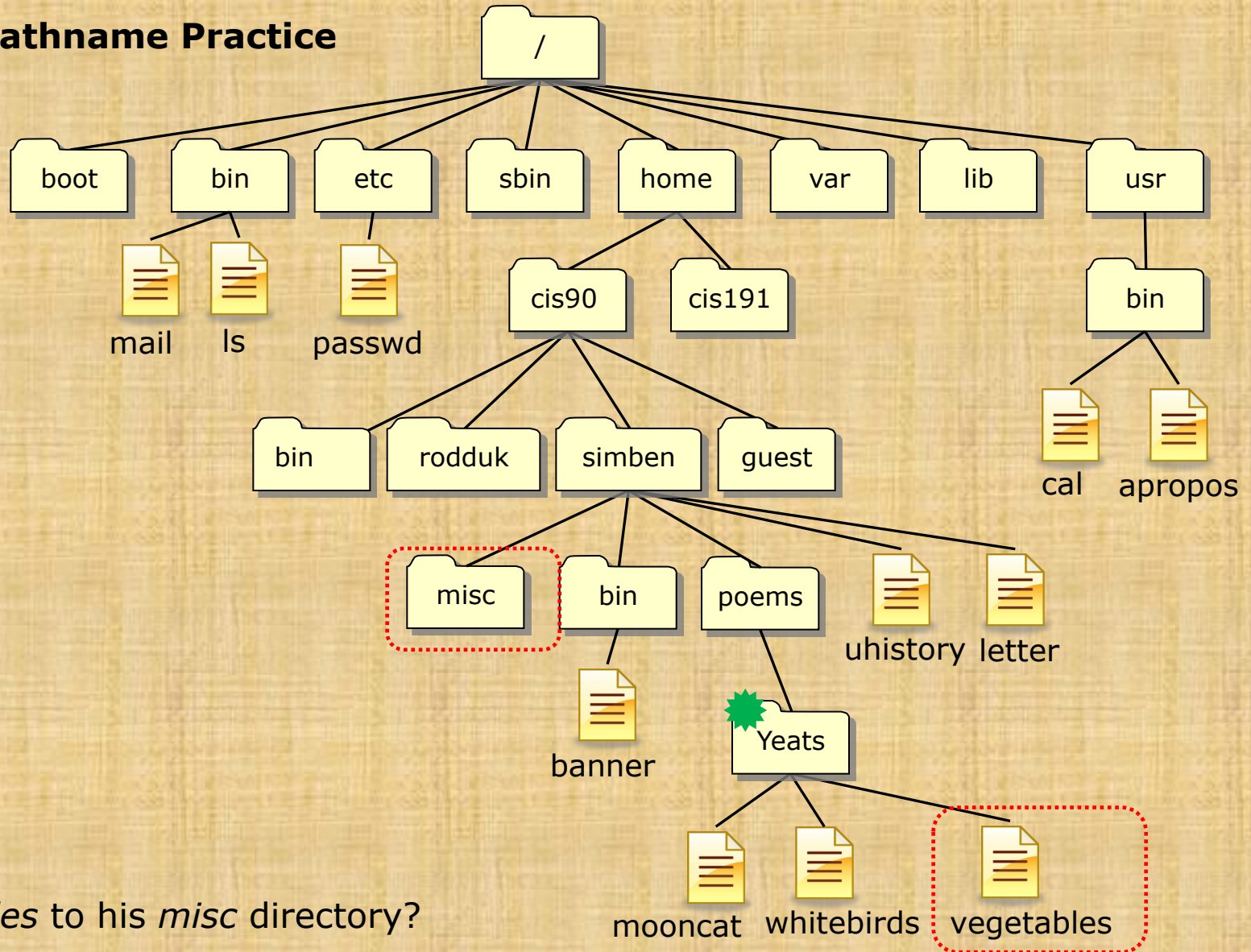
From  which command will generate an error message?

touch /bin/banner
touch bin/banner

```
/home/cis90/simben $ touch /bin/banner
touch: cannot touch `/bin/banner': Permission denied
```

banner is in your local bin directory

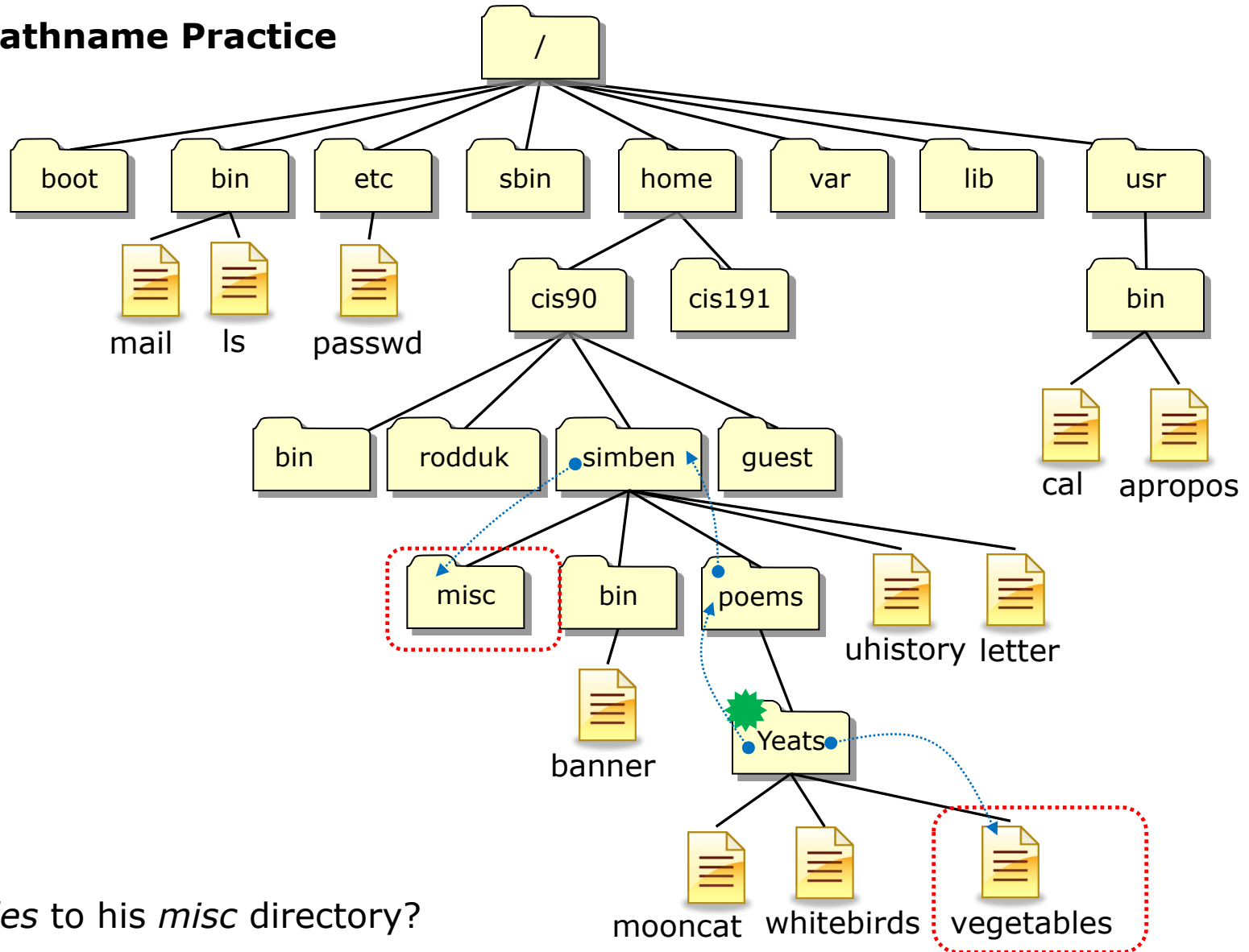
File Tree Pathname Practice



From  how
does Benji:

Move *vegetables* to his *misc* directory?

File Tree Pathname Practice



From  how
does Benji:

Move *vegetables* to his *misc* directory?

`/home/cis90/simben/poems/Yeats $ mv vegetables ../../misc/`

*Other answers
are also
acceptable*

From  how
does Benji:

Move *vegetables* to his
misc directory?

***mv* <path-to-file> <path-to-directory>**

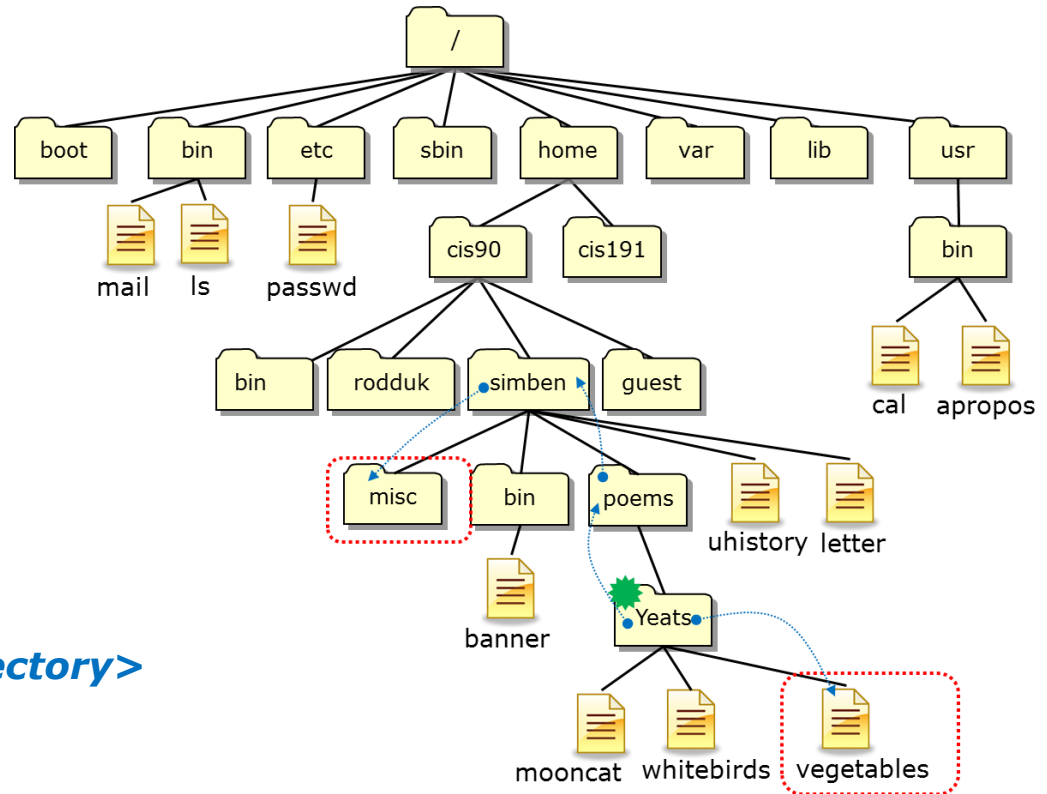
***mv* vegetables ../../misc/**

***mv* vegetables /home/cis90/simben/misc/**

***mv* /home/cis90/simben/poems/Yeats/vegetables ../../misc/**

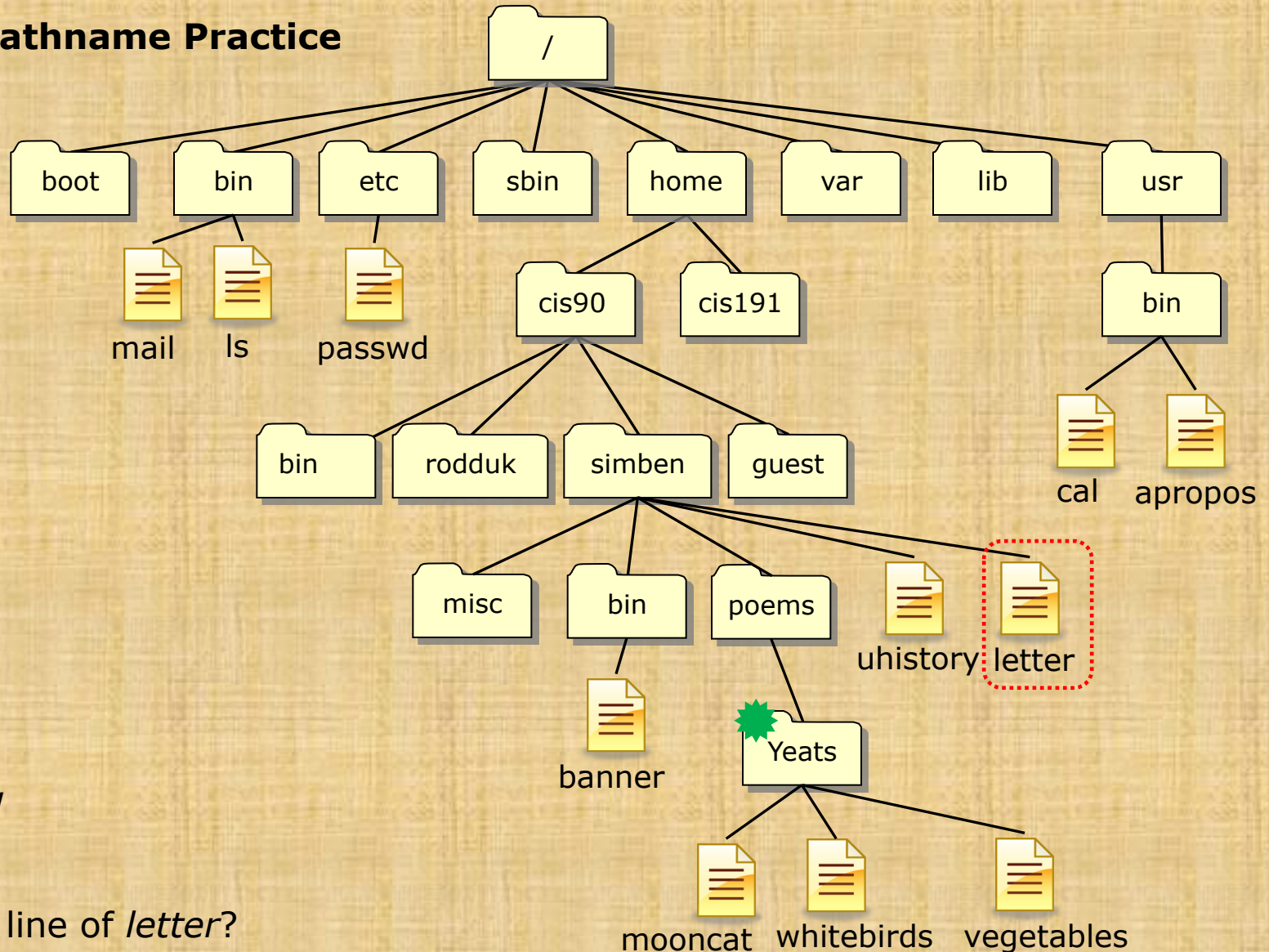
***mv* /home/cis90/simben/poems/Yeats/vegetables /home/cis90/simben/misc/**

***mv* vegetables ~/misc/**



All these answers are correct

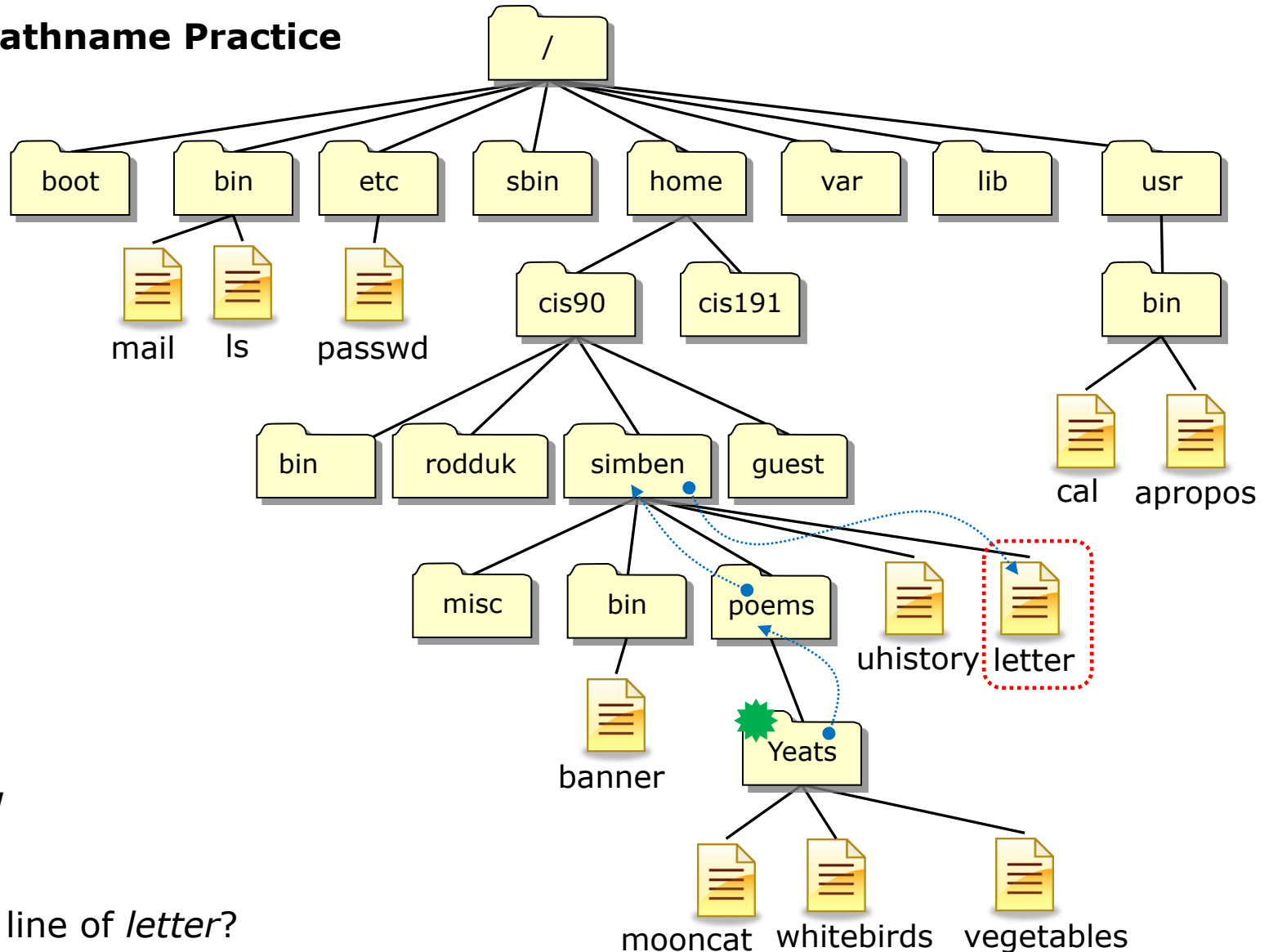
File Tree Pathname Practice



From  how
does Benji:

Print the last line of *letter*?

File Tree Pathname Practice



/home/cis90/simben/poems/Yeats \$ **tail -n1 ../../letter**

*Other answers
are also
acceptable*

From  how
does Benji:

Print the last line of *letter*?

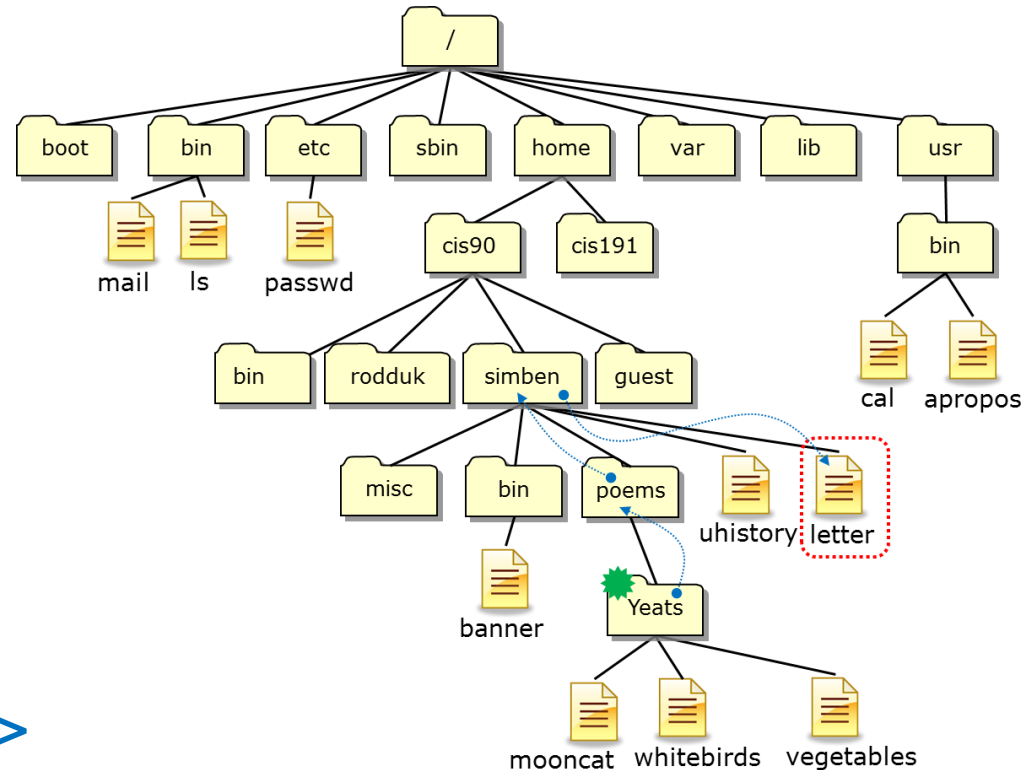
tail -n<number> <path-to-file>

tail -n1 ../../letter

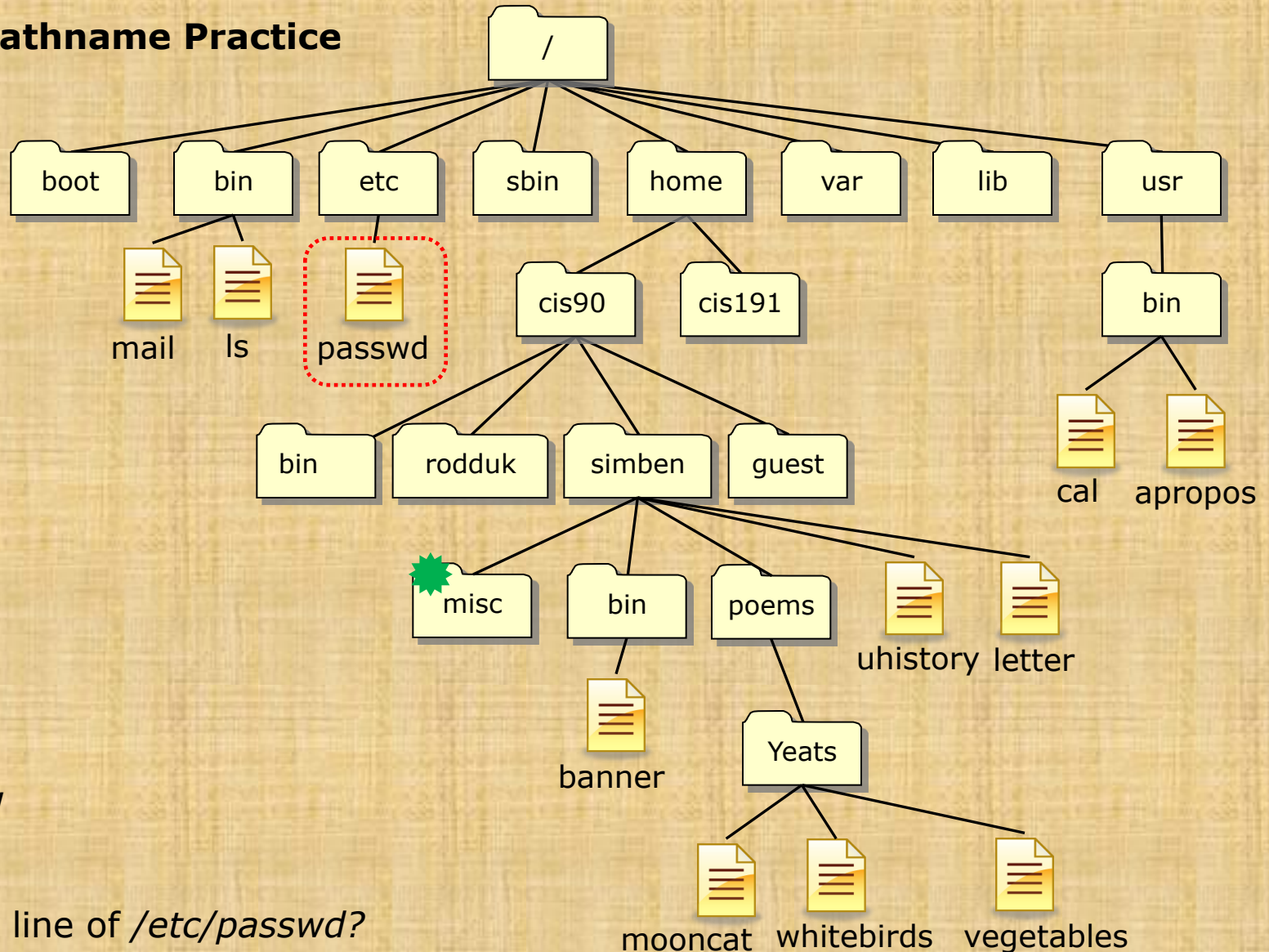
tail -n1 /home/cis90/simben/letter

tail -n1 ~/letter

All these answers are correct



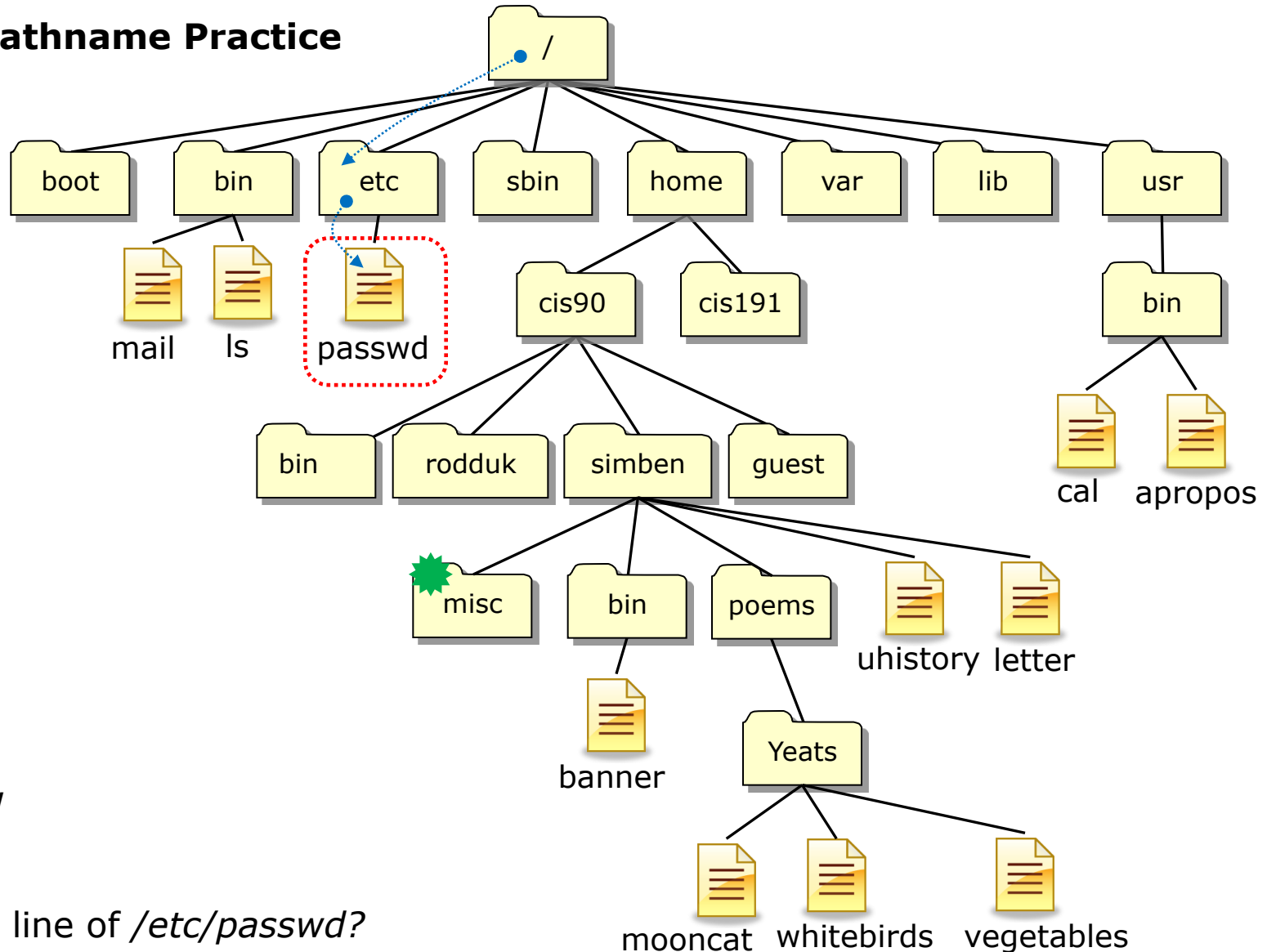
File Tree Pathname Practice



From  how
does Benji:

Print the first line of `/etc/passwd`?

File Tree Pathname Practice



From  how
does Benji:

Print the first line of `/etc/passwd`?

```
/home/cis90/simben/misc $ head -n1 /etc/passwd
```

*Other answers
are also
acceptable*

From  how
does Benji:

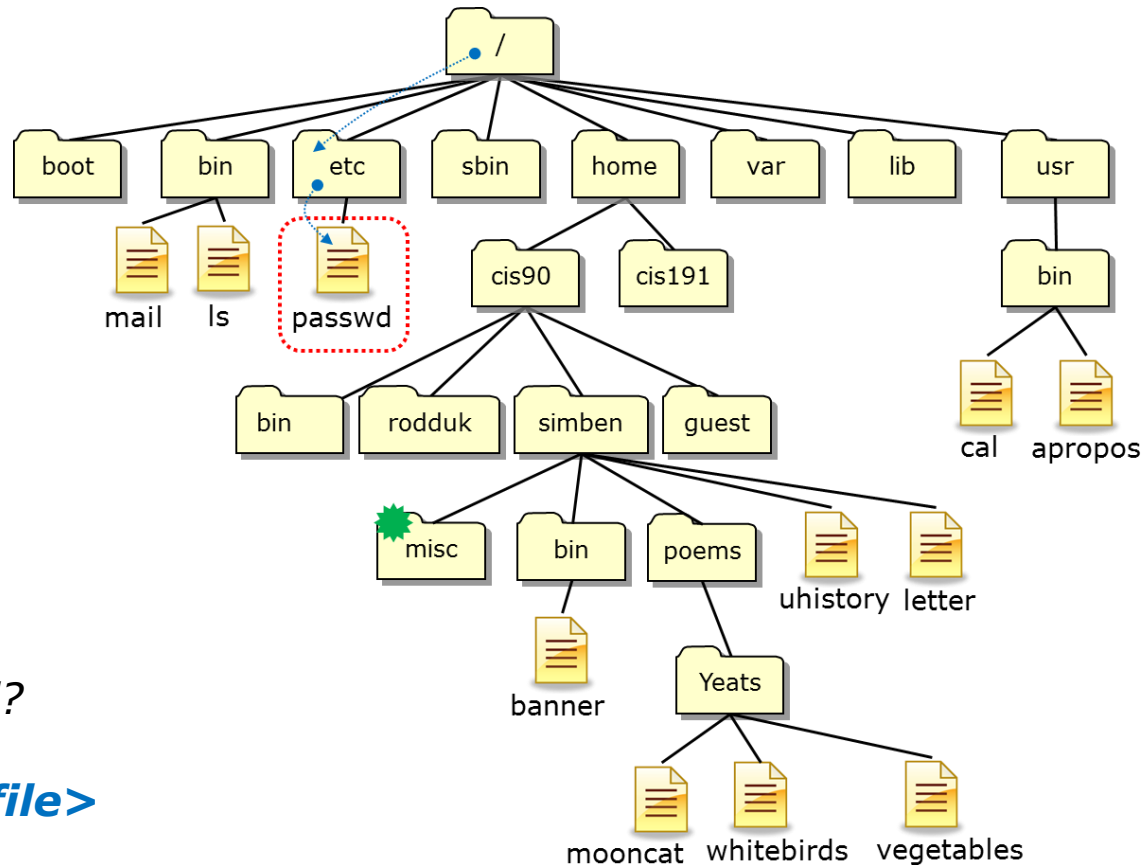
Print the first line of `/etc/passwd`?

`head -n<number> <path-to-file>`

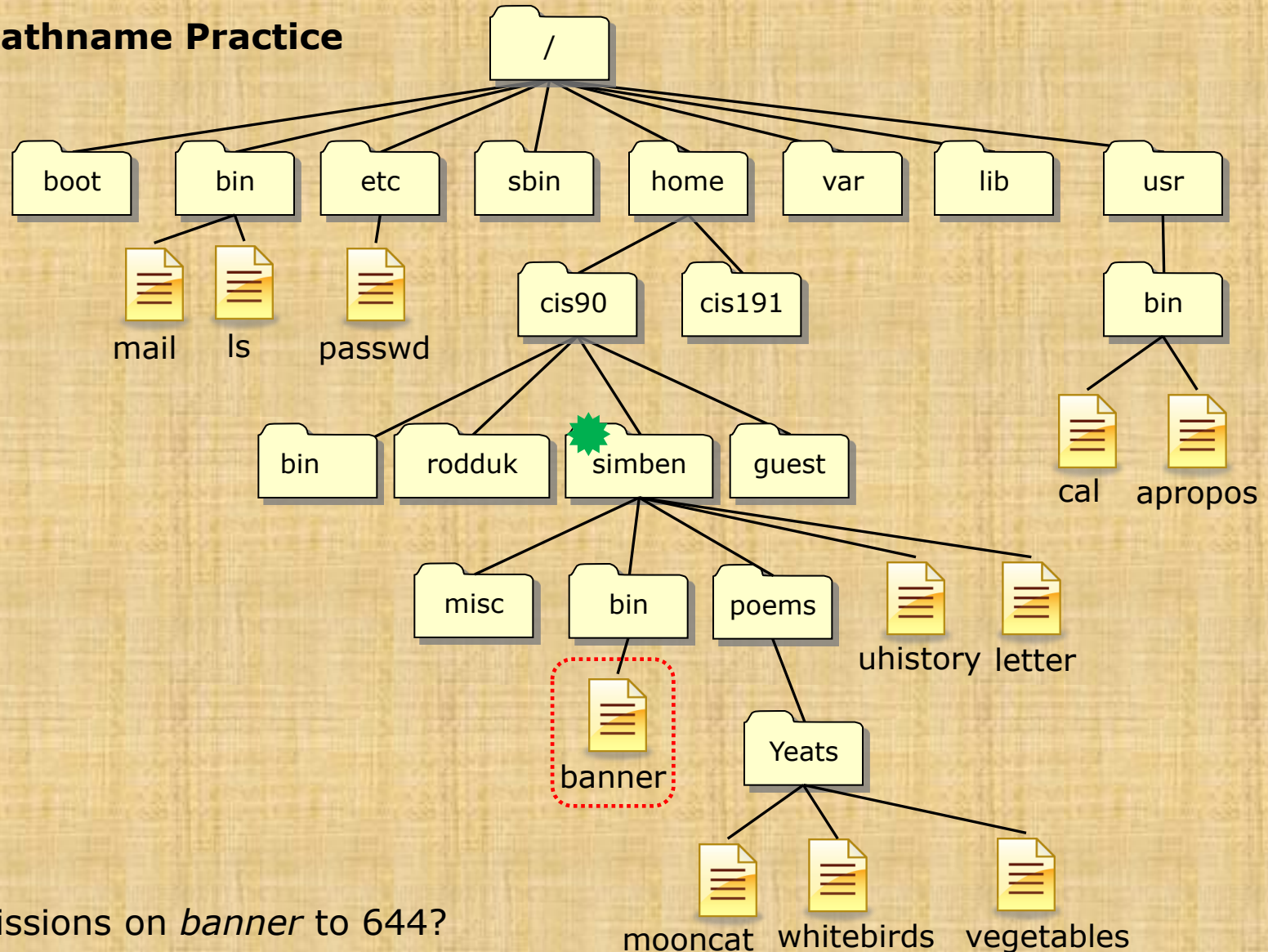
`head -n1 /etc/passwd`

`head -n1 ../../../../etc/passwd`

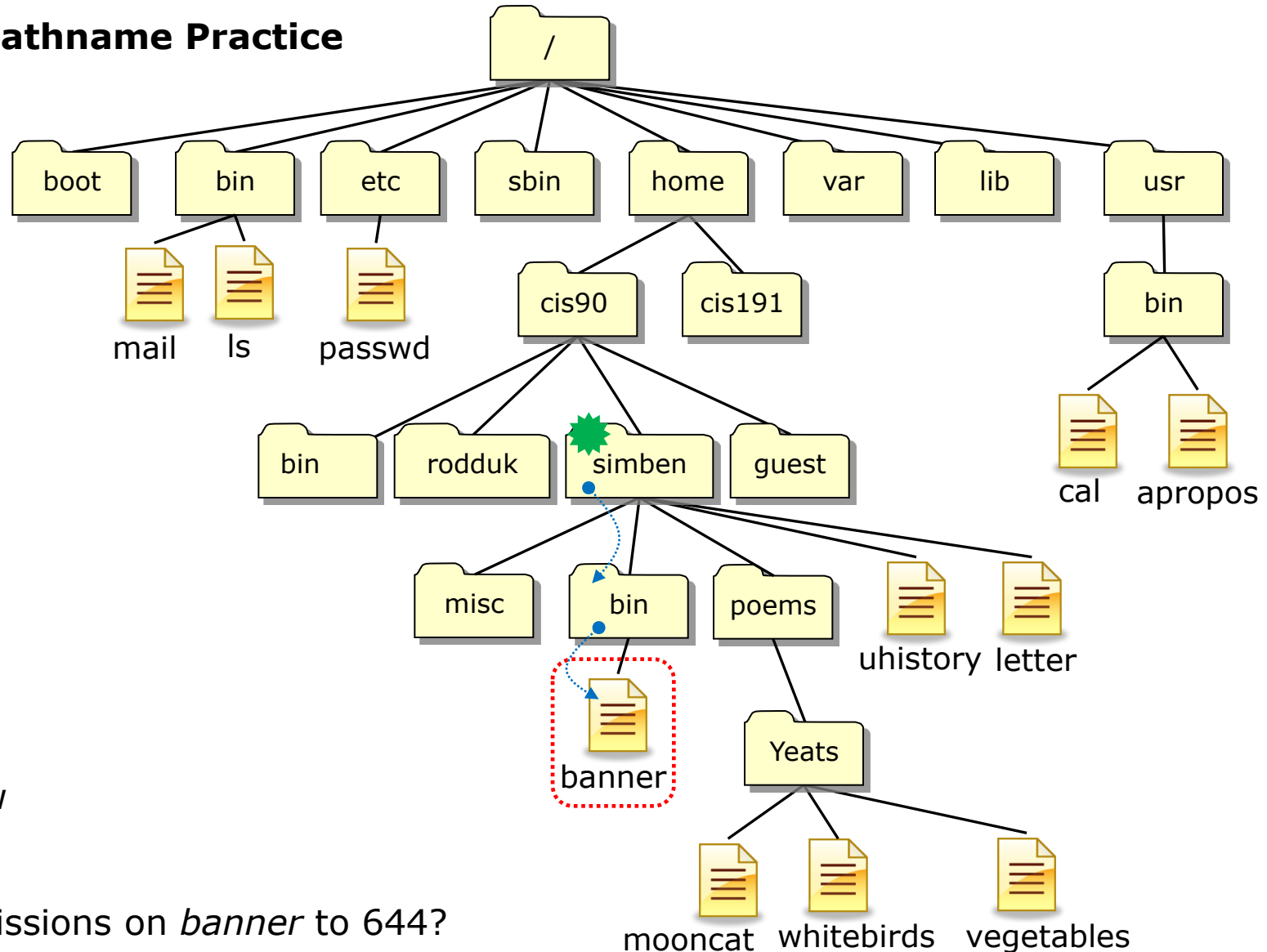
Both these answers are correct



File Tree Pathname Practice



File Tree Pathname Practice

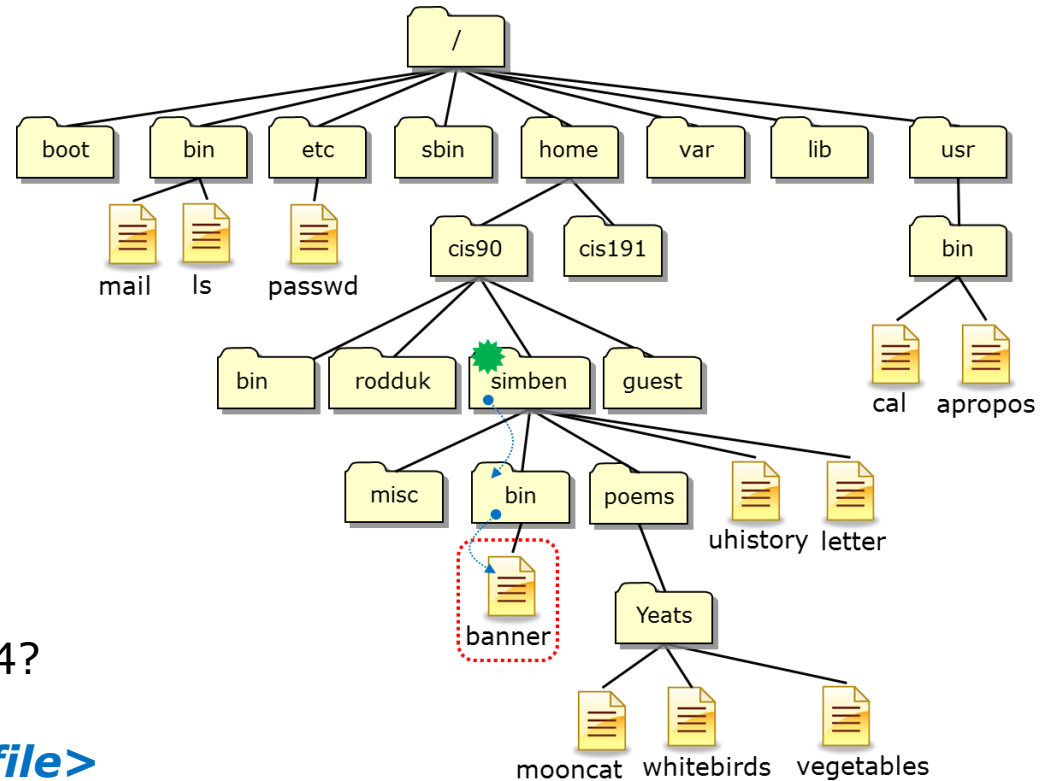


From  how
does Benji:

Change permissions on *banner* to 644?

`/home/cis90/simben $ chmod 644 bin/banner`

*Other answers
are also
acceptable*



From  how
does Benji:

Change permissions on *banner* to 644?

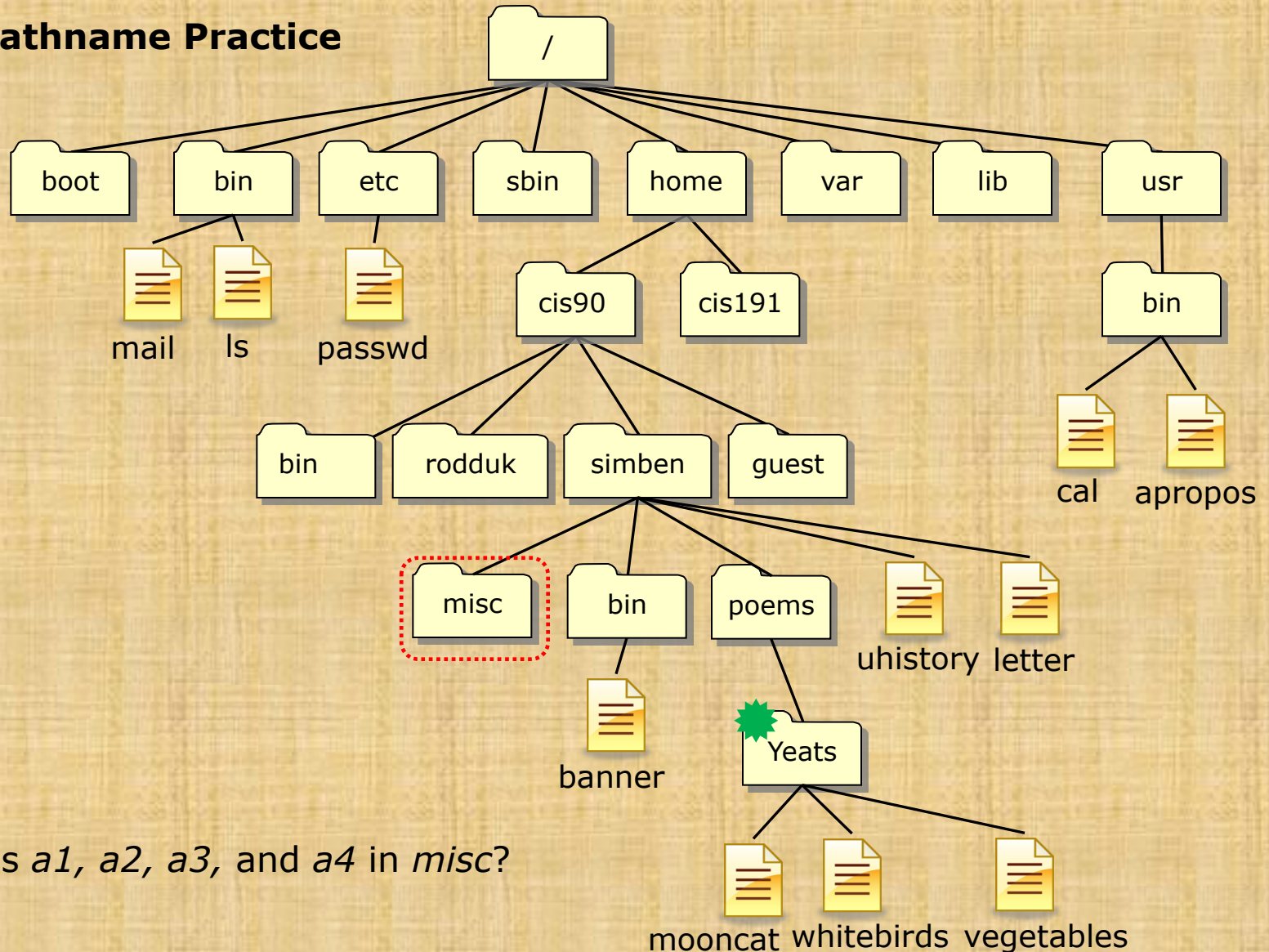
chmod *<permissions>* *<path-to-file>*

chmod 644 bin/banner

chmod 644 /home/cis90/simben/bin/banner

Both these answers are correct

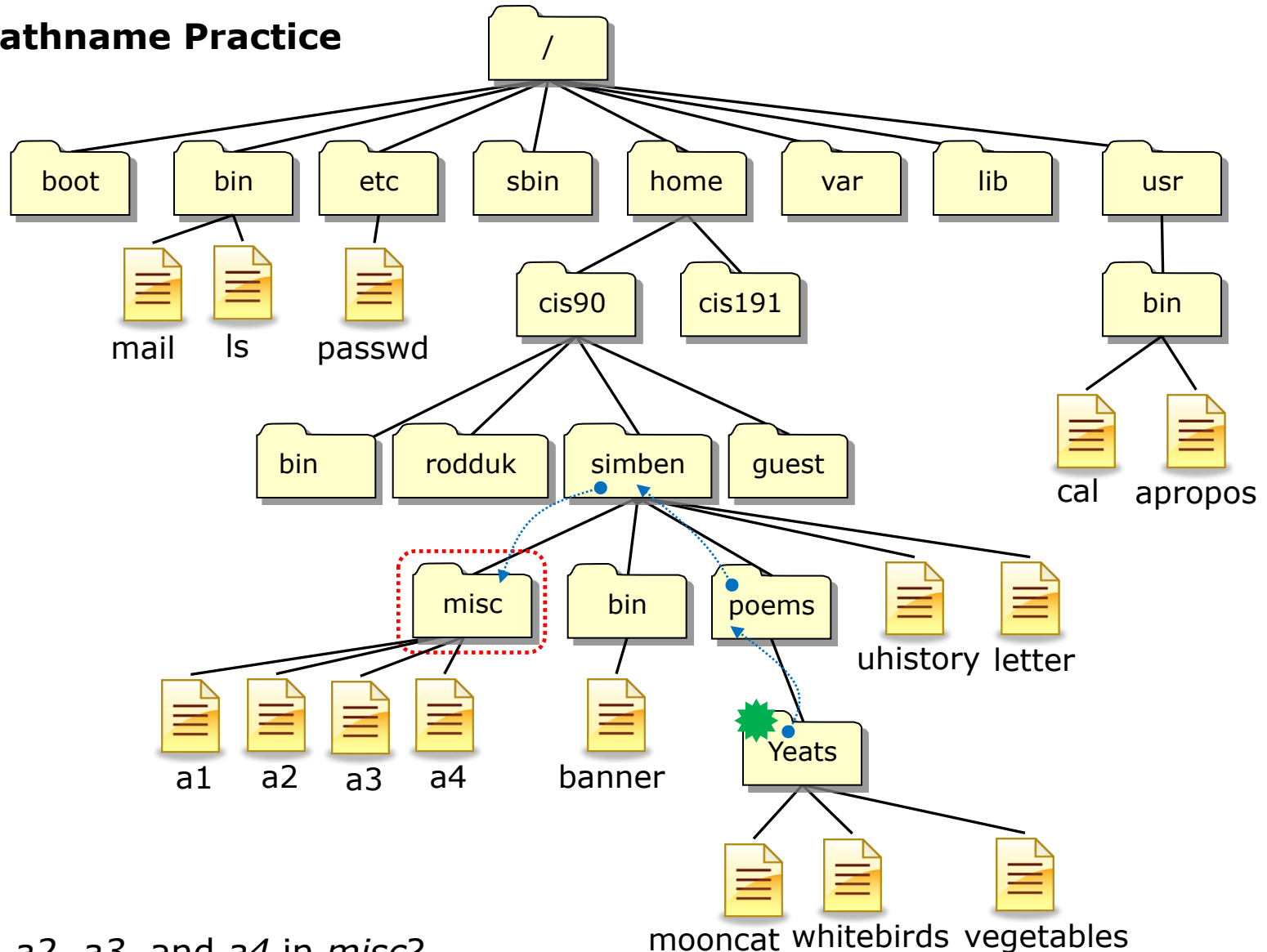
File Tree Pathname Practice



From  how
does Benji:

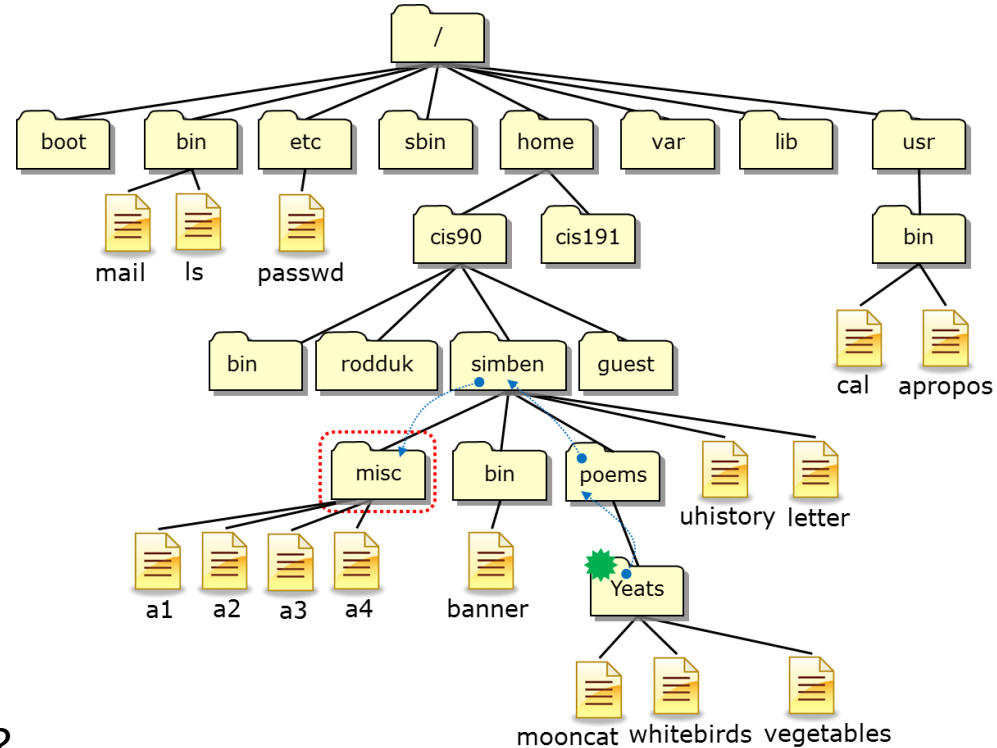
Create new files *a1*, *a2*, *a3*, and *a4* in *misc*?

File Tree Pathname Practice



/home/cis90/simben/poems/Yeats \$ **touch** ../../misc/a1 ../../misc/a2 ../../misc/a3 ../../misc/a4

*Other answers
are also
acceptable*



From  how
does Benji:

Create files *a1*, *a2*, *a3*, and *a4* in *misc*?

touch *<path-to-file>* *<path-to-file>* *<path-to-file>* *<path-to-file>*

touch ../../misc/a1 ../../misc/a2 ../../misc/a3 ../../misc/a4

touch ~/misc/a1 ~/misc/a2 ~/misc/a3 ~/misc/a4

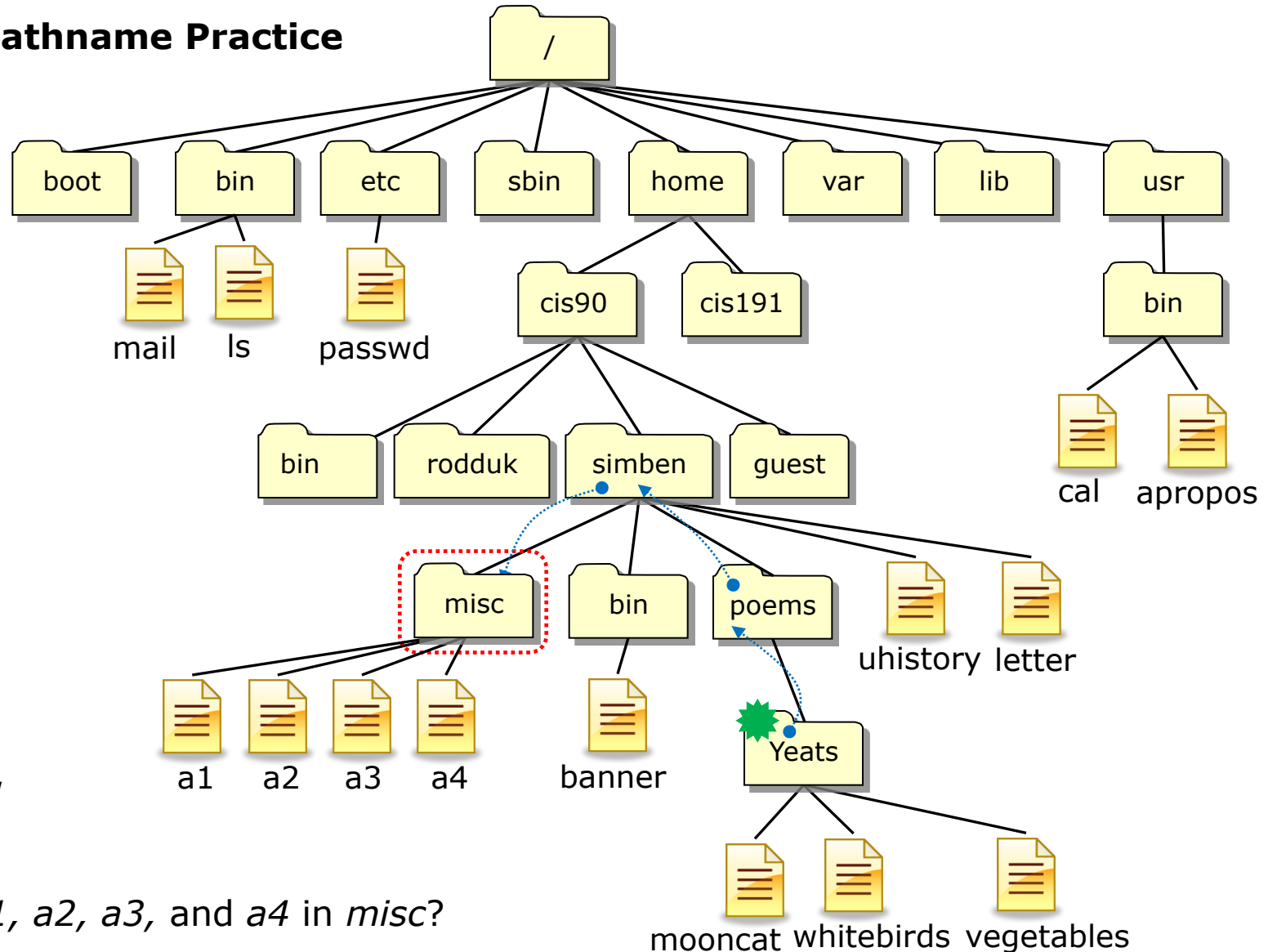
touch /home/cis90/simben/misc/a1 /home/cis90/simben/misc/a2
/home/cis90/simben/misc/a3 /home/cis90/simben/misc/a4 *(all on one line)*

All these answers are correct



*For the aspiring gurus
there is an even better
way to do the last
operation!*

File Tree Pathname Practice



FYI
only

From  how
does Benji:

Create files *a1*, *a2*, *a3*, and *a4* in *misc*?

/home/cis90/simben/poems/Yeats \$ **touch** ~/misc/a{1,2,3,4}

umask continued

Why umask?

Allows users and system administrators to disable specific permissions on new files and directories when they are created.

*Unlike **chmod**, it does **NOT** change the permissions on existing files or directories.*

umask summary

To determine permissions on a new file or directory apply the umask to the initial starting permissions:

- For new files, start with **666**
- For new directories, start with **777**
- For file copies, start with **the permission on the source file**



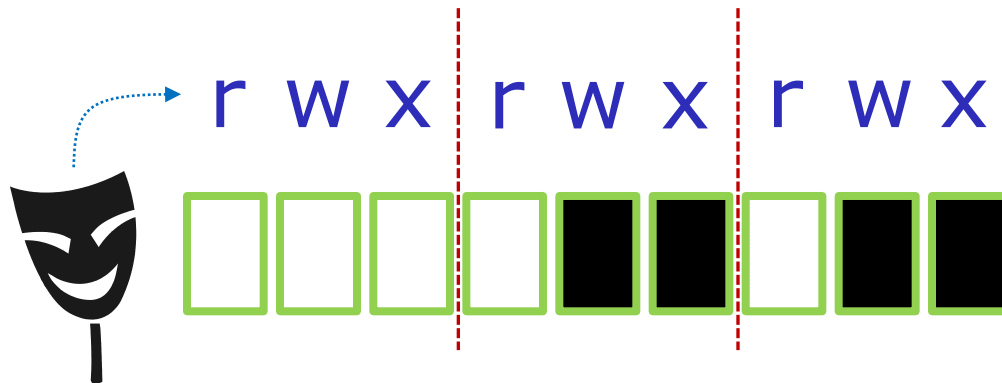
Case 1 – a new directory

With a umask of 033 what permissions would a newly created DIRECTORY have?

Write your answer in the chat window

Case 1 – a new directory

With a umask of 033 what permissions would a newly created DIRECTORY have?



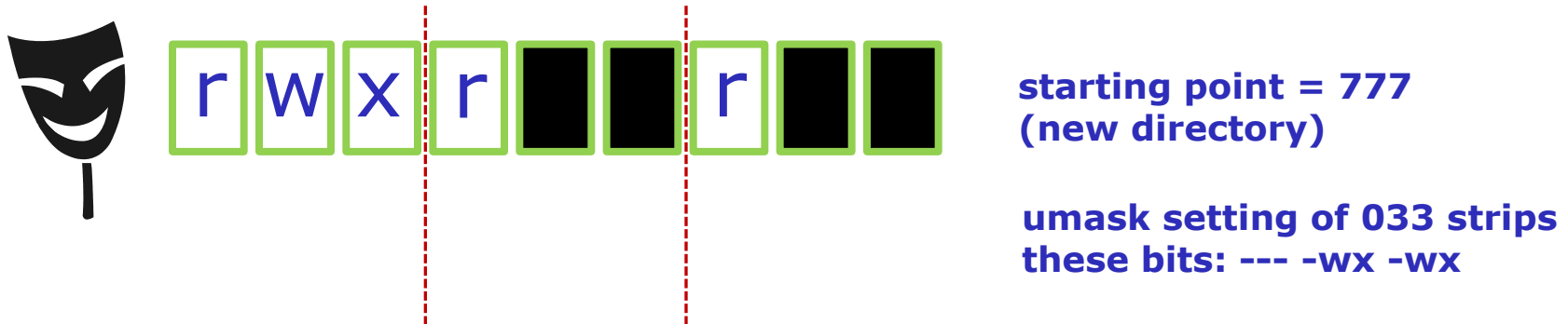
starting point = 777
(new directory)

umask setting of 033 strips
these bits: --- -wx -wx

Now slide the mask up and over the starting point permissions

Case 1 – a new directory

With a umask of 033 what permissions would a newly created DIRECTORY have?



Answer: 744

Prove it to yourself on Opus as shown here

```
/home/cis90ol/simmsben $ umask 033
/home/cis90ol/simmsben $ mkdir brandnewdir
/home/cis90ol/simmsben $ ls -ld brandnewdir/
drwxr--r-- 2 simmsben cis90ol 4096 Apr 21 12:46 brandnewdir/
 7  4  4
```

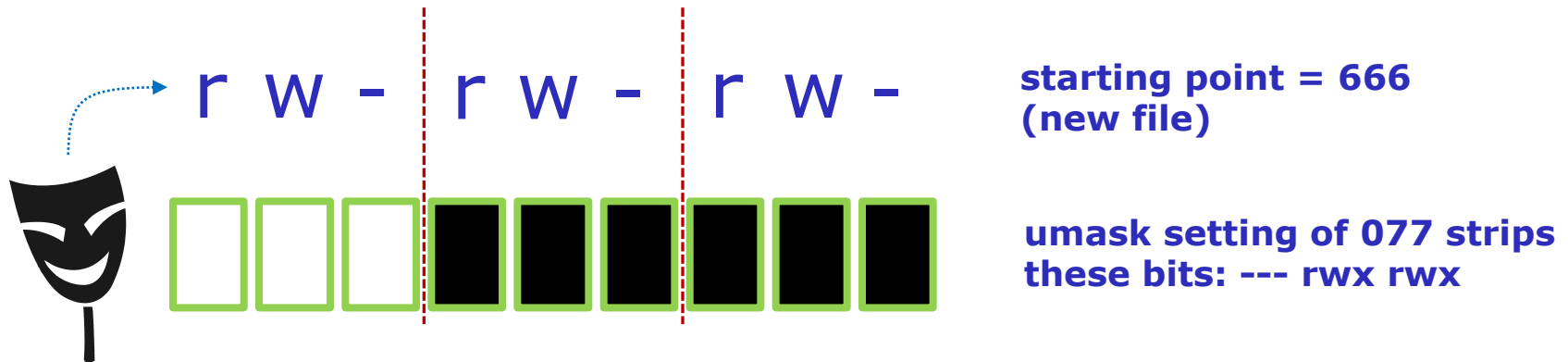
Case 2 – new file

With a umask of 077 what permissions would a newly created FILE have?

Write your answer in the chat window

Case 2 – new file

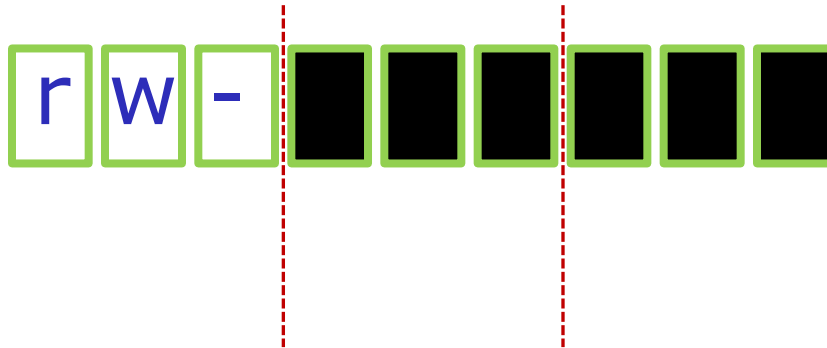
With a umask of 077 what permissions would a newly created FILE have?



Now slide the mask up and over the starting point permissions

Case 2 – new file

With a umask of 077 what permissions would a newly created FILE have?



starting point = 666
(new file)

umask setting of 077 strips
these bits: --- rwx rwx

Answer: 600

Prove it to yourself on Opus as shown here

```
/home/cis90ol/simmsben $ umask 077
/home/cis90ol/simmsben $ touch brandnewfile
/home/cis90ol/simmsben $ ls -l brandnewfile
-rw----- 1 simmsben cis90ol 0 Apr 21 12:50 brandnewfile
 6 0 0
```

Case 3 – file copy

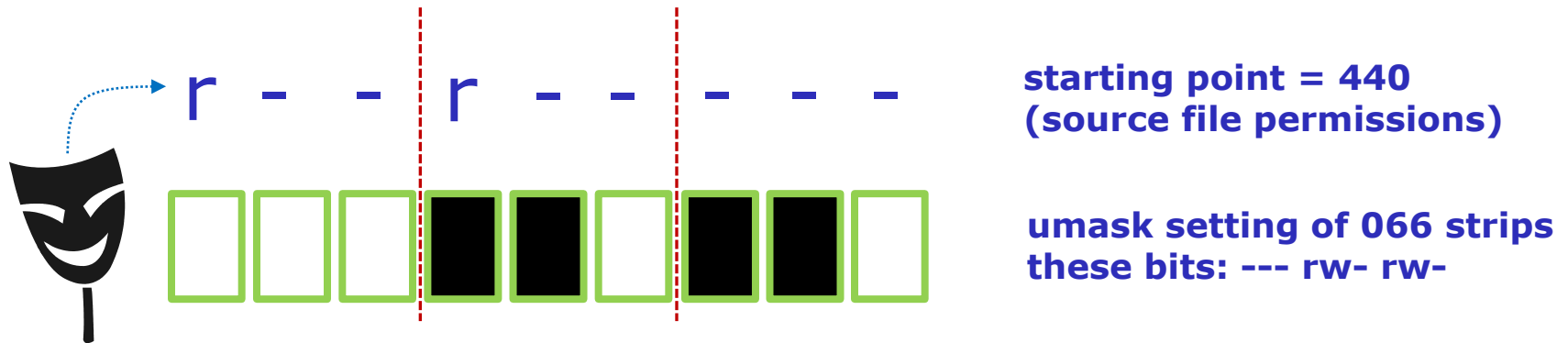
If `umask=066` and the *cinderella* file permissions are 440
What would the permissions be on *cinderella.bak* after:
`cp cinderella cinderella.bak`

Write your answer in the chat window

Case 3 – file copy

If `umask=066` and the *cinderella* file permissions are 440
What would the permissions be on *cinderella.bak* after:

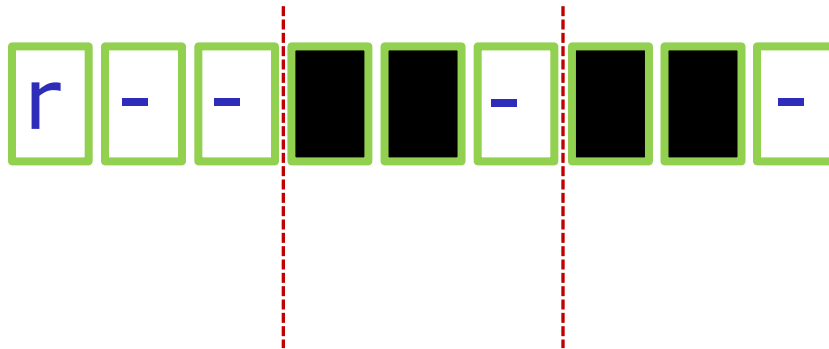
```
cp cinderella cinderella.bak
```



Now slide the mask up and over the starting point permissions

Case 3 – file copy

If **umask=066** and the *cinderella* file permissions are **440**
What would the permissions be on *cinderella.bak* after:
`cp cinderella cinderella.bak`



starting point = 440
(source file permissions)

umask setting of 066 strips
these bits: --- rw- rw-

Answer: 400

Prove it to yourself on Opus as shown here

```
/home/cis90/simben $ touch cinderella
/home/cis90/simben $ chmod 440 cinderella
/home/cis90/simben $ umask 066
/home/cis90/simben $ cp cinderella cinderella.bak
/home/cis90/simben $ ls -l cinderella.bak
-r----- 1 simben90 cis90 0 Oct 22 09:17 cinderella.bak
  4   0   0
```



Housekeeping

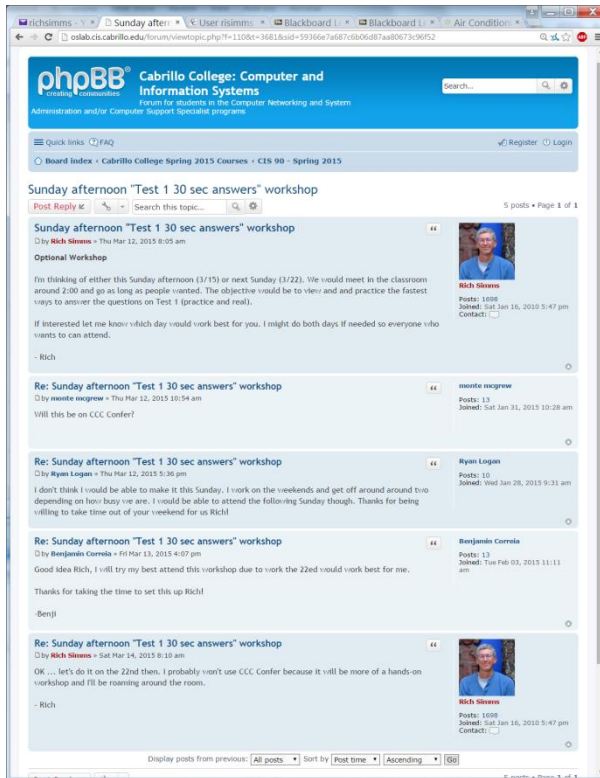
Previous material and assignment

1. Lab 6 due 11:59PM
2. A **check6** script is available



**Don't forget to submit
Lab 6 with the submit script!**

3. Five more posts due 11:59PM
4. Early preview of Lab X2 is now available. This is recommended for anyone wanting more practice with pathnames.



Sunday Afternoon Optional Workshop

Date: March 22nd

Time: 2:00PM till whenever

Place: Room 828

Objective: Learn how to do Test 1 questions in 30 seconds or less

In person workshop, no CCC Confer

Perkins/VTEA Survey

Carl D. Perkins Vocational and Technical Education Act

POSTREPLY Search this topic... Search 2 posts • Page 1 of 1

Carl D. Perkins Vocational and Technical Education Act
by Rich Simms » Wed Sep 24, 2014 7:24 am

The Carl D. Perkins Vocational and Technical Education Act was originally authorized by Congress in 1984. It was reauthorized in 1998 and again in 2006. This act provides federal funding for improving career technical education (CTE) within the United States in order to help the economy.

For Cabrillo College to receive a portion of this funding students in technical classes must fill out a survey. The more surveys completed the more funds the college will receive. The survey only needs to be completed once per term by each student.

This survey can be completed online using web advisor:

Log on to WEBADVISOR at <https://wave.cabrillo.edu>

Select "STUDENTS: Click Here" (navy blue bar)

- Under "Academic Profile" Click on "Student Update Form"
- Use drop down list under "Select the earliest term for which you are registered" and click on the current term.
- Select "SUBMIT"

Scroll down to the "Career Technical Information"

- Answer questions by clicking on the circle to the left of your "Yes" or "No" answers
- You can get details about a question by clicking on blue underlined phrase
- After answering all questions Select "SUBMIT"

Then "LOG OUT"

Thank you for taking a few minutes to help Cabrillo College CS/CIS programs!

- Rich

Rich Simms
Posts: 1401
Joined: Sat Jan 16, 2010 6:47 pm

This is an important source of funding for Cabrillo College.

*Send me an email stating you completed this survey for **three points extra credit!***

(even if you filled out the survey in another class)

Career Technical Information
Your answers to these questions will help qualify Cabrillo College for Perkins/VTEA grant funds.

Are you currently receiving benefits from:

☐ Yes ☒ No TANF/CALWORKS

☐ Yes ☒ No SSI (Supplemental Security Income)

☐ Yes ☒ No GA (General Assistance)

☐ Yes ☒ No Does your income qualify you for a fee waiver?

☐ Yes ☒ No Are you a single parent with custody of one or more minor children?

☐ Yes ☒ No Are you a displaced homemaker attending Cabrillo to develop job skills?

☐ Yes ☒ No Have you moved in the preceding 36 months to obtain, or to accompany parents or spouses to obtain, temporary or seasonal employment in agriculture, dairy, or fishing?

<http://oslab.cis.cabrillo.edu/forum/viewtopic.php?f=101&t=3191>

Monitoring your grades

Points that could have been earned:

5 quizzes:	15 points
5 labs:	150 points
1 test:	30 points
1 forum quarter:	20 points
Total:	215 points

The CIS 90 website

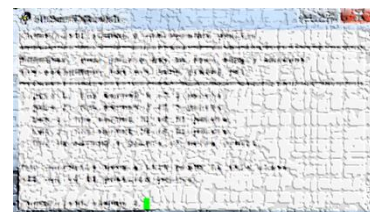


*Send me your
survey to get your
LOR code name.*

<http://simms-teach.com/cis90grades.php>

Or on Opus

checkgrades *codename*
(where *codename* is your LOR codename)



*The checkgrades script was written by
Jessie a past CIS 90 Alumnus*

Percentage	Total Points	Letter Grade	Pass/No Pass
90% or higher	504 or higher	A	Pass
80% to 89.9%	448 to 503	B	Pass
70% to 79.9%	392 to 447	C	Pass
60% to 69.9%	336 to 391	D	No pass
0% to 59.9%	0 to 335	F	No pass

*At the end of the term I'll add up all
your points and assign you a grade
using this table*

sort command



Tools for your toolbox



sort - sorts the lines in a file

sort command

Basic syntax

(see man page for the rest of the story)

sort *<options>* *<filepath>*

The **sort** command can read lines from a file or *stdin* and sort them.

The **-r** option will do a reverse sort

sort command

Get the *names* file to use for the next series of slides

```
/home/cis90/simben $ cd
```

return to home directory

```
/home/cis90/simben $ cp ../depot/names .
```

relative path to the names file in the depot directory

Think of the single dot file as "here" (it is hard linked to the current directory)

```
/home/cis90/simben $ cat names  
duke  
benji  
star  
homer
```

Pretend you are a
command

(use your
imagination)

Shell Steps

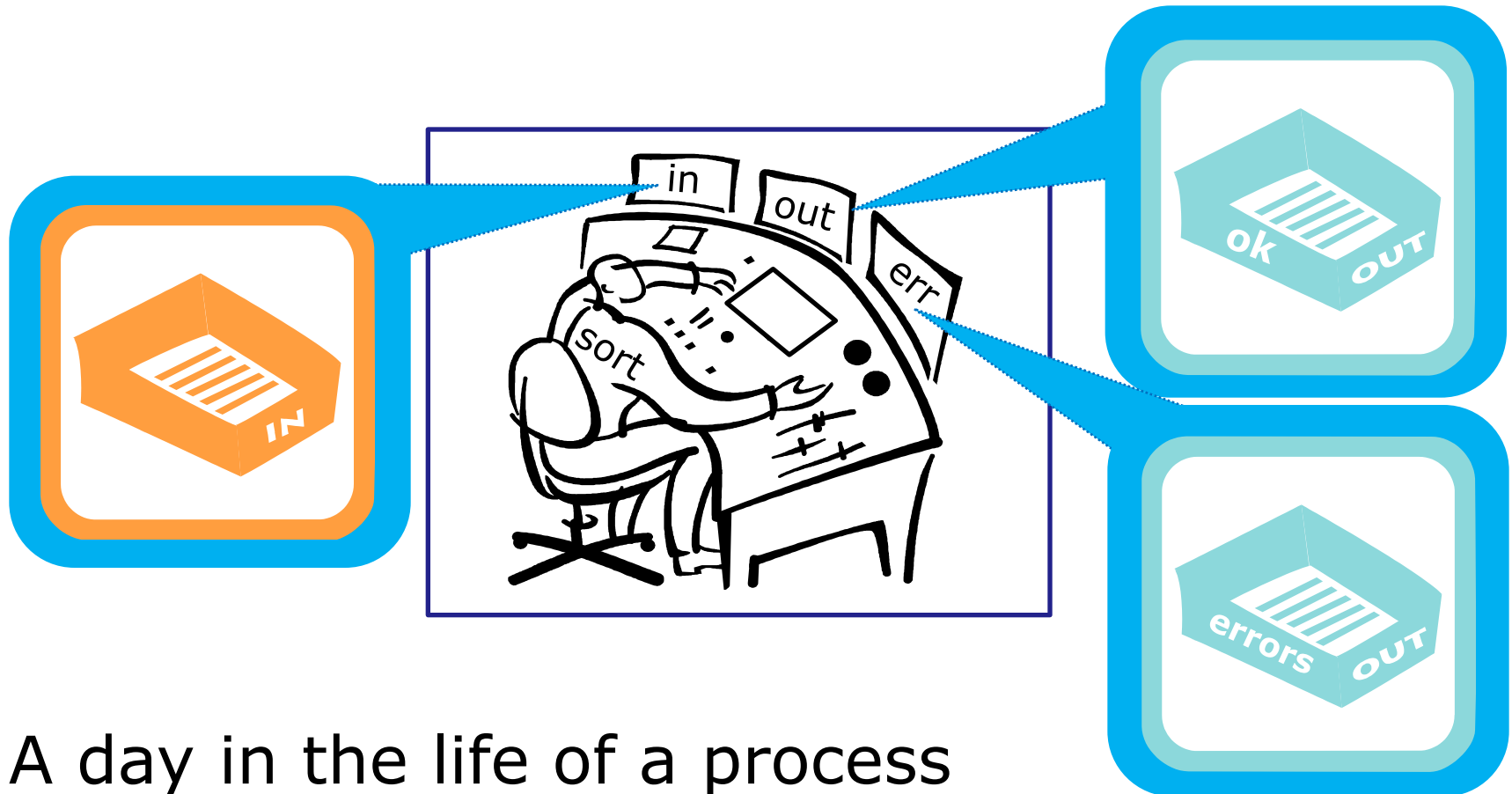
- 1) Prompt
- 2) Parse
- 3) Search
- 4) Execute
- 5) Nap
- 6) Repeat

Let's visualize being the sort program and being loaded into memory and executing



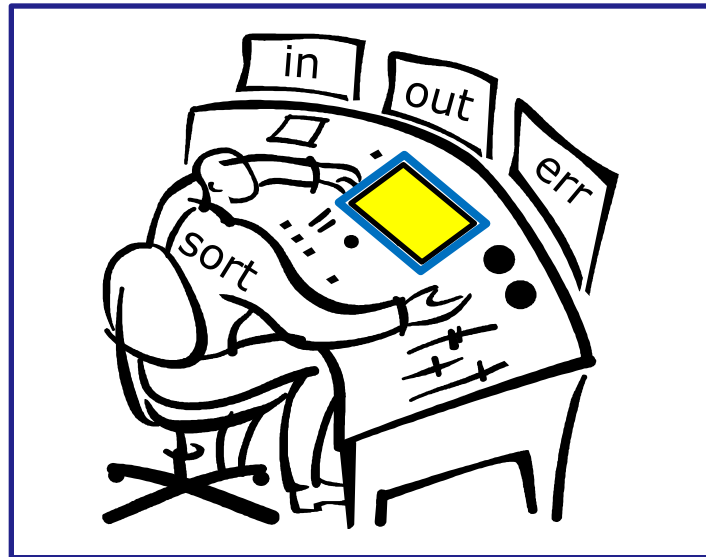
A day in the life of a process

*Looking around you notice there is one
in tray and two out trays*



A day in the life of a process

You also notice an instruction window on your desk. This is where you find out about any options or arguments the shell passes on to you.



A day in the life of a process

sort command

sort command with a filename argument

```
/home/cis90/simben $ cat names
```

```
duke  
benji  
star  
homer
```

```
/home/cis90/simben $ sort names
```

```
benji  
duke  
homer  
star
```

The sort command will sort the lines in a file and output the sorted lines

sort names (deep dive)

```
/home/cis90/simben $ sort names
```

```
benji
```

```
duke
```

```
homer
```

```
star
```

```
/home/cis90/simben $
```

One argument

```
/home/cis90/simben $ sort names
```

Shell Steps

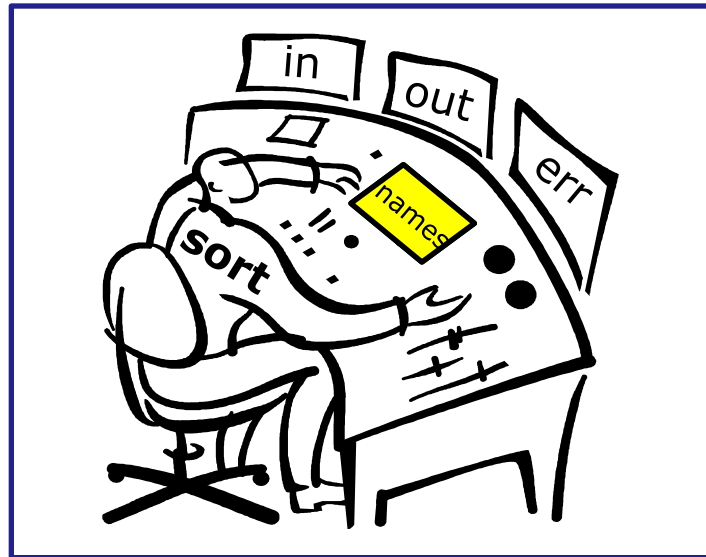
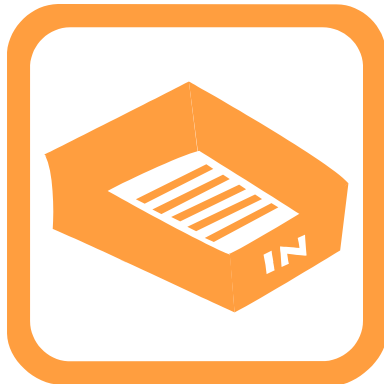
- 1) Prompt
- 2) Parse
- 3) Search
- 4) Execute
- 5) Nap
- 6) Repeat

1. Prompt string is: `/home/cis90/simben $ "`
2. Parsing results:
 - `command = sort`
 - `no options`
 - `1 argument = "names"`
 - `no redirection`
3. Search user's path and locate the sort program in `/bin`
4. Sort loaded into memory and execution begins

Shell Steps

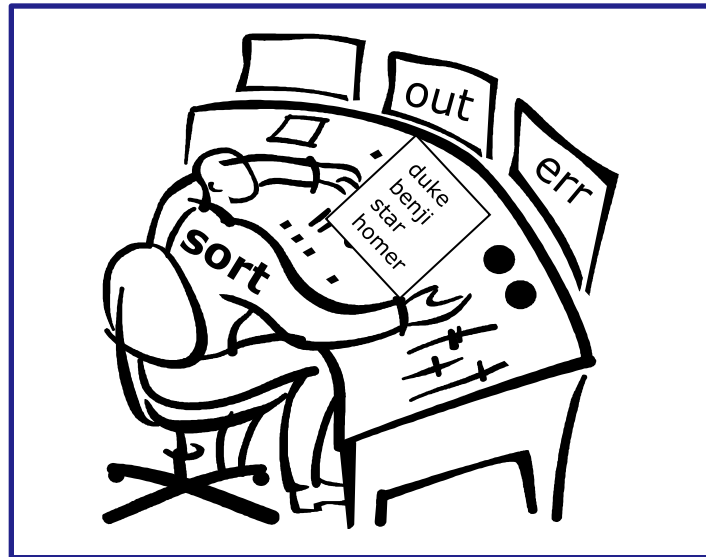
- 1) Prompt
- 2) Parse
- 3) Search
- 4) Execute
- 5) Nap
- 6) Repeat

/home/cis90/simben \$ **sort names**



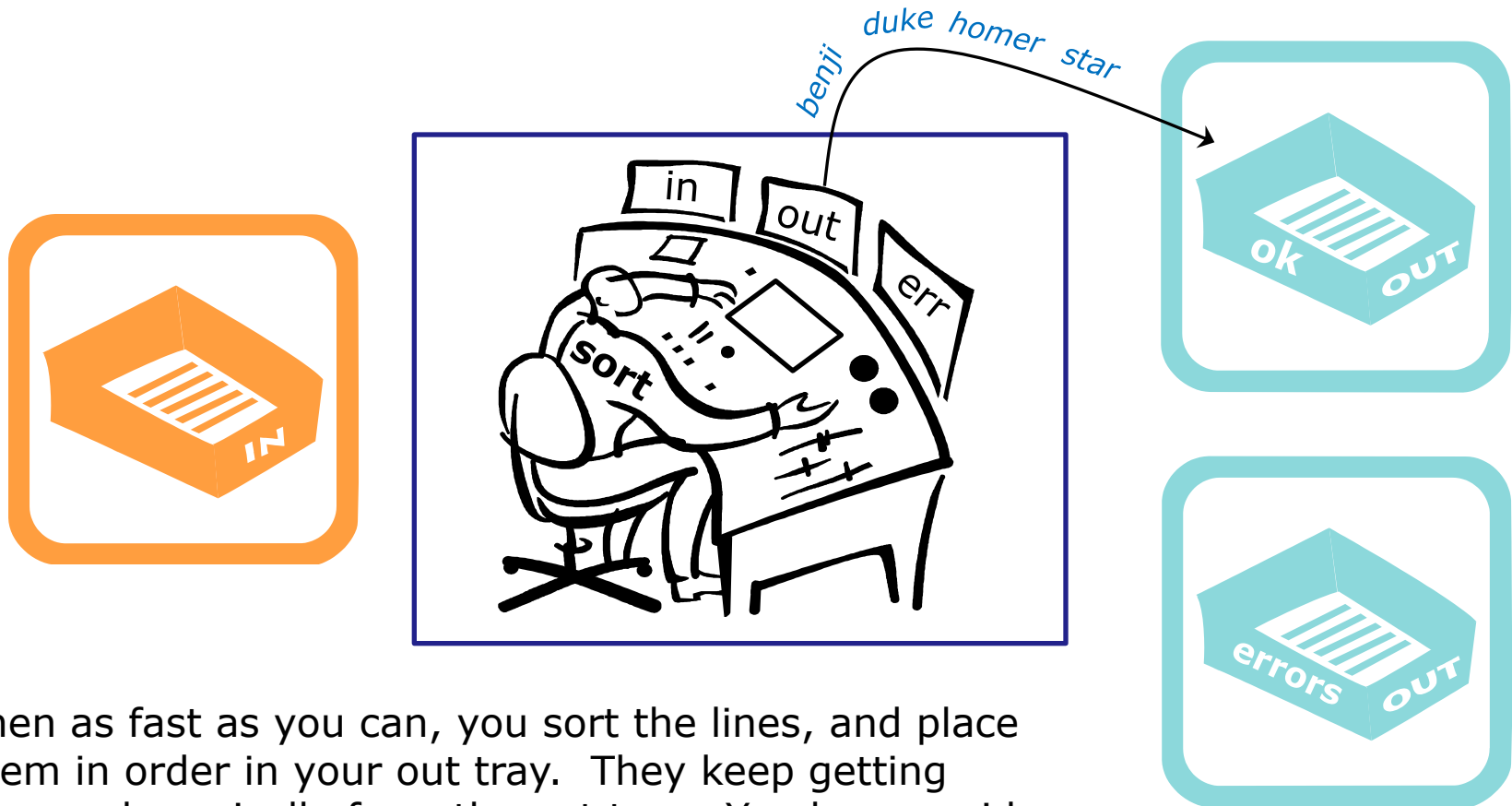
You (the sort process) check your instruction window and see the shell passed one argument "names" to you. You know (given your internal DNA code) that you must contact the kernel and request this file be opened and the contents read.

/home/cis90/simben \$ **sort names**



Note: Once the names file is opened you read in each line one at a time until you reach the EOF (End of File).

/home/cis90/simben \$ **sort names**



Then as fast as you can, you sort the lines, and place them in order in your out tray. They keep getting removed magically from the out tray. You have no idea where they go after that. You are done.

sort command

sort command with no arguments

```
/home/cis90/simben $ sort
```

*If no filename was specified, **sort** will read input from the keyboard*

kayla

sky

bella

benji

charlie

bella

benji

charlie

kayla

sky



Ctrl-D specifies the EOF (End Of File).

After sort receives the EOF it sorts the lines and outputs them

sort

(deep dive)

```
/home/cis90/simben $ sort
```

```
kayla
```

```
sky
```

```
bella
```

```
benji
```

```
charlie
```

```
bella
```

```
benji
```

```
charlie
```

```
kayla
```

```
sky
```

```
/home/cis90/simben $
```

No arguments

EOF

```
/home/cis90/simben $ sort
```

Shell Steps

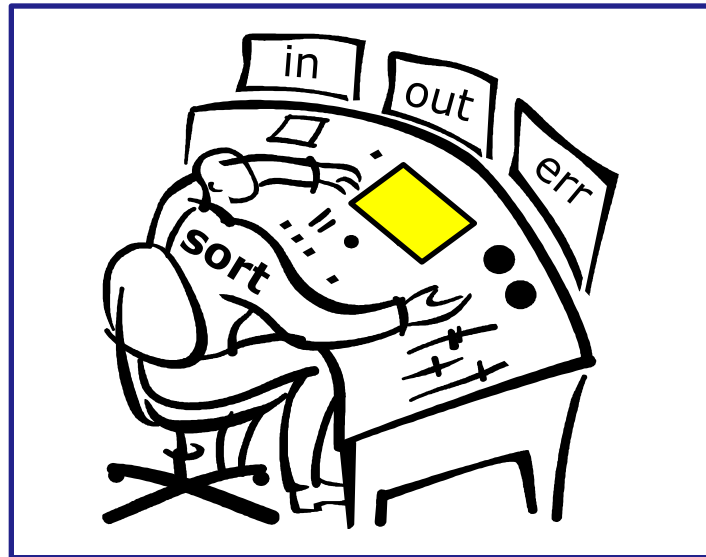
- 1) Prompt
- 2) Parse
- 3) Search
- 4) Execute
- 5) Nap
- 6) Repeat

1. Prompt string is: `"/home/cis90/simben $ "`
2. Parsing results:
 - `command = sort`
 - no options
 - no arguments
 - no redirection
3. Search user's path and locate the sort program in `/bin`
4. Sort loaded into memory and execution begins

Shell Steps

- 1) Prompt
- 2) Parse
- 3) Search
- 4) Execute
- 5) Nap
- 6) Repeat

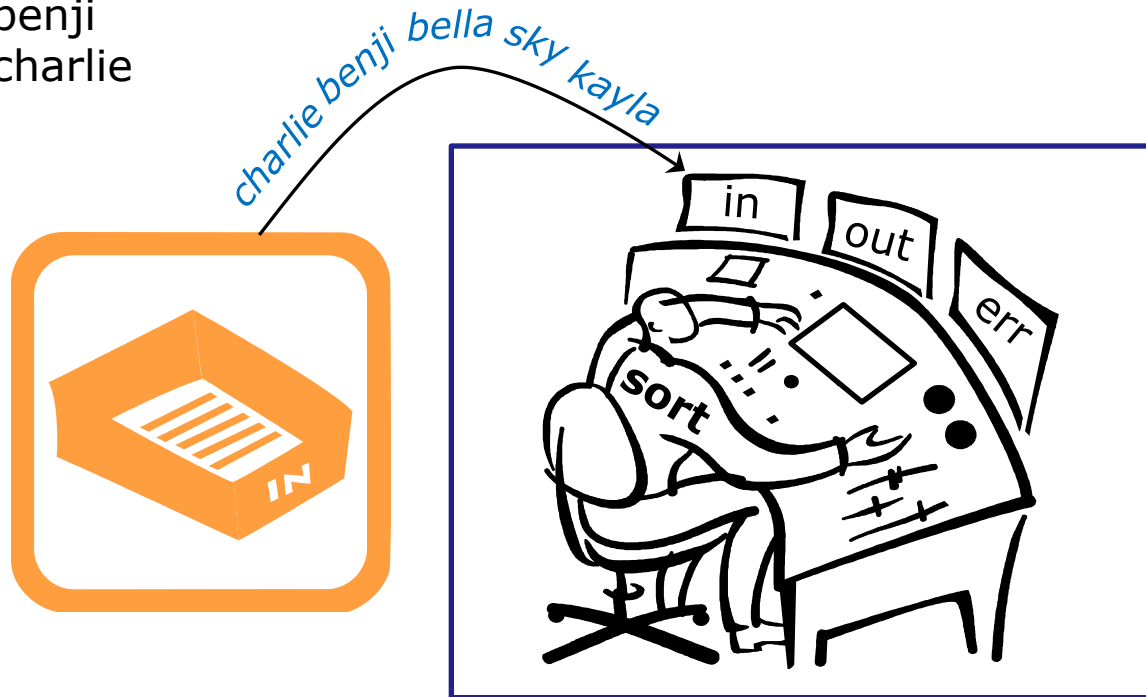
```
/home/cis90/simben $ sort
```



You (the sort process) check your instruction window and see that no options or arguments were passed to you from the shell to handle. You know (given your internal DNA code) that with no arguments you must look for lines to sort in your in tray, so you reach in to grab the first line to sort.

/home/cis90/simben \$ **sort**

kayla
sky
bella
benji
charlie



You work hard and fast. Each time you reach into the in tray there is another line! They just magically keep appearing into your in tray. You have no idea where they are coming from.


```
/home/cis90/simben $ sort
```

```
kayla  
sky  
bella  
benji  
charlie
```

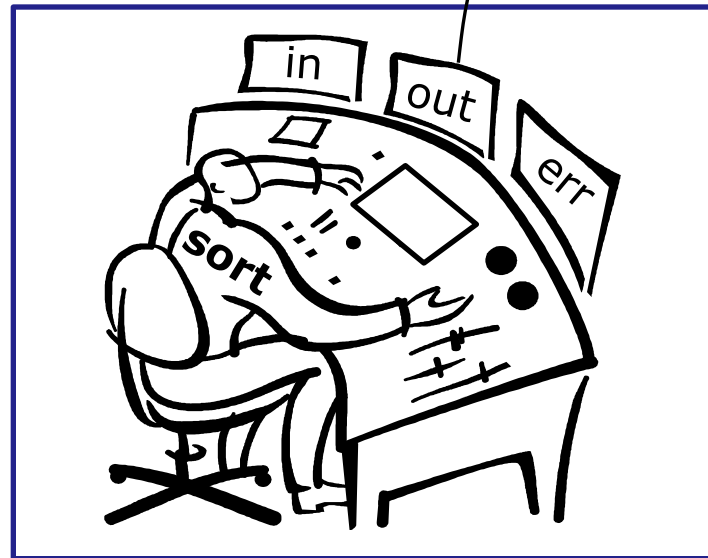


EOF

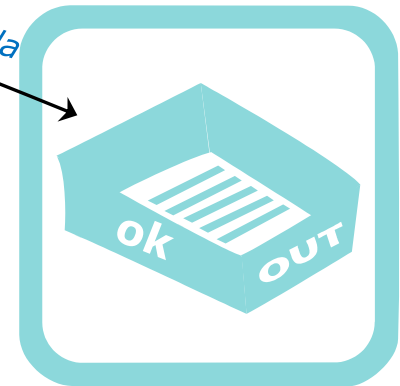


Then suddenly, when you reach for the next line, you find an EOF. You know (your internal DNA code) that this EOF means no more lines coming. You must sort what you have collected so far and place them, in order, into your out tray.

bella
benji
charlie
kayla
sky
/home/cis90/simben \$



sky kayla charlie benji bella



As fast as you can, you sort them, and place them in order in your out tray. They keep getting removed magically from the out tray. You have no idea where they go after that. You are done.

sort command

sort command with bad argument

```
/home/cis90/simben $ sort bogus  
sort: open failed: bogus: No such file or directory  
/home/cis90/simben $
```

The sort program will try and open any file it receives as an argument and print an error message if the file does not exist

sort bogus (deep dive)

```
/home/cis90/simben $ sort bogus  
sort: open failed: bogus: No such file or directory  
/home/cis90/simben $
```

 *No such file*

```
/home/cis90/simben $ sort bogus
```

Shell Steps

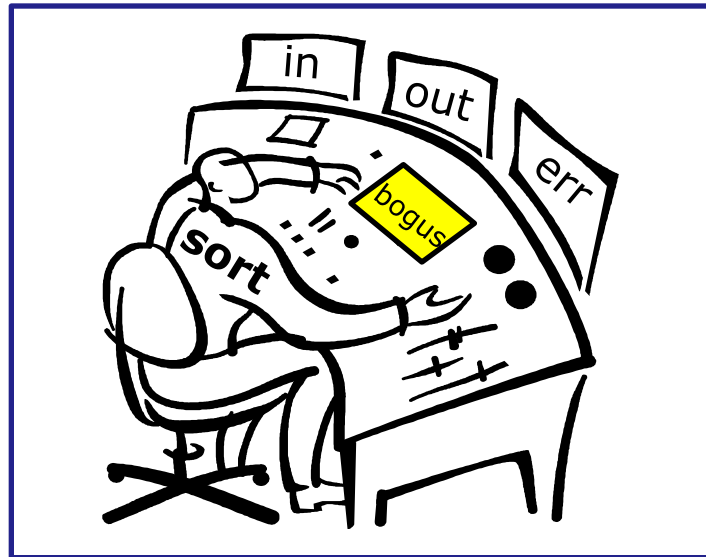
- 1) Prompt
- 2) Parse
- 3) Search
- 4) Execute
- 5) Nap
- 6) Repeat

1. Prompt string is: `"/home/cis90/simben $ "`
2. Parsing results:
 - `command = sort`
 - `no options`
 - `1 argument = bogus`
 - `no redirection`
3. Search user's path and locate the sort program in `/bin`
4. Sort command loaded into memory and execution begins

/home/cis90/simben \$ **sort bogus**

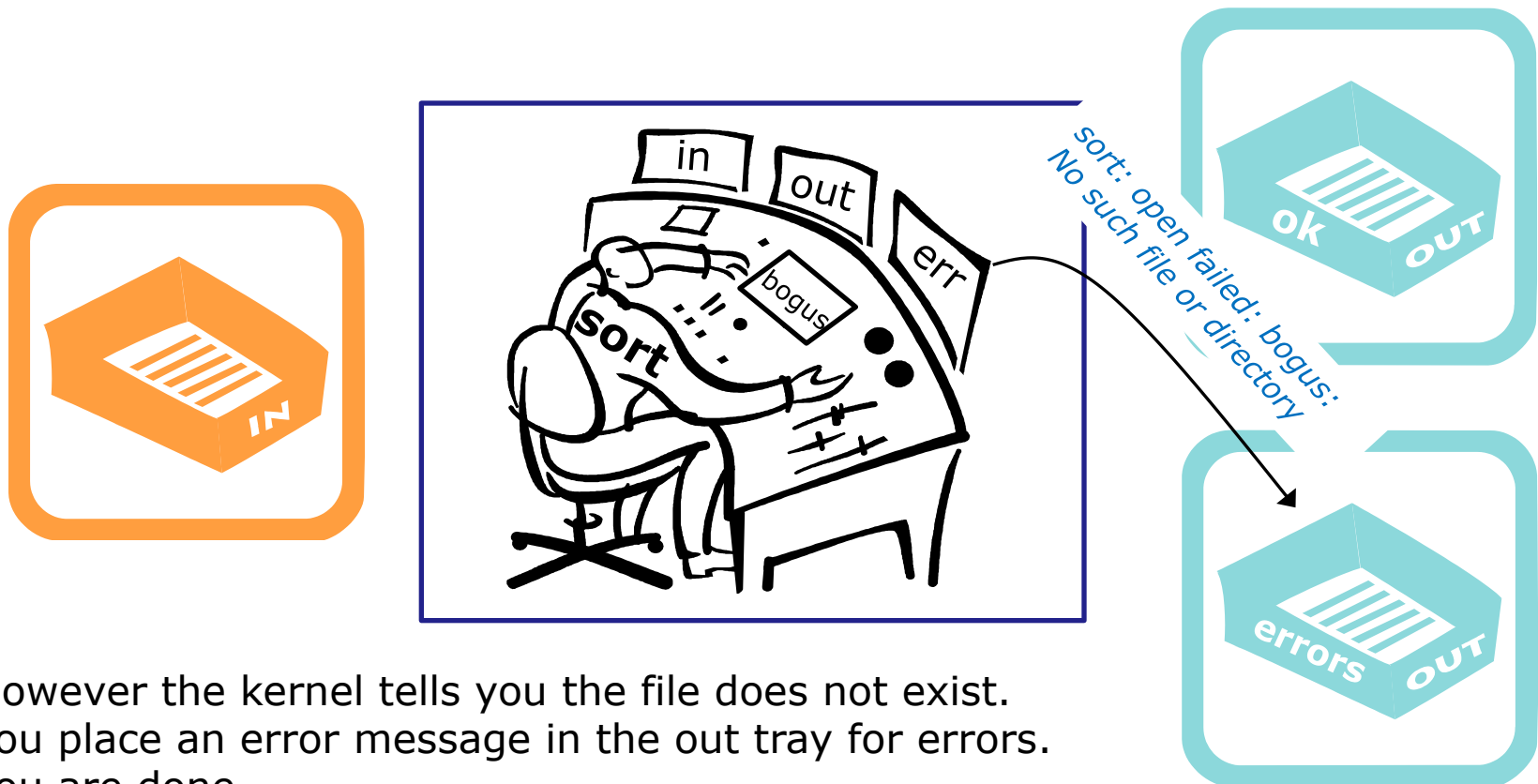
Shell Steps

- 1) Prompt
- 2) Parse
- 3) Search
- 4) Execute
- 5) Nap
- 6) Repeat



You check the instruction window and notice the shell passed you one argument: "bogus". You know (given your internal DNA code) that you must contact the kernel and request this file be opened.

```
/home/cis90/simben $ sort bogus
sort: open failed: bogus: No such file or directory
```



However the kernel tells you the file does not exist.
You place an error message in the out tray for errors.
You are done.

File Descriptors

Input and Output

File Descriptors

Every process is given three open files upon its execution. These open files are inherited from the shell.

stdin

Standard Input (0)

defaults to the user's terminal keyboard

stdout

Standard Output (1)

defaults to the user's terminal screen

stderr

Standard Error (2)

defaults to the user's terminal screen

bringing it
home

Ok, lets make the visualization a little more realistic

The in and out trays are really the three open file descriptors inherited from the shell:
stdin (0), **stdout (1)** and **stderr (2)**.

stdin (0)



stdout (1)



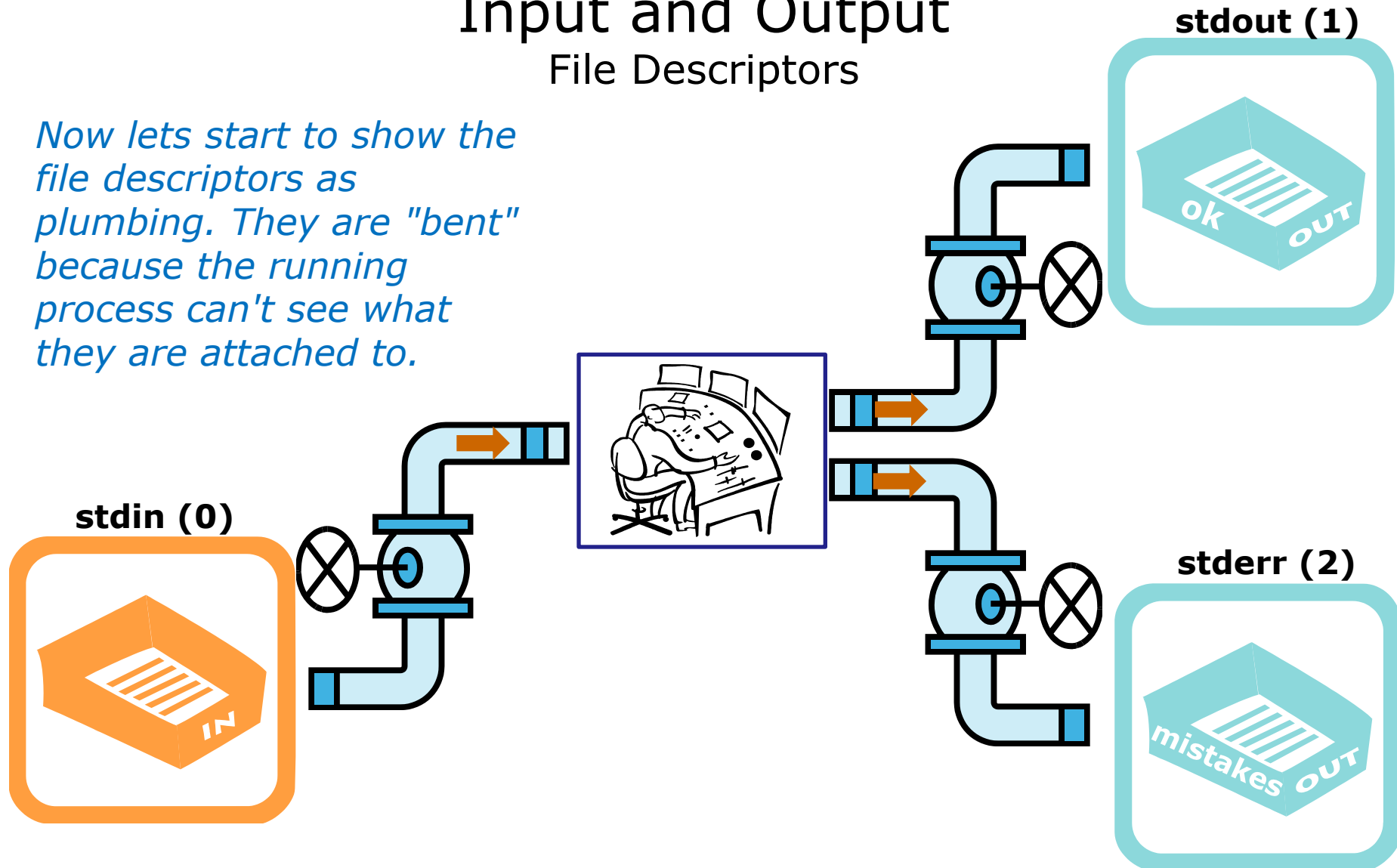
stderr (2)



Input and Output

File Descriptors

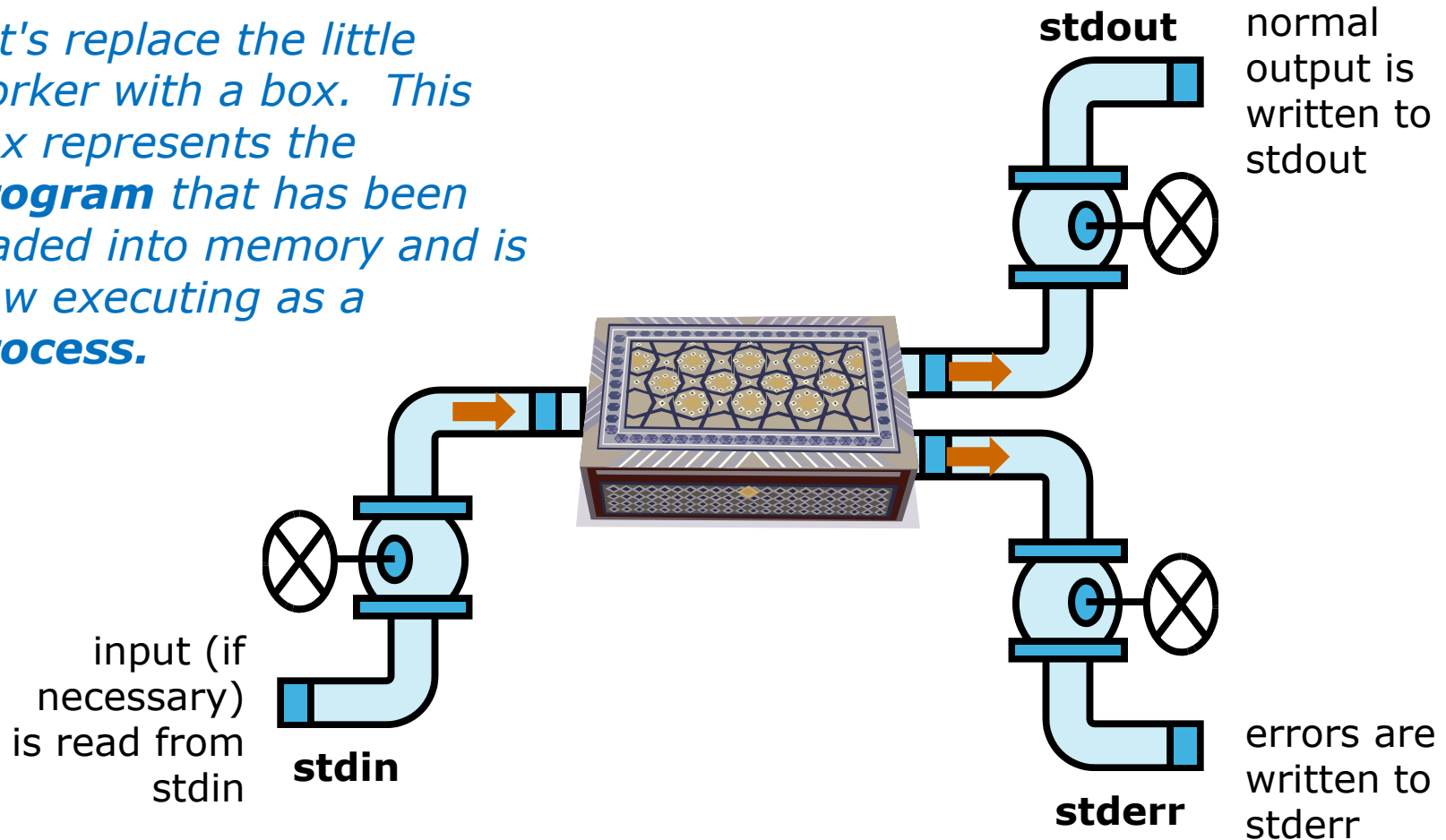
Now lets start to show the file descriptors as plumbing. They are "bent" because the running process can't see what they are attached to.



Input and Output

File Descriptors

*Let's replace the little worker with a box. This box represents the **program** that has been loaded into memory and is now executing as a **process**.*

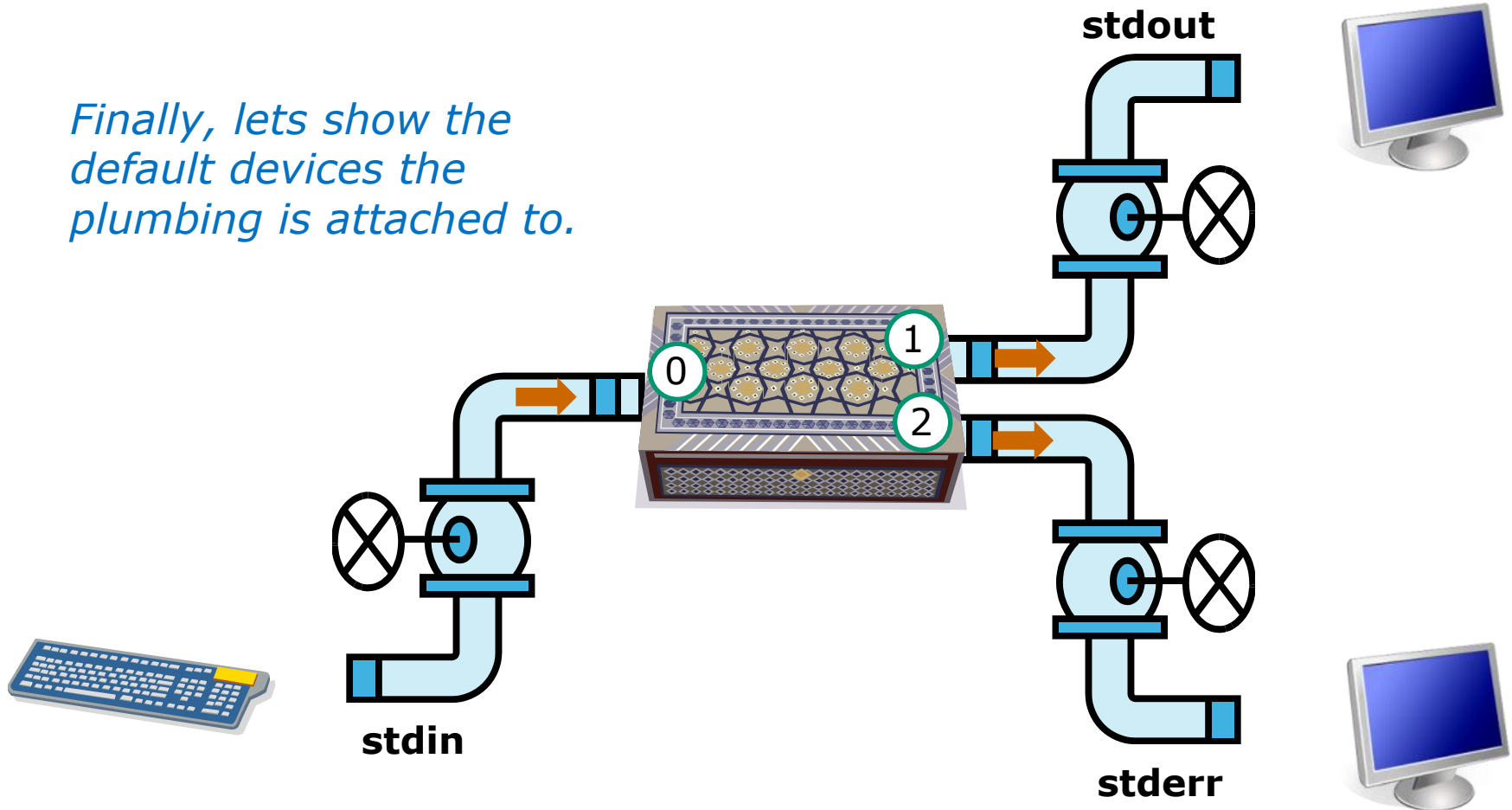


Input and Output

File Descriptors

By default is attached to the user's terminal device (screen)

Finally, lets show the default devices the plumbing is attached to.



By default is attached to the user's terminal device (keyboard)

By default is attached to the user's terminal device (screen)

Input and Output

File Descriptors

```
[simmsben@opus ~]$ sort
```

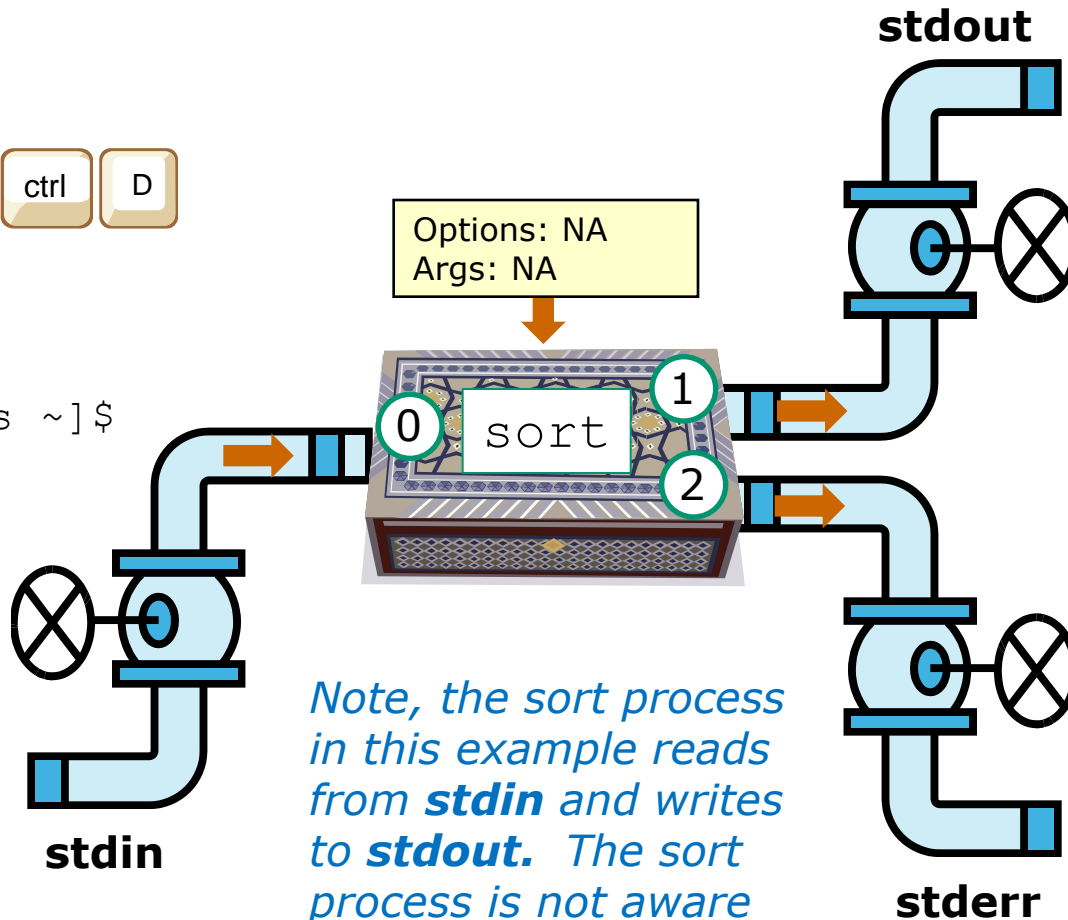
```
star
benji
duke
homer
benji
duke
homer
star
```



```
[simmsben@opus ~]$
```



```
star
benji
duke
homer
```



*Note, the sort process in this example reads from **stdin** and writes to **stdout**. The sort process is not aware what **stdin** or **stdout** are attached to*



```
benji
duke
homer
star
```

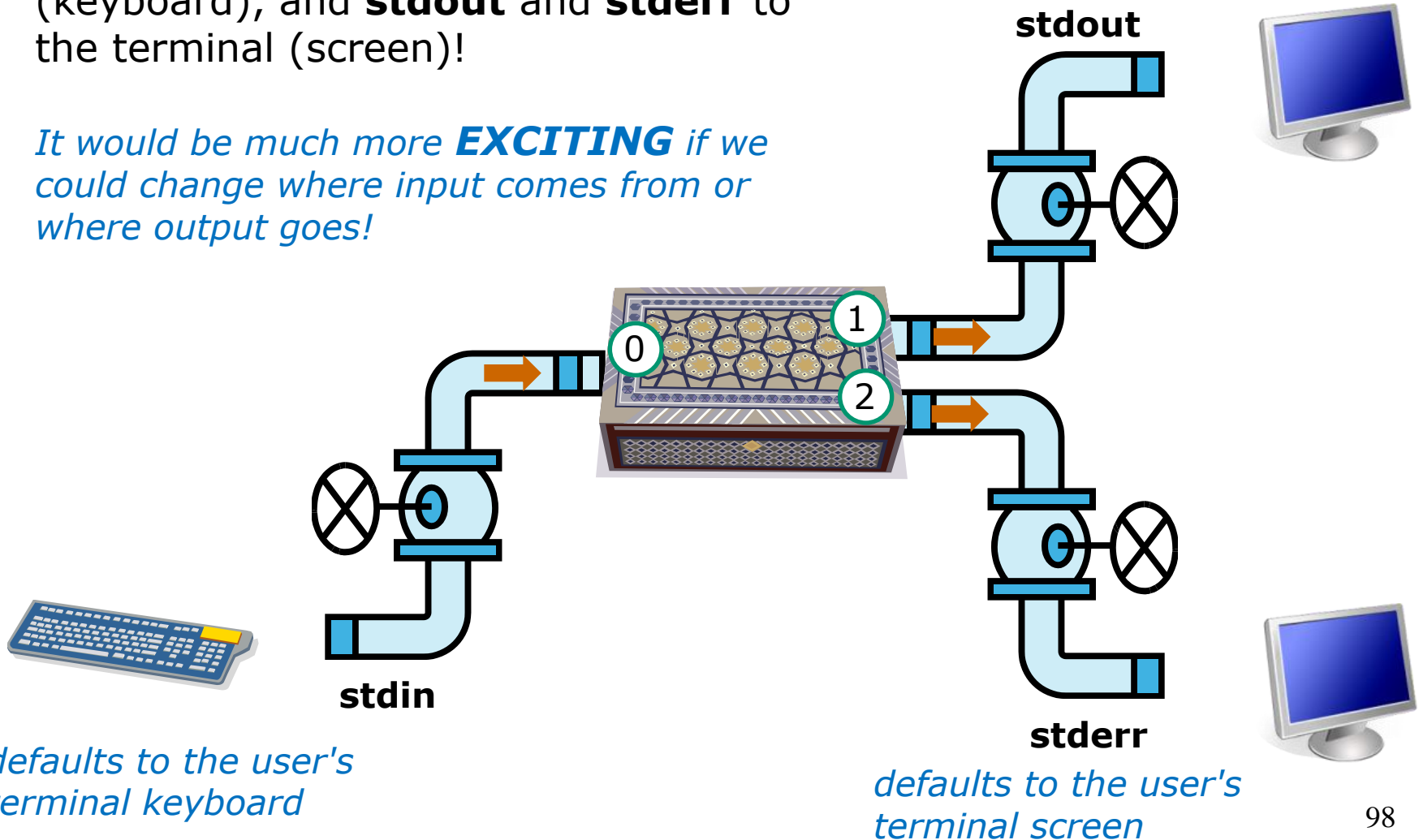


File Redirection

Life would be **BORING** if **stdin** was always attached to the terminal (keyboard), and **stdout** and **stderr** to the terminal (screen)!

defaults to the user's terminal screen

*It would be much more **EXCITING** if we could change where input comes from or where output goes!*

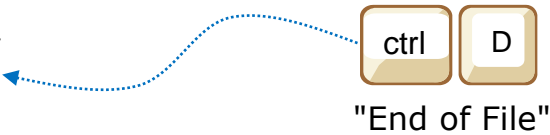


Input and Output

File Redirection

*Let's look at the
sort example again*

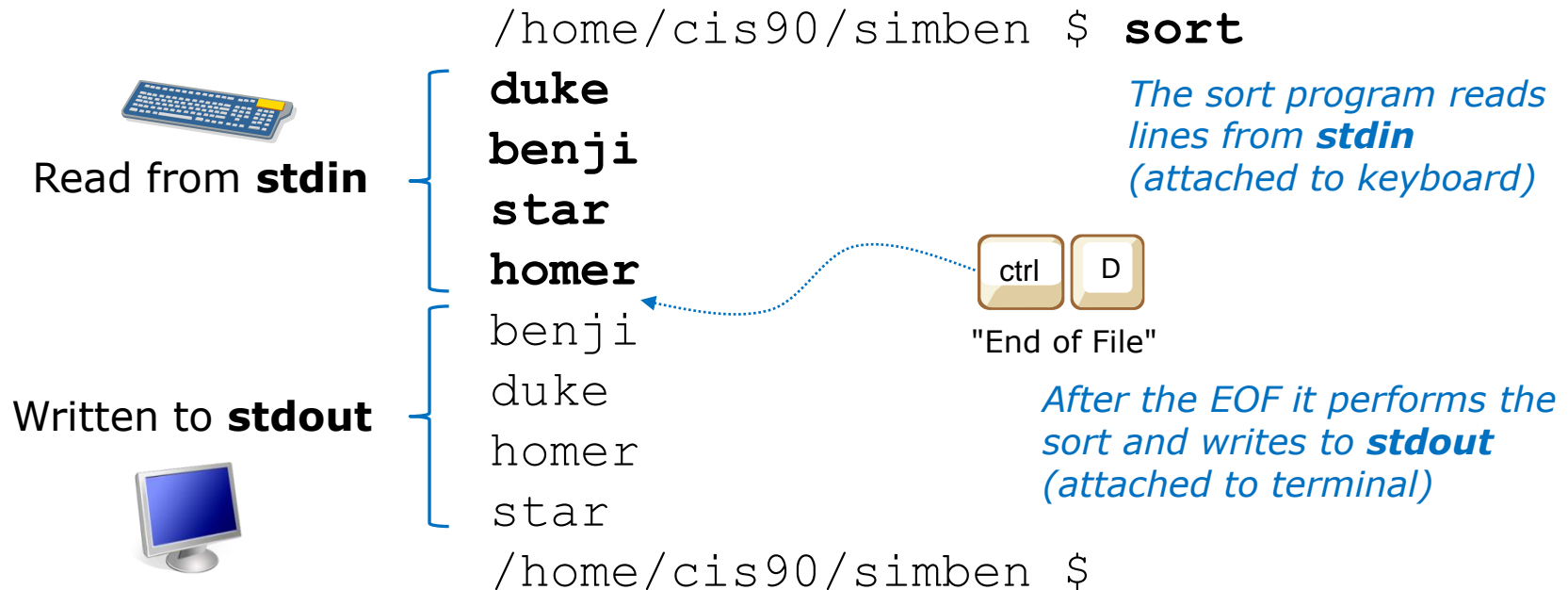
```
/home/cis90/simben $ sort  
duke  
benji  
star  
homer  
benji  
duke  
homer  
star  
/home/cis90/simben $
```



The diagram illustrates the effect of pressing Ctrl+D (End of File) during the execution of the `sort` command. A dotted blue line originates from a box containing the text "ctrl" and "D", with the label "End of File" below it. This line points to the word "benji" in the output, which is the first line of the sorted list. The output shows the names in sorted order: benji, duke, homer, star. The prompt indicates the command was run in the directory /home/cis90/simben.

Input and Output

File Redirection



sort command (no arguments)

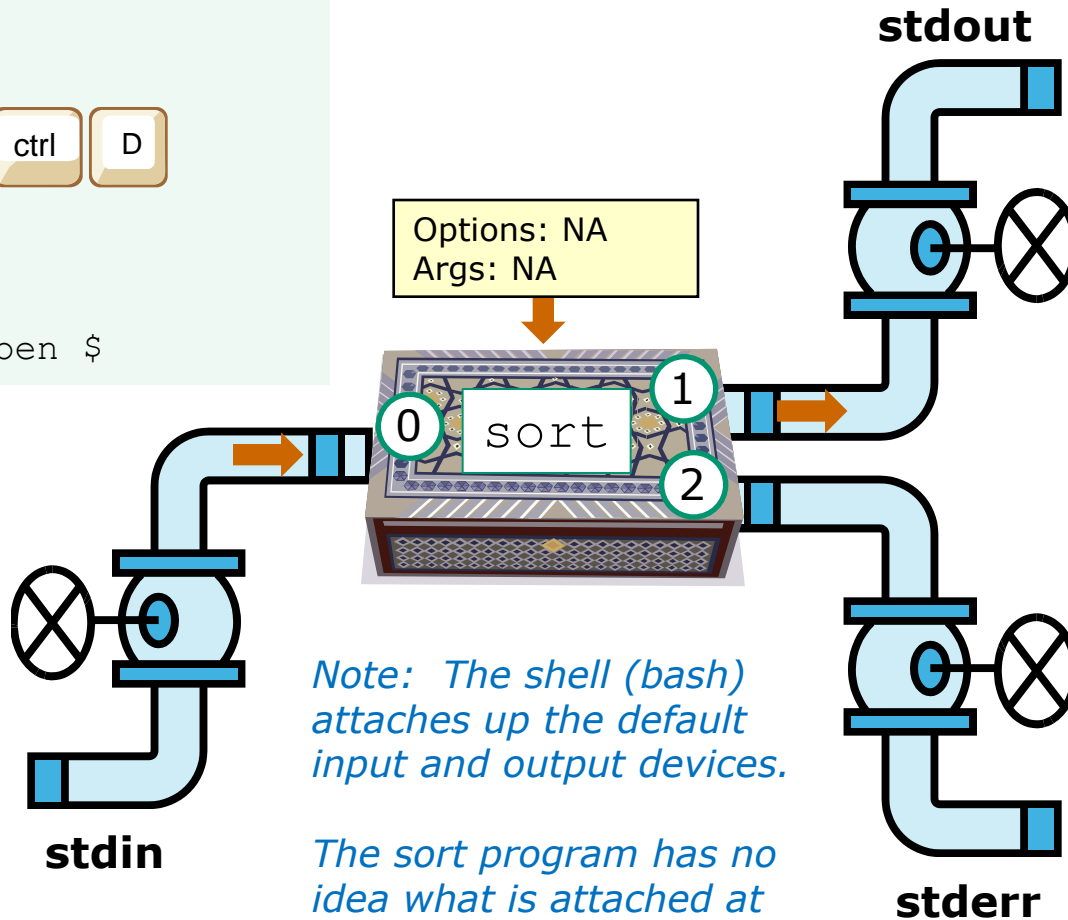
```
/home/cis90/simben $ sort
duke
benji
star
homer
benji
duke
homer
star
/home/cis90/simben $
```



/dev/pts/0



duke
benji
star
homer



Note: The shell (bash) attaches up the default input and output devices.

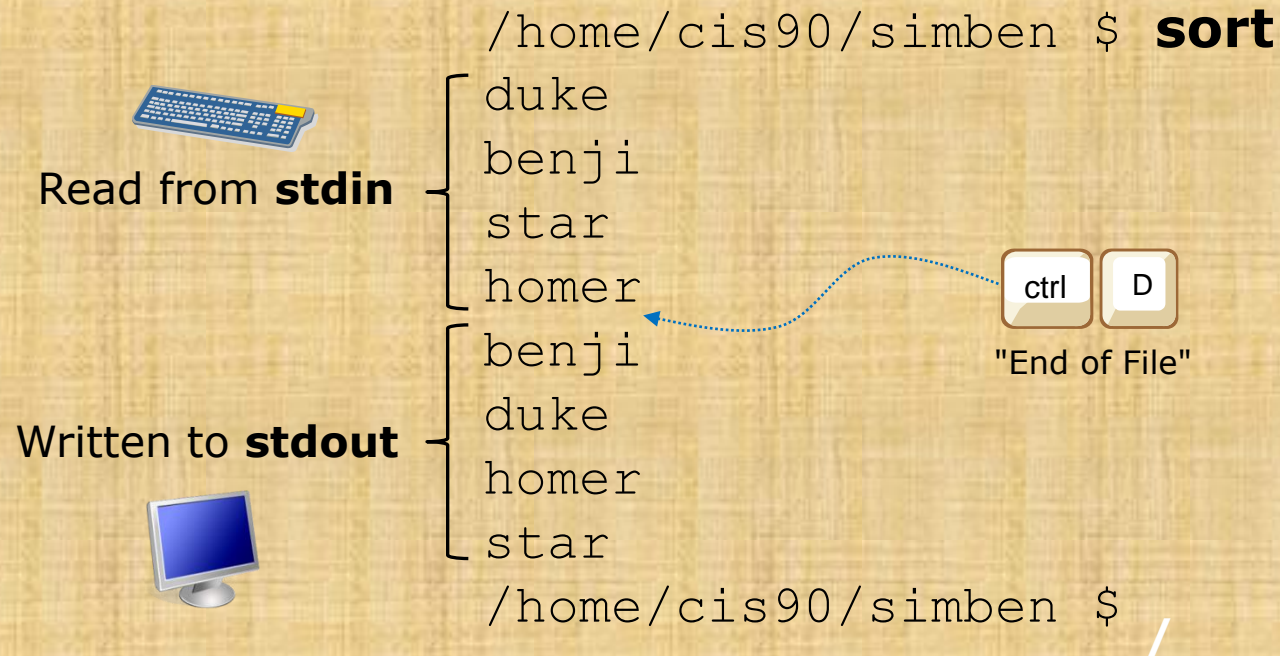
The sort program has no idea what is attached at the end of the pipes.

/dev/pts/0



benji
duke
homer
star

Activity

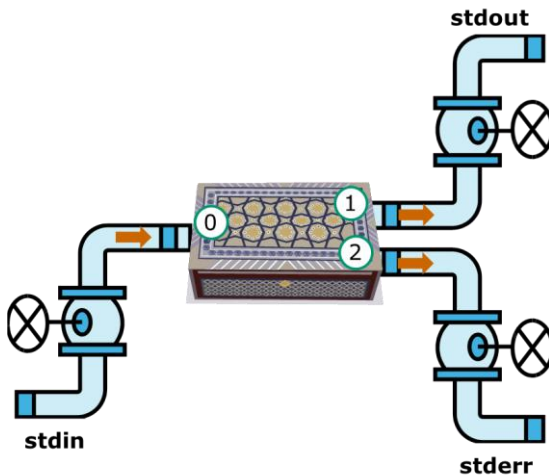


Now you try it

Input and Output

File Redirection

The input and output of a program can be **redirected** from and to other files using **<**, **>**, **2>** and **>>**:



~~0~~< **filename**

*To redirect **stdin** (either 0< or just <)*

~~1~~> **filename**

*To redirect **stdout** (either 1> or just >)*

2> filename

*To redirect **stderr***

>> filename

*To redirect **stdout** and append*

No arguments, redirecting stdout

sort just reads from **stdin**
and writes to **stdout**

stdout has been
redirected to the file
dogsinorder

```
[simmsben@opus ~]$ sort > dogsinorder
duke
benji
star
homer
```

If the file *dogsinorder* does not exist, it is
created. If it does exist it is emptied!



```
[simmsben@opus ~]$ cat dogsinorder
benji
duke
homer
star
[simmsben@opus ~]$
```

No arguments, redirecting stdout

```
$ sort > dogsinorder
```

```
duke  
benji  
star  
homer  
$
```



Options: NA
Args: NA



stdout



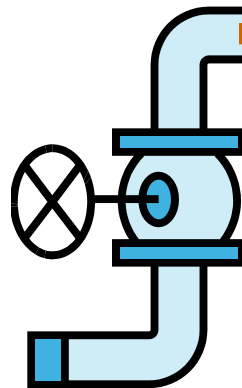
dogsinorder

```
$ cat dogsinorder  
benji  
duke  
homer  
star
```

/dev/pts/0



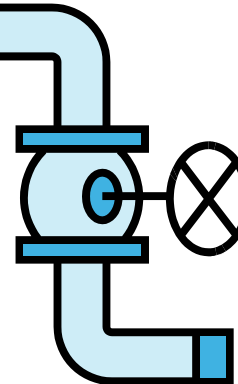
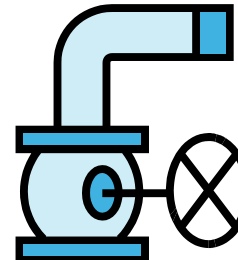
```
duke  
benji  
star  
homer
```



stdin

Note: *sort* doesn't know that input comes from the keyboard or that output will be sent to the *dogsinorder* file.

It just reads from **stdin** and writes to **stdout**.



stderr

Now you try it

sort just reads from **stdin**
and writes to **stdout**

stdout has been
redirected to the file
dogsinorder

```
[simmsben@opus ~]$ sort > dogsinorder
```

duke

benji

star

homer

If the file *dogsinorder* does not exist, it is
created. If it does exist it is emptied!



```
[simmsben@opus ~]$ cat dogsinorder
```

benji

duke

homer

star

```
[simmsben@opus ~]$
```

No arguments, redirecting stdin and stdout

```
[simben@opus ~]$ cat names
```

```
duke
```

```
benji
```

```
star
```

```
homer
```

input is redirected to come
from the file *names*

output is redirected to the
file *dogsinorder*

```
[simben@opus ~]$ sort < names > dogsinorder
```

```
[simben@opus ~]$ cat dogsinorder
```

```
benji
```

```
duke
```

```
homer
```

```
star
```

```
[simben@opus ~]$
```

Note: The bash shell handles the
command line parsing and redirection.
The sort command has no idea what
stdin or ***stdout*** are attached to.



No arguments, redirecting stdin and stdout

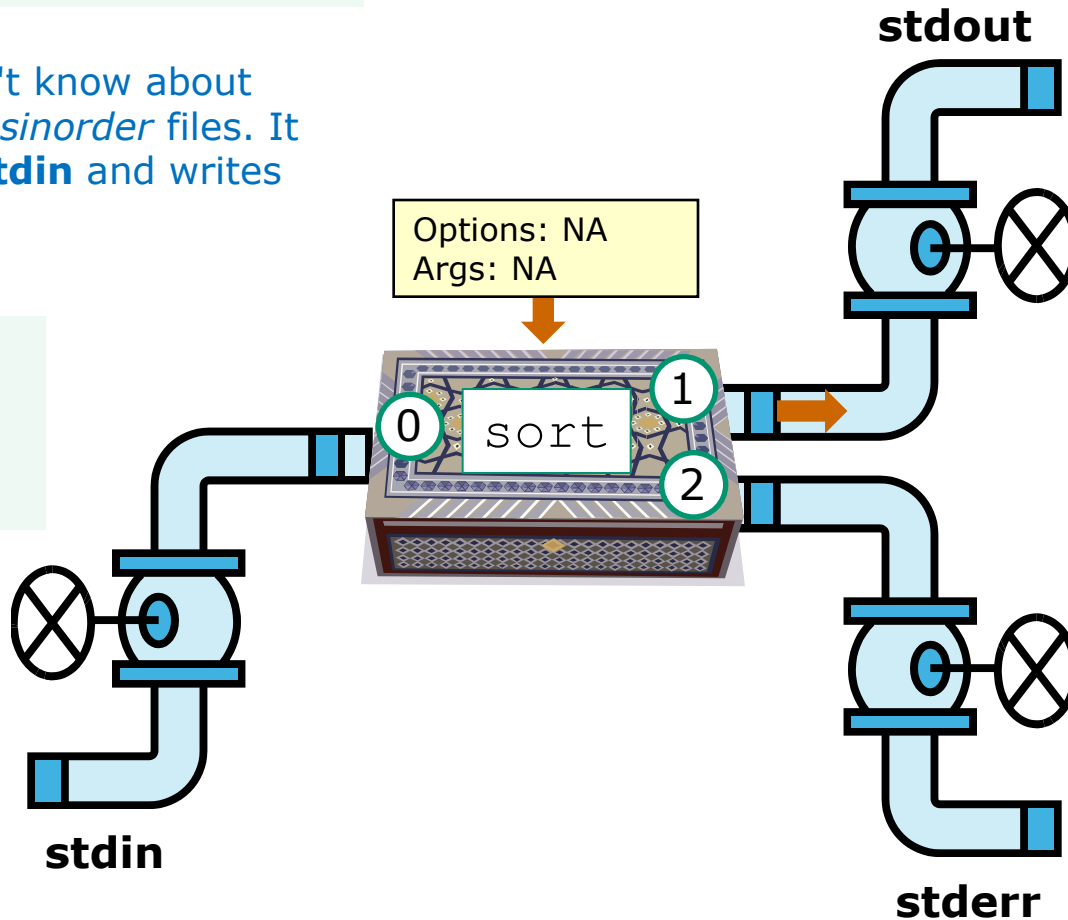
```
$ sort < names > dogsinorder
```

Note: `sort` doesn't know about the *names* or *dogsinorder* files. It just reads from **stdin** and writes to **stdout**.

```
$ cat names
duke
benji
star
homer
```



names



```
$ cat dogsinorder
benji
duke
homer
star
```

In this example, `sort` is getting its input from **stdin**, which has been redirected to the *names* file

Now you try it

```
[simben@opus ~]$ cat names
```

```
duke
```

```
benji
```

```
star
```

```
homer
```

input is redirected to come
from the file *names*

output is redirected to the
file *dogsorder*

```
[simben@opus ~]$ sort < names > dogsorder
```

```
[simben@opus ~]$ cat dogsorder
```

```
benji
```

```
duke
```

```
homer
```

```
star
```

```
[simben@opus ~]$
```

*Does the sort program know that its
input came from the names file?*

Put your answer in the chat window

One argument, redirecting stdout

The *names* file is parsed as an **argument** and is passed to the sort process to handle.

Output written to **stdout** is redirected to the file *dogsinorder*.

The shell, not the sort program, opens the *dogsinorder* file.

```
[simben@opus ~]$ sort names > dogsinorder
[simben@opus ~]$ cat dogsinorder
benji
duke
homer
star
[simben@opus ~]$
```

The sort program, not the shell, opens and reads directly from the *names* file.

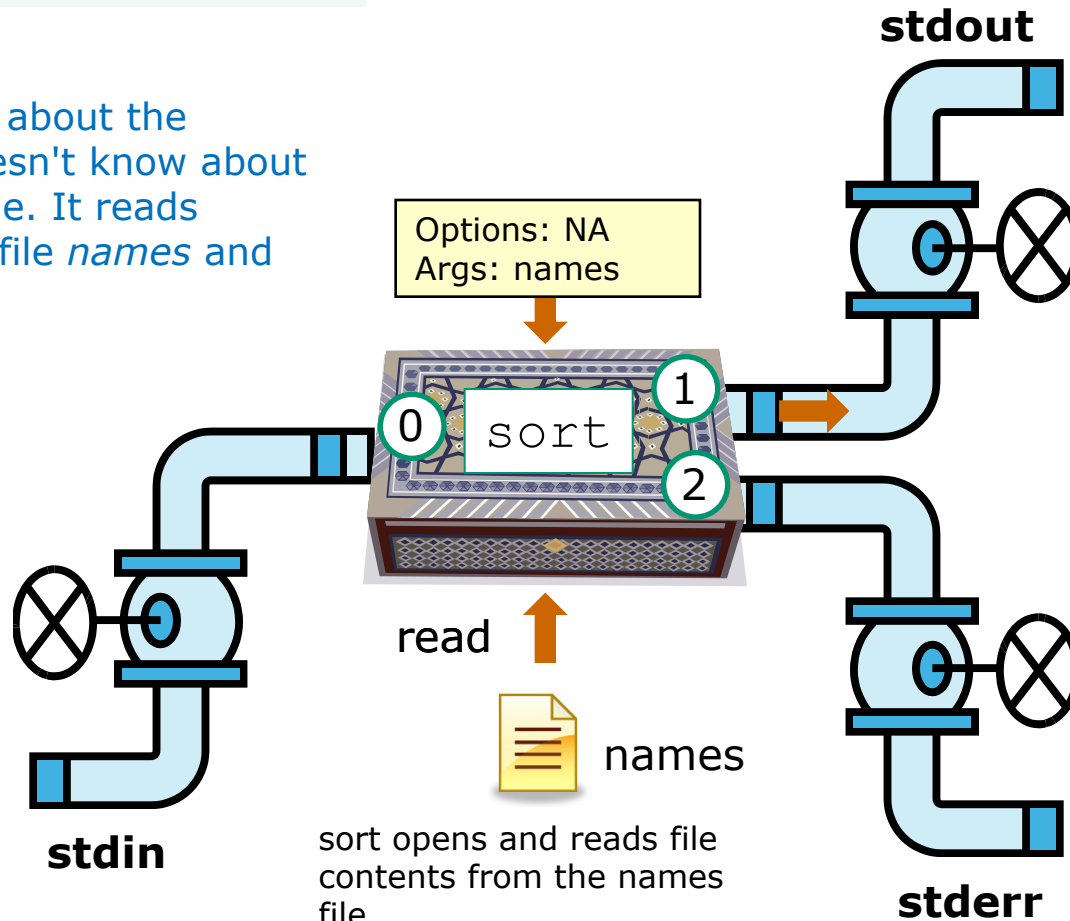
Корисне для
наступного
вікторини!

One argument, redirecting stdout

```
$ sort names > dogsinorder
```

Note: *sort* knows about the *names* file but doesn't know about the *dogsinorder* file. It reads directly from the file *names* and writes to **stdout**.

Корисне для
наступного
вікторини!



```
$ cat dogsinorder
benji
duke
homer
star
```

In this example, *sort* is getting its input directly from the *names* file

Now you try it

The *names* file is parsed as an **argument** and is passed to the sort process to handle.

Output written to **stdout** is redirected to the file *dogsinorder*.

The shell, not the sort program, opens the *dogsinorder* file.

```
[simben@opus ~]$ sort names > dogsinorder
[simben@opus ~]$ cat dogsinorder
benji
duke
homer
star
[simben@opus ~]$
```

Does the sort program know that its input came from the *names* file?

Put your answer in the chat window

Корисне для
наступного

One option, one argument, redirecting stdout

specifying an option
(for reverse order)

names is parsed as an
argument and passed to the
sort command

sort writes to **stdout**, which is
redirected to the file *dogsিনorder*

```
[simben@opus ~]$ sort -r names > dogsিনorder
```

```
[simben@opus ~]$ cat dogsিনorder
```

```
star
```

```
homer
```

```
duke
```

```
benji
```

```
[simben@opus ~]$
```

This -r option does the sort in
reverse order

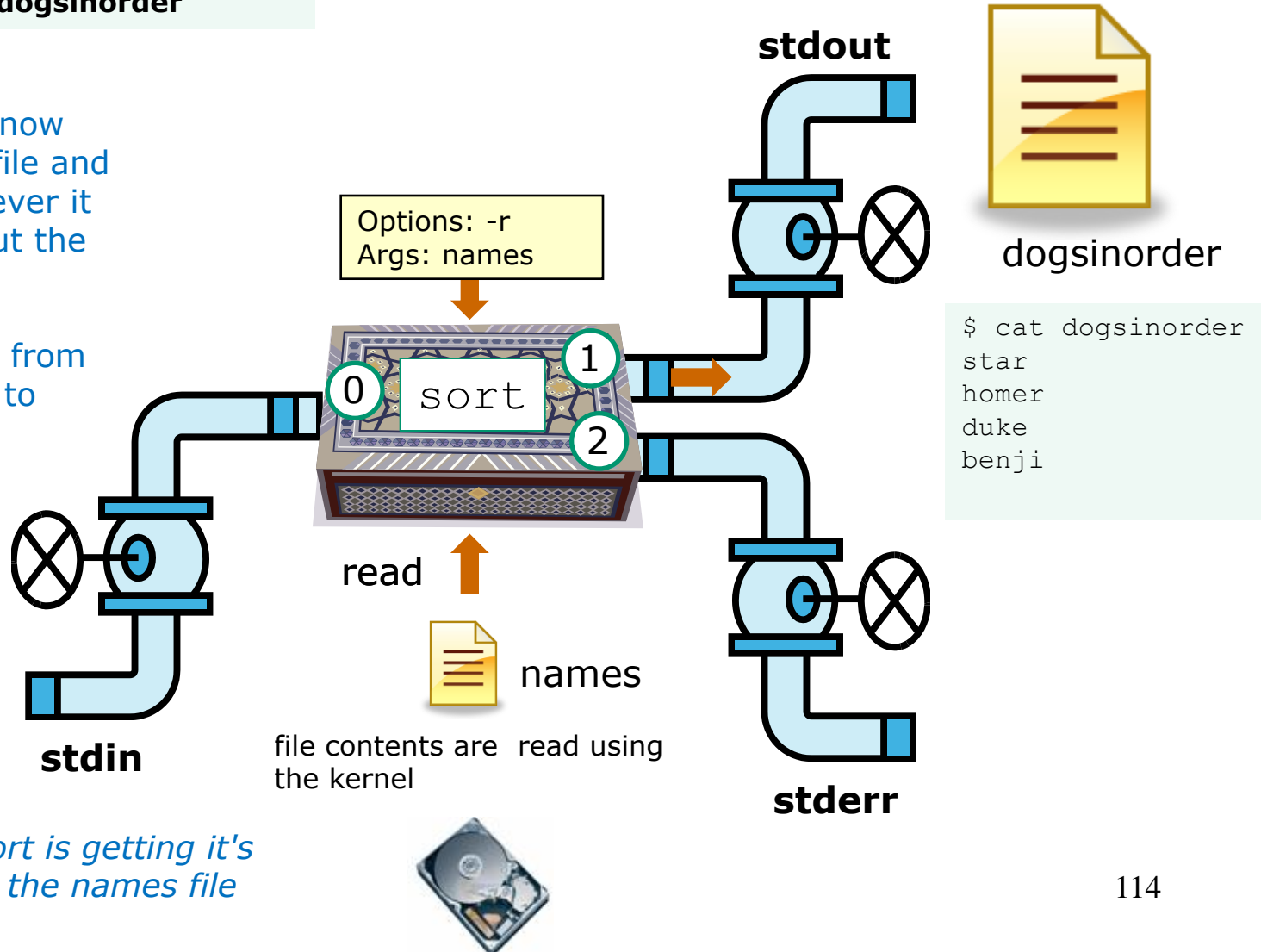
The shell opens the *dogsিনorder*
file. The sort process is not aware
that output is redirected there.

One option, one argument, redirecting stdout

```
$ sort -r names > dogsinorder
```

Note: `sort` does know about the *names* file and the `-r` option however it doesn't know about the *dogsinorder* file.

`sort` reads directly from *names* and writes to **stdout**.



*In this example, `sort` is getting its input directly from the *names* file*

Now you try it

specifying an option
(for reverse order)

names is parsed as an
argument and passed to the
sort command

sort writes to **stdout**,
which is redirected to
the file *dogsinorder*

```
/home/cis90/simben $ sort -r names > dogsinorder
/home/cis90/simben $ cat dogsinorder
star
homer
duke
benji
/home/cis90/simben $
```

Корисне для
наступного
вікторини!

*Does the sort program know that its
output is going to the dogsinorder file?*

Put your answer in the chat window

© 2008

Append vs Overwrite

> (overwrites) vs >> (appends)

```
[simben@opus ~]$ echo "Hello World" > message
```

```
[simben@opus ~]$ cat message
```

```
Hello World
```

```
[simben@opus ~]$ echo "Hello Universe" >> message
```

```
[simben@opus ~]$ cat message
```

```
Hello World
```

```
Hello Universe
```

*>> does not empty
file, just appends to
the end*

```
[simben@opus ~]$ echo "Oops" > message
```

```
[simben@opus ~]$ cat message
```

```
Oops
```

*> empties then
overwrites anything
already in the file!*

```
[simben@opus ~]$ > message
```

```
[simben@opus ~]$ cat message
```

```
[simben@opus ~]$
```


2> (overwrites) vs 2>> (appends)

```
/home/cis90/simben $ ls bogus 2> errors
/home/cis90/simben $ cat errors
ls: cannot access bogus: No such file or directory
/home/cis90/simben $ ls crud 2> errors
/home/cis90/simben $ cat errors
ls: cannot access crud: No such file or directory
```

2> causes the file errors to be emptied and overwritten with error output

```
/home/cis90/simben $ ls bogus 2> errors
/home/cis90/simben $ ls crud 2>> errors
/home/cis90/simben $ cat errors
ls: cannot access bogus: No such file or directory
ls: cannot access crud: No such file or directory
/home/cis90/simben $
```

2>> appends error output to the errors file

More redirection examples

Example 1 session

Input from stdin, redirecting stdout to another terminal device

/dev/pts/0

```
[simben@opus ~]$ cat names
duke
benji
star
homer
[simben@opus ~]$
[simben@opus ~]$ tty
/dev/pts/0
[simben@opus ~]$ sort names > /dev/pts/1
[simben@opus ~]$
```

*Note, everything in
UNIX is a file so we
can even redirect to
another terminal*

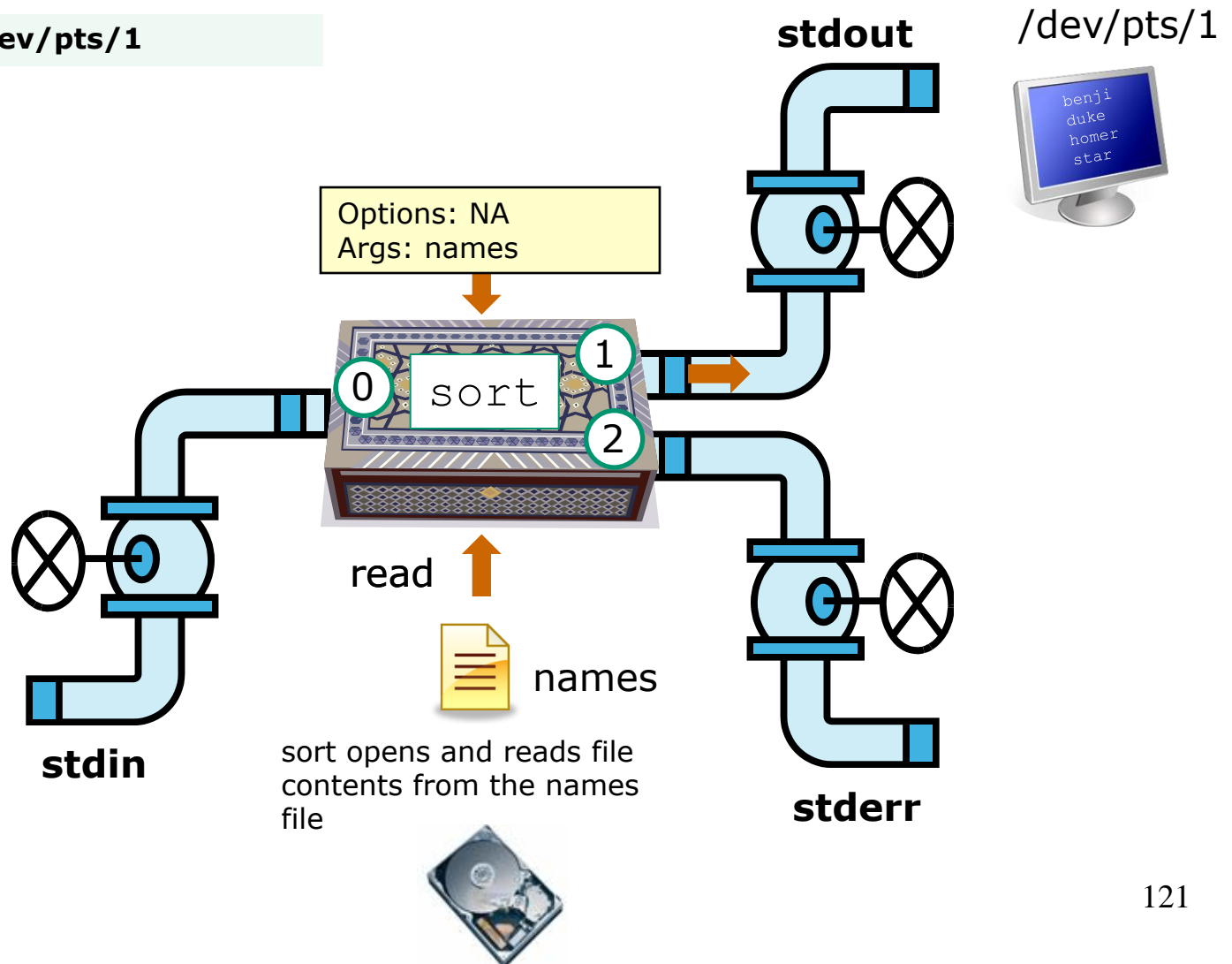
/dev/pts/1

```
[simben@opus ~]$ tty
/dev/pts/1
[simben@opus ~]$ benji
duke
homer
star
```

Example 1 diagram

Input from stdin, redirecting stdout to another terminal device

```
$ sort names > /dev/pts/1
```



Example 2 session

Input from the command line, redirecting stdout to file

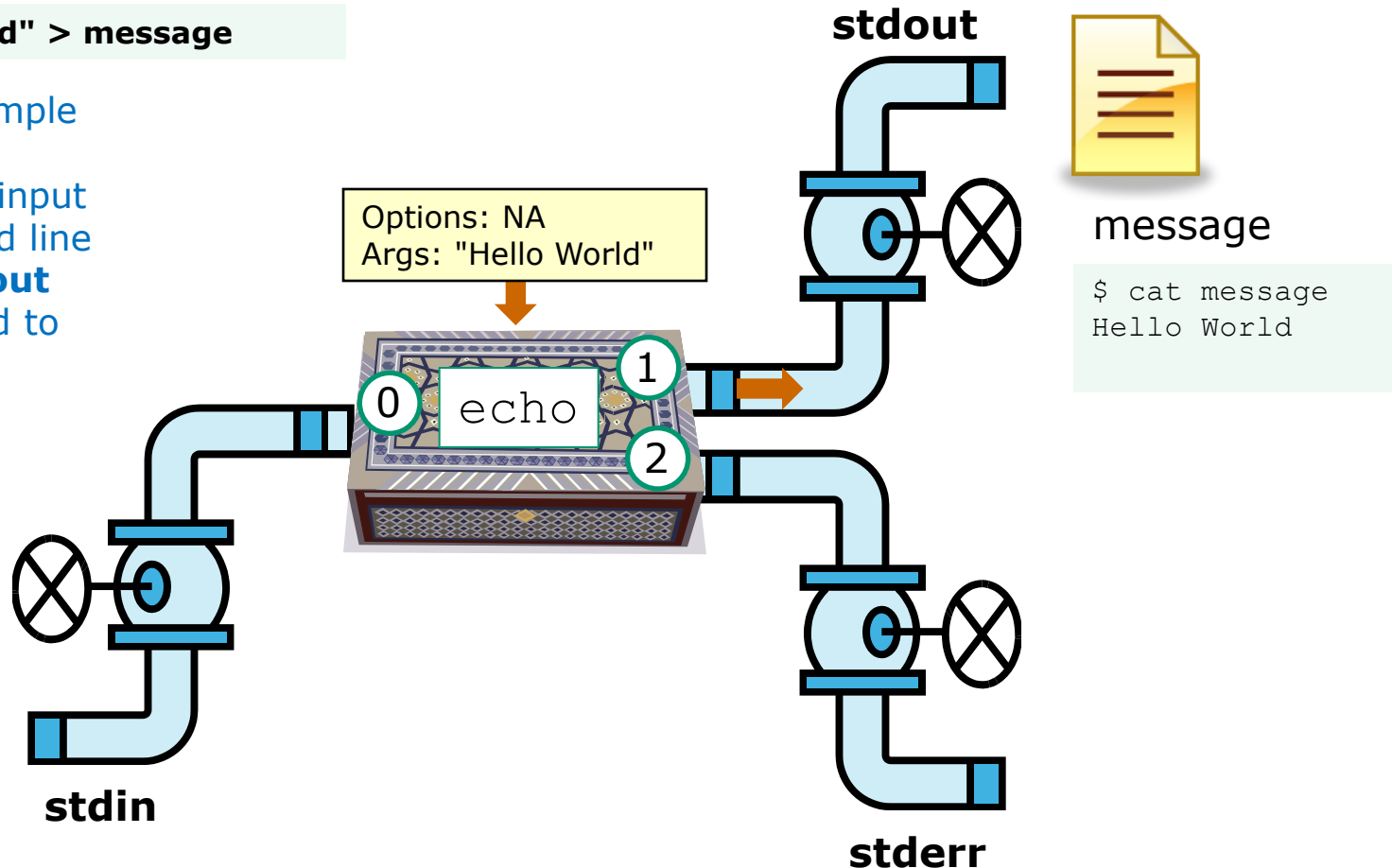
```
/home/cis90/simben $ echo "Hello World" > message  
/home/cis90/simben $ cat message  
Hello World  
/home/cis90/simben $
```

Example 2 diagram

Input from the command line, redirecting stdout to file

```
$ echo "Hello World" > message
```

Note: In this example echo does not use **stdin**. It gets its input from the command line and writes to **stdout** which is redirected to the file *message*.



Example 3 session

Input from command line and OS, redirecting stdout and stderr

```
[simben@opus ~]$ ls -lR > snapshot
ls: ./Hidden: Permission denied
[simben@opus ~]$ head -10 snapshot
.:
total 296
-rw-rw-r--  1 simben cis90      51 Sep 24 17:13 1993
-rw-r--r-- 21 guest90  cis90  10576 Jul 20  2001 bigfile
drwxr-x---  2 simben cis90   4096 Oct  8 09:05 bin
drwx--x---  4 simben cis90   4096 Oct  8 09:00 class
-rw-----  1 simben cis90    484 Sep 24 18:13 dead.letter
drwxrwxr-x  2 simben cis90   4096 Oct  8 09:05 docs
-rw-rw-r--  1 simben cis90     22 Oct 20 10:51 dogsinorder
drwx-----  2 simben cis90   4096 Oct 16 09:17 edits
[simben@opus ~]$
```

*Note: errors are written to **stderr**, which is attached by default to the terminal*

```
[simben@opus ~]$ ls -lR > snapshot 2> errors
[simben@opus ~]$ cat errors
ls: ./Hidden: Permission denied
[simben@opus ~]$
```

*> redirects
stdout to file
named snapshot*

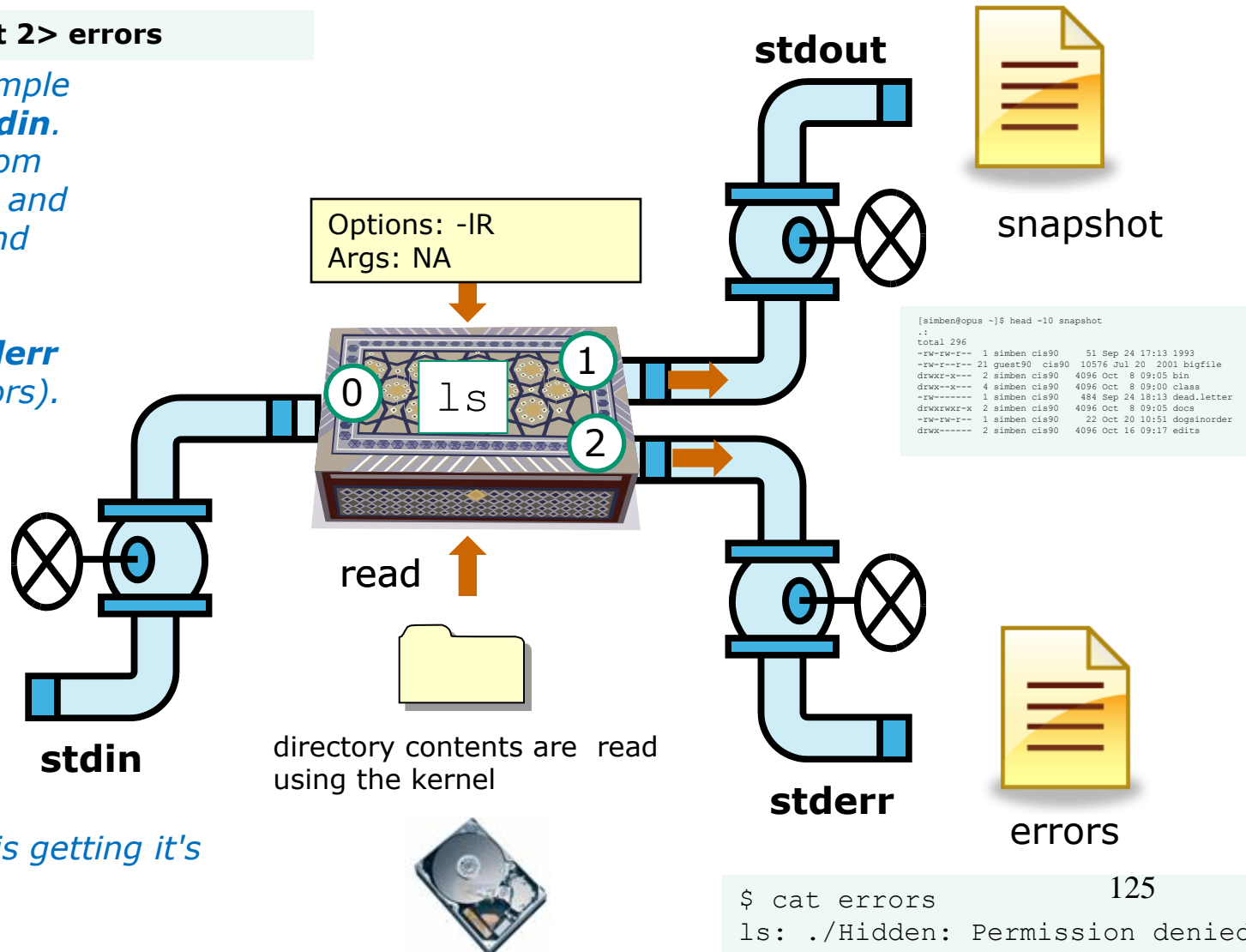
*2> redirects **stderr**
to file named errors*

Example 3 diagram

Input from command line and OS, redirecting stdout and stderr

```
$ ls -lR > snapshot 2> errors
```

*Note: In this example ls does not use **stdin**. It gets its input from the command line and the OS (kernel) and writes to **stdout** (redirected to snapshot) and **stderr** (redirected to errors).*



In this example, ls is getting its input from the OS

Example 4 session

Redirecting stdin, stdout and stderr

```
[simben@opus ~]$ echo 2+2 > math
```

```
[simben@opus ~]$ bc < math
```

```
4
```

bc reads input from **stdin** (redirected to *math*) and writes to **stdout** (attached to the terminal)

```
[simben@opus ~]$ echo 4/0 >> math
```

```
[simben@opus ~]$ cat math
```

```
2+2
```

```
4/0
```

```
[simben@opus ~]$ bc < math
```

```
4
```

bc reads inputs from **stdin** (redirected to *math*), writes to **stdout** (attached to the terminal) and writes errors to **stderr** (attached to the terminal)

```
Runtime error (func=(main), adr=5): Divide by zero
```

```
[simben@opus ~]$ bc < math > answers 2> errors
```

```
[simben@opus ~]$ cat answers
```

```
4
```

bc reads inputs from **stdin** (redirected to *math*), writes to **stdout** (redirected to *answers*) and writes errors to **stderr** (redirected to *errors*)

```
[simben@opus ~]$ cat errors
```

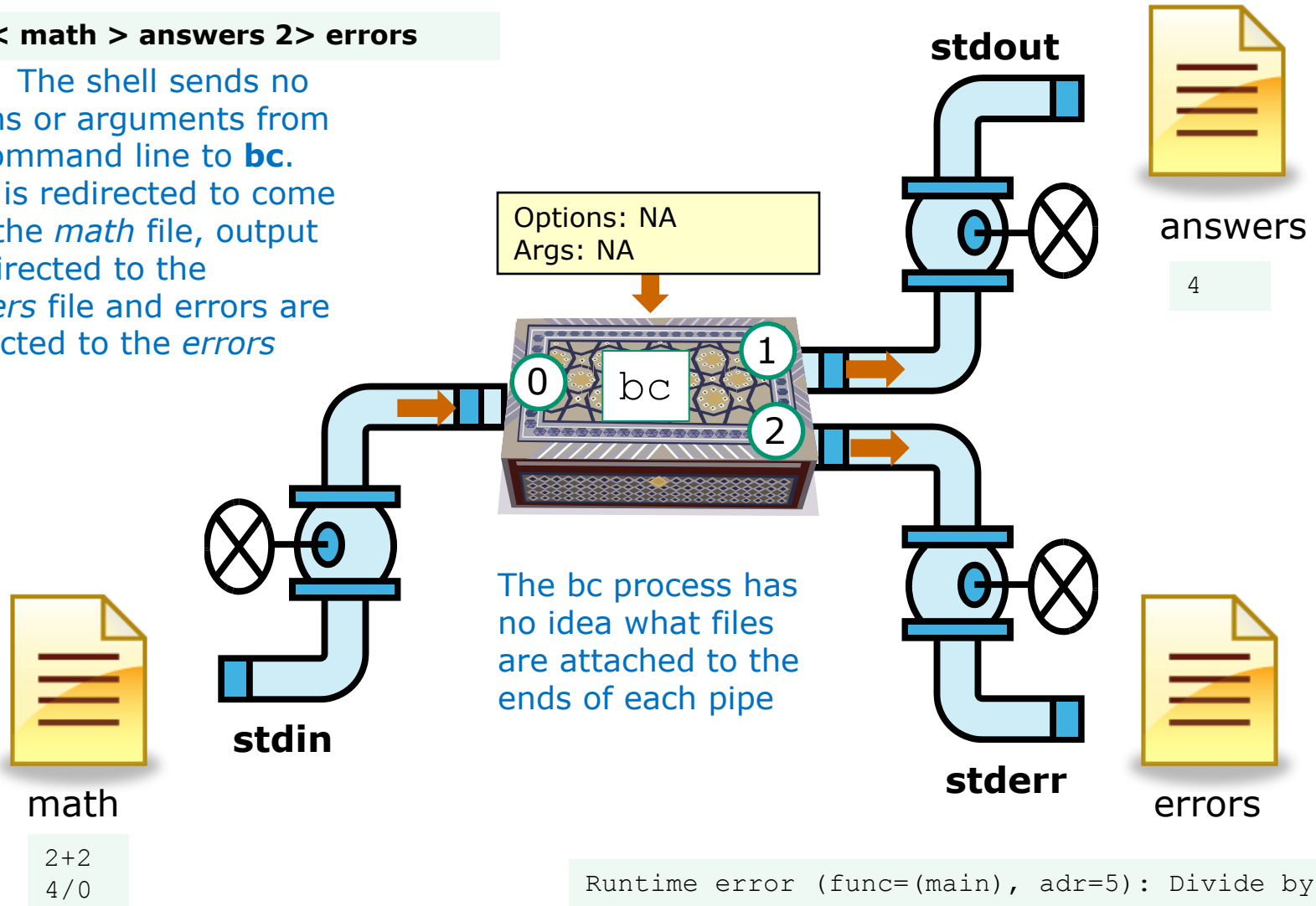
```
Runtime error (func=(main), adr=5): Divide by zero
```


Example 4 diagram

Redirecting stdin, stdout and stderr

```
$ bc < math > answers 2> errors
```

Note: The shell sends no options or arguments from the command line to **bc**. Input is redirected to come from the *math* file, output is redirected to the *answers* file and errors are redirected to the *errors* file.

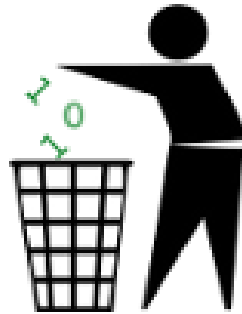


The bit bucket `/dev/null`

`/dev/null` = “bit bucket”

A bit bucket is very handy. You can throw stuff into it and never see it again!

<http://www.adrianmouat.com/bit-bucket/>



<http://didyouknowarchive.com/?p=1755>

It's like having your own black hole to discard those unwanted bits into!

/dev/null = "bit bucket"

*Whatever you redirect to /dev/null/
is gone forever*

```
/home/cis90/simben $ echo Clean up your room! > orders  
/home/cis90/simben $ cat orders  
Clean up your room!  
/home/cis90/simben $
```

```
/home/cis90/simben $ echo Clean up your room! > /dev/null  
/home/cis90/simben $ cat /dev/null  
/home/cis90/simben $
```

Корисне для
наступного
вікторини!

This is how you redirect output to the bit bucket

Pipelines

Input and Output

Pipelines

Commands may be chained together in such a way that the **stdout** of one command is "piped" into the **stdin** of a second process.

Filters

A program that both reads from **stdin** and writes to **stdout**.

Tees

A filter program that reads **stdin** and writes it to **stdout and the file** specified as the argument.

Input and Output

Pipelines

Note:

Use **redirection** operators (<, >, >>, 2>) to redirect input and output from and to **files**

Use the **pipe** operator (|) to pipe output from one **command** for use as input to another **command**

Pipeline Example

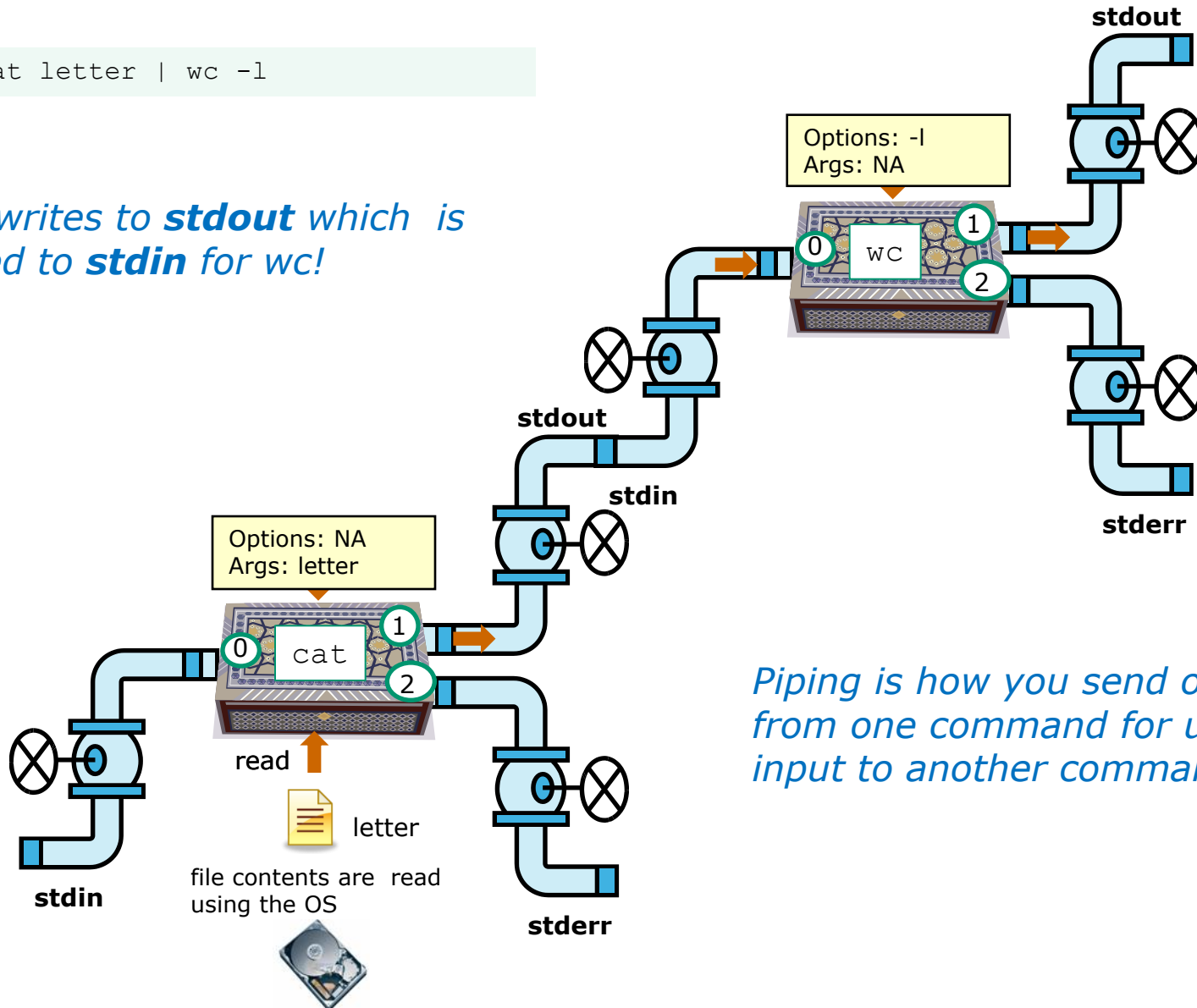
```
[simben@opus ~]$ cat letter | wc -l  
28
```

Counting the lines in the letter file

Counting lines in the letter file

```
$ cat letter | wc -l
```

*cat writes to **stdout** which is piped to **stdin** for wc!*



Piping is how you send output from one command for use as input to another command

You try it

Counting the lines in the letter file

```
/home/cis90/simben $ cat letter | wc -l  
28
```

Counting the number of Shakespeare poems

```
/home/cis90/simben $ ls poems/Shakespeare/ | wc -l  
15
```

More Commands



Tools for your toolbox

NEW

find - Find file or content of a file

NEW

grep - "Global Regular Expression Print"

sort - sort

NEW

spell - spelling correction

wc - word count

tee - split output

NEW

cut - cut fields from a line

find command

Find Command

Basic syntax

(see man page for the rest of the story)

```
find <start-directory> -name <filename>  
                        -type <filetype>  
                        -user <username>  
                        -group <groupname>  
                        -exec <command> {} \;
```

Use the **find** command to find files by their name, type, owner, group (or other attributes) and optionally run a command on each of the files found.

The find command is **recursive** by default. It will start finding files at the <start directory> and includes all files and sub-directories in that branch of the file tree.

find command with no options or arguments

*The **find** command by itself lists all files in the current directory and recursively down into any sub-directories.*

```
[simben@opus poems]$ find
```

```
.
./Blake
./Blake/tiger
./Blake/jerusalem
./Shakespeare
./Shakespeare/sonnet1
./Shakespeare/sonnet2
./Shakespeare/sonnet3
./Shakespeare/sonnet4
./Shakespeare/sonnet5
./Shakespeare/sonnet7
./Shakespeare/sonnet9
./Shakespeare/sonnet10
./Shakespeare/sonnet15
./Shakespeare/sonnet17
./Shakespeare/sonnet26
./Shakespeare/sonnet35
./Shakespeare/sonnet11
./Shakespeare/sonnet6
./Yeats
./Yeats/whitebirds
./Yeats/mooncat
./Yeats/old
./Anon
./Anon/ant
./Anon/nursery
./Anon/twister
```

Because no start directory was specified the find command will start listing files in the current directory (poems)

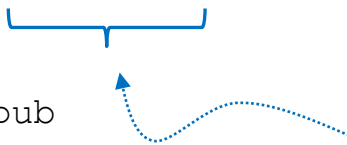
*note: reduced font size
so it will fit on this slide*

```
[simben@opus poems]$
```

find command - the starting directory

One or more starting directories in the file tree can be specified as an argument to the find command which will list recursively all files and sub-folders from that directory and down

```
/home/cis90/simben $ find /etc/ssh
/etc/ssh
/etc/ssh/ssh_config
/etc/ssh/ssh_host_dsa_key.pub
/etc/ssh/moduli
/etc/ssh/ssh_host_key
/etc/ssh/ssh_host_dsa_key
/etc/ssh/ssh_host_rsa_key.pub
/etc/ssh/ssh_host_rsa_key
/etc/ssh/ssh_host_key.pub
/etc/ssh/sshd_config
/home/cis90/simben $
```



*this find command will
start listing files from the
/etc/ssh directory*

The find command -name option

Since no starting directory was specified find will start in the current directory (simben90's home directory).

Directs the find command to only look for files whose names start with "sonnet"

```
/home/cis90/simben $ find -name 'sonnet*'
find: './Hidden': Permission denied
./poems/Shakespeare/sonnet10
./poems/Shakespeare/sonnet15
./poems/Shakespeare/sonnet26
./poems/Shakespeare/sonnet3
./poems/Shakespeare/sonnet35
./poems/Shakespeare/sonnet6
./poems/Shakespeare/sonnet2
./poems/Shakespeare/sonnet4
./poems/Shakespeare/sonnet1
./poems/Shakespeare/sonnet11
./poems/Shakespeare/sonnet7
./poems/Shakespeare/sonnet5
./poems/Shakespeare/sonnet9
./poems/Shakespeare/sonnet17
/home/cis90/simben $
```

All those permission errors

An error is printed for every directory lacking read permission!

Where to start finding files

*only include files
named sonnet6*

```
[simben@opus ~]$ find /home/cis90 -name sonnet6
```

```
find: /home/cis90/guest/.ssh: Permission denied
find: /home/cis90/guest/Hidden: Permission denied
/home/cis90/guest/Poems/Shakespeare/sonnet6
find: /home/cis90/guest/.gnupg: Permission denied
find: /home/cis90/guest/.gnome2: Permission denied
find: /home/cis90/guest/.gnome2_private: Permission denied
find: /home/cis90/guest/.gconf: Permission denied
find: /home/cis90/guest/.gconfd: Permission denied
find: /home/cis90/simben/Hidden: Permission denied
```

*Yuck! How
annoying is this?*

<snipped>

```
find: /home/cis90/wichemic/class: Permission denied
find: /home/cis90/crivejoh/Hidden: Permission denied
/home/cis90/crivejoh/poems/Shakespeare/sonnet6
[simben@opus ~]$
```



Redirecting find errors to the bit bucket

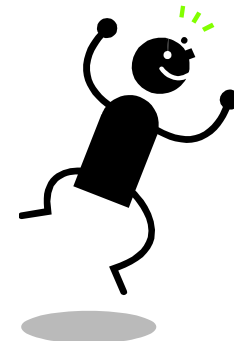
*redirecting stderr
to the "bit bucket"*

```
[simben@opus ~]$ find /home/cis90 -name sonnet6 2> /dev/null
```

```
/home/cis90/guest/Poems/Shakespeare/sonnet6  
/home/cis90/simben/poems/Shakespeare/sonnet6  
/home/cis90/stanlcha/poems/Shakespeare/sonnet6  
/home/cis90/seatocol/poems/Shakespeare/sonnet6  
/home/cis90/wrigholi/poems/Shakespeare/sonnet6  
/home/cis90/dymesdia/poems/Shakespeare/sonnet6  
/home/cis90/lyonsrob/poems/Shakespeare/sonnet6  
/home/cis90/ybarrser/poems/Shakespeare/sonnet6  
/home/cis90/ybarrser/poems/Sonnets/sonnet6  
/home/cis90/valdemar/poems/Shakespeare/sonnet6  
/home/cis90/elliokat/poems/Shakespeare/sonnet6  
/home/cis90/jessuwes/poems/Shakespeare/sonnet6  
/home/cis90/luisjus/poems/Shakespeare/sonnet6  
/home/cis90/meyerjas/poems/Shakespeare/sonnet6  
/home/cis90/bergelyl/sonnet6  
/home/cis90/bergelyl/poems/Shakespeare/sonnet6  
/home/cis90/gardnnic/poems/Shakespeare/sonnet6  
/home/cis90/mohanchi/poems/Shakespeare/sonnet6  
/home/cis90/whitfbob/poems/Shakespeare/sonnet6  
/home/cis90/crivejoh/poems/Shakespeare/sonnet6  
[simben@opus ~]$
```

Ahhh ... much better!

*All the annoying error
messages are redirected
to the bit bucket*



*This is why we want a
bit bucket*

find command examples

*start finding in /
(the top of the file tree)*

```
[simben@opus ~]$ find / 2> /dev/null | wc -l
```

154033

*redirect permission
errors into the bit
bucket (discard them)*

*pipe the output of the **find**
command as input to the **wc**
command*

***wc** counts the number of
lines read from stdin*

Корисне для
наступного
вікторини!

*Getting an approximate count of all the files on
Opus and suppressing any permission errors*

find command examples

The directory to start finding files

Redirect errors written to stderr to the bit bucket

```
/home/cis90/simben $ find /home -user root 2> /dev/null
```

The user that owns the files

```
/home  
/home/cis175  
/home/cis172  
/home/cis172/computers.txt  
/home/cis172/science.txt  
/home/lost+found  
/home/cis90/simben $
```

Find all files in the /home directory that belong to the root user and discard any error messages

find command examples

*The directory to
start finding files*

*Redirect errors to
the bit bucket*

```
/home/cis90/simben $ find /home -type d -user milhom90 2> /dev/null
/home/turnin/cis90/milhom90
/home/cis90/milhom
/home/cis90/milhom/Hidden
/home/cis90/milhom/Lab2.0
/home/cis90/milhom/Miscellaneous
/home/cis90/milhom/bin
/home/cis90/milhom/Poems
/home/cis90/milhom/Poems/Shakespeare
/home/cis90/milhom/Poems/Yeats
/home/cis90/milhom/Poems/Blake
/home/cis90/milhom/Lab2.1
/home/cis90/milhom/Lab2.1/filename
/home/cis90/milhom/cis90_html
/home/cis90/milhom/cis90_html/images
/home/cis90/milhom/cis90_html/css
/home/cis90/milhom/.ssh
/home/cis90/simben $
```

*Only find type
d files
(directories)*

*Only those that
belong to
milhom90*

Find all directories starting in /home that belong to milhom90 and suppress permission errors

find command examples

start from "here" → *specifies directories only* *specifies only files whose names start with a B, S, Y or A*

```
[simben@opus ~]$ find . -type d -name '[BSYA]*'
find: ./Hidden: Permission denied
./poems/Blake
./poems/Shakespeare
./poems/Yeats
./poems/Anon
[simben@opus ~]$
```

Find all directories, starting from the current directory that start with a capital B, S, Y or A.

find command examples

No start directory specified so start in current directory

file type "f" (regular)

file names contain the letter "k"

The command to run on each file found

```
/home/cis90/simben $ find -type f -name '*k*' -exec file {} \;
```

The {} are replaced by filenames as they are found

Escape the ; so it will be passed to the find command

```
find: `./Hidden': Permission denied
./editors/spellk: ASCII English text
./kshrc: ASCII text
./docs/MarkTwain: ASCII English text
./ssh/known_hosts: ASCII text, with very long lines
/home/cis90/simben $
```

Run the file command on all regular files found starting in the current directory whose names contain the letter "k"

Now you try it

start from "here"

specifies only files whose names contain "town"

```
[simben@opus ~]$ find . -name '*town*'
find: ./Hidden: Permission denied
./edits/small_town
./edits/better_town
[simben@opus ~]$
```

Find all files starting from your current location whose names contain "town"

More filter commands

A command is called a "**filter**" if it can read from *stdin* and write to *stdout*

cat - concatenate

grep - "Global Regular Expression Print"

sort - sort

spell - spelling correction

wc - word count

tee - split output

cut - cut fields from a line

Filters enable building useful pipelines

grep command

grep command

Basic syntax

(see man page for the rest of the story)

grep *<options>* "search string" *<filenames...>*

grep -R *<options>* "search string" *<start-directory>*

Use the **grep** command to search the **contents** of files. Use the **-R** option to do a recursive search starting from a directory

Some other useful options:

- i (case insensitive)
- w (whole word)
- v (does not contain)
- n (show line number)
- color (uses color to show matches)

grep for text string

string to search for *files to search contents of*

[simben@opus poems]\$ **grep love Shakespeare/son***

```
Shakespeare/sonnet10:For shame deny that thou bear'st love to any,
Shakespeare/sonnet10:Shall hate be fairer lodg'd then gentle love?
Shakespeare/sonnet10:    Make thee another self for love of me,
Shakespeare/sonnet15:    And all in war with Time for love of you,
Shakespeare/sonnet26:Lord of my love, to whom in vassalage
Shakespeare/sonnet26:    Then may I dare to boast how I do love thee,
Shakespeare/sonnet3:Of his self-love, to stop posterity?
Shakespeare/sonnet3:Calls back the lovely April of her prime,
Shakespeare/sonnet4:Unthrifty loveliness, why dost thou spend
Shakespeare/sonnet5:The lovely gaze where every eye doth dwell
Shakespeare/sonnet9:    No love toward others in that bosom sits
```

files that contain love

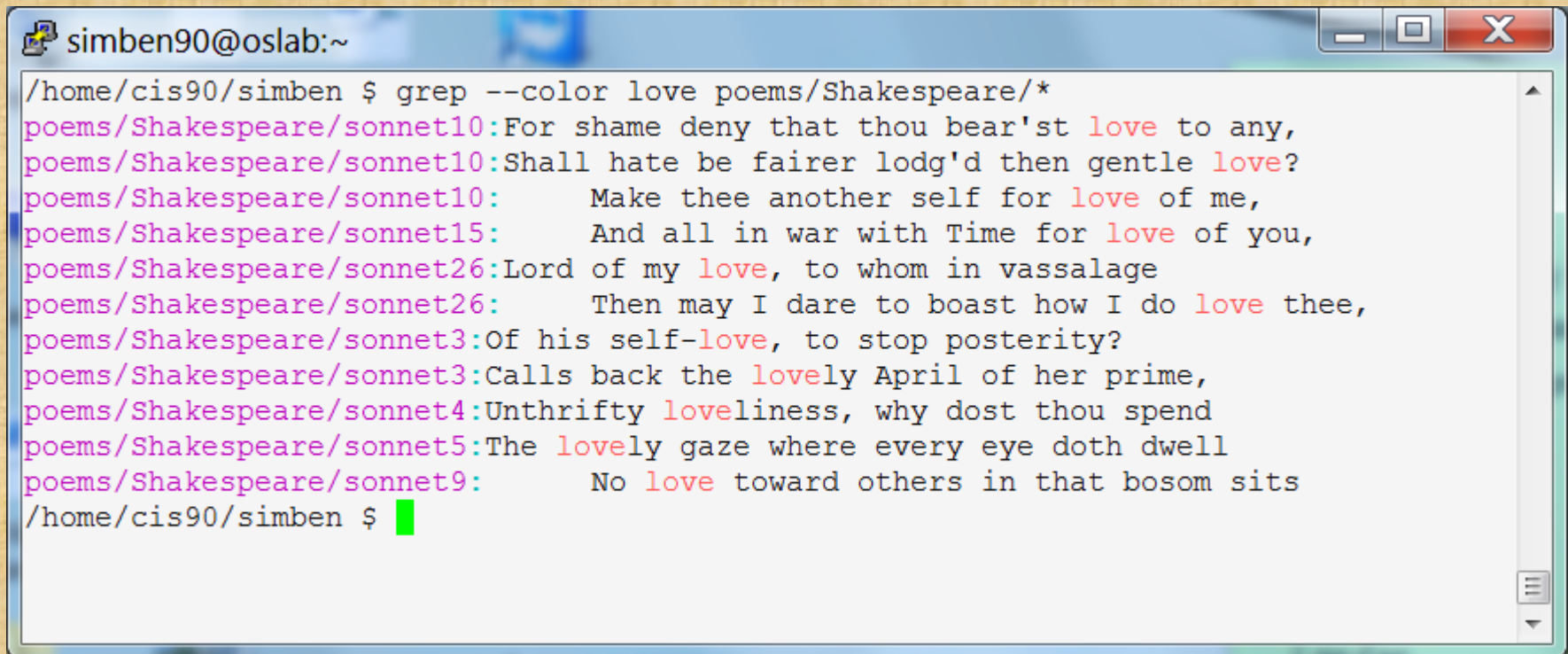
Looking for love in all the wrong places?

Find the word love in Shakespeare's sonnets

Now you try it

The color option

grep --color love poems/Shakespeare/*



```
simben90@oslab:~  
/home/cis90/simben $ grep --color love poems/Shakespeare/*  
poems/Shakespeare/sonnet10:For shame deny that thou bear'st love to any,  
poems/Shakespeare/sonnet10:Shall hate be fairer lodg'd then gentle love?  
poems/Shakespeare/sonnet10:      Make thee another self for love of me,  
poems/Shakespeare/sonnet15:      And all in war with Time for love of you,  
poems/Shakespeare/sonnet26:Lord of my love, to whom in vassalage  
poems/Shakespeare/sonnet26:      Then may I dare to boast how I do love thee,  
poems/Shakespeare/sonnet3:Of his self-love, to stop posterity?  
poems/Shakespeare/sonnet3:Calls back the lovely April of her prime,  
poems/Shakespeare/sonnet4:Unthrifty loveliness, why dost thou spend  
poems/Shakespeare/sonnet5:The lovely gaze where every eye doth dwell  
poems/Shakespeare/sonnet9:      No love toward others in that bosom sits  
/home/cis90/simben $
```

Searching for love with colors

grep the output of a grep

string to search for in the output of the previous command

string to search for *files to search contents of*

```
[simben@opus poems]$ grep love Shakespeare/son* | grep hate  
Shakespeare/sonnet10:Shall hate be fairer lodg'd then gentle love?  
[simben@opus poems]$
```

Find all lines with both love and hate

grep using the -n (line number) option

*string to
search for* *file to search
contents of*

```
/home/cis90/simben $ grep simben90 /etc/passwd  
simben90:x:1201:190:Benji Simms:/home/cis90/simben:/bin/bash
```

Show account in /etc/passwd for simben90

*Option to show
line number* *string to
search for* *file to search
contents of*

```
/home/cis90/simben $ grep -n simben90 /etc/passwd  
52:simben90:x:1201:190:Benji Simms:/home/cis90/simben:/bin/bash
```

*Found in line 52 of
/etc/passwd*

Same as before but include line number it was found on

grep using the -i (case insensitive) option

```
/home/cis90/simben $ grep "so" poems/Shakespeare/sonnet[345]
poems/Shakespeare/sonnet3:Thou dost beguile the world, unbless some mother.
poems/Shakespeare/sonnet3:For where is she so fair whose unear'd womb
poems/Shakespeare/sonnet3:Or who is he so fond will be the tomb,
poems/Shakespeare/sonnet5:A liquid prisoner pent in walls of glass,
```

Look for "so" in sonnet3, sonnet4 and sonnet5

*Use the -i option to make
searches case insensitive*



```
/home/cis90/simben $ grep -i "so" poems/Shakespeare/sonnet[345]
poems/Shakespeare/sonnet3:Thou dost beguile the world, unbless some mother.
poems/Shakespeare/sonnet3:For where is she so fair whose unear'd womb
poems/Shakespeare/sonnet3:Or who is he so fond will be the tomb,
poems/Shakespeare/sonnet3:So thou through windows of thine age shalt see,
poems/Shakespeare/sonnet4:So great a sum of sums, yet canst not live?
poems/Shakespeare/sonnet5:A liquid prisoner pent in walls of glass,
```

Look for "so" (case insensitive) in sonnet3, sonnet4 and sonnet5

grep using the -w (whole word) option

```
/home/cis90/simben $ grep so poems/Shakespeare/sonnet[345]  
poems/Shakespeare/sonnet3:Thou dost beguile the world, unbless some mother.  
poems/Shakespeare/sonnet3:For where is she so fair whose unear'd womb  
poems/Shakespeare/sonnet3:Or who is he so fond will be the tomb,  
poems/Shakespeare/sonnet5:A liquid prisoner pent in walls of glass,
```

Look for "so" in sonnet3, sonnet4 and sonnet5

*Use the -w option for whole
word only searches*

```
/home/cis90/simben $ grep -w so poems/Shakespeare/sonnet[345]  
poems/Shakespeare/sonnet3:For where is she so fair whose unear'd womb  
poems/Shakespeare/sonnet3:Or who is he so fond will be the tomb,
```

Look for "so" (whole word only) in sonnet3, sonnet4 and sonnet5

grep recursively with the -R option

Text string to search for

*Search recursively
(all sub-directories)*

*starting directory
(. is the current directory)*

discard permission errors

```
/home/cis90/simben $ grep -R kind . 2> /dev/null
./poems/Shakespeare/sonnet10:Be as thy presence is gracious and kind,
./poems/Shakespeare/sonnet10:Or to thyself at least kind-hearted prove:
./poems/Shakespeare/sonnet35: Let no unkind, no fair beseechers kill;
./poems/Yeats/mooncat:When two close kindred meet,
./poems/Anon/ant:distorted out of kind,
./letter:Mother, Father, kindly disregard this letter.
./bin/enlightenment: echo "to find out what kind of file \"what_am_i\" is"
./misc/mystery: echo "to find out what kind of file \"what_am_i\" is"
```

Search recursively for files containing "kind"

grep command

Background

Apache is the worlds most popular web server and it's installed on Opus. Try it, you can browse to oslab.cabrillo.edu.

Every Apache configuration file must specify the location (an absolute pathname) of the documents to publish on the world wide web. This is done with the **DocumentRoot** directive. This directive is found in every Apache configuration file.

All configuration files are kept in /etc.

Tasks

- Can you use **grep** to find the Apache configuration file?
Hint: use the -R option to recursively search all sub-directories
- What are the names of the files in Apache's document root directory on Opus?
Hint: Use the ls command on the document root directory

spell command

spell command

Basic syntax

(see man page for the rest of the story)

spell *<filepath>*

spell *<filepath>* *<filepath>* ...

The **spell** command is used to check spelling of words in one or more text files

spell command

Task: Run a spell check on the magna_cart file

```
/home/cis90/simben $ cd docs
/home/cis90/simben/docs $ ls
magna_carta MarkTwain policy
/home/cis90/simben/docs $ spell magna_carta
Anjou
Arundel
Aymeric
Bergh
Daubeney
de
honour
kingdon
Pandulf
Poitou
Poppeley
seneschal
subdeacon
Warin
```

*The spell command will
show any words not
found in the dictionary.*

spell command

Count the number of misspelled words in the magna_carta file

*The -l option instructs the **wc** command to just count the number of lines*

```
/home/cis90/simben/docs $ spell magna_carta | wc -l  
14
```

*Pipe the output of the **spell** command (the misspelled words) into the input of the **wc** command*

Activity

```
/home/cis90/simben $ cat edits/spellk
```

Spell Check

```
Eye halve a spelling chequer  
It came with my pea sea  
It plainly marques four my revue  
Miss steaks eye kin knot sea.  
Eye strike a key and type a word  
And weight four it two say  
Weather eye am wrong oar write  
It shows me strait a weigh.  
As soon as a mist ache is maid  
It nose bee fore two long  
And eye can put the error rite  
Its rare lea ever wrong.  
Eye have run this poem threw it  
I am shore your pleased two no  
Its letter perfect awl the weigh  
My chequer tolled me sew.
```

```
/home/cis90/simben $
```

*How many misspelled
word are in your spellk
file?*

*Write your answer in the
chat window.*

tee command

tee command

Basic syntax

(see man page for the rest of the story)

tee *<filepath>*

The **tee** command, a filter, reads from **stdin** and writes to **stdout** AND to the file specified as the argument.

tee command

For example, the following command sends a sorted list of the current users logged on to the system to the screen, and saves an unsorted list to a file named users.

```
/home/cis90/simben $ who | tee users | sort
```

```
caumar98 pts/5      2014-03-17 17:29 (75.140.158.6)
caumar98 pts/6      2014-03-17 17:41 (75.140.158.6)
chejul98 pts/1      2014-03-17 19:42 (acbe4f9e.ipt.aol.com)
goojun172 pts/7     2014-03-17 19:53 (c-67-169-144-100.hsd1.ca.comcast.net)
hovdav98 pts/2      2014-03-16 14:48 (c-76-126-1-130.hsd1.ca.comcast.net)
mmatera pts/4       2014-03-13 16:06 (2607:f380:80f:f828:e108:c48e:9e1a:57ff)
rsimms pts/0        2014-03-17 09:40 (2001:470:1f05:9b3:3044:7820:6ce0:8a4)
/home/cis90/simben $
```

```
/home/cis90/simben $ cat users
```

```
rsimms pts/0        2014-03-17 09:40 (2001:470:1f05:9b3:3044:7820:6ce0:8a4)
chejul98 pts/1      2014-03-17 19:42 (acbe4f9e.ipt.aol.com)
hovdav98 pts/2      2014-03-16 14:48 (c-76-126-1-130.hsd1.ca.comcast.net)
mmatera pts/4       2014-03-13 16:06 (2607:f380:80f:f828:e108:c48e:9e1a:57ff)
caumar98 pts/5      2014-03-17 17:29 (75.140.158.6)
caumar98 pts/6      2014-03-17 17:41 (75.140.158.6)
goojun172 pts/7     2014-03-17 19:53 (c-67-169-144-100.hsd1.ca.comcast.net)
/home/cis90/simben $
```

tee command

```
/home/cis90/simben $ head edits/spellk
Spell Check
```

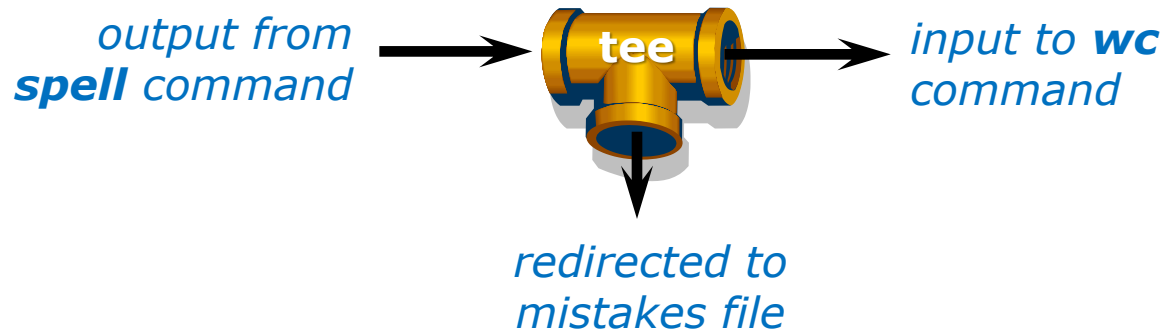
```
Eye halve a spelling chequer
It came with my pea sea
It plainly marques four my revue
Miss steaks eye kin knot sea.
Eye strike a key and type a word
And weight four it two say
Weather eye am wrong oar write
```

The misspelled words from spell are piped to the tee command

*The **tee** command copies the misspelled words to stdout and to the file named mistakes*

```
/home/cis90/simben $ spell edits/spellk | tee mistakes | wc -l
1
/home/cis90/simben $ cat mistakes
chequer
```

*The **wc** command counts the misspelled words*



cut command

cut command

Basic syntax

(see man page for the rest of the story)

cut -f *<num>* **-d** "*<delimiter-character>*" *<pathname>*

cut -c *<start column>-<end column>* *<pathname>*

*The **cut** command can cut text from a line by delimited fields or by a range of columns.*

cut command

(cut text using delimited fields)

```
[rsimms@oslab ~]$ grep $LOGNAME /etc/passwd
rsimms:x:201:503:Rich Simms:/home/rsimms:/bin/bash
```

1st
field

2nd
field

3rd
field

4th
field

5th
field

6th
field

7th
field

```
[rsimms@oslab ~]$ grep $LOGNAME /etc/passwd | cut -f 7 -d ":"
/bin/bash
```

Cut the 7th field

Using ":" as the delimiter

cut command (cut text by column numbers)

```
/home/cis90/simben $ ls -l letter
```

```
-rw-r--r--. 1 simben90 cis90 1044 Jul 20 2001 letter
```

```
123456789012345678901234567890123456789012345678901234567890
```

column 2

column 10

```
/home/cis90/simben $ ls -l letter | cut -c 2-10
```

```
rw-r--r--
```

*Cut columns
2 through 10*

```
/home/cis90/simben $ perm=$(ls -l letter | cut -c 2-10)
```

*This puts the output of the pipeline
above into a variable named perm*

```
/home/cis90/simben $ echo The permissions on letter are $perm
```

```
The permissions on letter are rw-r--r--
```

*Which we can use to
build a custom message*

Pipeline Practice

Class Exercise

Pipeline Tasks

Background

The **last** command searches through /var/log/wtmp and prints out a list of users logged in since that file was created.

Task

Can you see the last times you were logged in on a Wednesday and then count them?

```
last | grep $LOGNAME
```

```
last | grep $LOGNAME | grep "Wed"
```

```
last | grep $LOGNAME | grep "Wed" | wc -l
```

On what days do you log in the most? the least?

Class Exercise

Pipeline Tasks

Background

The **cut** command can cut a field out of a line of text where each field is delimited by some character.

The */etc/passwd* file uses the ":" as the delimiter between fields. The 5th field is a comment field for the user account.

Task

Build up a pipeline, one pipe at a time:

```
cat /etc/passwd
```

```
cat /etc/passwd | grep $LOGNAME
```

```
cat /etc/passwd | grep $LOGNAME | cut -f 5 -d ":"
```

What gets printed with the last pipeline?

ONLY
If Time Allows

Permissions

“The rest of the story”

- Special Permissions
- ACLs
- Extended Attributes
- SELinux



This module is for your information only. We won't use this in CIS 90 but its good to know they exist. More in CIS 191, 192 and 193



Special Permissions

Sticky bit - used on directories, e.g. /tmp, so that only owners can rename or remove files even though other users may have write permission on the directory.

SetUID or SetGID - allows a user to run an program file with the permissions of the file's owner (Set User ID) or the file's group (Set Group ID). Examples include **ping** and **passwd** commands.

FYI
only

Special Permissions

Sticky bit - used on directories, e.g. /tmp, so that only owners can rename or remove files even though other users may have write permission on the directory.

```
/home/cis90/simben $ ls -ld /tmp
```

```
drwxrwxrwt. 3 root root 4096 Oct 16 16:13 /tmp
```

*green background
with black text*

```
/home/cis90/simben $ mkdir tempdir
```

```
/home/cis90/simben $ chmod 777 tempdir/
```

```
/home/cis90/simben $ ls -ld tempdir/
```

```
drwxrwxrwx. 2 simben90 cis90 4096 Oct 16 15:25 tempdir/
```

*green background
with blue text*

set sticky bit

```
/home/cis90/simben $ chmod 1777 tempdir
```

```
/home/cis90/simben $ ls -ld tempdir/
```

```
drwxrwxrwt. 2 simben90 cis90 4096 Oct 16 15:25 tempdir/
```

sticky bit set

*green background
with black text*

FYI
only

Special Permissions

SetUID or SetGID - allows a user to run a program file with the permissions of the file's owner (Set User ID) or the file's group (Set Group ID). Examples include **ping** and **passwd** commands.

```
/home/cis90/simben $ ls -l /bin/ping /usr/bin/passwd
-rwsr-xr-x. 1 root root 36892 Jul 18 2011 /bin/ping
-rwsr-xr-x. 1 root root 25980 Feb 22 2012 /usr/bin/passwd
```

*red background
with gray text*

```
/home/cis90/simben $ echo banner Hola > hola; chmod +x hola; ls -l hola
-rwxrwxr-x. 1 simben90 cis90 12 Oct 16 16:45 hola
```

```
/home/cis90/simben $ chmod 4775 hola
/home/cis90/simben $ ls -l hola
-rwsrwxr-x. 1 simben90 cis90 12 Oct 16 16:45 hola
/home/cis90/simben $ chmod 2775 hola
/home/cis90/simben $ ls -l hola
-rwxrwsr-x. 1 simben90 cis90 12 Oct 16 16:45 hola
```



ACLs (Access Control Lists)

ACLs - offer a finer granularity of control allowing additional permissions to be set for specific users or groups.

FYI
only

ACLs (Access Control Lists)

ACLs - offer a finer granularity of control allowing additional permissions to be set for specific users or groups.

```
/home/cis90/simben $ echo yabadabadoo > yogi
/home/cis90/simben $ chmod 400 yogi
/home/cis90/simben $ ls -l yogi
-r-----. 1 simben90 cis90 12 Oct 16 17:02 yogi

/home/cis90/simben $ getfacl yogi
# file: yogi
# owner: simben90
# group: cis90
user::r--
group::---
other::---
```

*Create a file and
set permissions
to 400*

*Use **getfacl** to
show ACLs*

```
[milhom90@oslab ~]$ cat ../simben/yogi
cat: ../simben/yogi: Permission denied
```

*Homer, a member of the cis90
group can't read the file*

```
[rodduk90@oslab ~]$ cat ../simben/yogi
cat: ../simben/yogi: Permission denied
```

*Duke, a member of the cis90
group can't read the file either*

FYI
only

ACLs (Access Control Lists)

Let's give special permissions to one user

```
/home/cis90/simben $ setfacl -m u:milhom90:rw yogi
/home/cis90/simben $ ls -l yogi
-r--rw----+ 1 simben90 cis90 12 Oct 16 17:02 yogi
/home/cis90/simben $ getfacl yogi
# file: yogi
# owner: simben90
# group: cis90
user::r--
user:milhom90:rw-
group:---
mask::rw-
other:---
```

modify

*Allow milhom90 to
have read/write
access*

```
[milhom90@oslab ~]$ cat ../simben/yogi
yabadabadoo
```

Homer can now read the file

```
[rodduk90@oslab ~]$ cat ../simben/yogi
cat: ../simben/yogi: Permission denied
```

But not Duke

FYI
only

ACLs (Access Control Lists)

Let's remove the special permissions to that user

remove all base ACLs

```
/home/cis90/simben $ setfacl -b yogi
/home/cis90/simben $ ls -l yogi
-r----- . 1 simben90 cis90 12 Oct 16 17:02 yogi
/home/cis90/simben $ getfacl yogi
# file: yogi
# owner: simben90
# group: cis90
user::r--
group::---
other::---
```

*Remove all ACLs on
yogi file*

```
[milhom90@oslab ~]$ cat ../simben/yogi
cat: ../simben/yogi: Permission denied
```

Now Homer can't read it again

```
[rodduk90@oslab ~]$ cat ../simben/yogi
cat: ../simben/yogi: Permission denied
```

Same for Duke



Extended File Attributes

Extended Attributes - the root user can set some extended attribute bits to enhance security.

FYI
only

Extended File Attributes

Let's use extended file attributes to totally lock down a file against changes, even by its owner!

```
/home/cis90/simben $ echo yabadabadoo > yogi
/home/cis90/simben $ ls -l yogi
-rw-rw-r--. 1 simben90 cis90 12 Oct 16 17:29 yogi
```

Create a sample file to work on

*The root user sets the **immutable bit (i)** so Benji cannot remove his own file*

```
[root@oslab ~]# lsattr /home/cis90/simben/yogi
-----e- /home/cis90/simben/yogi
[root@oslab ~]# chattr +i /home/cis90/simben/yogi
[root@oslab ~]# lsattr /home/cis90/simben/yogi
----i-----e- /home/cis90/simben/yogi
```

```
/home/cis90/simben $ ls -ld ~
drwxr-xr-x. 17 simben90 cis90 4096 Oct 16 17:29 /home/cis90/simben
/home/cis90/simben $ rm yogi
rm: remove write-protected regular file `yogi'? yes
rm: cannot remove `yogi': Operation not permitted
```

!!



Extended File Attributes

Extended Attributes - the root user can set some extended attribute bits to enhance security.

*The root user removes the **immutable bit (i)** so Benji can remove his own file again*

```
[root@oslab ~]# chattr -i /home/cis90/simben/yogi
[root@oslab ~]# lsattr /home/cis90/simben/yogi
-----e- /home/cis90/simben/yogi
```

```
/home/cis90/simben $ ls -ld ~
drwxr-xr-x. 17 simben90 cis90 4096 Oct 16 17:29 /home/cis90/simben
/home/cis90/simben $ rm yogi
/home/cis90/simben $
```

FYI
only

Extended File Attributes

Let's use extended file attributes to allow the file to be appended (but still not emptied or removed)

```
/home/cis90/simben $ ls -l yogi
-rw-rw-r--. 1 simben90 cis90 12 Oct 16 17:41 yogi
```

*The root user sets the **append only bit (a)** so Benji can only append to his file*

```
[root@oslab ~]# lsattr /home/cis90/simben/yogi
-----e- /home/cis90/simben/yogi
[root@oslab ~]# chattr +a /home/cis90/simben/yogi
[root@oslab ~]# lsattr /home/cis90/simben/yogi
----a-----e- /home/cis90/simben/yogi
```

```
/home/cis90/simben $ rm yogi
rm: cannot remove `yogi': Operation not permitted
/home/cis90/simben $ > yogi
-bash: yogi: Operation not permitted
/home/cis90/simben $ echo yowser >> yogi
/home/cis90/simben $
```



SELinux context

SELinux - Security Enhanced Linux. SELinux is a set of kernel modifications that provide Mandatory Access Control (MAC). In MAC-enabled systems there is a strict set of security policies for all operations which users cannot override. The primary original developer of SELinux was the NSA (National Security Agency).



SELinux context

Use the Z option on the ls command to show the SELinux context on a file

```
[root@oslab selinux]# ls -lZ test*
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 test01.html
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 test02.html
```

user
role
type
level



SELinux context

Create two identical web pages with identical permissions

```
[root@oslab selinux]# cp test01.html test02.html  
cp: overwrite `test02.html'? yes
```

```
[root@oslab selinux]# ls -lZ test*  
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 test01.html  
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 test02.html
```

Use chcon command to change the SELinux context on one file

```
[root@oslab selinux]# chcon -v -t home_root_t test02.html  
changing security context of `test02.html'
```

```
[root@oslab selinux]# ls -lZ test*  
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 test01.html  
-rw-r--r--. root root unconfined_u:object_r:home_root_t:s0 test02.html
```

*Note, the root user's home files are
not appropriate web content*

FYI
only

SELinux context

SELinux won't let Apache publish a file with an inappropriate context

```
[root@oslab selinux]# ls -lZ test*
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 test01.html
-rw-r--r--. root root unconfined_u:object_r:home_root_t:s0 test02.html
[root@oslab selinux]#
```

test01.html



type = httpd_sys_content_t



test02.html



type = home_root_t

Wrap up

New commands:

find	find files or content
grep	look for text strings
last	show last logins
sort	perform sorts
spell	spell checking
tee	save output to a file
wc	count lines or words in a file

Next Class

Assignment: Check Calendar Page on web site to see what is due next week.

Lab 7

Quiz questions for next class:

- How do you redirect error messages to the bit bucket?
- What command could you use to get an approximate count of all the files on Opus and ignore the permission errors?
- For **sort dognames > dogsinorder** where does the sort process obtain the actual names of the dogs to sort?
 - a) stdin
 - b) the command line
 - c) directly from the file dognames

Backup

Lab 6

Tips

```
/home/cis90/simben $ tree poems/
```

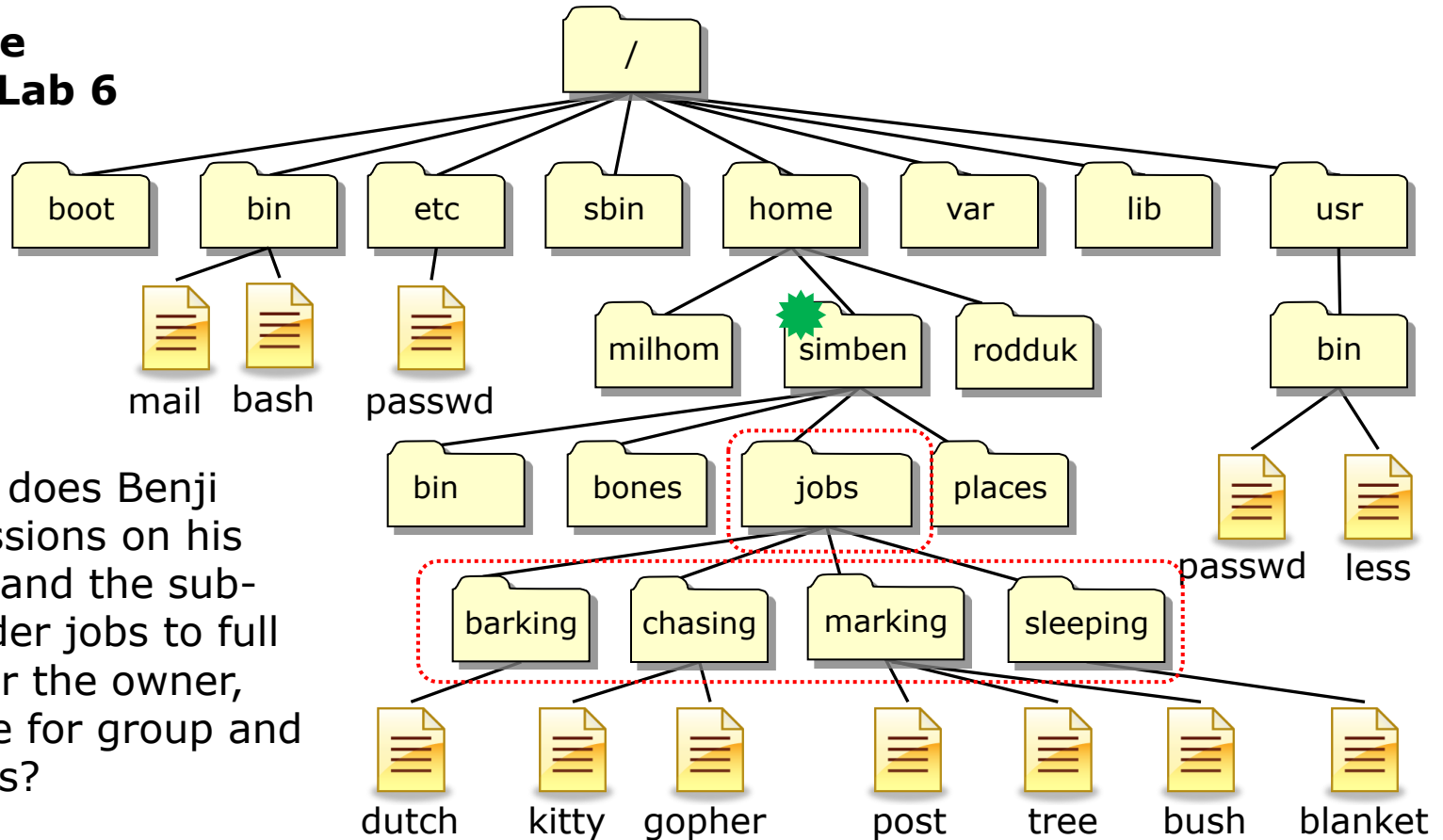
```
poems/
├── Anon
│   ├── ant
│   ├── nursery
│   └── twister
├── Blake
│   ├── jerusalem
│   └── tiger
├── Shakespeare
│   ├── sonnet1
│   ├── sonnet10
│   ├── sonnet11
│   ├── sonnet15
│   ├── sonnet17
│   ├── sonnet2
│   ├── sonnet26
│   ├── sonnet3
│   ├── sonnet35
│   ├── sonnet4
│   ├── sonnet5
│   ├── sonnet6
│   ├── sonnet7
│   └── sonnet9
└── Yeats
    ├── mooncat
    ├── old
    └── whitebirds
```


One of the steps in Lab 6

9. Set the permissions of your poems directory and its subdirectories so that you have full permissions as owner, but group and others have no write permission. Group and others should still have read and execute permission.

```
4 directories, 22 files
/home/cis90/simben $
```

An example related to Lab 6 Q9



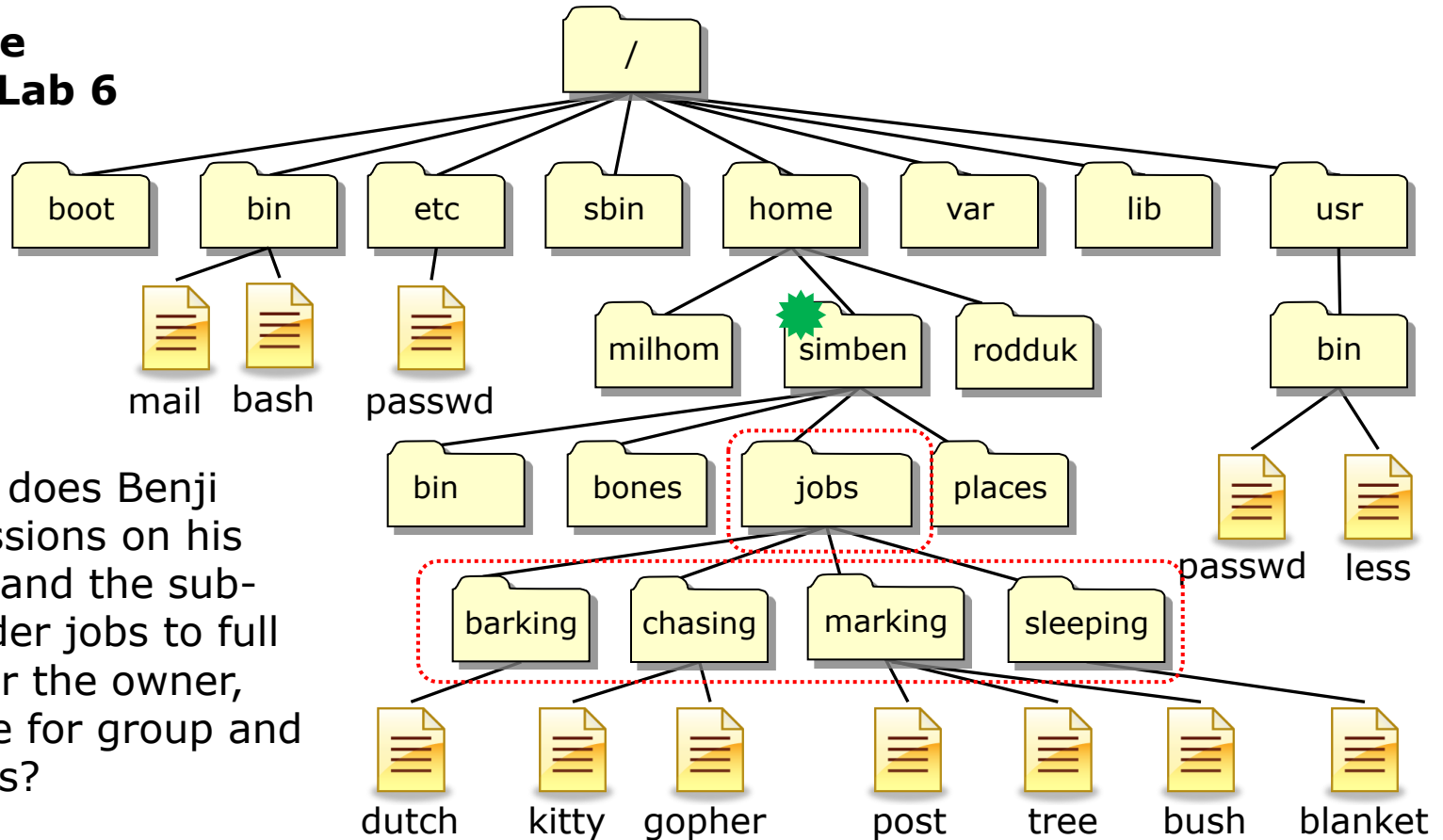
From  how does Benji change permissions on his jobs directory and the sub-directories under jobs to full permissions for the owner, read & execute for group and none for others?


```

chmod 750 jobs
cd jobs
chmod 750 barking
chmod 750 chasing
chmod 750 marking
chmod 750 sleeping
  
```

*The "elbow grease" method:
It works and takes 6 commands to
complete*

An example related to Lab 6 Q9



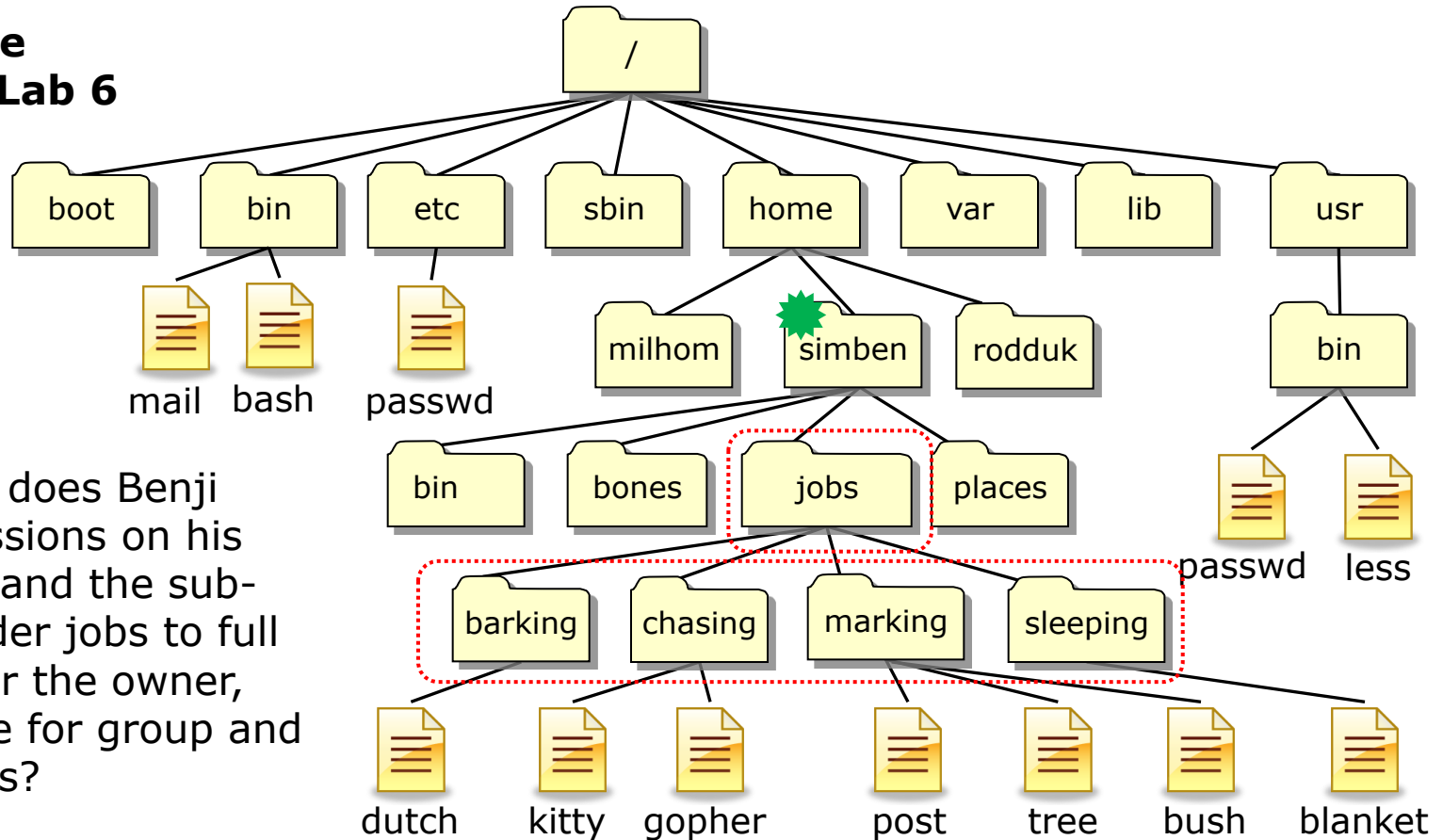
From  how does Benji change permissions on his jobs directory and the sub-directories under jobs to full permissions for the owner, read & execute for group and none for others?


```

chmod 750 jobs
chmod 750 jobs/barking
chmod 750 jobs/chasing
chmod 750 jobs/markings
chmod 750 jobs/sleeping
  
```

Using relative paths allows us to do the same thing and uses one less command

An example related to Lab 6 Q9

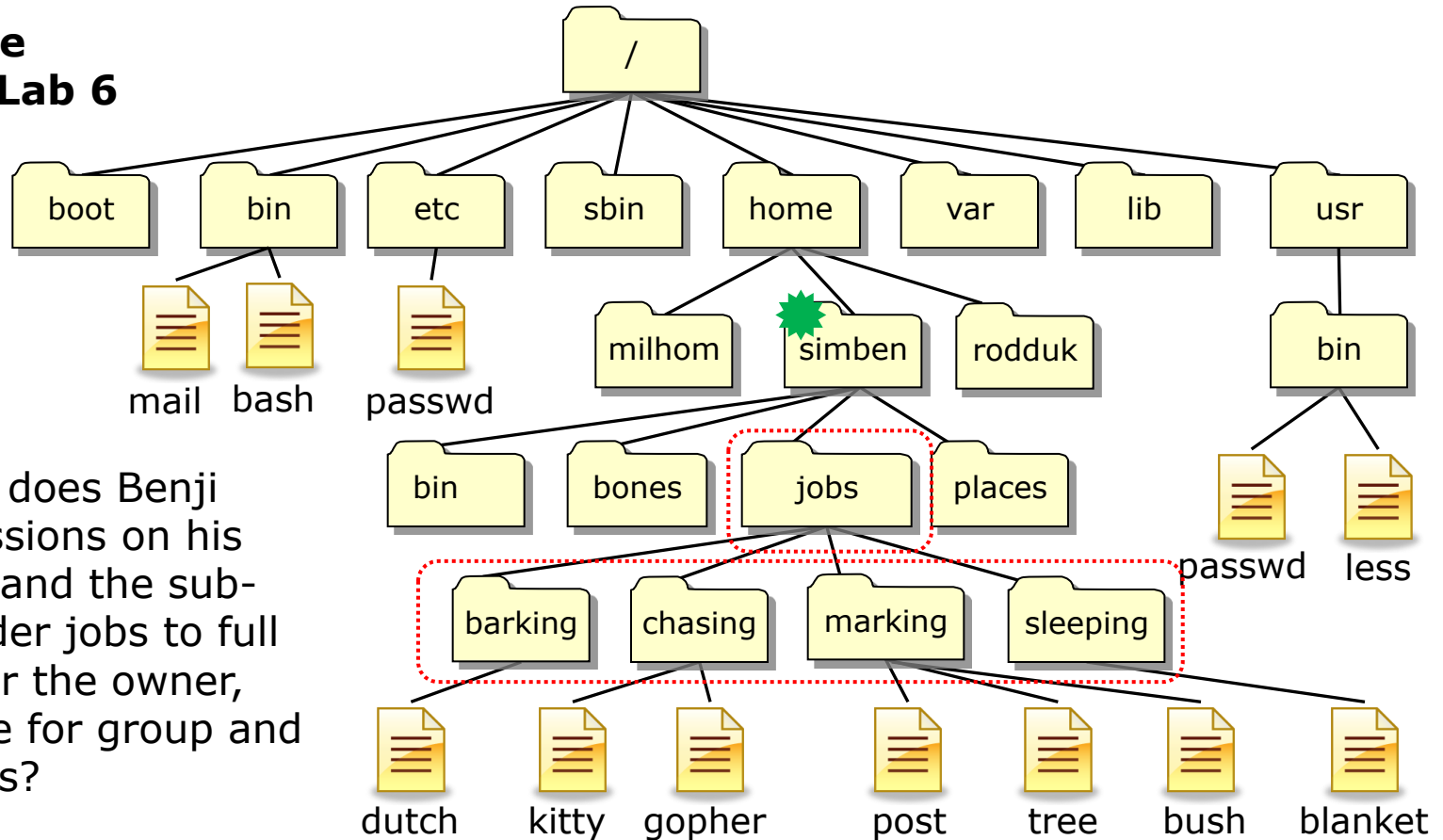



From  how does Benji change permissions on his jobs directory and the sub-directories under jobs to full permissions for the owner, read & execute for group and none for others?

chmod 750 jobs
chmod 750 jobs/*

Using relative paths and a filename expansion metacharacter lets us do the same things with only two commands

An example related to Lab 6 Q9

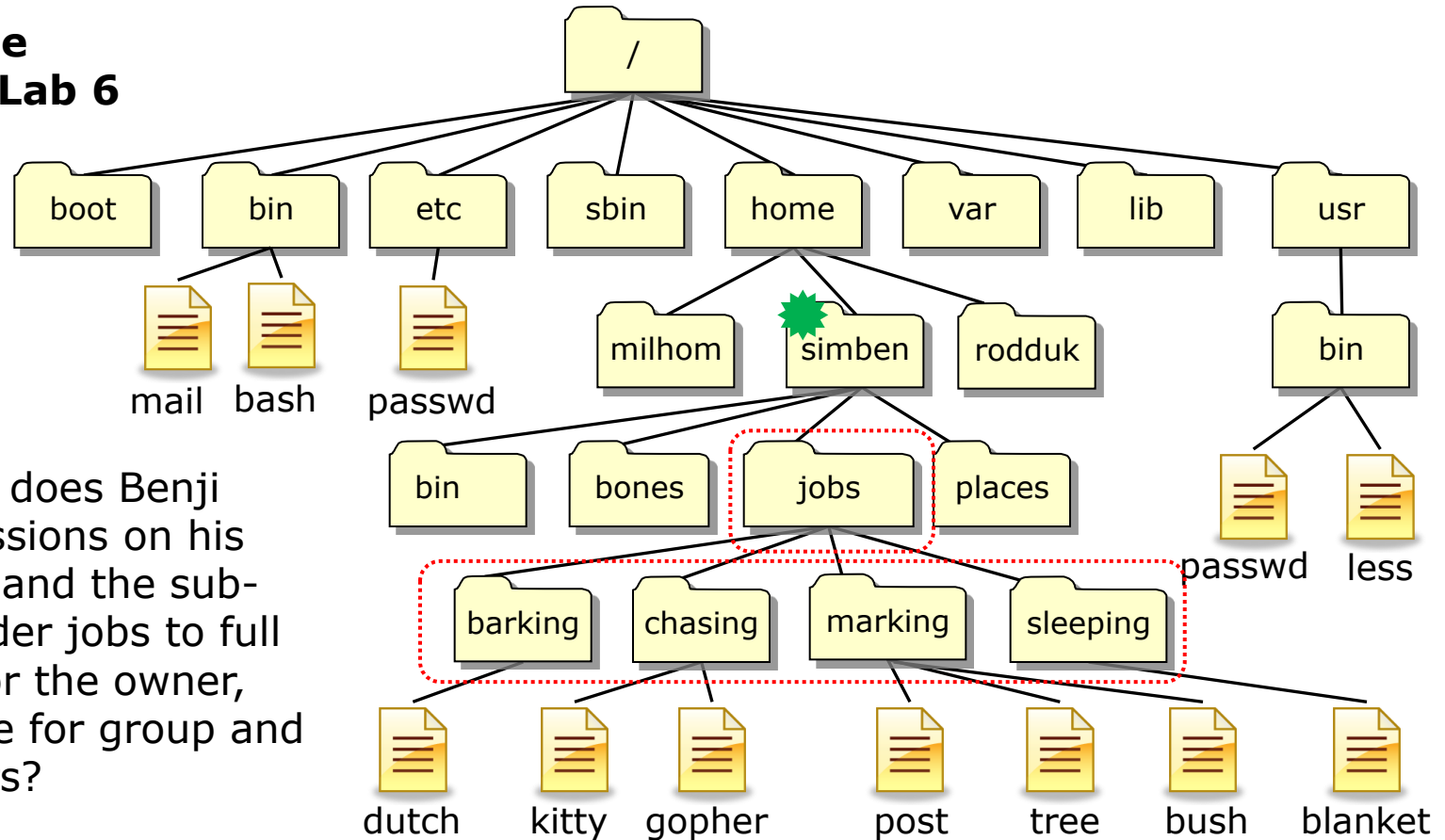



From  how does Benji change permissions on his jobs directory and the sub-directories under jobs to full permissions for the owner, read & execute for group and none for others?

chmod 750 jobs jobs/*

*The "Linux guru" method:
Using relative paths, filename expansion
metacharacter and multiple arguments lets us
do the same thing with one command!*

An example related to Lab 6 Q9



From  how does Benji change permissions on his jobs directory and the sub-directories under jobs to full permissions for the owner, read & execute for group and none for others?

The "elbow grease" method:
chmod 750 jobs
cd jobs
chmod 750 barking
chmod 750 chasing
chmod 750 marking
chmod 750 sleeping

*Both ways
work, the
choice is yours!*

The "Linux guru" method:
chmod 750 jobs jobs/*

```
/home/cis90/simben $ tree poems/
```

```
poems/
├── Anon
│   ├── ant
│   ├── nursery
│   └── twister
├── Blake
│   ├── jerusalem
│   └── tiger
├── Shakespeare
│   ├── sonnet1
│   ├── sonnet10
│   ├── sonnet11
│   ├── sonnet15
│   ├── sonnet17
│   ├── sonnet2
│   ├── sonnet26
│   ├── sonnet3
│   ├── sonnet35
│   ├── sonnet4
│   ├── sonnet5
│   ├── sonnet6
│   ├── sonnet7
│   └── sonnet9
└── Yeats
    ├── mooncat
    ├── old
    └── whitebirds
```

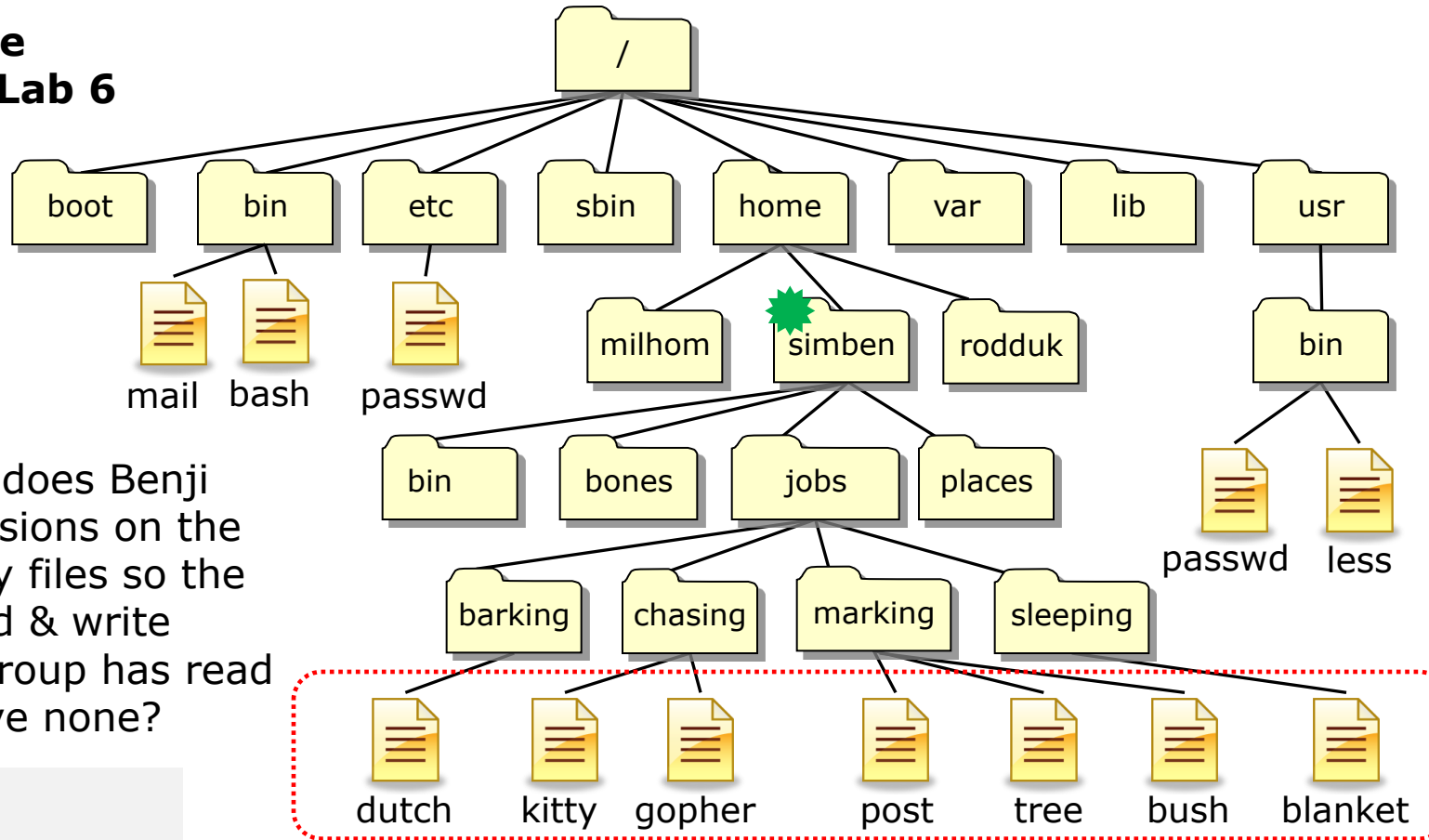
Another step in Lab 6


10. Set all ordinary files under the poems directory to be read only for user, group, and others. We want everyone to read our poetry, but no one should modify it, including ourselves.

See if you can do this using a minimum number of commands. (hint: use filename expansion characters).

```
4 directories, 22 files
/home/cis90/simben $
```

An example related to Lab 6 Q10



From  how does Benji change permissions on the circled ordinary files so the owner has read & write permissions, group has read and others have none?

```

cd jobs
cd barking
chmod 640 dutch
cd ..

```

```

cd chasing
chmod 640 kitty
chmod 640 gopher
cd ..

```

```

cd marking
chmod 640 post
chmod 640 tree
chmod 640 bush
cd ..

```

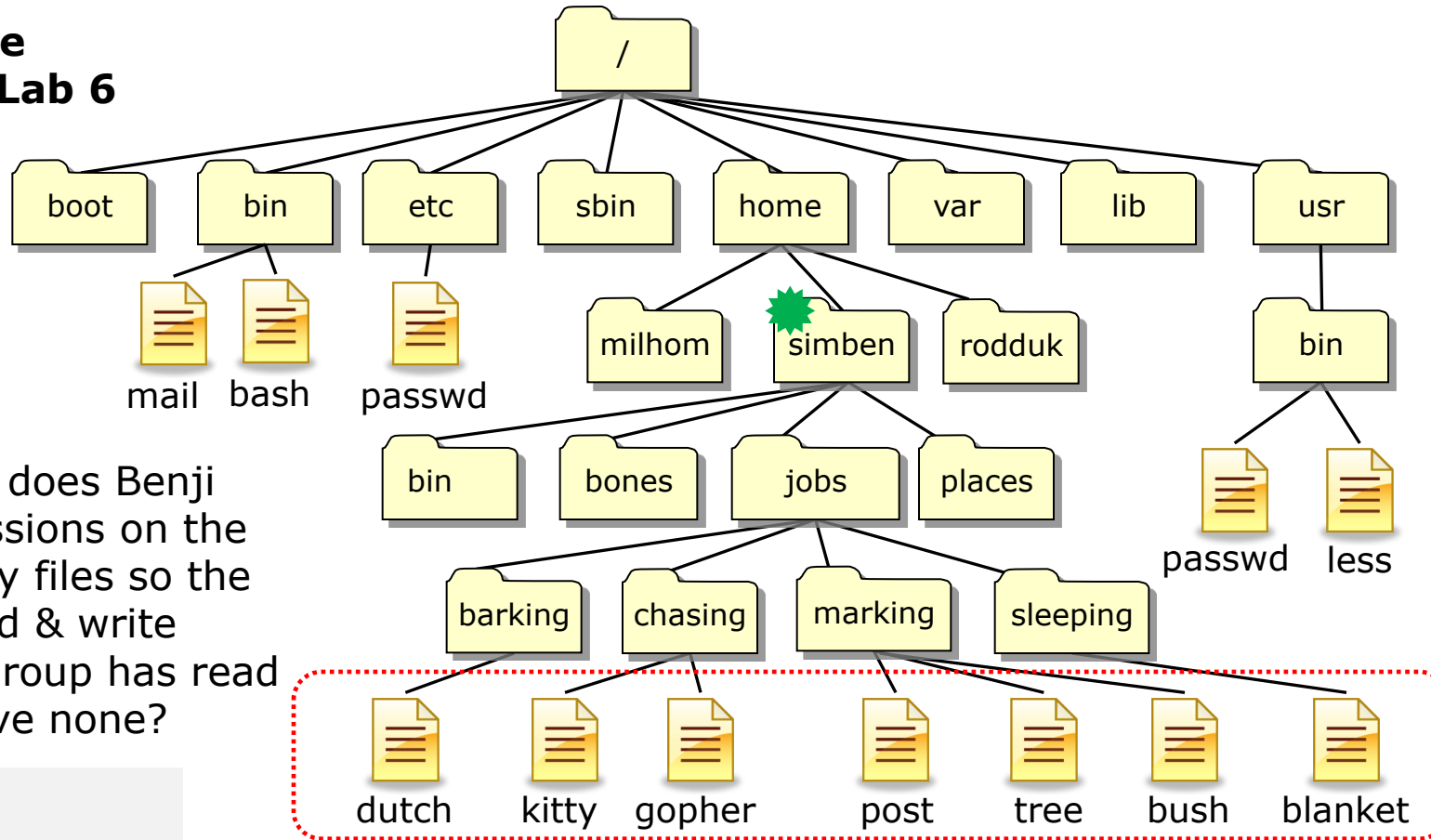
```


cd sleeping
chmod 640 blanket
cd

```

The "elbow grease" method takes 16 commands

An example related to Lab 6 Q10



From  how does Benji change permissions on the circled ordinary files so the owner has read & write permissions, group has read permissions, and others have none?

cd jobs
cd barking
chmod 640 dutch
cd ..

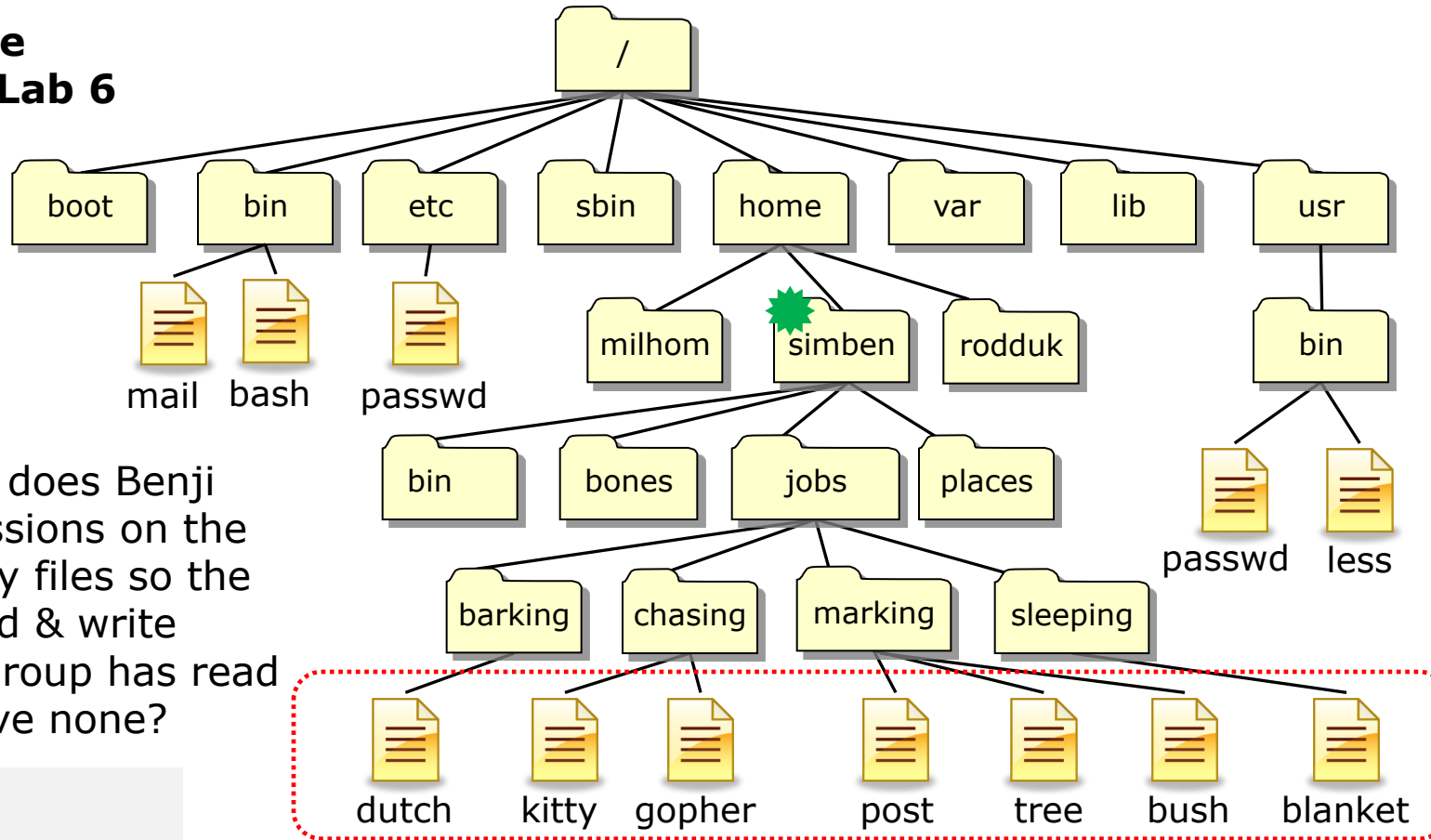
cd chasing
chmod 640 kitty gopher
cd ..


cd marking
chmod 640 post tree bush
cd ..

cd sleeping
chmod 640 blanket
cd

*Using multiple arguments
on chmod:
takes 13 commands*

An example related to Lab 6 Q10



From  how does Benji change permissions on the circled ordinary files so the owner has read & write permissions, group has read permissions, group has read and others have none?

cd jobs
cd barking
chmod 640 *
cd ..

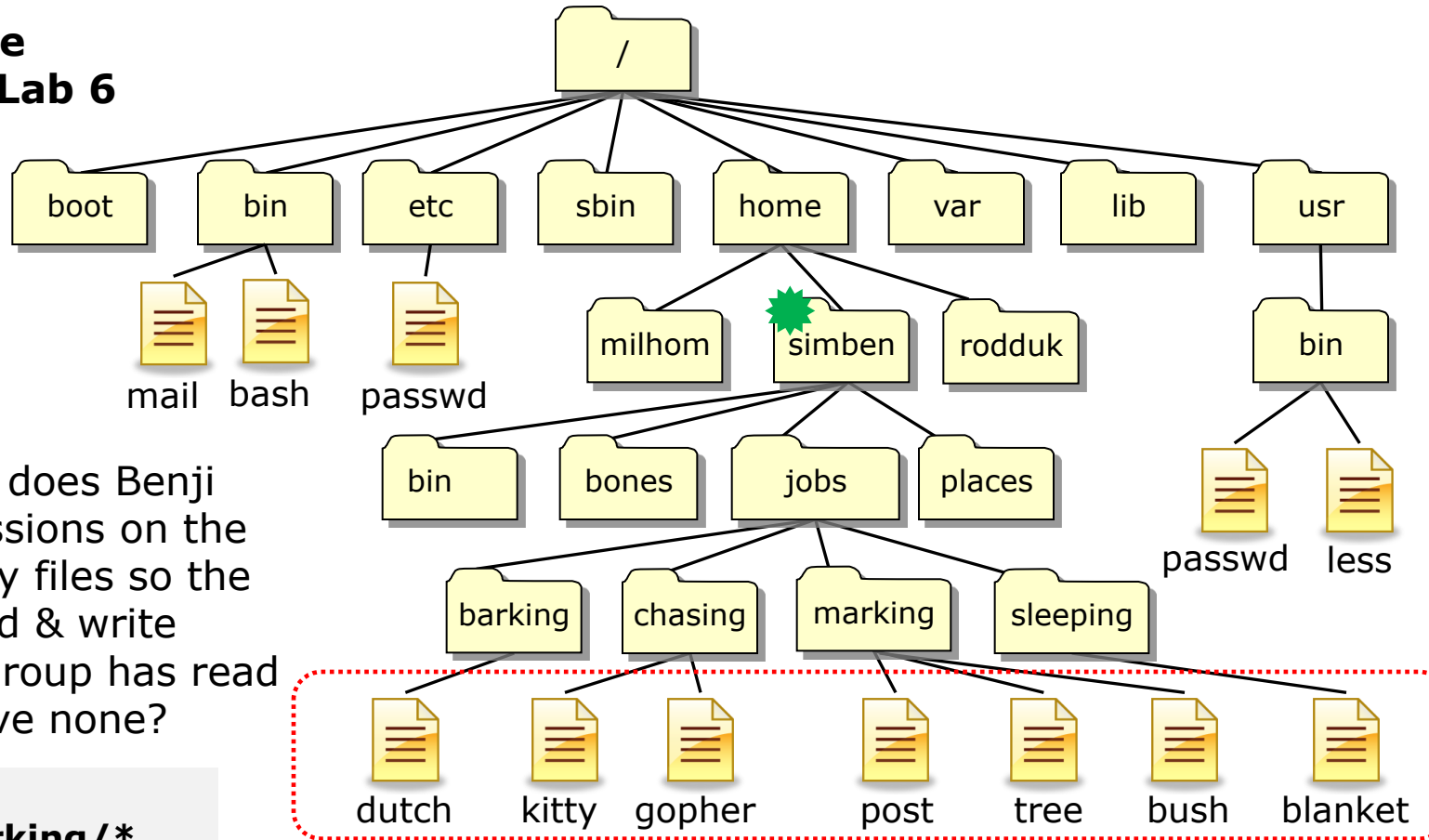
cd chasing
chmod 640 *
cd ..


cd marking
chmod 640 *
cd ..

cd sleeping
chmod 640 *
cd

*Using * (filename expansion metacharacter) takes 13 commands but fewer keystrokes*

An example related to Lab 6 Q10



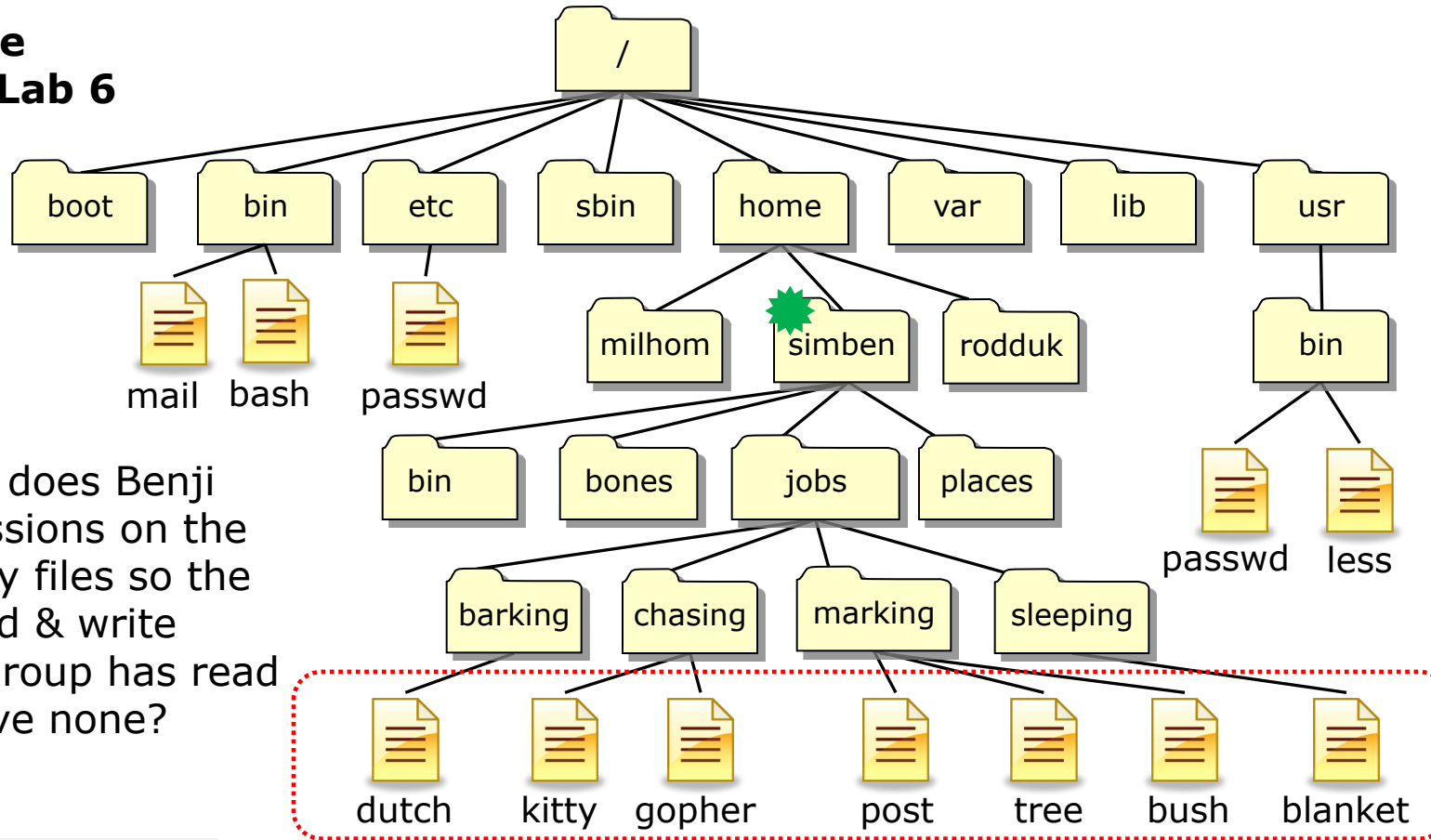
From  how does Benji change permissions on the circled ordinary files so the owner has read & write permissions, group has read permissions, and others have none?


```

cd jobs
chmod 640 barking/*
chmod 640 chasing/*
chmod 640 marking/*
chmod 640 sleeping/*
cd ..
    
```

Using relative paths and filename expansion characters takes 6 commands

An example related to Lab 6 Q10

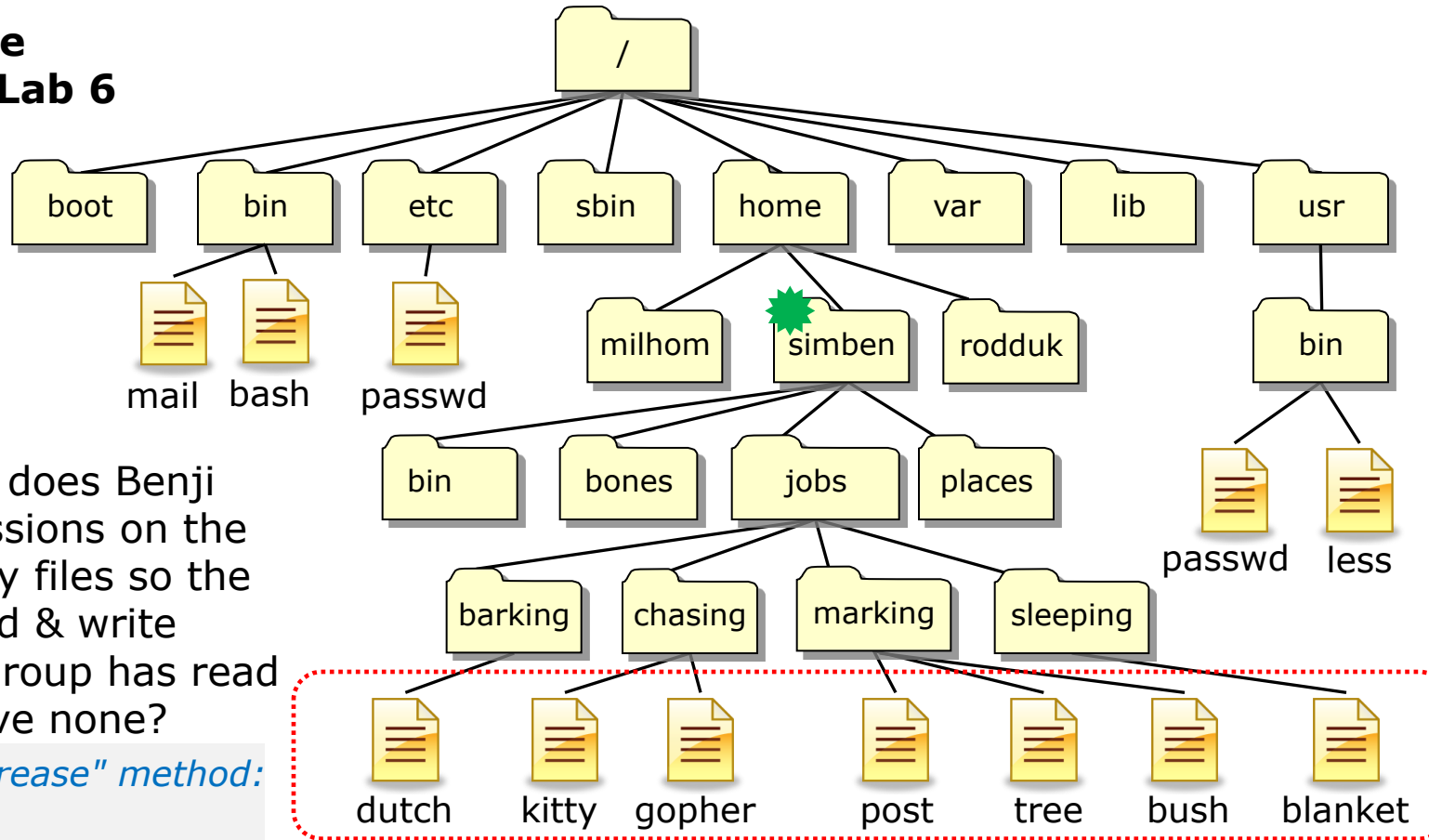



From  how does Benji change permissions on the circled ordinary files so the owner has read & write permissions, group has read permissions, and others have none?

chmod 640 jobs/*/*

*The Linux guru method:
Using relative paths, filename expansion characters and combining all arguments on a single command line takes one command*

An example related to Lab 6 Q10



From  how does Benji change permissions on the circled ordinary files so the owner has read & write permissions, group has read permissions, others have none?

The "elbow grease" method:

```

cd jobs
cd barking
chmod 640 dutch
cd ..
cd chasing
chmod 640 kitty
chmod 640 gopher
cd ..
cd marking
chmod 640 post
chmod 640 tree
chmod 640 bush
cd ..
cd sleeping
chmod 640 blanket
cd

```

*Both ways
work, the
choice is yours!*

The "Linux guru" method:
chmod 640 jobs/*/*

Permissions Review

File Permissions

Binary

Permissions are stored internally using binary numbers and they can be specified using decimal numbers

rwX	Binary	Convert	Decimal
- - -	0 0 0	0 + 0 + 0	0
- - x	0 0 1	0 + 0 + 1	1
- w -	0 1 0	0 + 2 + 0	2
- w x	0 1 1	0 + 2 + 1	3
r - -	1 0 0	4 + 0 + 0	4
r - x	1 0 1	4 + 0 + 1	5
r w -	1 1 0	4 + 2 + 0	6
r w x	1 1 1	4 + 2 + 1	7

r (read) is the 4's column

w (write) is the 2's column

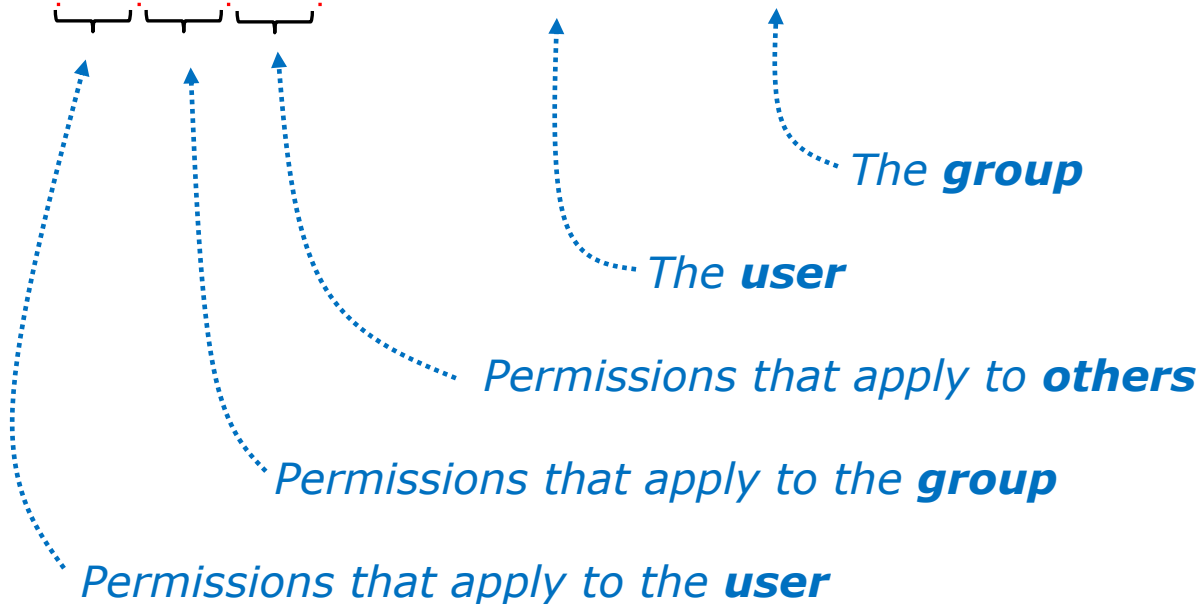
x (execute) is the 1's column

File Permissions

An example long listing

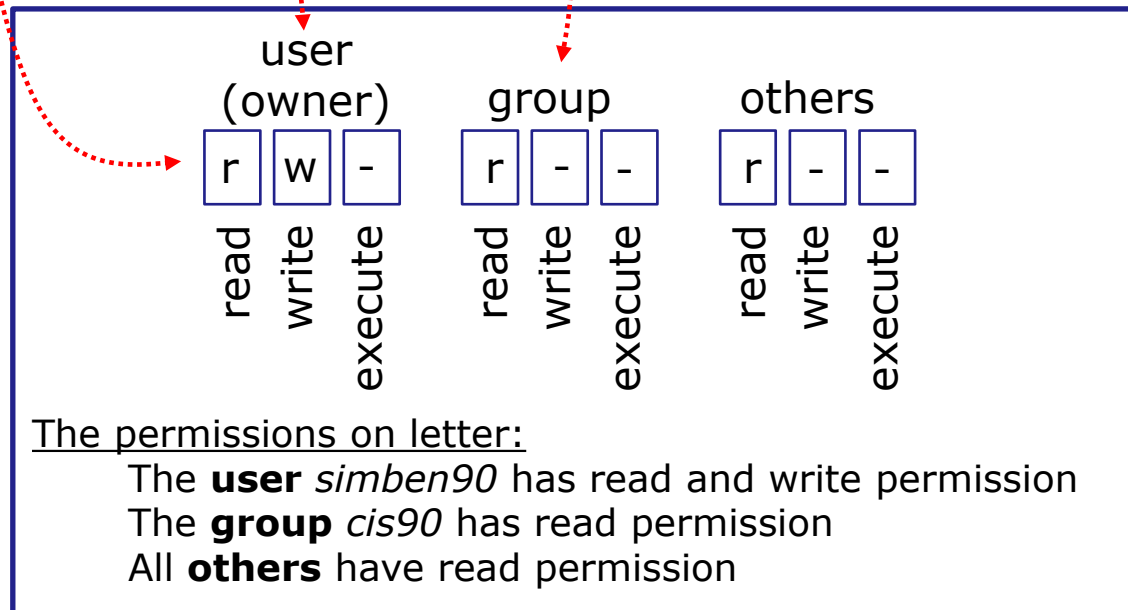
r=read
w=write
x=execute
-=none

```
/home/cis90/simben $ ls -l letter
-rw-r--r-- 1 simben90 cis90 1044 Oct 14 20:39 letter
```



File Permissions

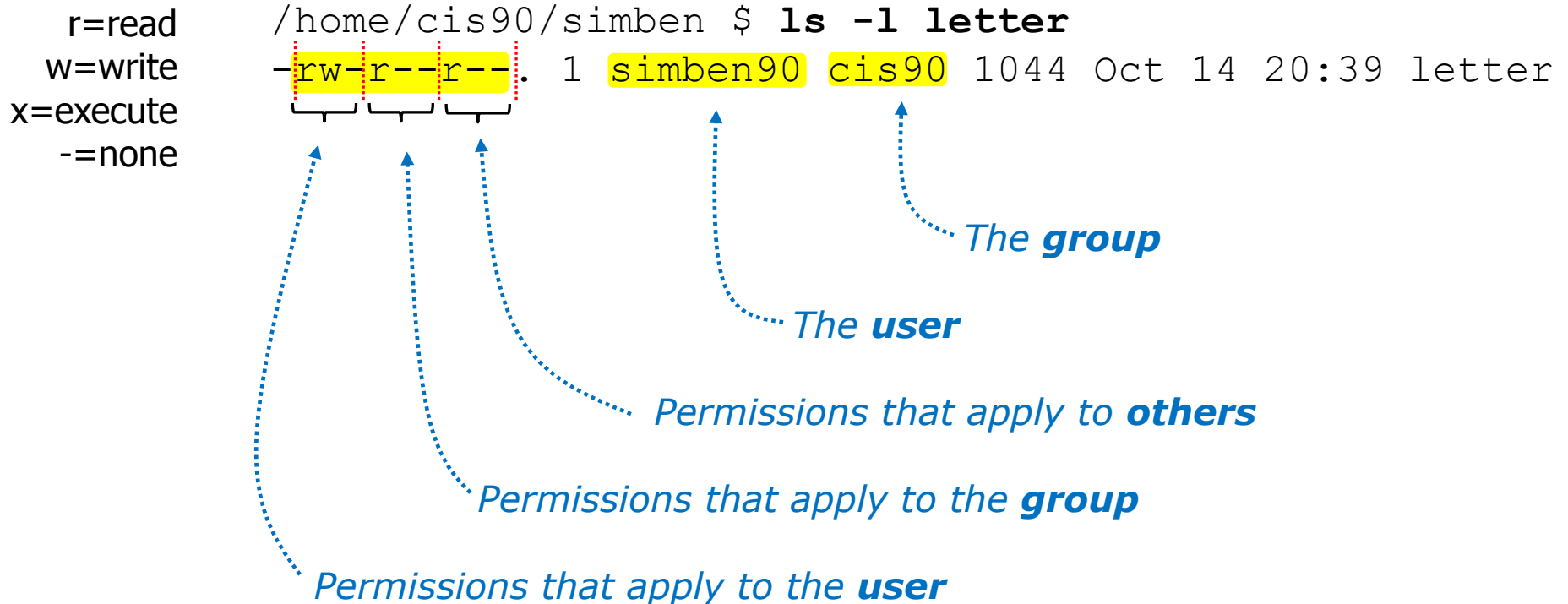
```
/home/cis90/simben $ ls -l letter
-rw-r--r--. 1 simben90 cis90 1044 Oct 14 20:39 letter
```



Use long listings to show permissions

File Permissions

Use long listings to show permissions



Does the simben90 user have execute permission on the letter file?

Type answer in chat window

File Permissions

Use long listings to show permissions

r=read
 w=write
 x=execute
 -=none

```

/home/cis90/simben $ ls -l letter
-rw-r--r-- 1 simben90 cis90 1044 Oct 14 20:39 letter
  
```

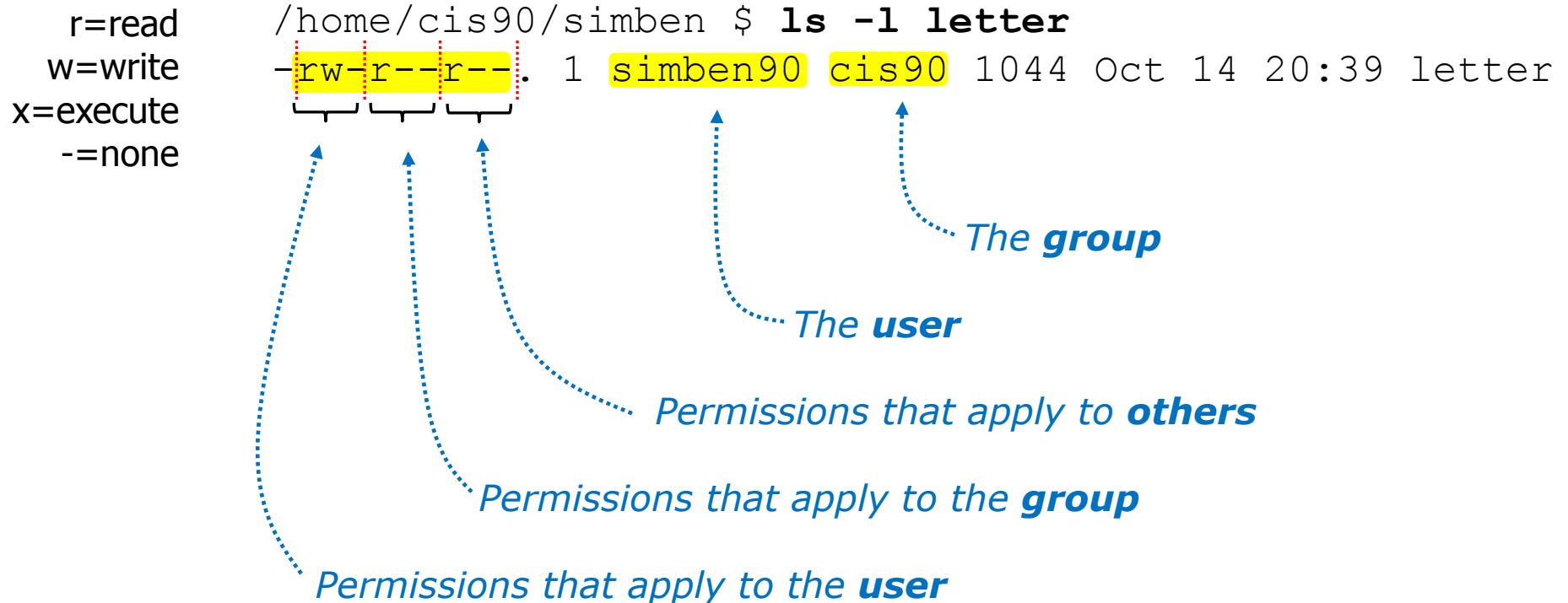
*Permissions that apply to the **user***
*Permissions that apply to the **group***
*Permissions that apply to **others***
*The **user***
*The **group***

Does the simben90 user have execute permission on the letter file?

No

File Permissions

Use long listings to show permissions



Does the zamhum90 user have write permission on the letter file?

Type answer in chat window

File Permissions

Use long listings to show permissions

r=read
 w=write
 x=execute
 -=none

```

/home/cis90/simben $ ls -l letter
-rw-r--r-- 1 simben90 cis90 1044 Oct 14 20:39 letter
  
```

The permissions `-rw-r--r--` are broken down as follows:

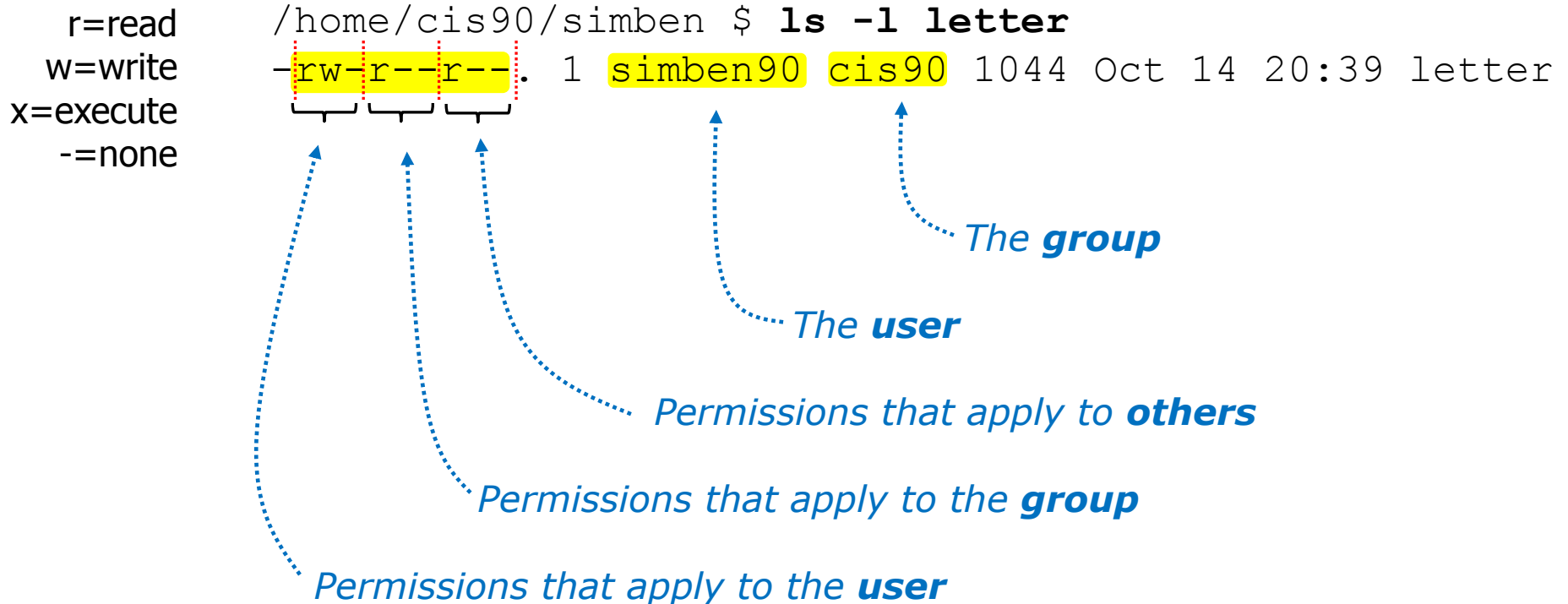
- `-`: File type (regular file)
- `rw`: Permissions for the **user** (simben90)
- `r--`: Permissions for the **group** (cis90)
- `r--`: Permissions for **others**

The **user** (simben90) has read and write permissions.
 The **group** (cis90) has read permission.
Others have read permission.

Does the zamhum90 user have write permission on the letter file?
No

File Permissions

Use long listings to show permissions

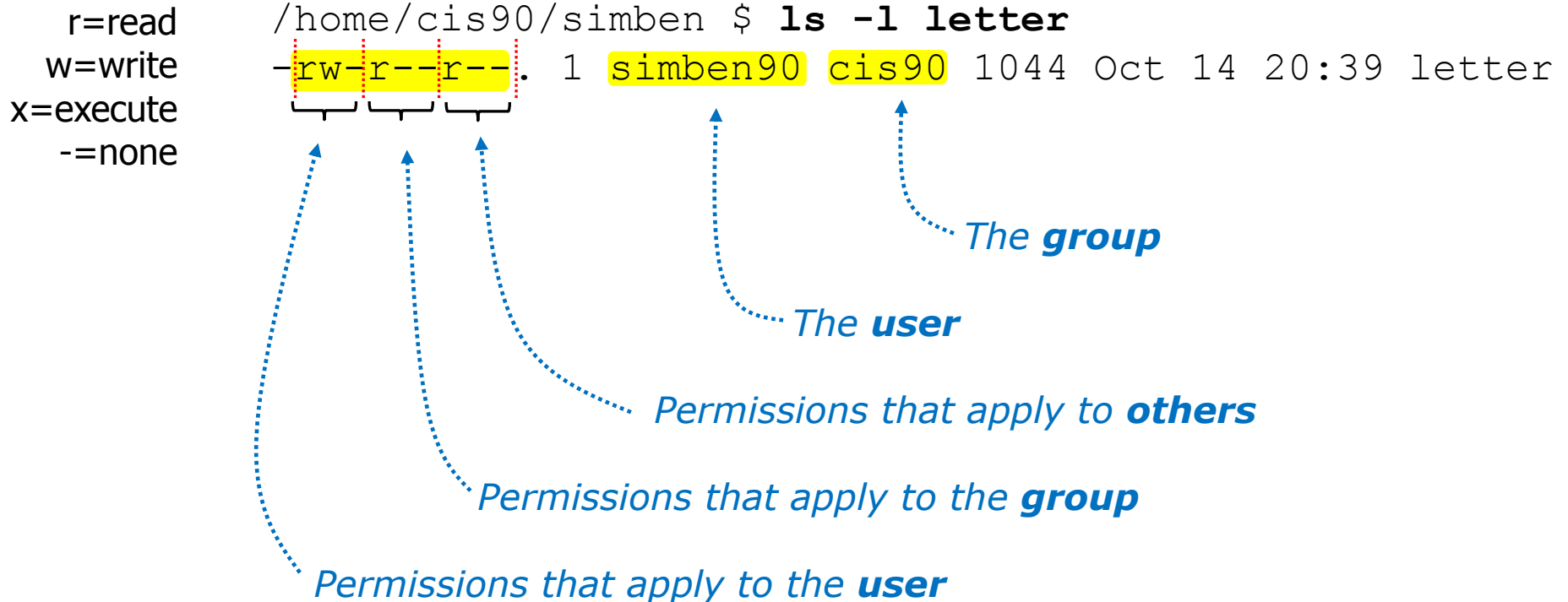


Does the zamhum90 user have read permission on the letter file?

Type answer in chat window

File Permissions

Use long listings to show permissions

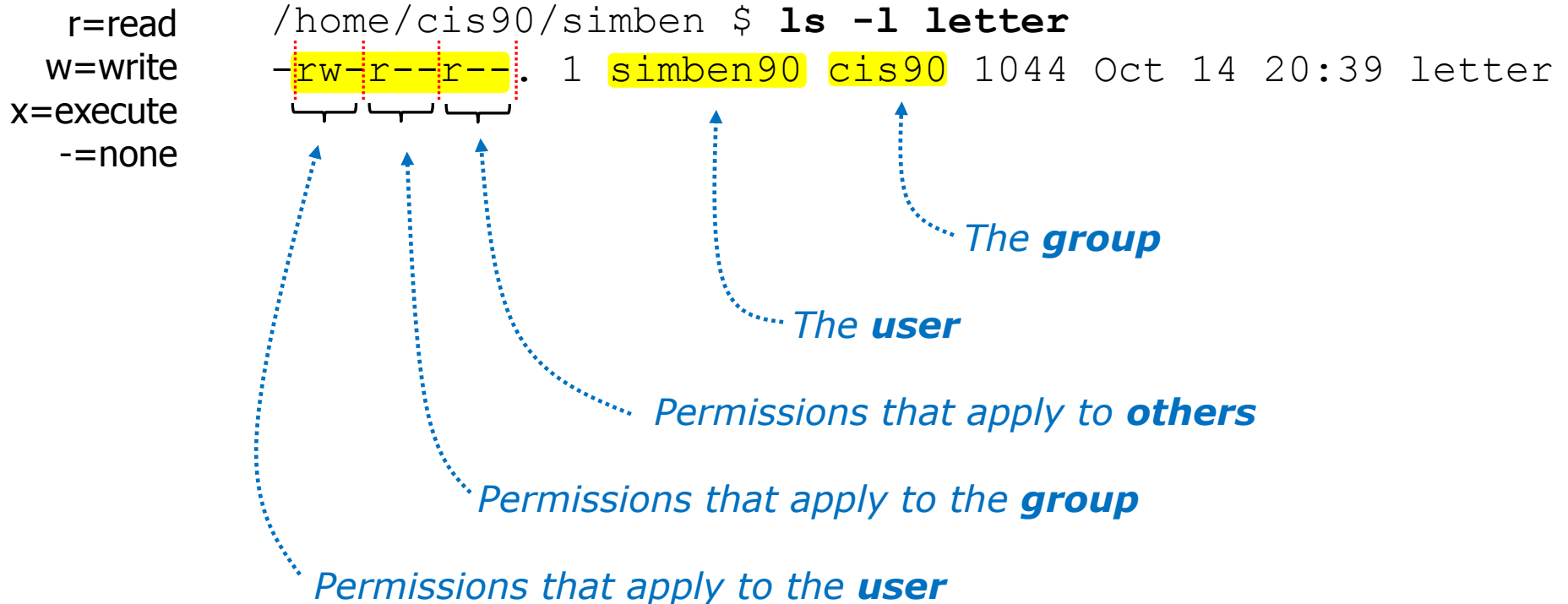


Does the zamhum90 user have read permission on the letter file?

Yes

File Permissions

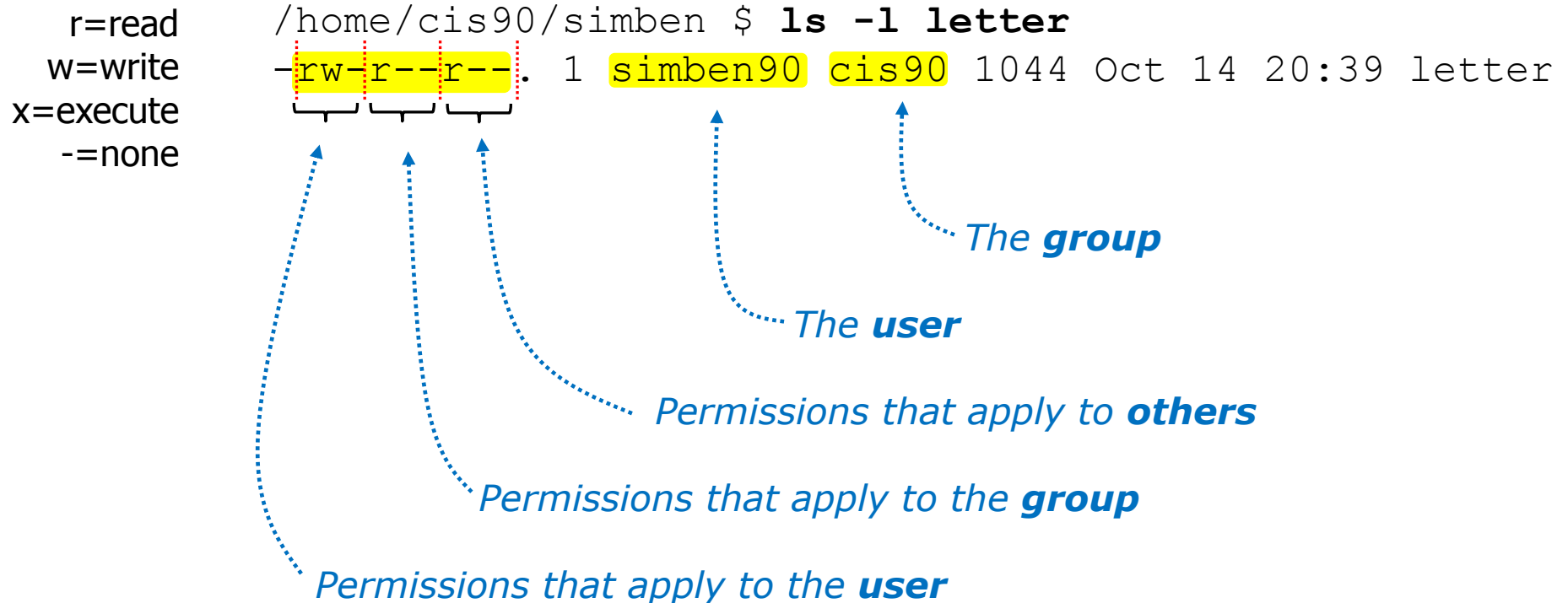
Use long listings to show permissions



Does the smimat172 user have read permission on the letter file?
Type answer in chat window

File Permissions

Use long listings to show permissions



Does the smimat172 user have read permission on the letter file?

Yes



Tools for managing permissions

chown - Changes the ownership of a file. (Only the superuser has this privilege)

chgrp - Changes the group of a file. (Only to groups that you belong to)

chmod - Changes the file mode “permission” bits of a file.

- Numeric: **chmod 640 letter** (sets the permissions)
- Mnemonic: **chmod ug+rw letter** (changes the permissions)
u=user(owner), **g**=group, **o**=other
r=read, **w**=write, **x**=execute

umask – Allows specific permissions to be removed on future newly created files and directories



Tools for managing permissions

chown

- Changes the ownership of a file. (Only the superuser has this privilege)
- Syntax: **chown** <owner> <pathname>

```
/home/cis90/simben $ ls -l letter  
-rw-r--r--. 1 simben90 cis90 1044 Oct 14 20:39 letter
```

```
/home/cis90/simben $ chown rsimms letter  
chown: changing ownership of `letter': Operation not permitted
```

Only root (superuser) can change the ownership of a file



Tools for managing permissions

chgrp

- Changes the group of a file. (Only to groups the owner belongs to)
- Syntax: **chgrp <group> <pathname>**

```
/home/cis90/simben $ ls -l letter
-rw-r--r--. 1 simben90 cis90 1044 Oct 14 20:39 letter
```

```
/home/cis90/simben $ groups
cis90 users
```

```
/home/cis90/simben $ chgrp users letter
```

```
/home/cis90/simben $ ls -l letter
-rw-r--r--. 1 simben90 users 1044 Oct 14 20:39 letter
```

The owner can change the group to any he/she belongs to



Tools for managing permissions

chmod

- Changes the file mode "permission" bits of a file
- "Numeric" syntax: **chmod <numeric permission> <pathname>**

```
/home/cis90/simben $ ls -l letter
-rw-r--r--. 1 simben90 cis90 1044 Oct 14 20:39 letter
```

```
/home/cis90/simben $ chmod 750 letter
/home/cis90/simben $ ls -l letter
-rwxr-x---. 1 simben90 cis90 1044 Oct 14 20:39 letter
```

```
/home/cis90/simben $ chmod 644 letter
/home/cis90/simben $ ls -l letter
-rw-r--r--. 1 simben90 cis90 1044 Oct 14 20:39 letter
```




Tools for managing permissions

chmod

- Changes the file mode "permission" bits of a file.
- "Mnemonic" syntax: **chmod <u|g|o><+|-|=><r|w|x> <pathname(s)>**
u=user(owner), **g**=group, **o**=other
r=read, **w**=write, **x**=execute

```
/home/cis90/simben $ ls -l letter
-rw-r--r--. 1 simben90 cis90 1044 Oct 14 20:39 letter
```

```
/home/cis90/simben $ chmod u+x,g+w,o-r letter
/home/cis90/simben $ ls -l letter
-rwxrw----. 1 simben90 cis90 1044 Oct 14 20:39 letter
```

```
/home/cis90/simben $ chmod u=rw,g=r,o=r letter
/home/cis90/simben $ ls -l letter
-rw-r--r--. 1 simben90 cis90 1044 Oct 14 20:39 letter
```



Tools for managing permissions

umask – Allows specific permissions to be removed on future newly created files and directories

sort command

```
/home/cis90/simben $ cat misc/salad
```

```
orange  
mango  
banana  
peach  
apple  
grapes  
pear  
apricot  
kiwi  
watermelon  
pineapple
```

*Try the sort command on the
salad file in your misc/
directory*

```
/home/cis90/simben $ sort misc/salad
```

```
apple  
apricot  
banana  
grapes  
kiwi  
mango  
orange  
peach  
pear  
pineapple  
watermelon
```

Pipeline example

```
[simben@opus ~]$ who | sort | tee users | wc -l  
4
```

```
[simben@opus ~]$ cat users  
bolasale pts/4      2008-10-21 10:43 (dsl-63-249-97-17.cruzio.com)  
simben pts/0       2008-10-19 18:36 (dsl-63-249-103-107.cruzio.com)  
simben pts/1       2008-10-19 18:27 (dsl-63-249-103-107.cruzio.com)  
rsimms  pts/2       2008-10-20 17:33 (dsl-63-249-103-107.cruzio.com)
```

Counting, sorting and recording the currently logged in users

Why pipelines?

Without pipelines we would have to save the results of each intermediate step in a temporary file

```
[simben@opus ~]$ who
simben pts/0          2008-10-19 18:36 (dsl-63-249-103-107.cruzio.com)
simben pts/1          2008-10-19 18:27 (dsl-63-249-103-107.cruzio.com)
rsimms pts/2          2008-10-20 17:33 (dsl-63-249-103-107.cruzio.com)
bolasale pts/4        2008-10-21 10:43 (dsl-63-249-97-17.cruzio.com)
[simben@opus ~]$ who > tempfile
[simben@opus ~]$ sort tempfile
bolasale pts/4        2008-10-21 10:43 (dsl-63-249-97-17.cruzio.com)
simben pts/0          2008-10-19 18:36 (dsl-63-249-103-107.cruzio.com)
simben pts/1          2008-10-19 18:27 (dsl-63-249-103-107.cruzio.com)
rsimms pts/2          2008-10-20 17:33 (dsl-63-249-103-107.cruzio.com)
[simben@opus ~]$ sort tempfile > users
[simben@opus ~]$ wc -l users
4 users
[simben@opus ~]$ cat users
bolasale pts/4        2008-10-21 10:43 (dsl-63-249-97-17.cruzio.com)
simben pts/0          2008-10-19 18:36 (dsl-63-249-103-107.cruzio.com)
simben pts/1          2008-10-19 18:27 (dsl-63-249-103-107.cruzio.com)
rsimms pts/2          2008-10-20 17:33 (dsl-63-249-103-107.cruzio.com)
```

Best practices: build pipelines one command at a time so you can see what you are doing

```
[simben@opus ~]$ who      who is logged in
simben pts/0      2008-10-19 18:36 (dsl-63-249-103-107.cruzio.com)
simben pts/1      2008-10-19 18:27 (dsl-63-249-103-107.cruzio.com)
rsimms  pts/2      2008-10-20 17:33 (dsl-63-249-103-107.cruzio.com)
bolasale pts/4      2008-10-21 10:43 (dsl-63-249-97-17.cruzio.com)
```

```
[simben@opus ~]$ who | sort    who is logged in and sorted
bolasale pts/4      2008-10-21 10:43 (dsl-63-249-97-17.cruzio.com)
simben pts/0      2008-10-19 18:36 (dsl-63-249-103-107.cruzio.com)
simben pts/1      2008-10-19 18:27 (dsl-63-249-103-107.cruzio.com)
rsimms  pts/2      2008-10-20 17:33 (dsl-63-249-103-107.cruzio.com)
```

```
[simben@opus ~]$ who | sort | wc -l    who is logged in, sorted and counted
4
```

```
[simben@opus ~]$ who | sort | tee users | wc -l    who is logged in, sorted, counted
4                                                    and saved in file named users
```

```
[simben@opus ~]$ cat users
bolasale pts/4      2008-10-21 10:43 (dsl-63-249-97-17.cruzio.com)
simben pts/0      2008-10-19 18:36 (dsl-63-249-103-107.cruzio.com)
simben pts/1      2008-10-19 18:27 (dsl-63-249-103-107.cruzio.com)
rsimms  pts/2      2008-10-20 17:33 (dsl-63-249-103-107.cruzio.com)
```