Rich's lesson module checklist

Last updated 10/19/2017

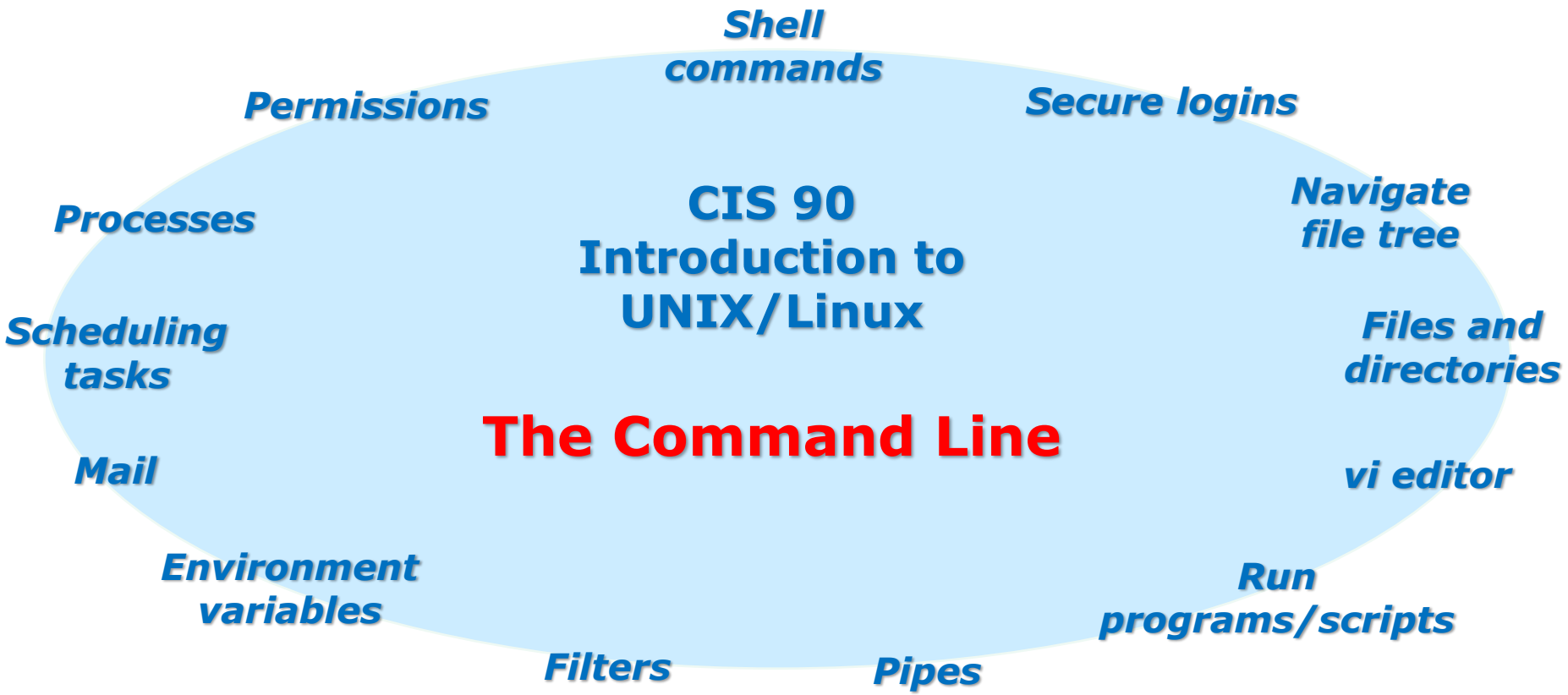
- ❑ Slides and labs 7 and X2 posted
- ❑ WB converted from PowerPoint
- ❑ Print out agenda slide and annotate page numbers

- ❑ Flash cards
- ❑ Page numbers
- ❑ 1st minute quiz
- ❑ Web Calendar summary
- ❑ Web book pages
- ❑ Commands

- ❑ Schedule lock of turnin directory and submit
  scripts/schedule-submit-locks
- ❑ Lab 7 tested
- ❑ Lab X2 updated with kernels and tested
- ❑ checkx2 updated

- ❑ 9V backup battery for microphone
- ❑ Backup slides, CCC info, handouts on flash drive
- ❑ Key card for classroom door

- ❑ Update CCC Confer and 3C Media portals



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.

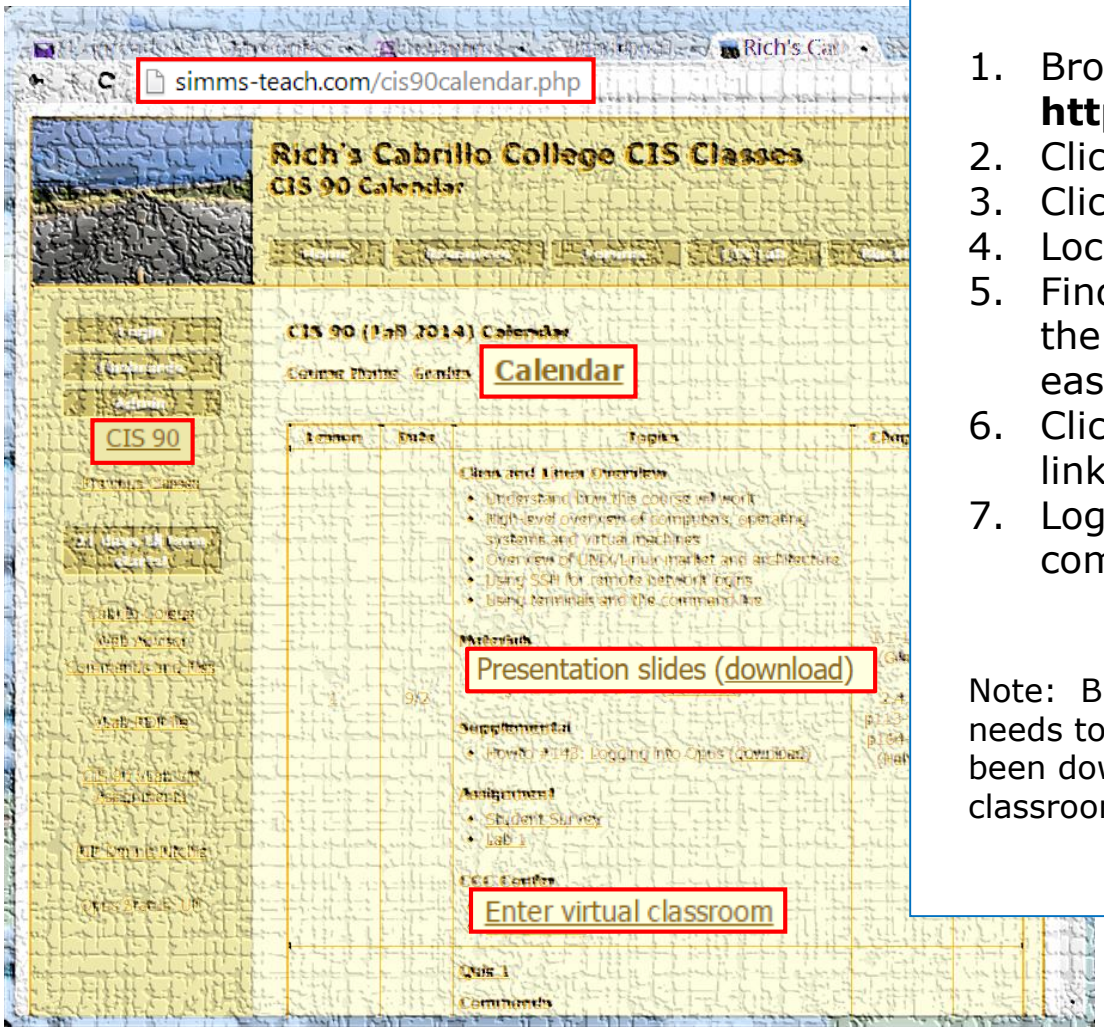
Introductions and Credits

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 for attending class

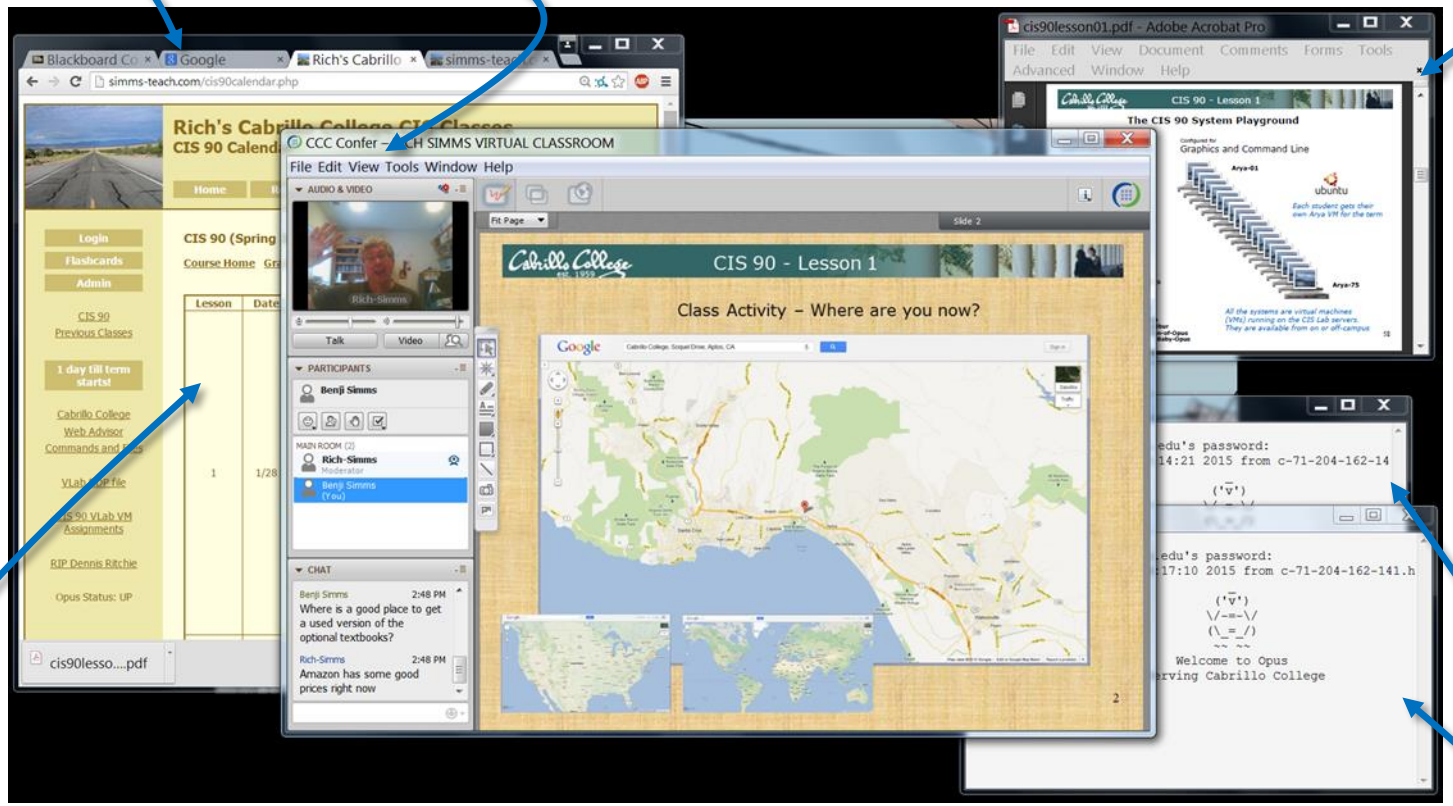


1. Browse to: **http://simms-teach.com**
2. Click the **CIS 90** link.
3. Click the **Calendar** link.
4. Locate today's lesson.
5. Find the **Presentation slides** for the lesson and **download** for easier viewing.
6. Click the **Enter virtual classroom** link to join CCC Confer.
7. Log into Opus-II with Putty or ssh command.

Note: Blackboard Collaborate Launcher only needs to be installed once. It has already been downloaded and installed on the classroom PC's.

Student checklist for suggested screen layout

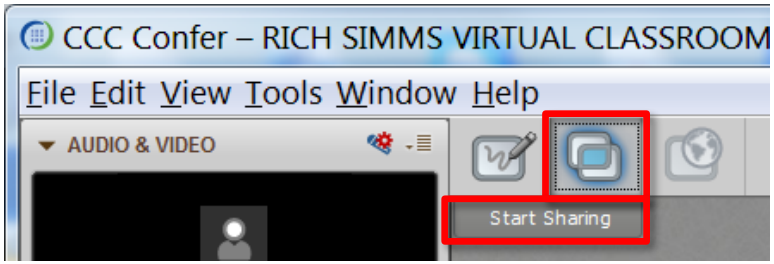
- ❑ *Google*
- ❑ *CCC Confer*
- ❑ *Downloaded PDF of Lesson Slides*



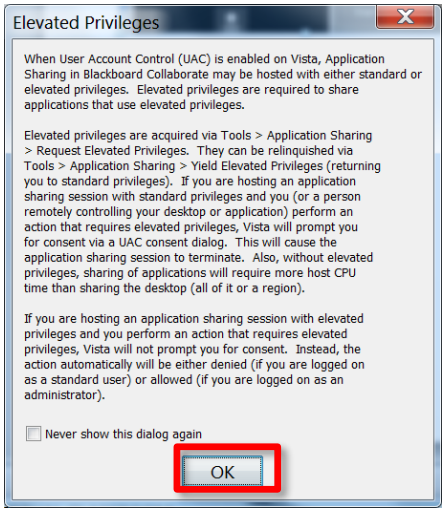
- ❑ *CIS 90 website Calendar page*
- ❑ *One or more login sessions to Opus*

Student checklist for sharing desktop with classmates

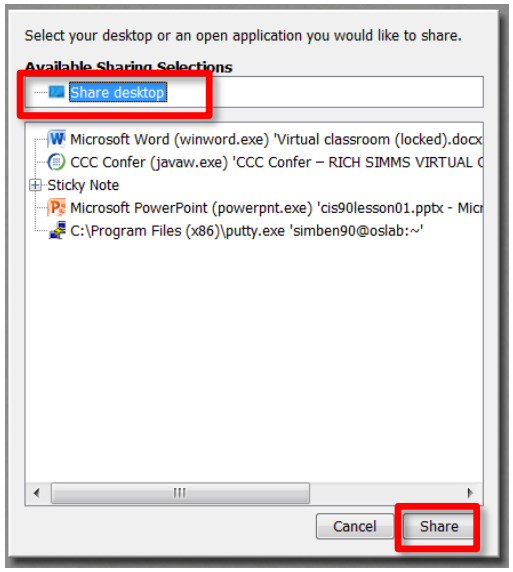
1) Instructor gives you sharing privileges



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



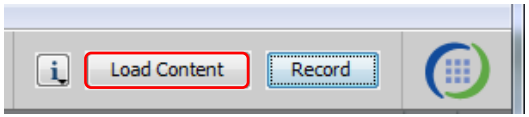
3) Click OK button.



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

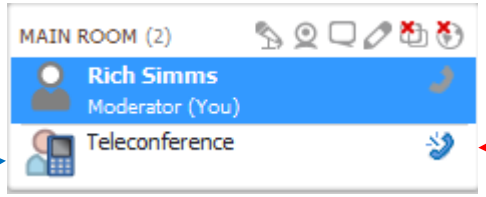
Rich's CCC Confer checklist - setup

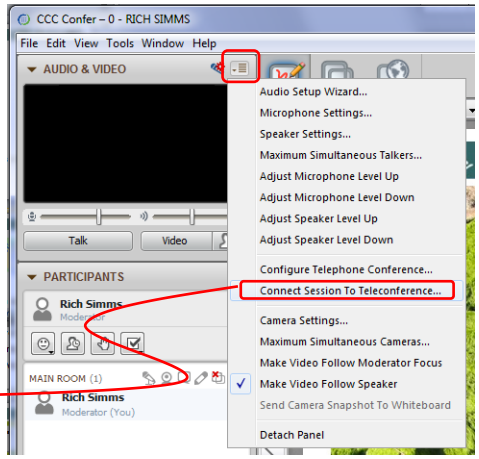
[ ] Preload White Board



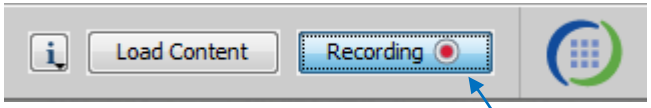
[ ] Connect session to Teleconference

Session now connected to teleconference





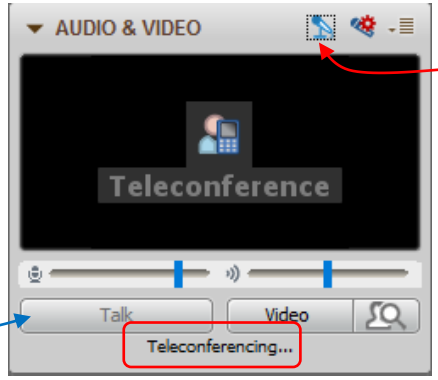
[ ] Is recording on?



Red dot means recording

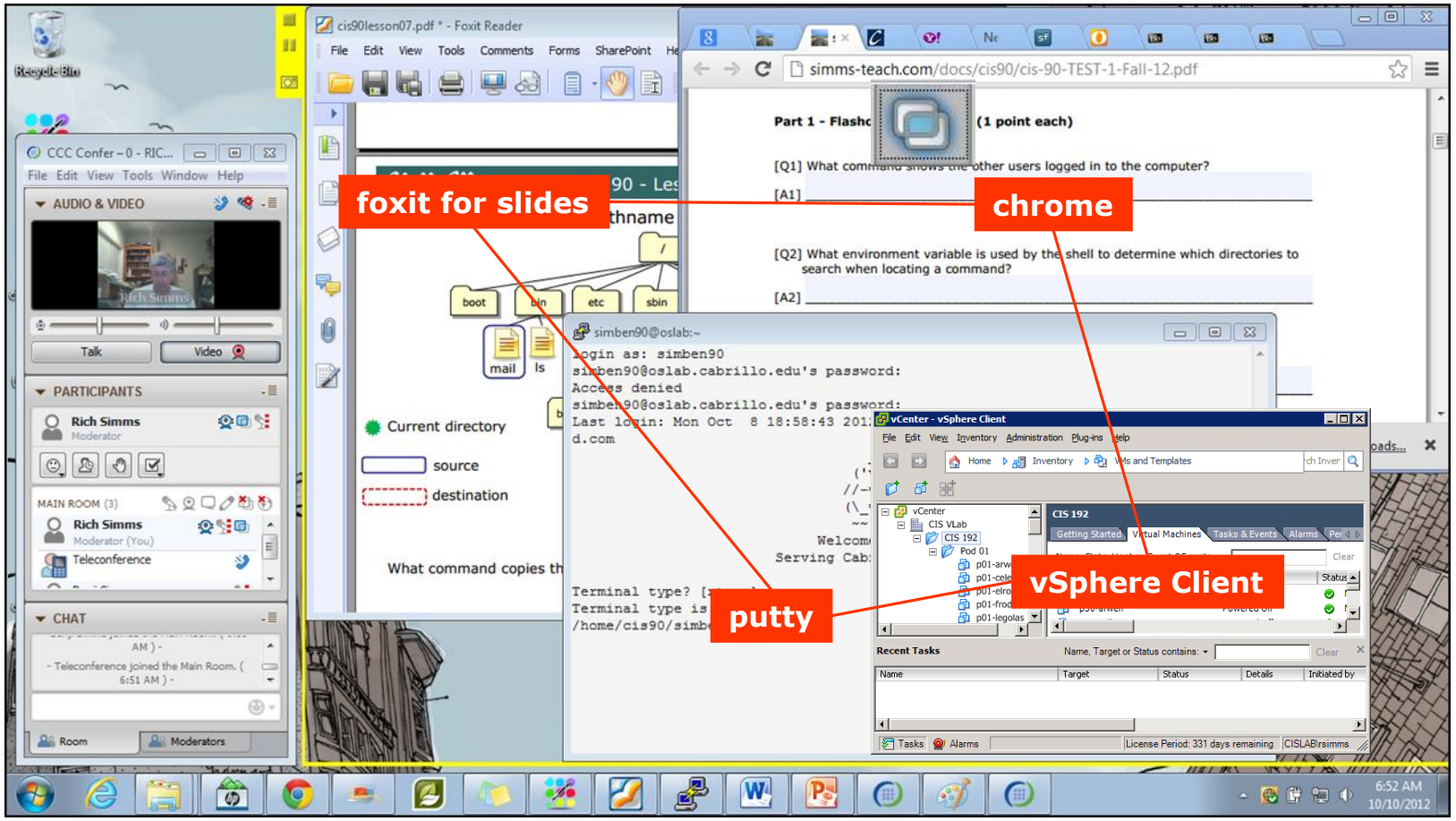
[ ] Use teleconferencing, not mic

Should be grayed out



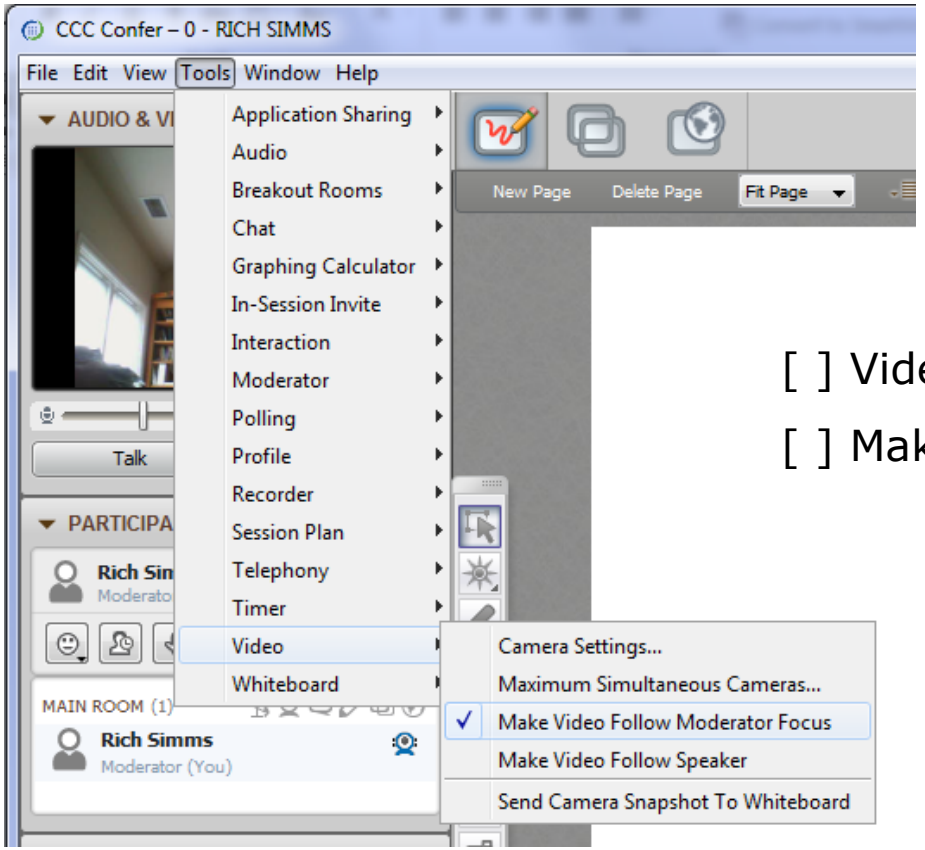
Should change from phone handset icon to little Microphone icon and the Teleconferencing … message displayed

Rich's CCC Confer checklist - screen layout



[ ] layout and share apps

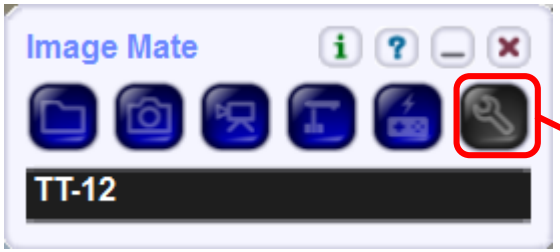
Rich's CCC Confer checklist - webcam setup

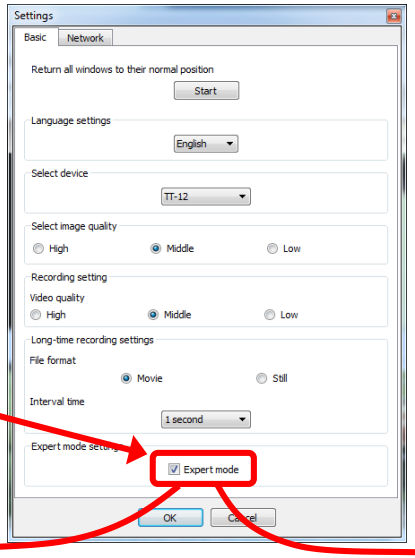


[ ] Video (webcam)

[ ] Make Video Follow Moderator Focus

Rich's CCC Confer checklist - Elmo



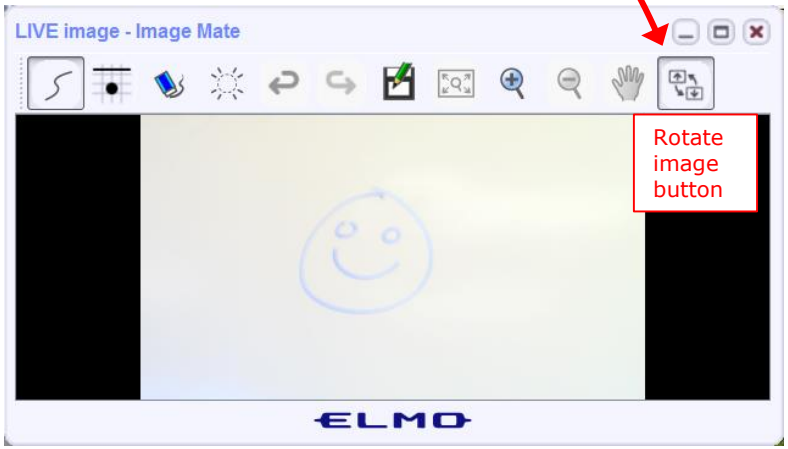


The "rotate image" button is necessary if you use both the side table and the white board.

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

Elmo rotated down to view side table

Elmo rotated up to view white board



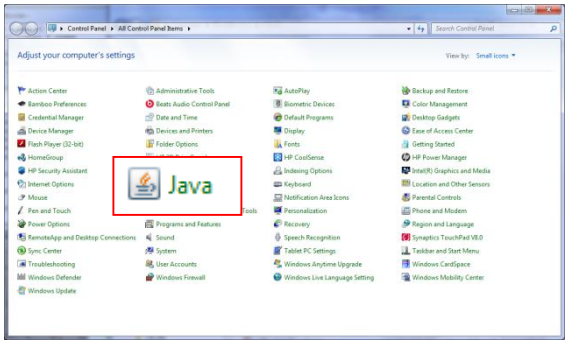
Run and share the Image Mate program just as you would any other app with CCC Confer

Rich's CCC Confer checklist - universal fixes

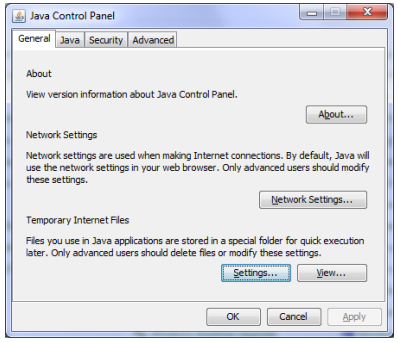
Universal Fix for CCC Confer:

1) Shrink (500 MB) and delete Java cache
2) Uninstall and reinstall latest Java runtime
3) http://www.cccconfer.org/support/technicalSupport.aspx

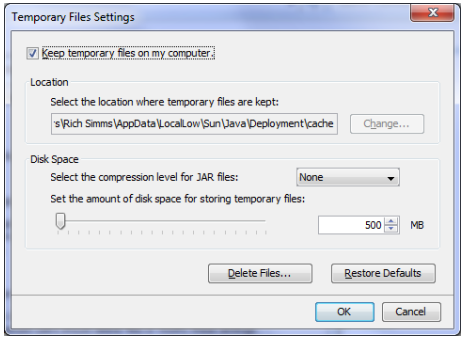
Control Panel (small icons)



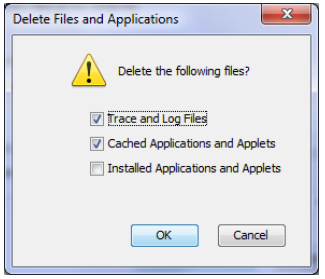
General Tab > Settings…



500MB cache size

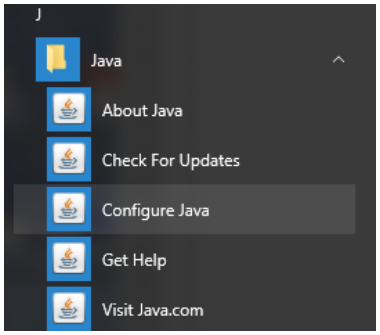


Delete these

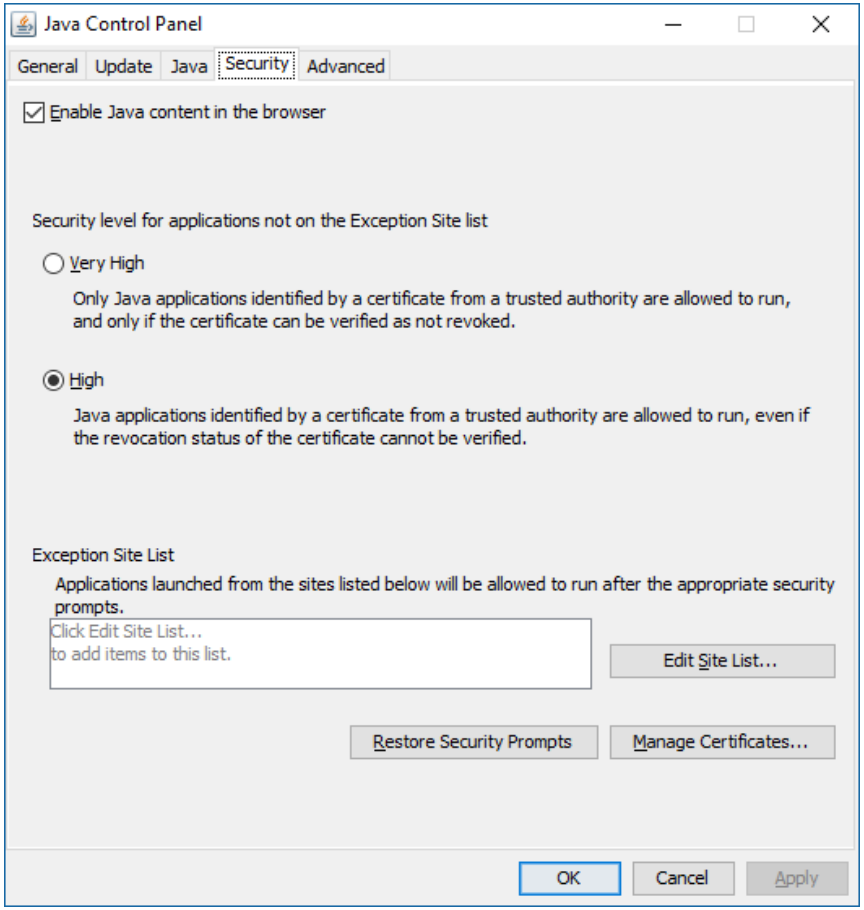


Google Java download

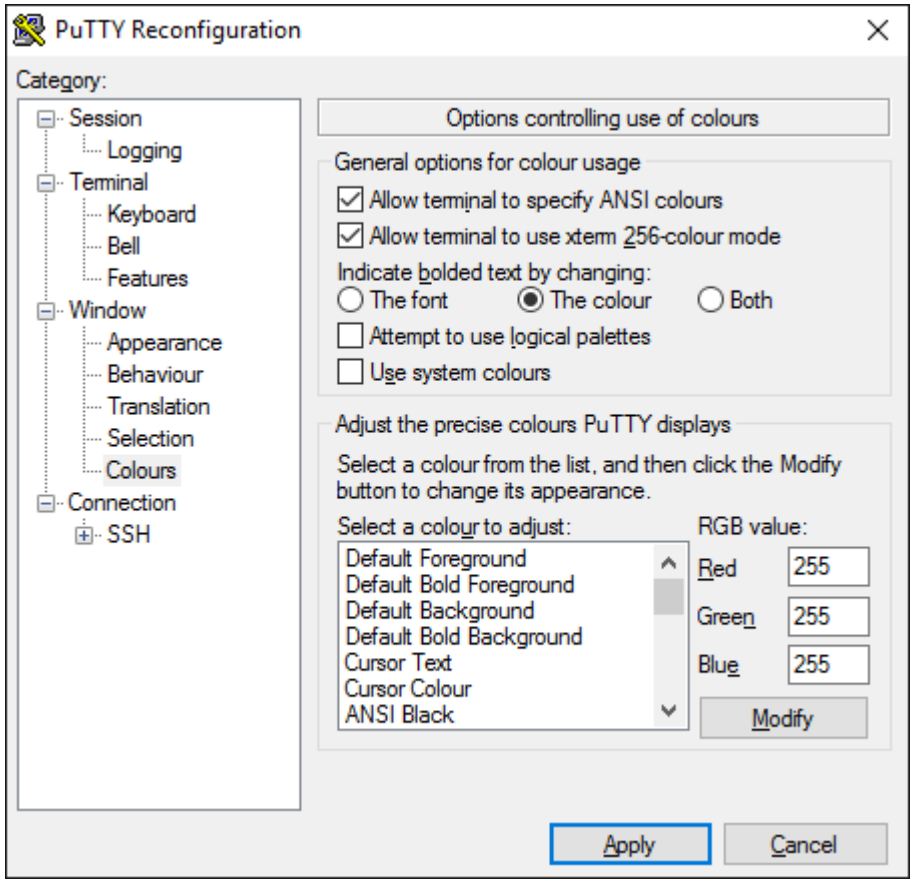
Rich's CCC Confer checklist - digital certificate work around



1. Open the Java Control Panel
2. Select the **Security** tab
3. Select **Edit Site List...**
4. Select **Add**
5. Click into the white box next to the red exclamation mark and type **https://na-downloads.elluminate.com**
6. Press OK
7. Press **Continue** on the pop-up message
8. Press OK
9. Access your session or recording once more



Rich's CCC Confer checklist - Putty Colors



http://looselytyped.blogspot.com/2013/02/zenburn-pleasant-color-scheme-for-putty.html

Putty Colors

Default Foreground 255 255 255
Default Bold Foreground 255 255 255
Default Background 51 51 51
Default Bold Background 255 2 85
Cursor Text 0 0 0
Cursor Color 0 255 0
ANSI Black 77 77 77
ANSI Black Bold 85 85 85
ANSI Red 187 0 0
ANSI Red Bold 255 85 85
ANSI Green 152 251 152
ANSI Green Bold 85 255 85
ANSI Yellow 240 230 140
ANSI Yellow Bold 255 255 85
ANSI Blue 205 133 63
ANSI Blue Bold 135 206 235
ANSI Magenta 255 222 173
ANSI Magenta Bold 255 85 255
ANSI Cyan 255 160 160
ANSI Cyan Bold 255 215 0
ANSI White 245 222 179
ANSI White Bold 255 255 255

Start

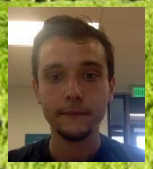
Sound Check

*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.*

Volume
**4 - increase conference volume.*
**7 - decrease conference volume.*
**5 - increase your voice volume.*
**8 - decrease your voice volume.*

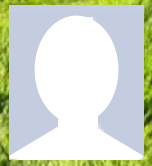
Instructor: **Rich Simms**
Dial-in: **888-886-3951**
Passcode: **136690**


Vinny

Marvin

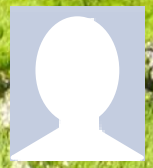
William

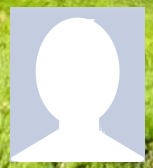
Dan C.


Hayden

Nick

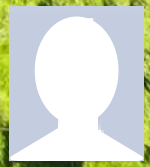
Ramon


Nicholas


Manuel

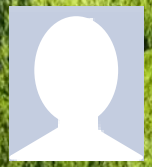
Damien

Adam


Oscar

Daniel P.

Jason

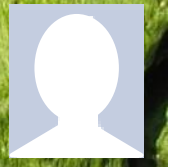

Brian

Vincent P.

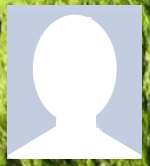
Kyle

Sam X.

Jacobs

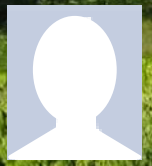

Tyler

Alejandro


Sean

Karina

Michael J.


Victor

Moises

Joseph

David

Emmanuel

Ben

Email me (risimms@cabrillo.edu) a relatively current photo of your face for 3 points extra credit

First Minute Quiz

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

Use CCC Confer 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
• 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: ❖ find ❖ grep ❖ wc ❖ sort ❖ spell	• Quiz • Questions • Warmup • umask continued • Housekeeping • New commands (sort) • Pretend you are a command (imagination) • Sort command deep dive (good arg, no args, bad arg) • Bringing it home (reality) • File redirection • The bit bucket • Pipelines • find command • Filter commands (grep, spell, tee, cut) • Pipeline practice • Permissions, the rest of the story • Assignment • Wrap up

Questions

Questions?

Lesson material?

Labs? Tests?

How this course works?

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

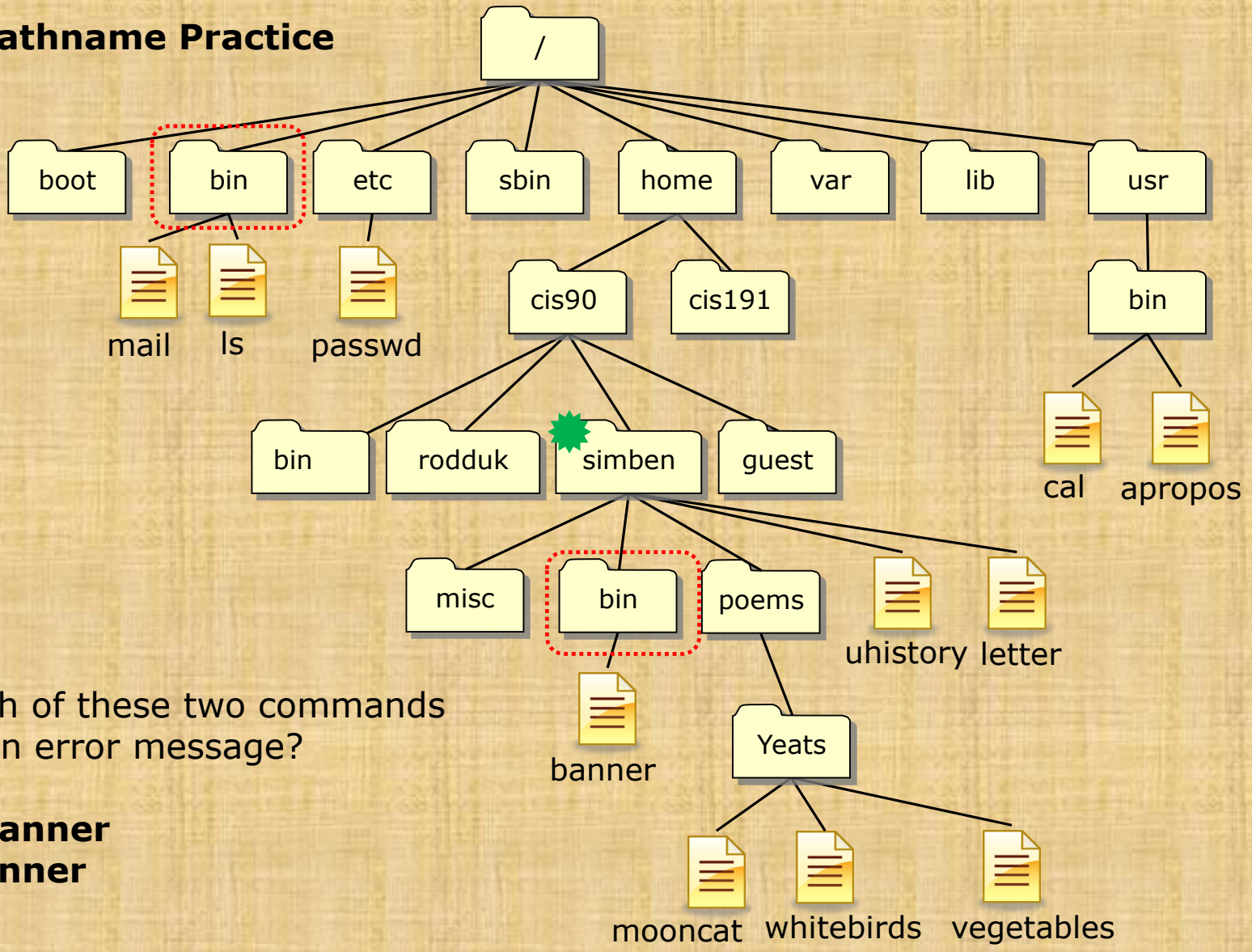
Who questions much, shall learn much, and retain much.
- Francis Bacon

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.*

Warmup

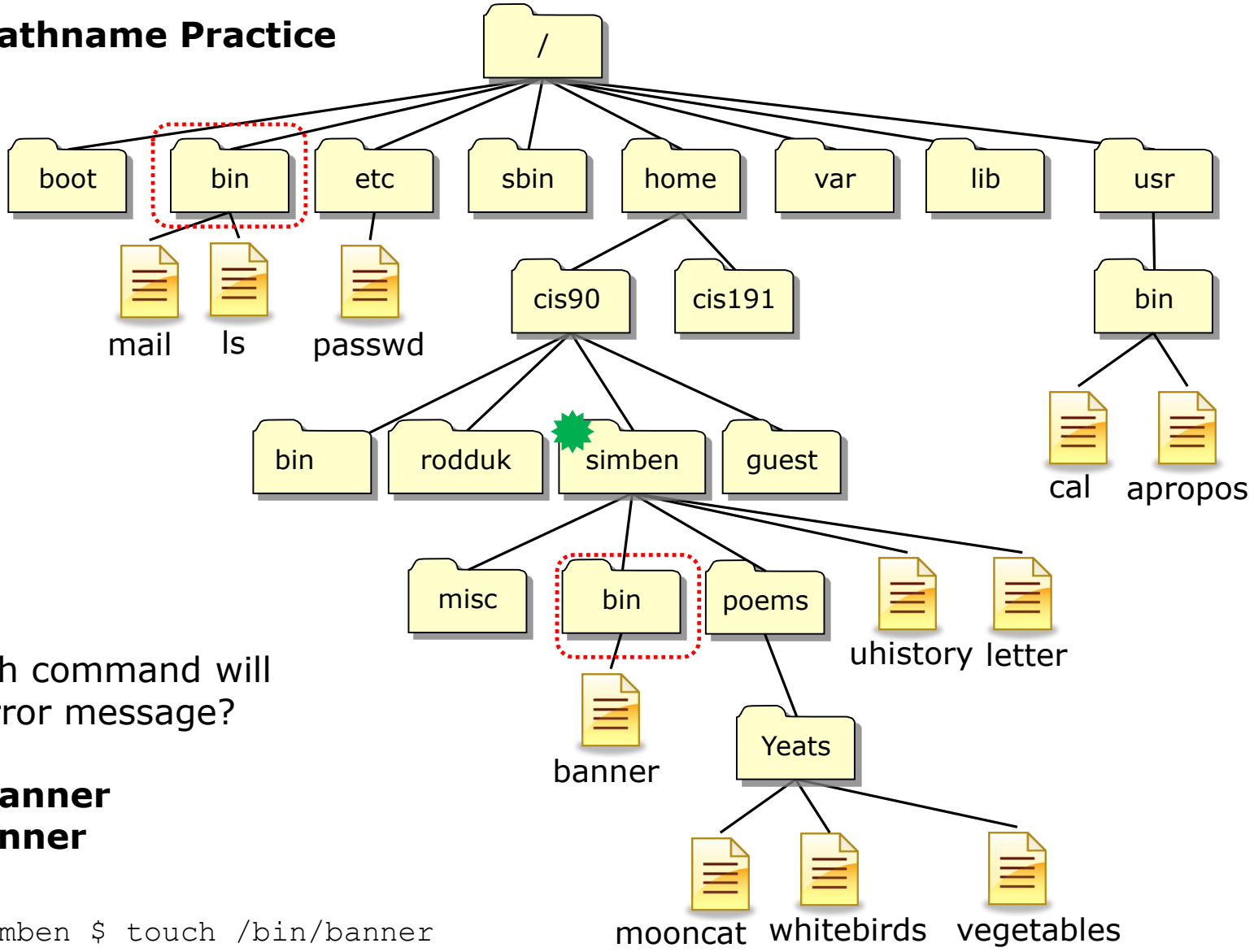
File Tree Pathname Practice



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

touch /bin/banner
touch bin/banner

File Tree Pathname Practice



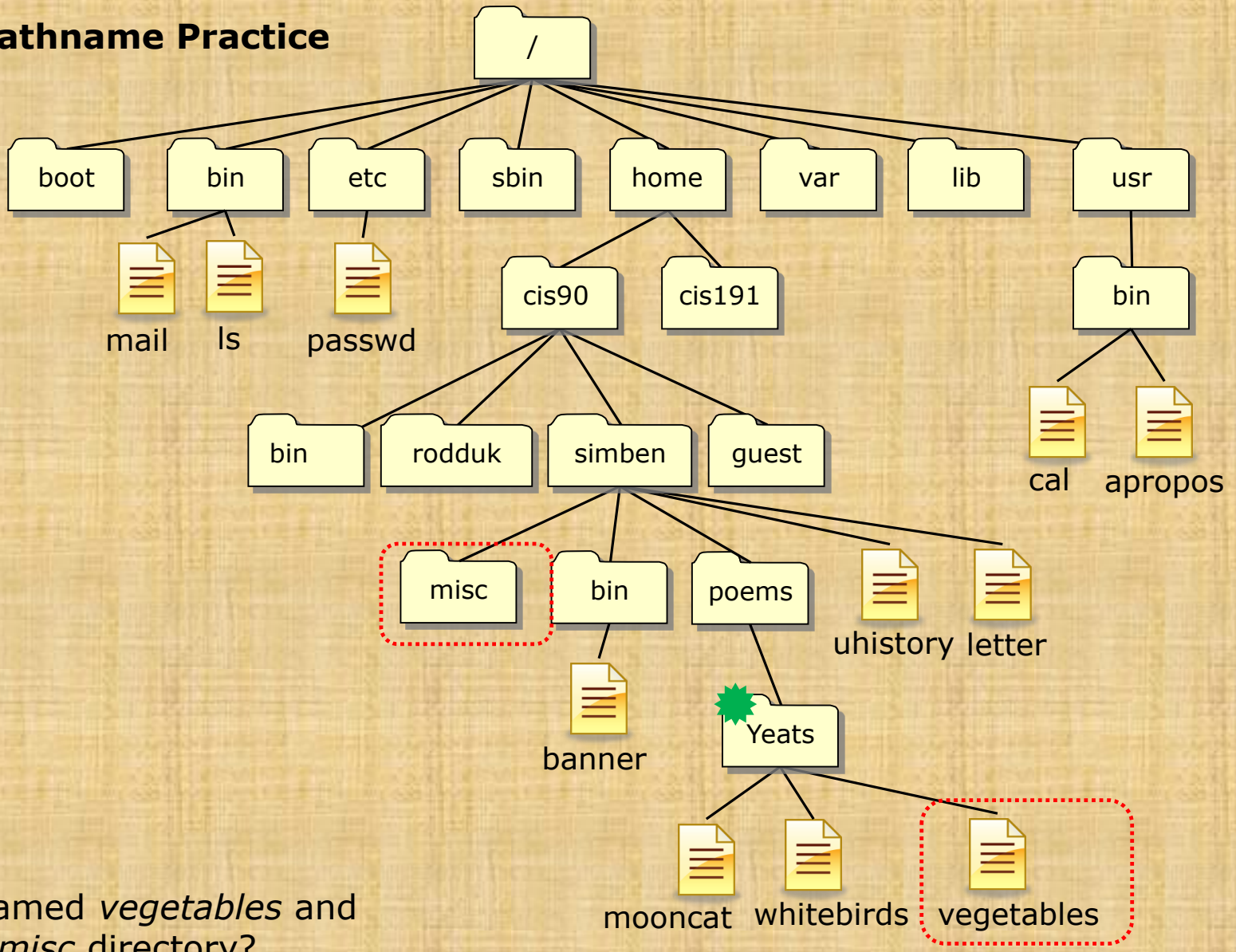
From ✹ which command will generate an error message?

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

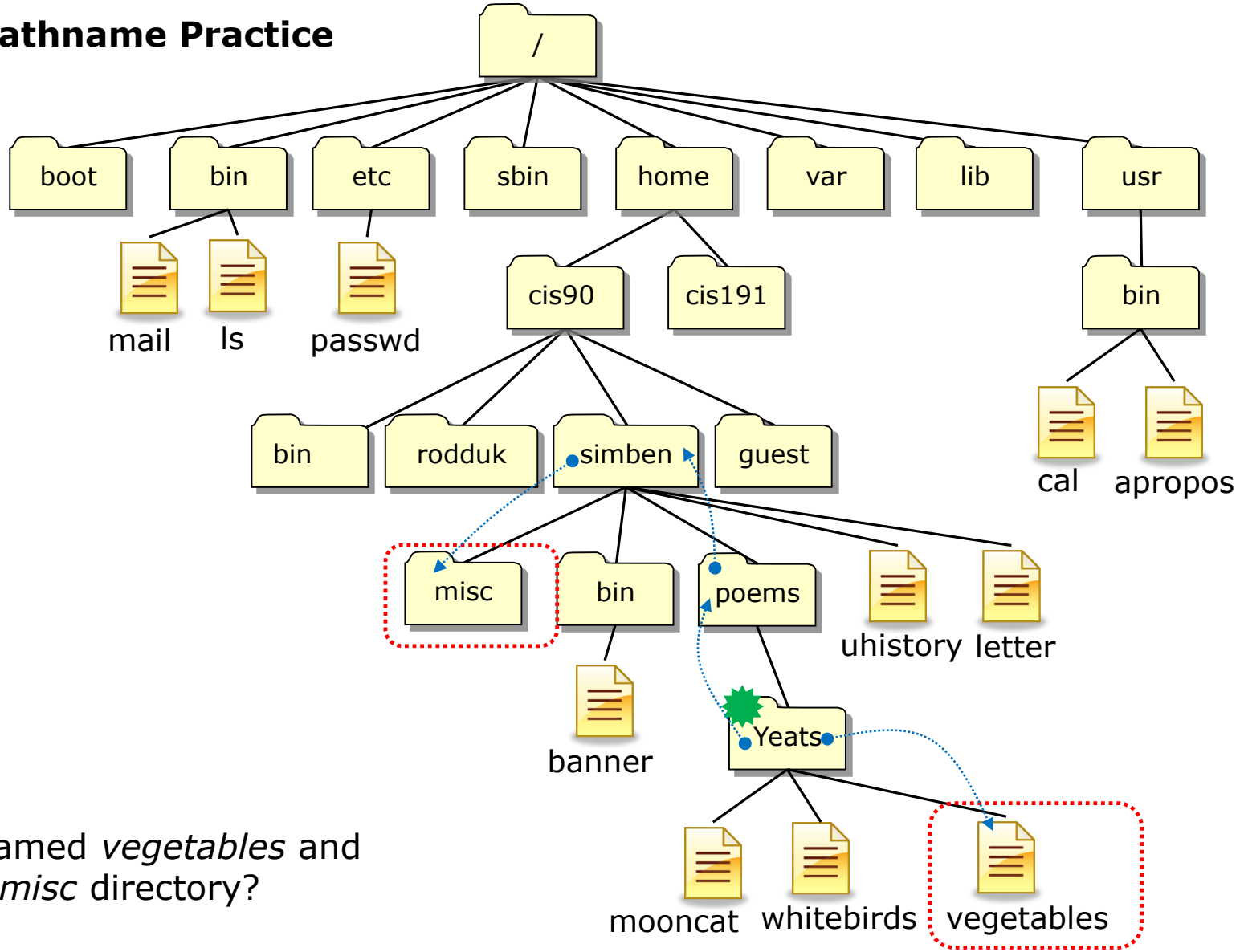
File Tree Pathname Practice



From how does Benji:

Create a file named *vegetables* and move it to his *misc* directory?

File Tree Pathname Practice



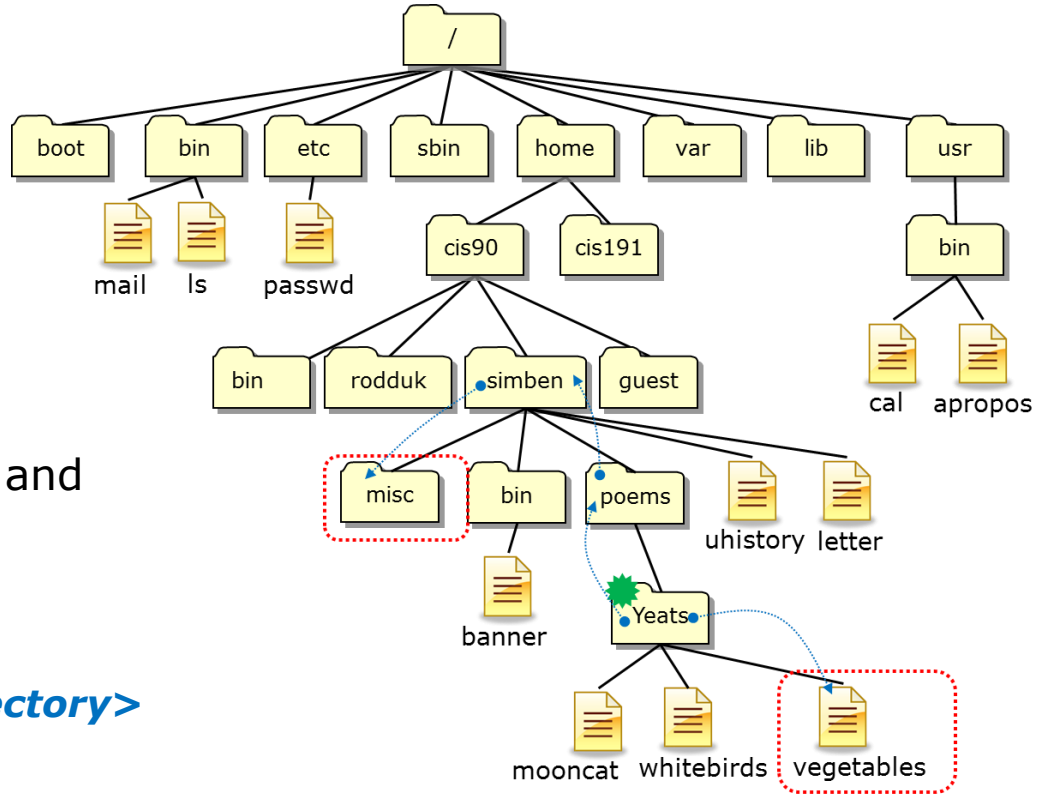
From how does Benji:

Create a file named *vegetables* and move it to his *misc* directory?

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

Other answers are also acceptable

From how does Benji:



Create a file named *vegetables* and move it to his *misc* directory?

touch vegetables

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

mv vegetables ../../misc/

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

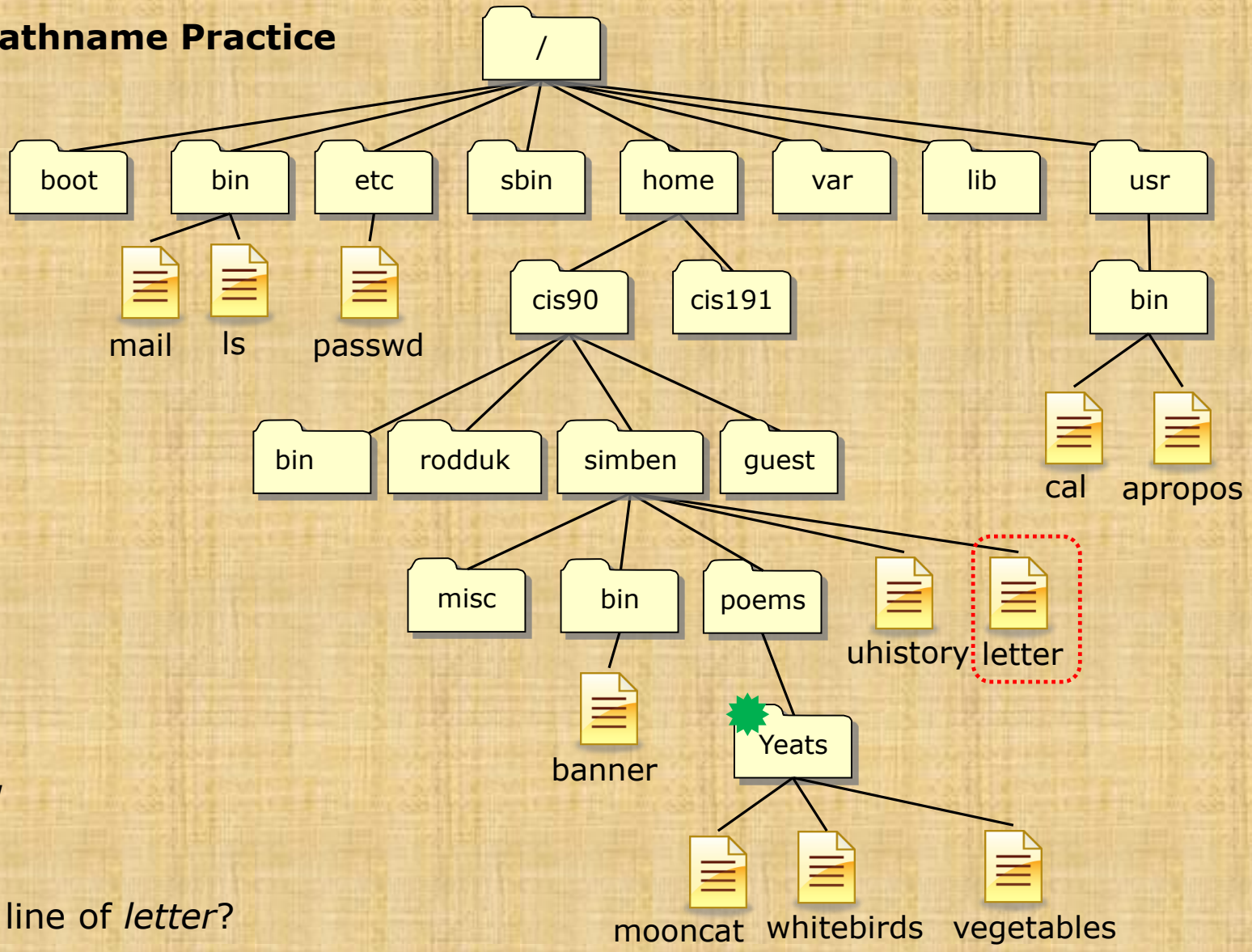
or **mv /home/cis90/simben/poems/Yeats/vegetables ../../misc/**

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

or **mv vegetables ~/misc/**

All these answers are correct

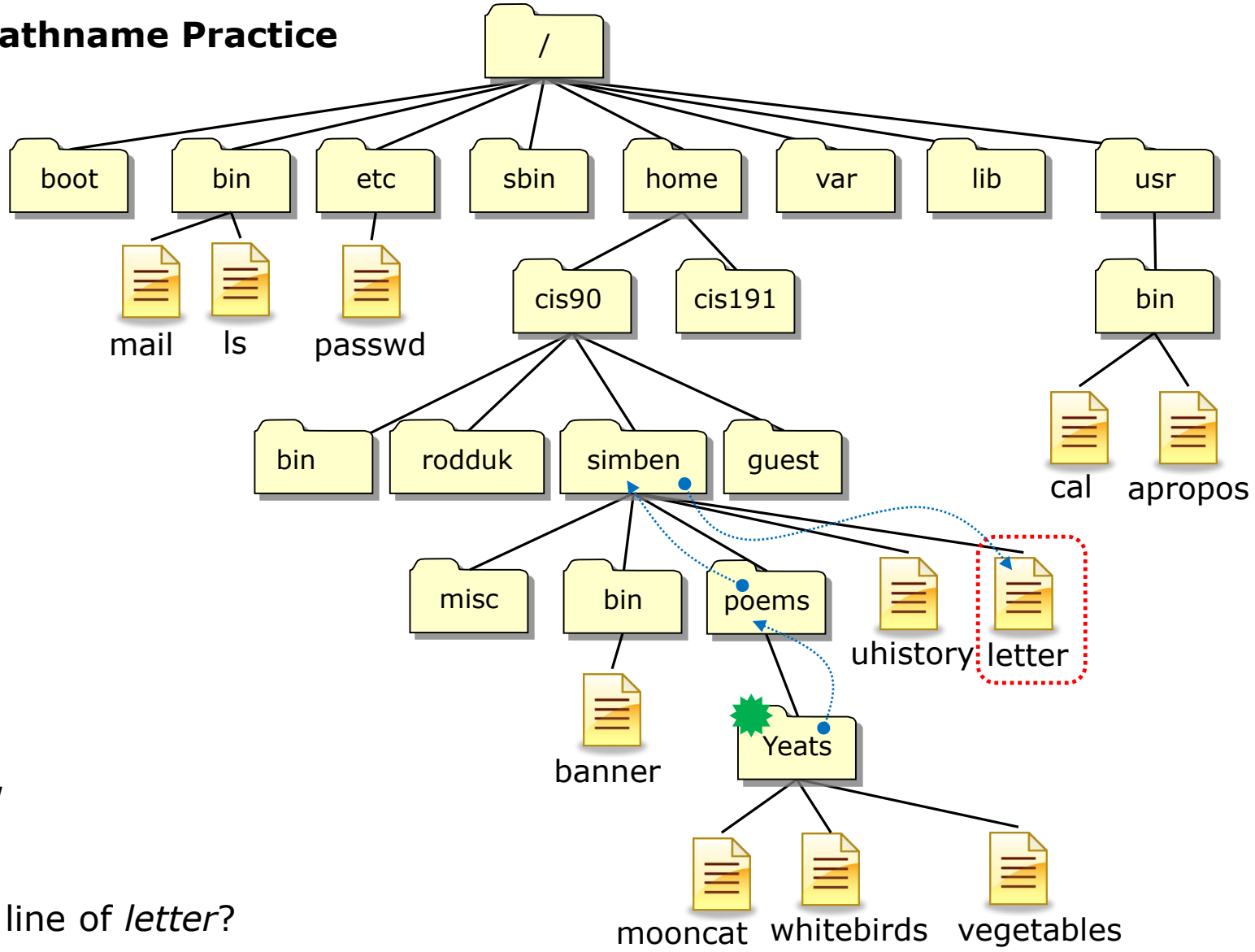
File Tree Pathname Practice



From how does Benji:

Print the last line of *letter*?

File Tree Pathname Practice

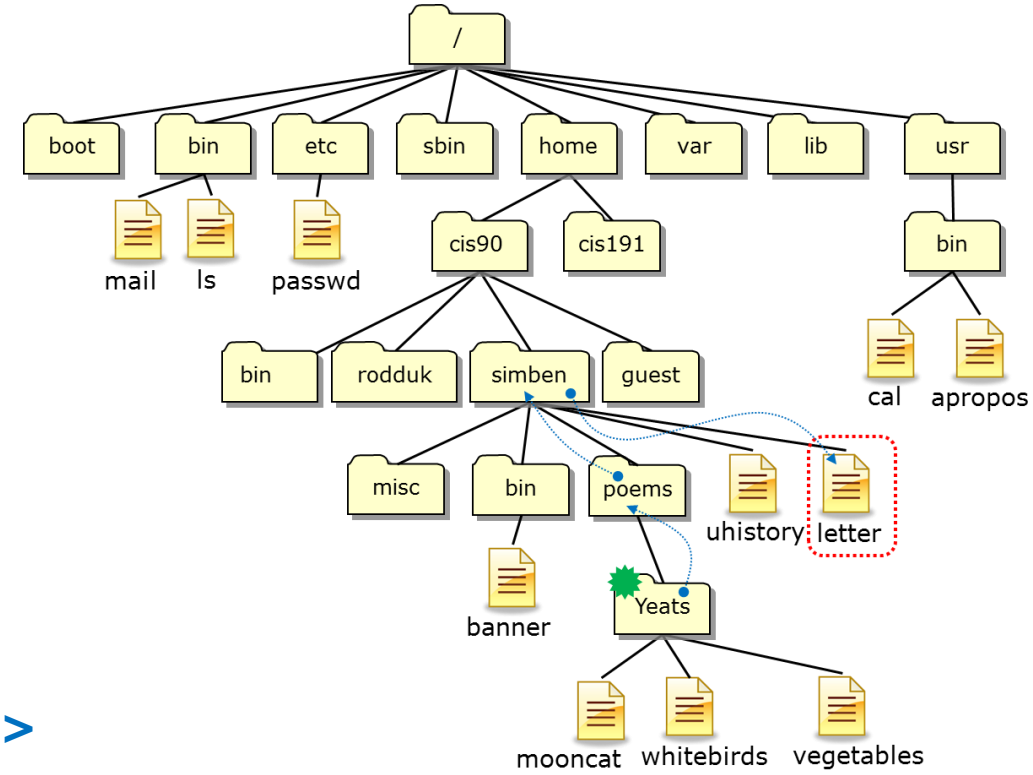


From ✹ how does Benji:

Print the last line of *letter*?

`/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>*

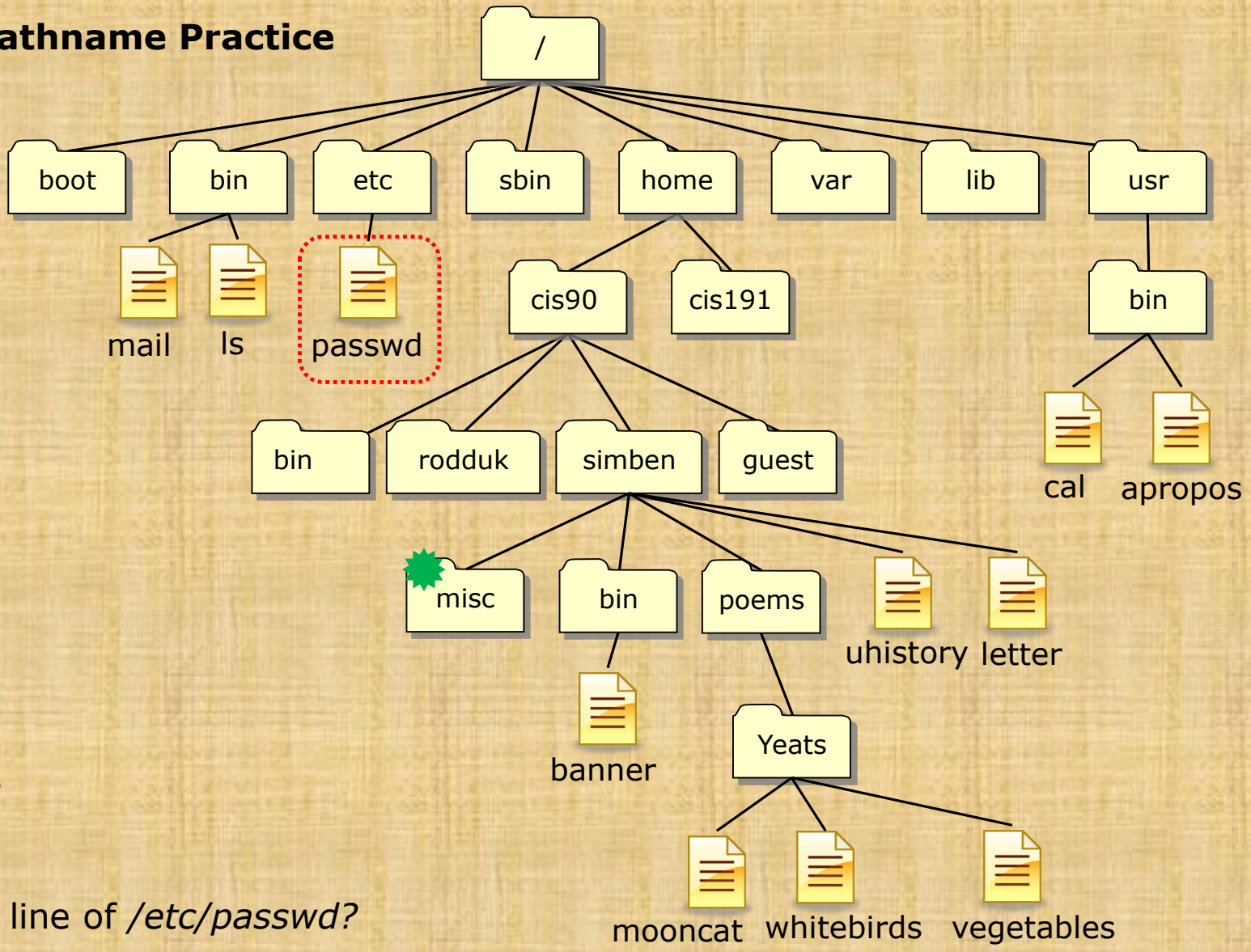
tail -n1 ../../letter

or **tail -n1 /home/cis90/simben/letter**

or **tail -n1 ~/letter**

All these answers are correct

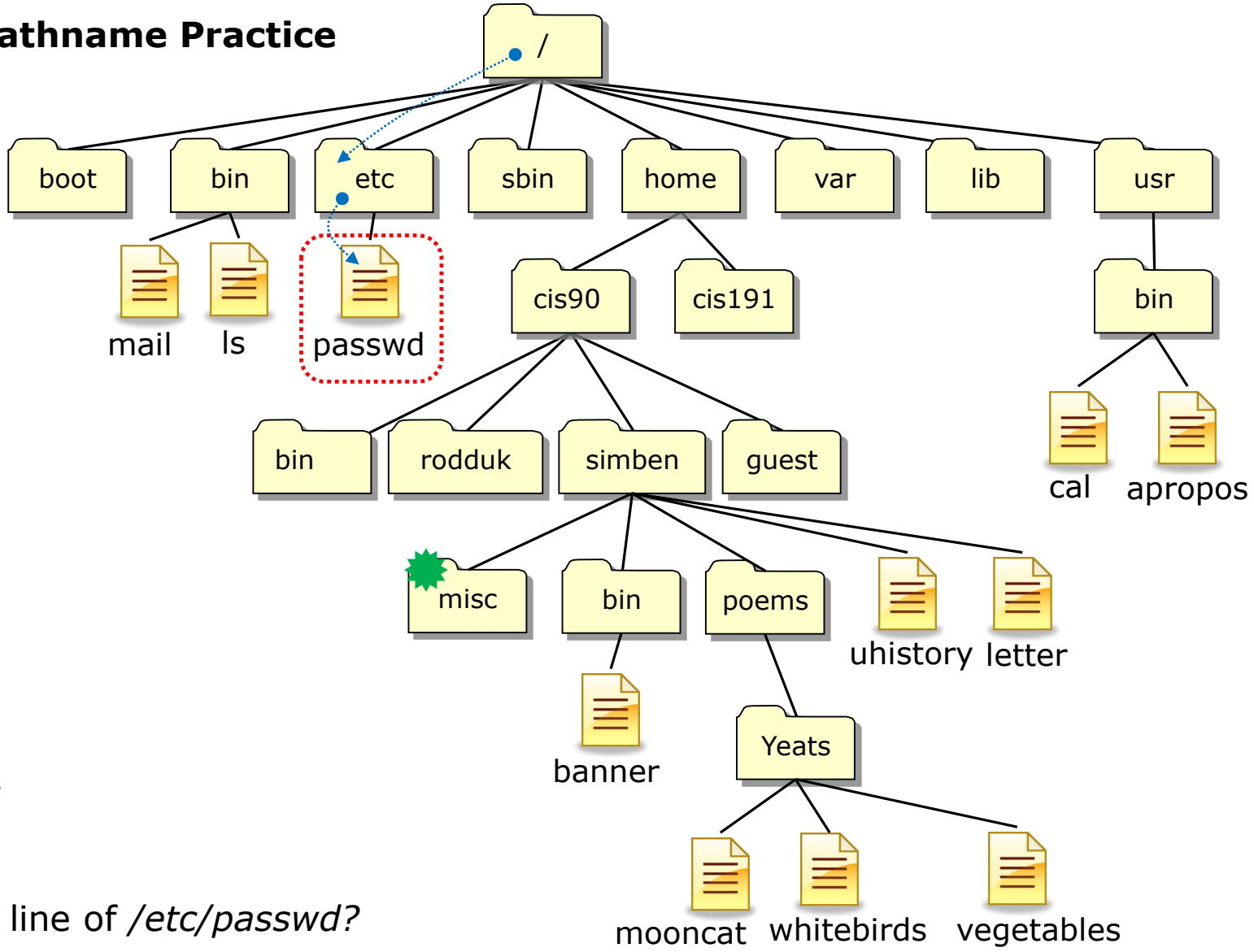
File Tree Pathname Practice



From how does Benji:

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

File Tree Pathname Practice

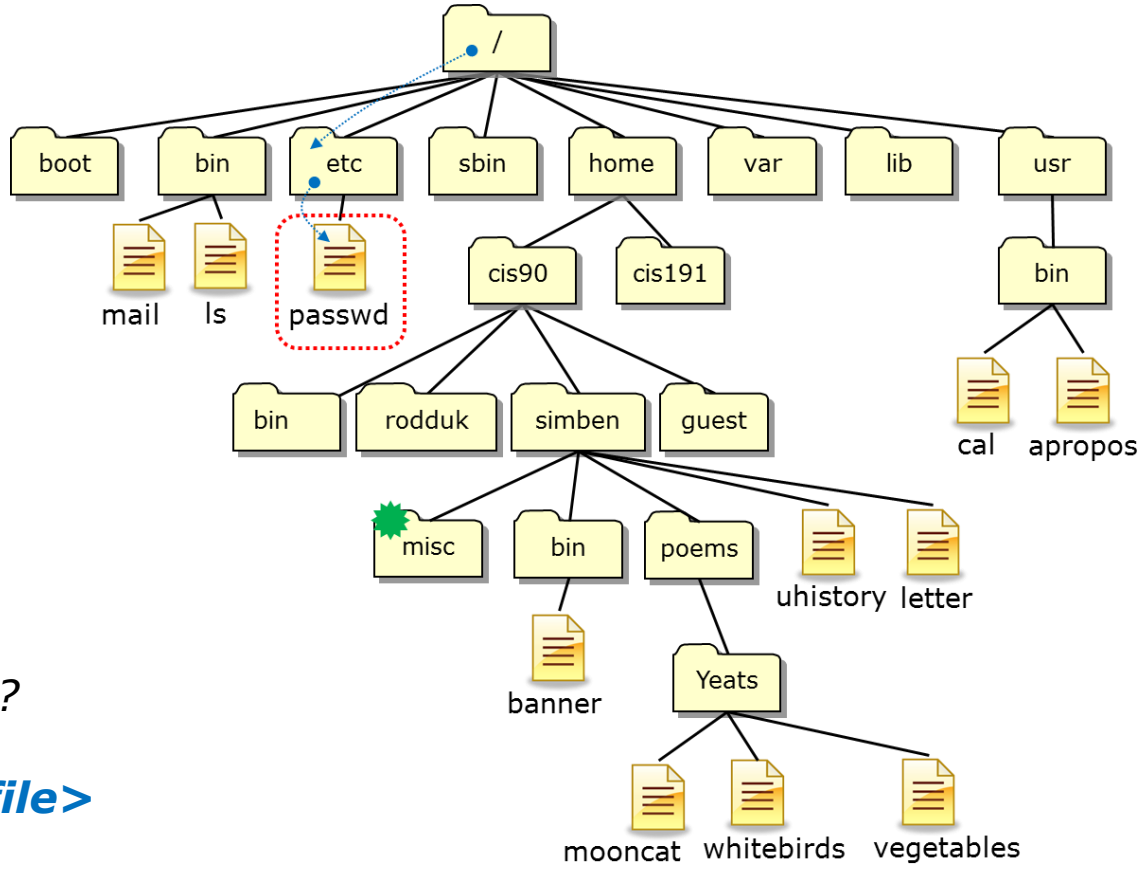


From ✹ how does Benji:

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?*

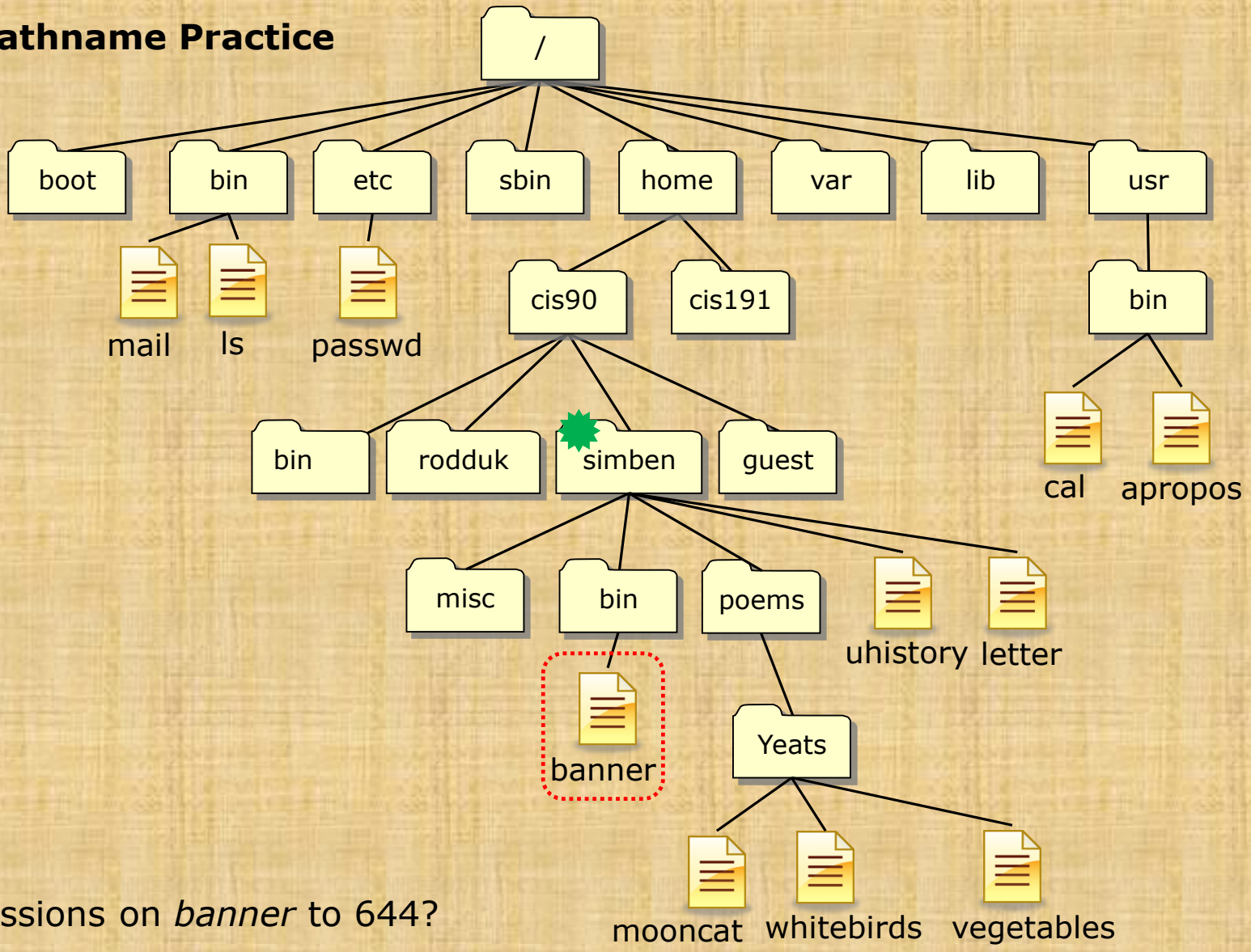
head -n<number> <path-to-file>

or **head -n1 /etc/passwd**

or **head -n1 ../../../../etc/passwd**

Both these answers are correct

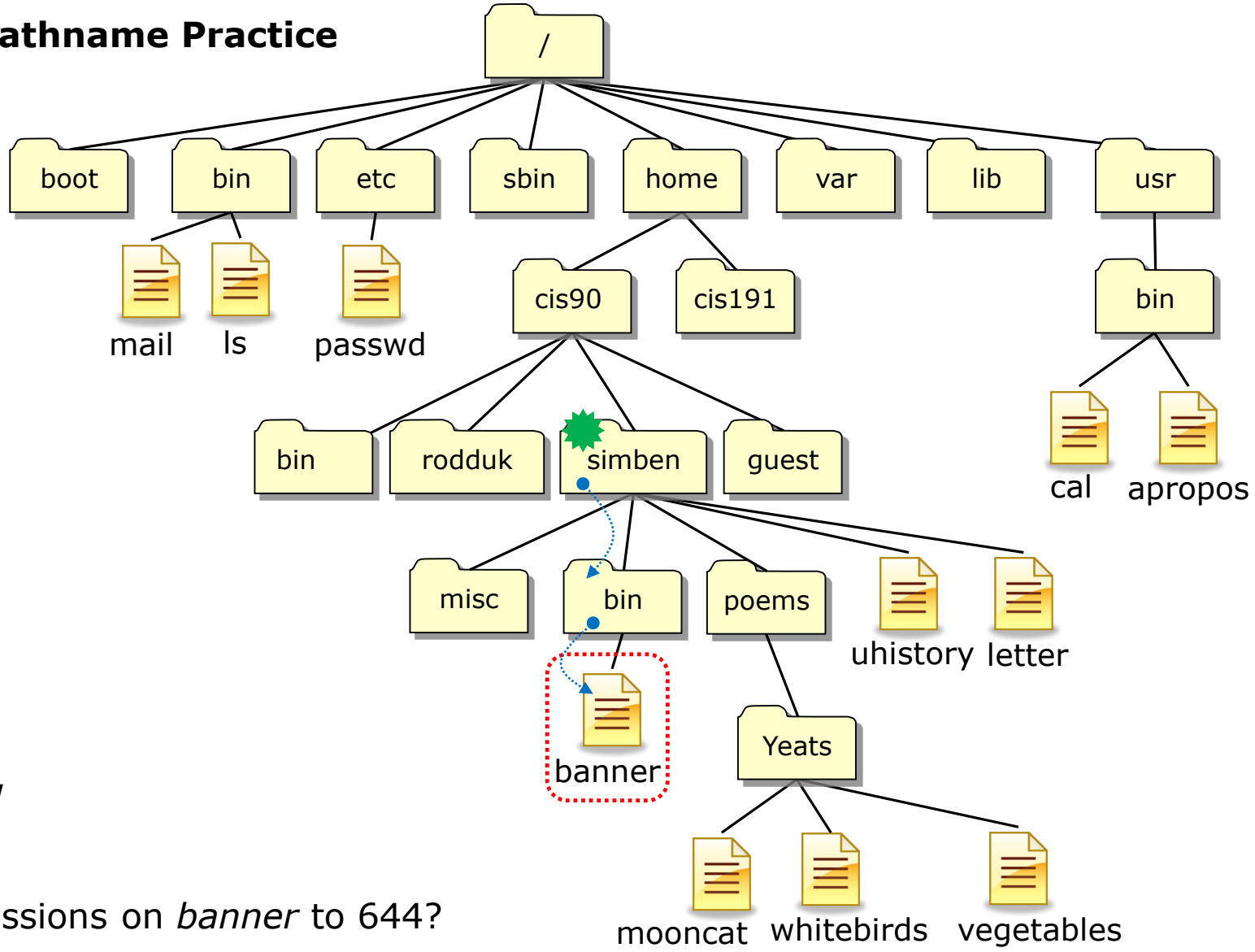
File Tree Pathname Practice



From how does Benji:

Change permissions on *banner* to 644?

File Tree Pathname Practice

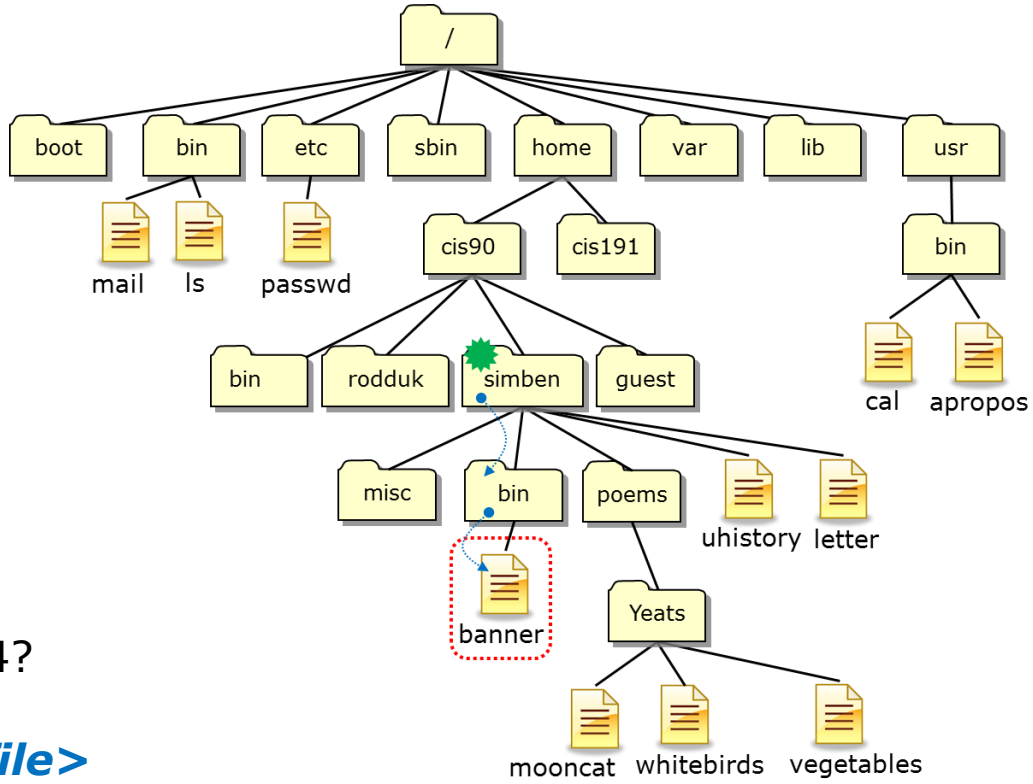


From ✹ how does Benji:

Change permissions on *banner* to 644?

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

Other answers are also acceptable



From how does Benji:

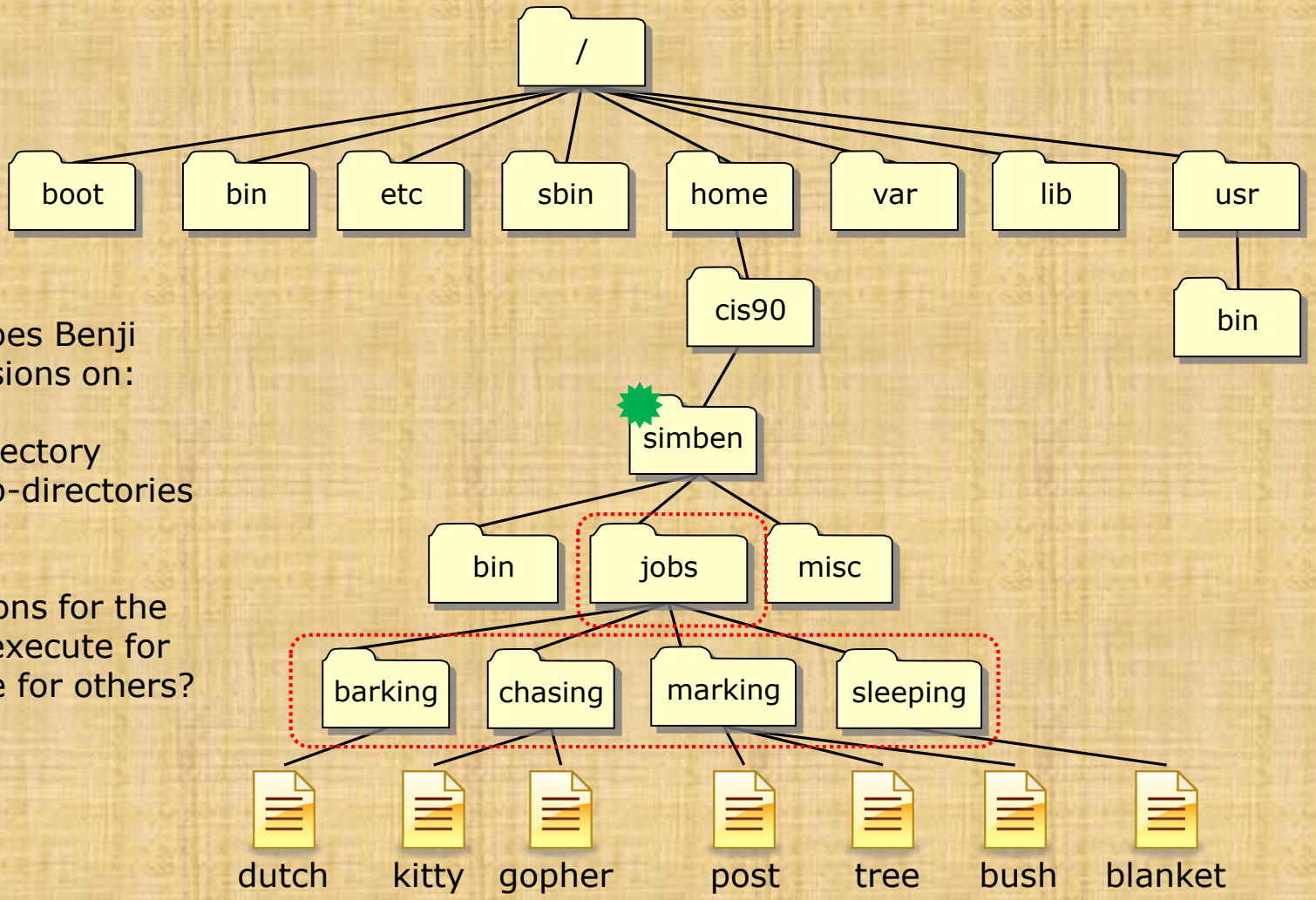
Change permissions on *banner* to 644?

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

or **chmod 644 bin/banner**

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

Both these answers are correct



From how does Benji change permissions on:

1. His *jobs/* directory
2. The four sub-directories under *jobs/*

to full permissions for the owner, read & execute for group and none for others?

You can make your own jobs directory by issuing:

cd
tar xvf ../depot/jobs.tar

This works

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

So does this

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

And this

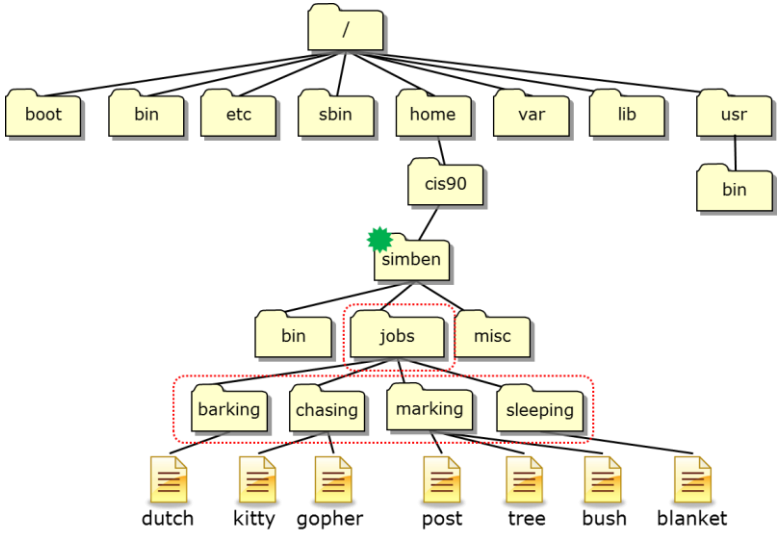
```
chmod 750 jobs
chmod 750 jobs/barking/ jobs/chasing/ jobs/marking/ jobs/sleeping/
```

This is better though

```
chmod 750 jobs
chmod 750 jobs/*
```

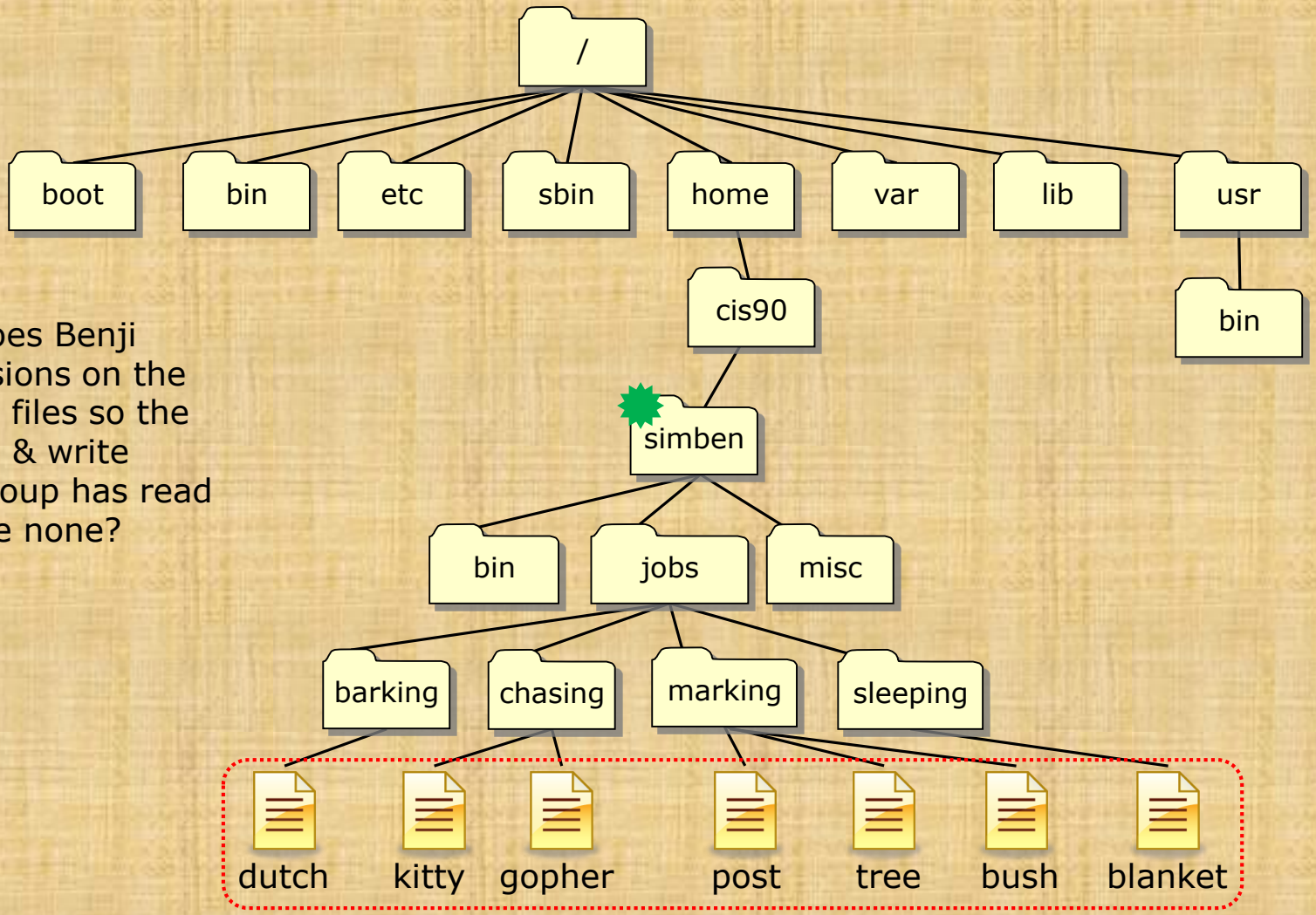
I like this the best!

```
chmod 750 jobs jobs/*
```



And so … which way did you do step 9 in Lab 6?

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?

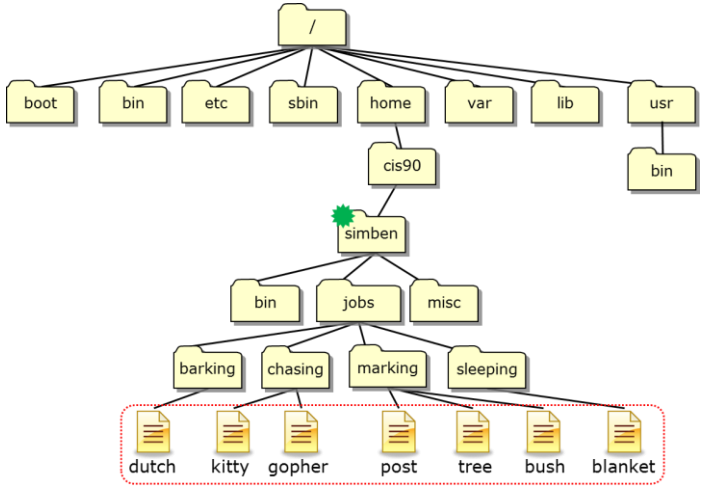


This will always work

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

This works too

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



So will this

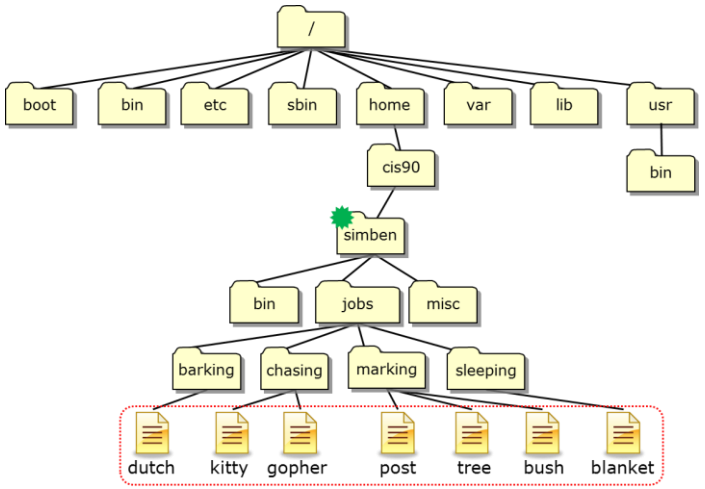
```
cd jobs
cd barking
chmod 640 *
cd ..
cd chasing
chmod 640 *
cd ..
cd marking
chmod 640 *
cd ..
cd sleeping
chmod 640 *
cd
```

This is better

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

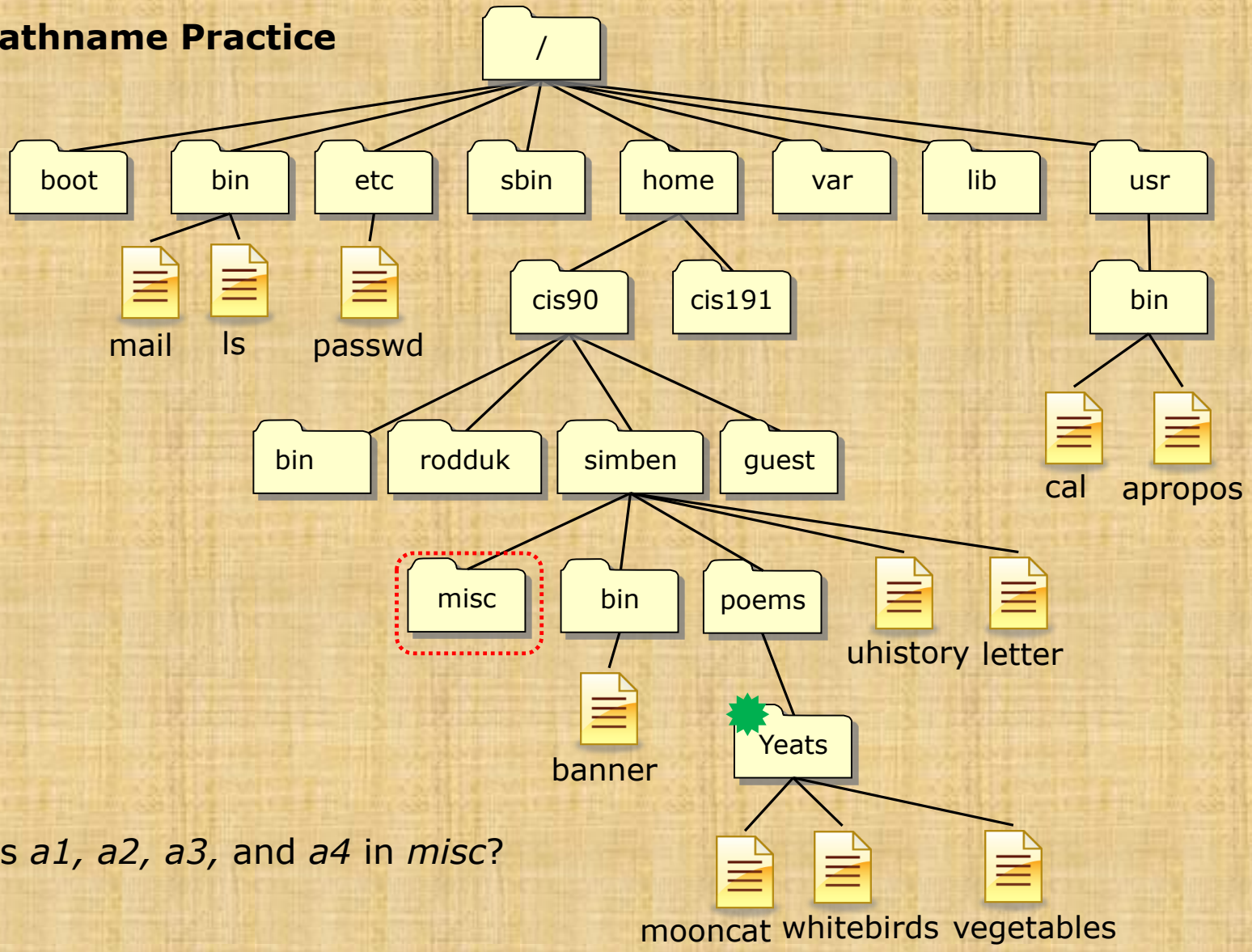
I like this the best!

```
chmod 640 jobs/*/*
```



And so … which way did you do step 10 in Lab 6?

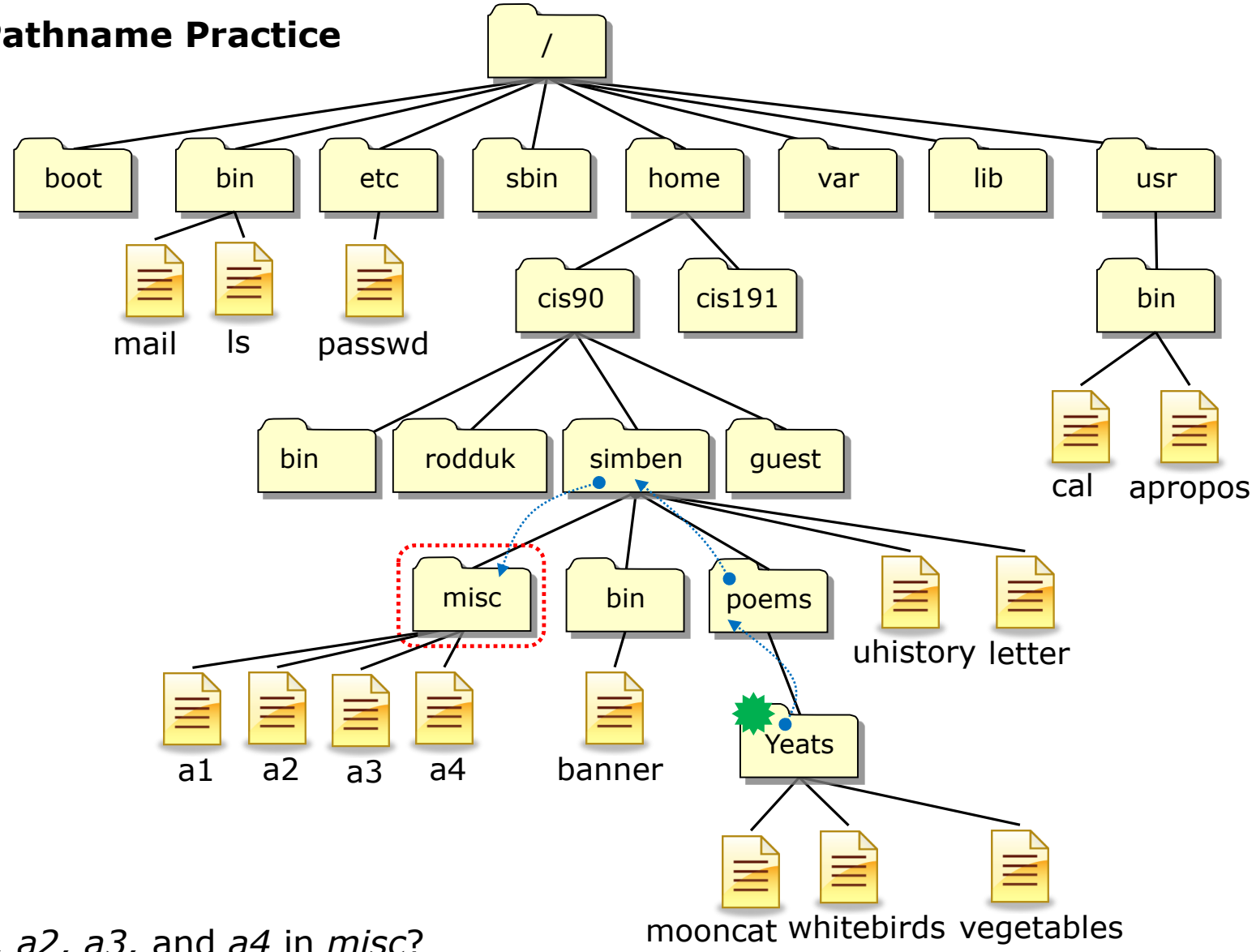
File Tree Pathname Practice



From how does Benji:

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

File Tree Pathname Practice

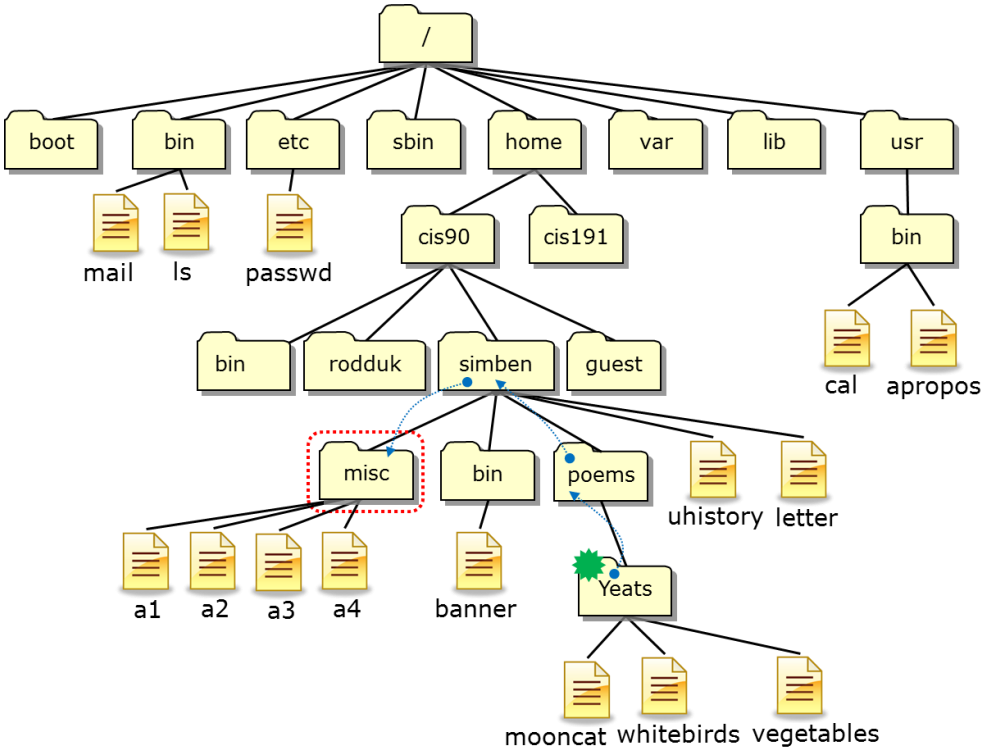


From ✹ how does Benji:

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

/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

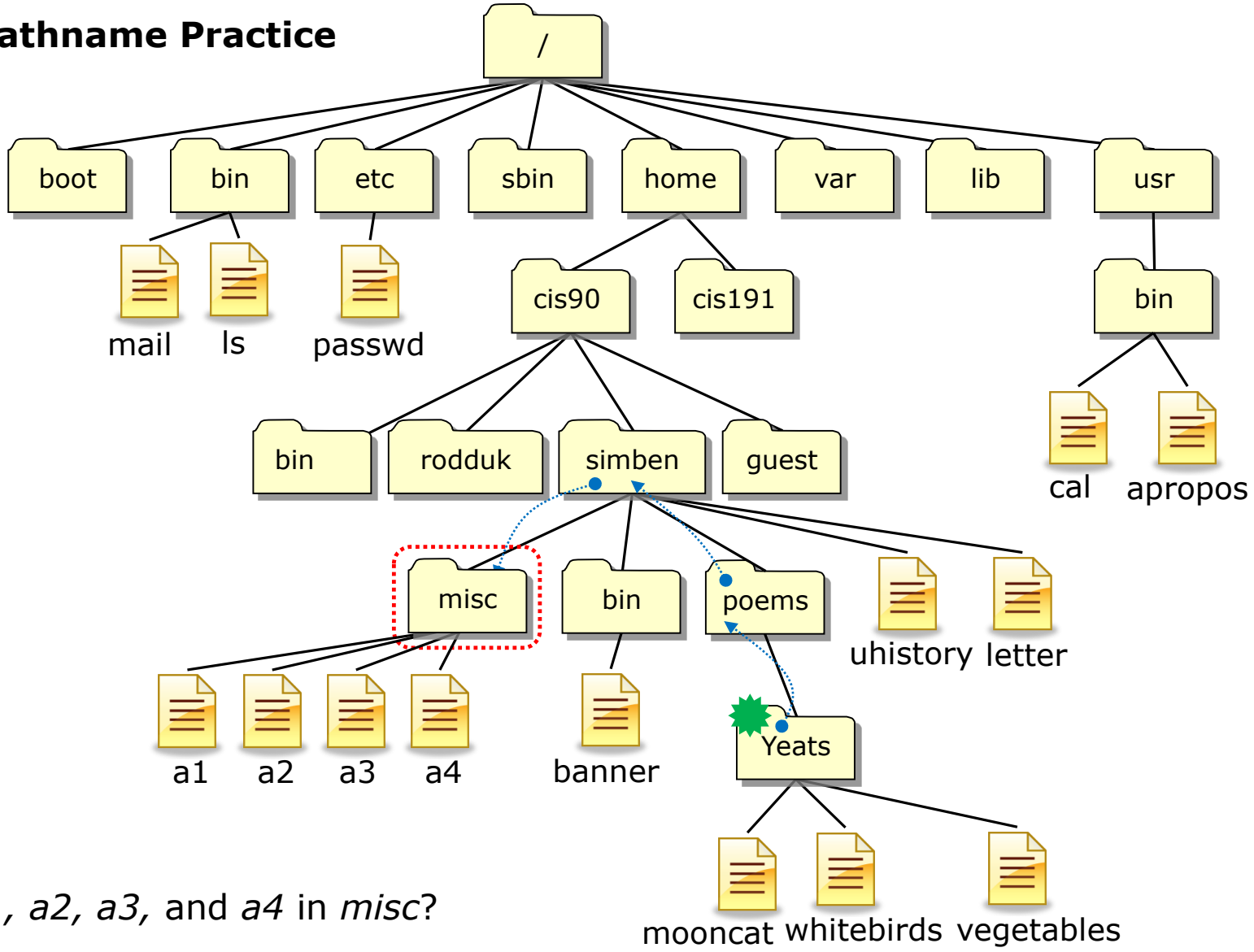
or **touch ~/misc/a1 ~/misc/a2 ~/misc/a3 ~/misc/a4**

or **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**

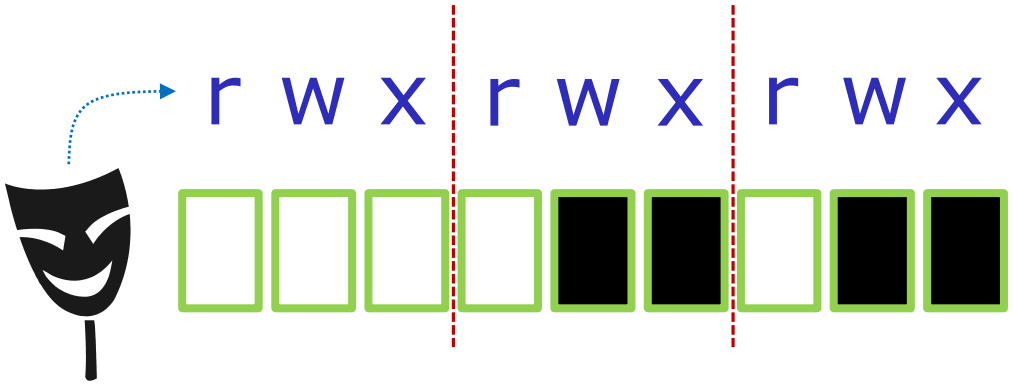
Case 1 – a new directory

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?



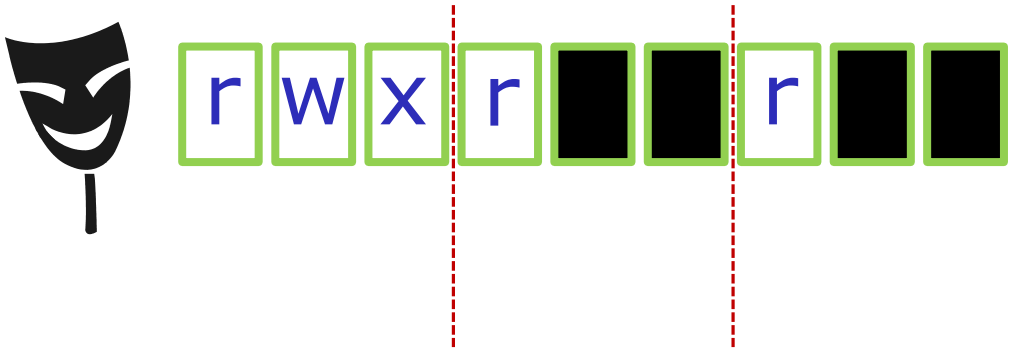
starting point = 777 (new directory)

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?



starting point = 777 (new directory)

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

Answer: 744

Prove it to yourself on Opus-II 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
```

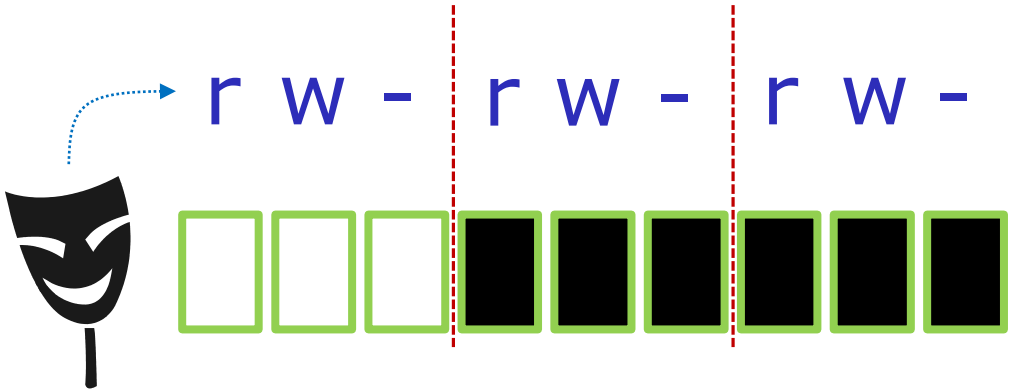
Case 2 – new file

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?



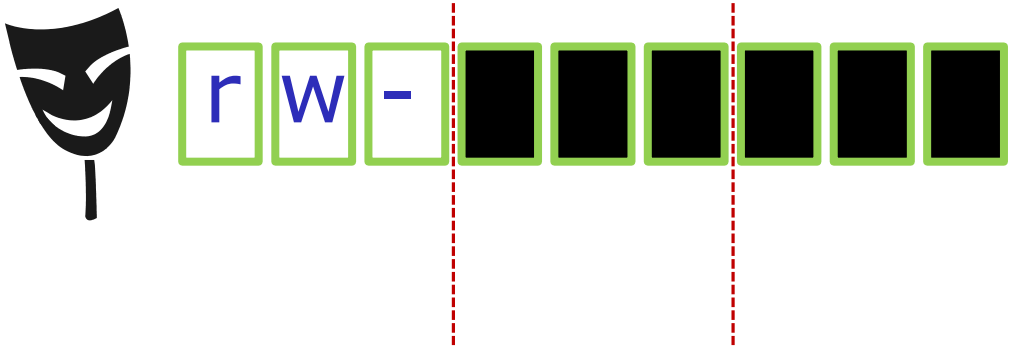
starting point = 666 (new file)

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

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-II 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:

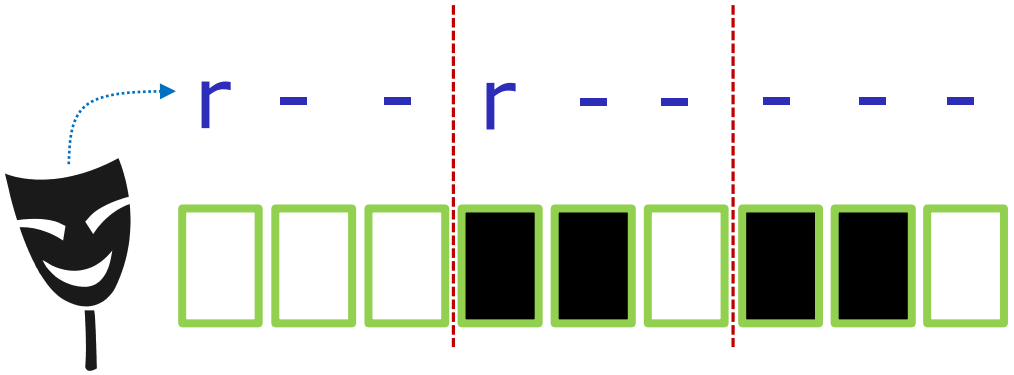
```
cp cinderella cinderella.bak
```

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
```



starting point = 440
(source file permissions)

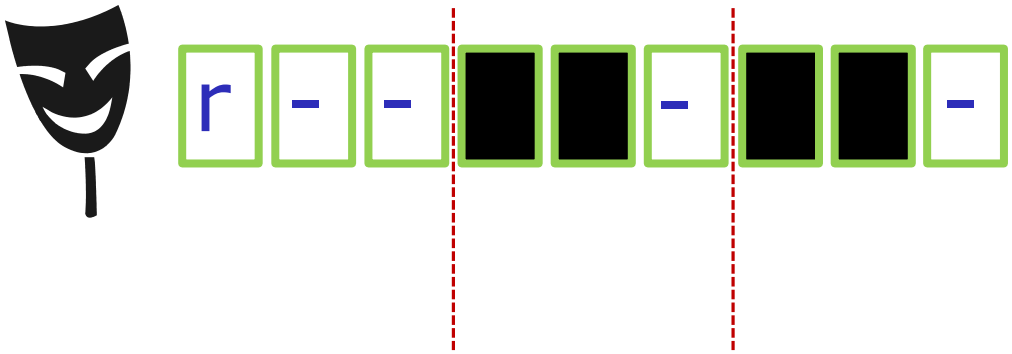
umask setting of 066 strips these bits: --- rw- rw-

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-II 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 your final Lab 6!

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.

Last Call -- Perkins/VTEA Survey

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"

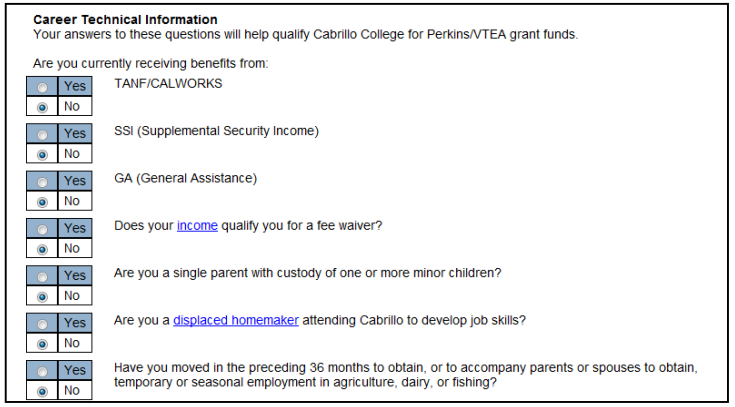
Scroll down to the "Career Technical Information"

- 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"

This is an important source of funding for Cabrillo College.

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



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?

iiED Institute for Innovation & Economic Development
CSU Monterey Bay | College of Business PRESENTS

STARTUP HACKATHON MONTEREY BAY

Hurricanes, Earthquakes and Wildfires: Tech for Disasters

Preparing, surviving and recovering from disasters present challenges to information flow, coordination of services, redirection of needed supplies, maintenance of order and safety, and even basic survival. Your mission, should you choose to accept it, is to create technology-based solutions to aid people (or animals!) in preparing for, surviving, and recovering from major natural disasters. We encourage students and teams of students from all fields, including computer science, environmental science, health and human services, cinematic arts and technology, and business to participate in the Hackathon.

Students are encouraged to bring laptops.
For questions or disability accommodations, please contact innovation@csumb.edu

Pre-registration is required at https://startuphackathonmb2017.eventbrite.com

Scan the code to register

Date: November 3rd– 5th
Time: 4 p.m. Nov 3rd – 4 p.m. Nov. 5th
Place: CSUMB BIT Building (1st Floor)
Cost: Free for CSUMB, Cabrillo college, and MPC Students (use code HACK 2017), other students $49, General $99

For more information visit csumb.edu/iied

HACKATHON PARTNERS

California State University MONTEREY BAY | College of Business and School of Computing & Design | her Script(♡); | MPC Monterey Peninsula College | Cabrillo College

Free for Cabrillo students

Date: November 3rd– 5th
Time: 4 p.m. Nov 3rd – 4 p.m. Nov. 5th
Place: CSUMB BIT Building (1st Floor)
Cost: Free for CSUMB, Cabrillo college, and MPC Students (use code HACK 2017), other students $49, General $99

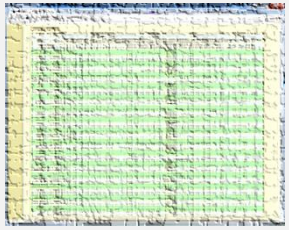
https://simms-teach.com/docs/Hackathon-Cabrillo-Flyer-2016-17.pdf

Where to find your grades

Send me your survey to get your LOR code name.

The CIS 90 website Grades page

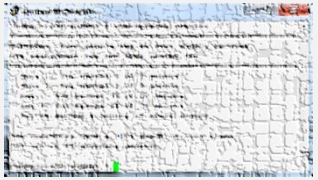
http://simms-teach.com/cis90grades.php



Or check on Opus

checkgrades *codename*

(where codename is your LOR codename)



Written by Jesse Warren 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

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**

New commands

Lesson 8 commands for your toolbox

find - Find file or content of a file

grep - "Global Regular Expression Print"

sort - sort

spell - spelling correction

wc - word count

tee - split output

cut - cut fields from a line

sort command

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

Activity

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

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

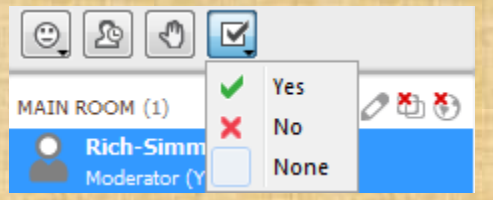
return to home directory

relative path to the names file in the depot directory

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

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
```



Give me a green Yes check if you get the same results

Pretend you are a command

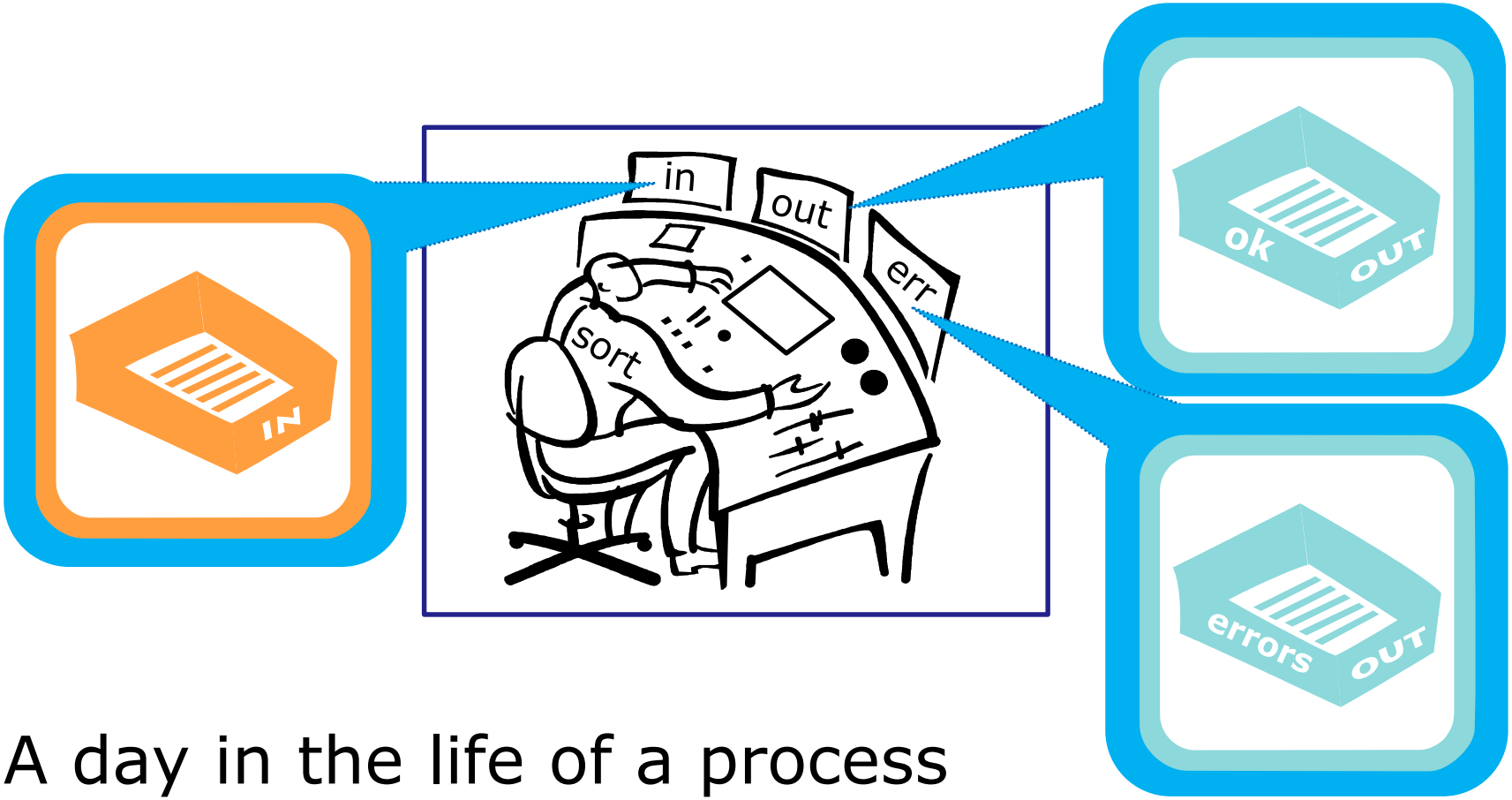
(use your great imagination)

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

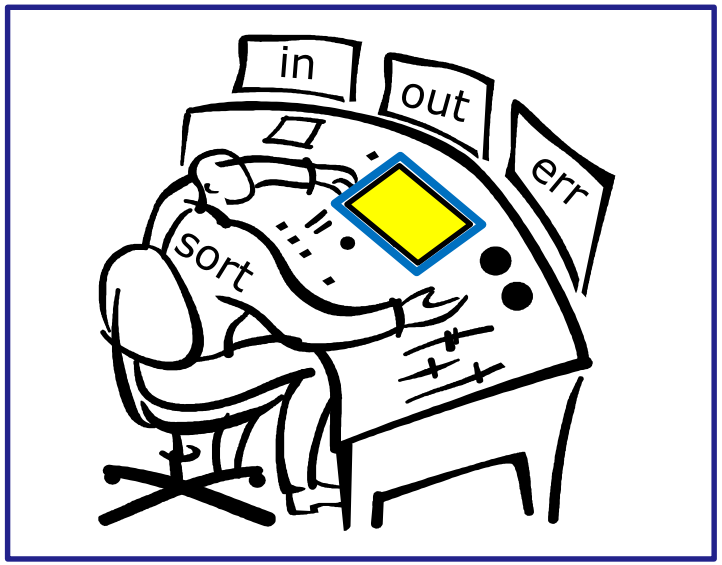
A day in the life of a process

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
deep dive examples

sort <*good filepath*>

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

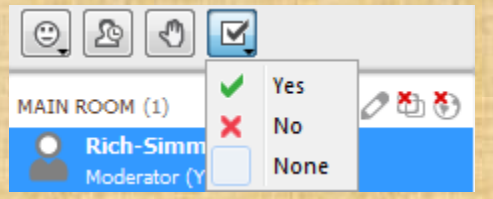
One argument which is a filename

Activity

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



Give me a green Yes check if you get the same results

```
/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 $ "`

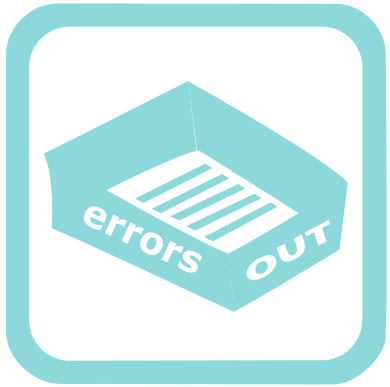
2. Parsing results:
   - command = `sort`
   - no options
   - 1 argument = "`names`"
   - no redirection

3. Search user's path and locate the sort program in */bin*

4. Sort loaded into memory and execution begins

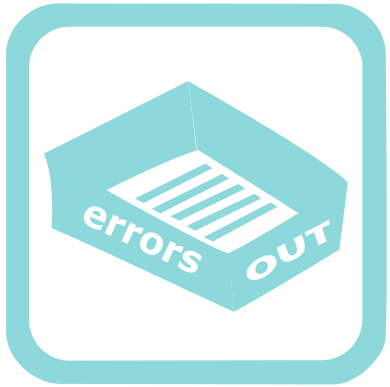
Shell Steps
1) Prompt
2) Parse
3) Search
4) Execute
5) Nap
6) Repeat

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



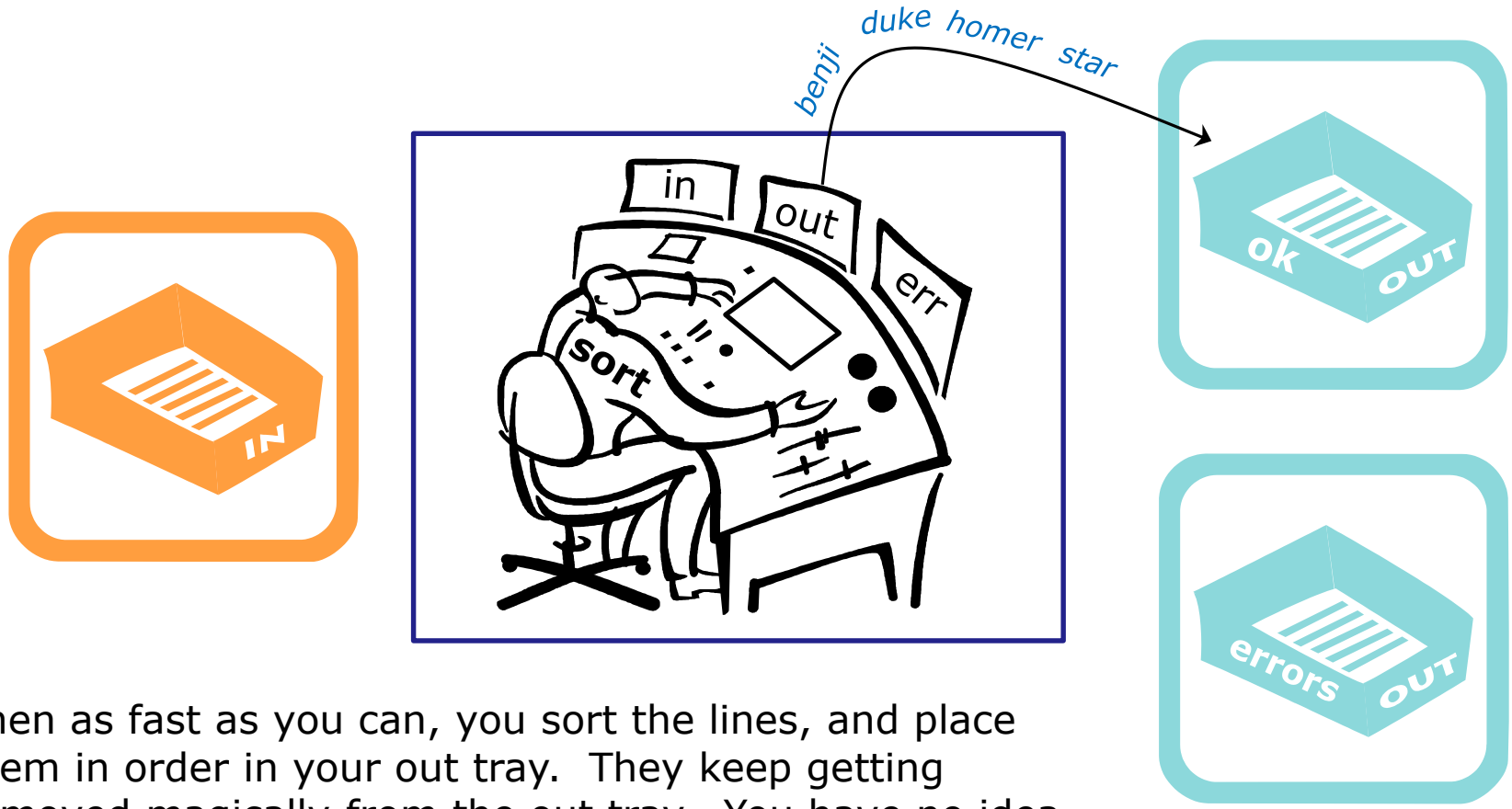
You (the sort process) check your instruction window and see the shell passed one argument "names" to you. You know (given your internal DNA code) that you must contact the kernel and request this file be opened and the contents read.

/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 (*no arguments*)

```
/home/cis90/simben $ sort
kayla
sky
bella
benji
charlie
bella
benji
charlie
kayla
sky
/home/cis90/simben $
```

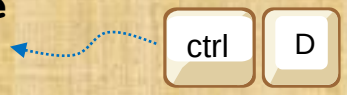
No arguments specified

EOF

Activity

sort command with no arguments

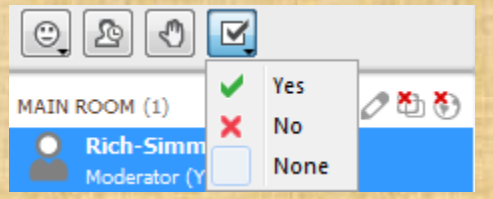
```
/home/cis90/simben $ sort
kayla
sky
bella
benji
charlie
bella
benji
charlie
kayla
sky
```



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

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

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



Give me a green Yes check if you get the same results

```
/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 $ "`

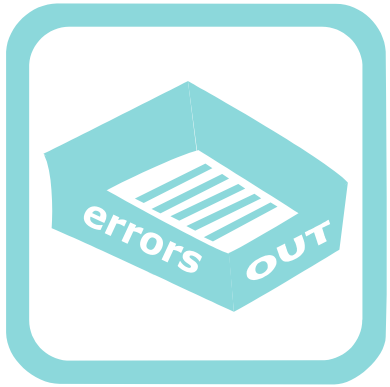
2. Parsing results:
   - command = `sort`
   - no options
   - no arguments
   - no redirection

3. Search user's path and locate the sort program in */bin*

4. Sort loaded into memory and execution begins

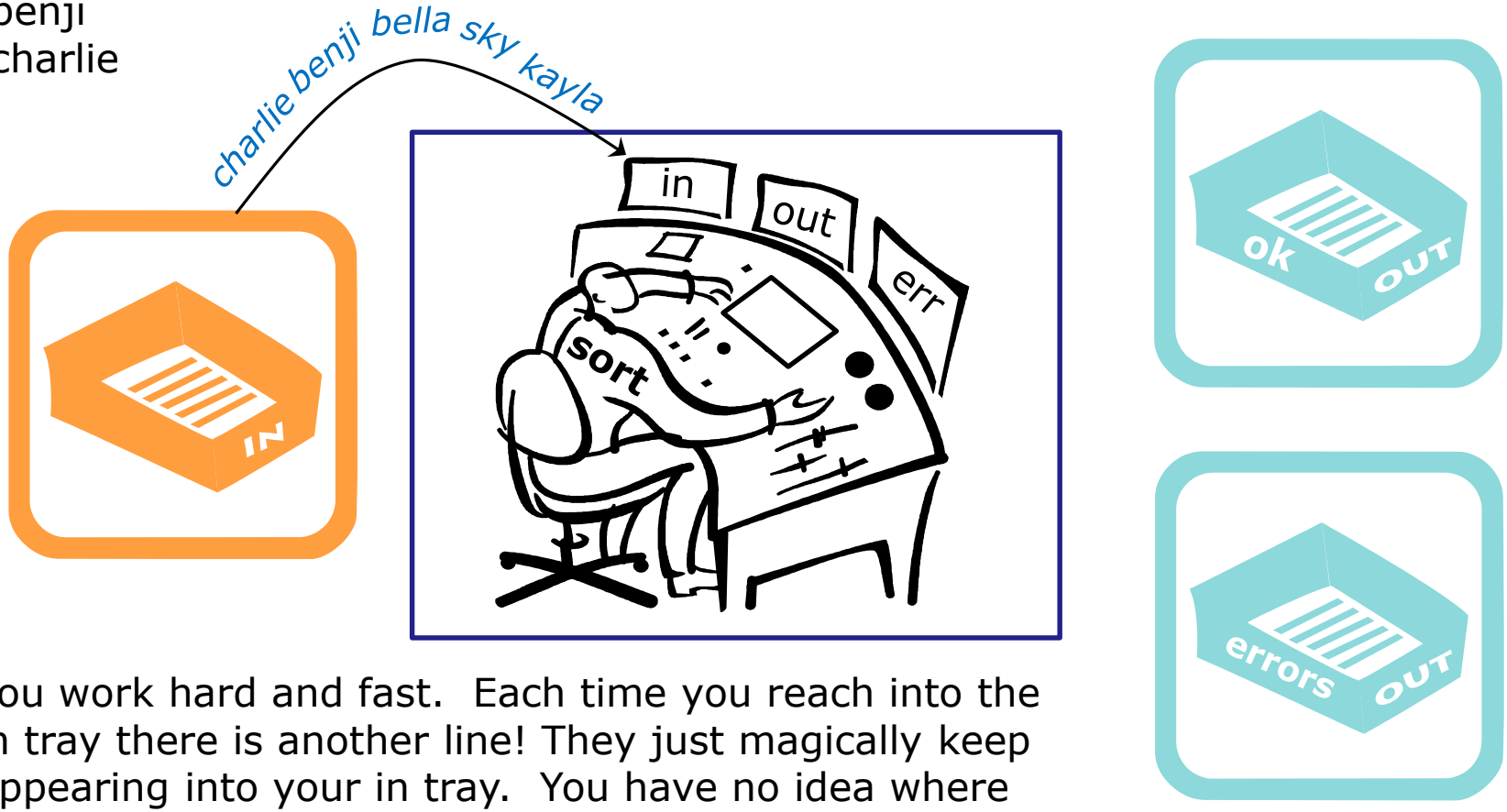
Shell Steps
1) Prompt
2) Parse
3) Search
4) Execute
5) Nap
6) Repeat

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



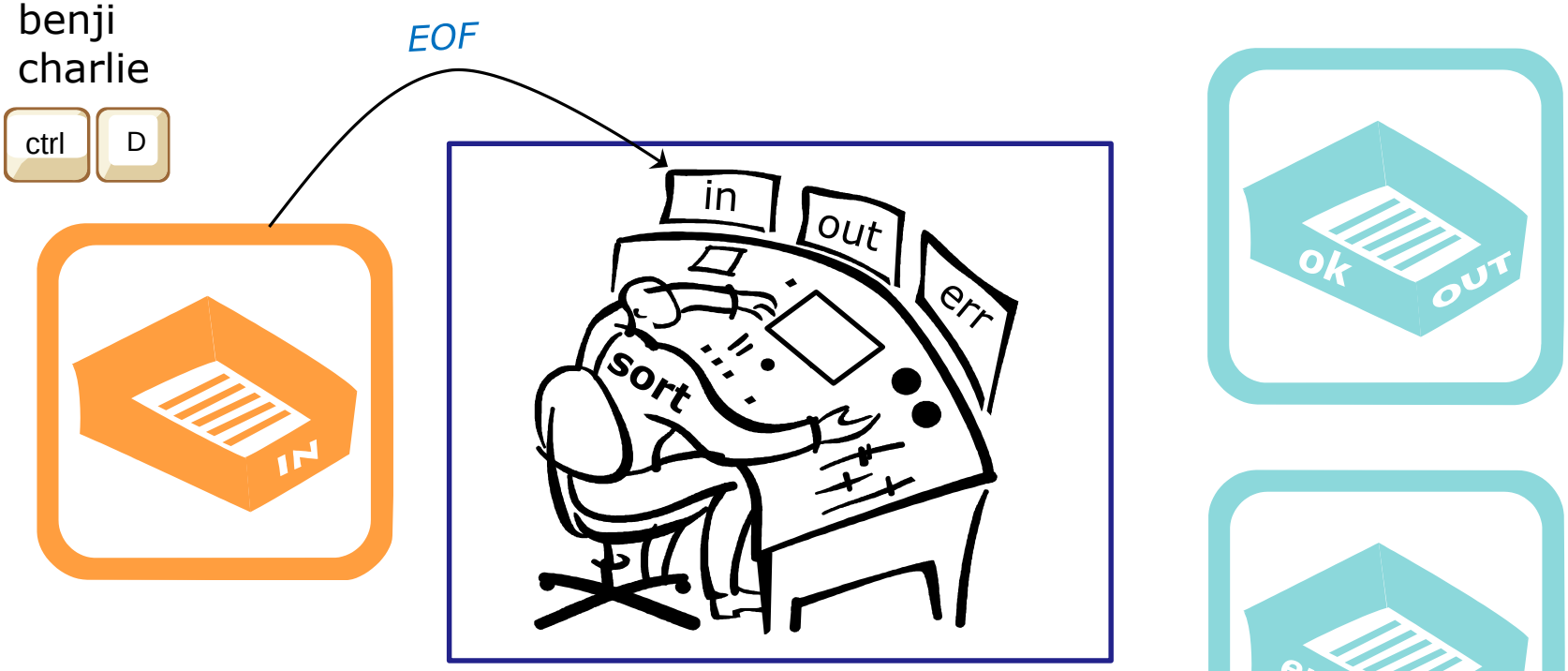
You (the sort process) check your instruction window and see that no options or arguments were passed to you from the shell to handle. You know (given your internal DNA code) that with no arguments you must look for lines to sort in your in tray, so you reach in to grab the first line to sort.

```
/home/cis90/simben $ sort
kayla
sky
bella
benji
charlie
```



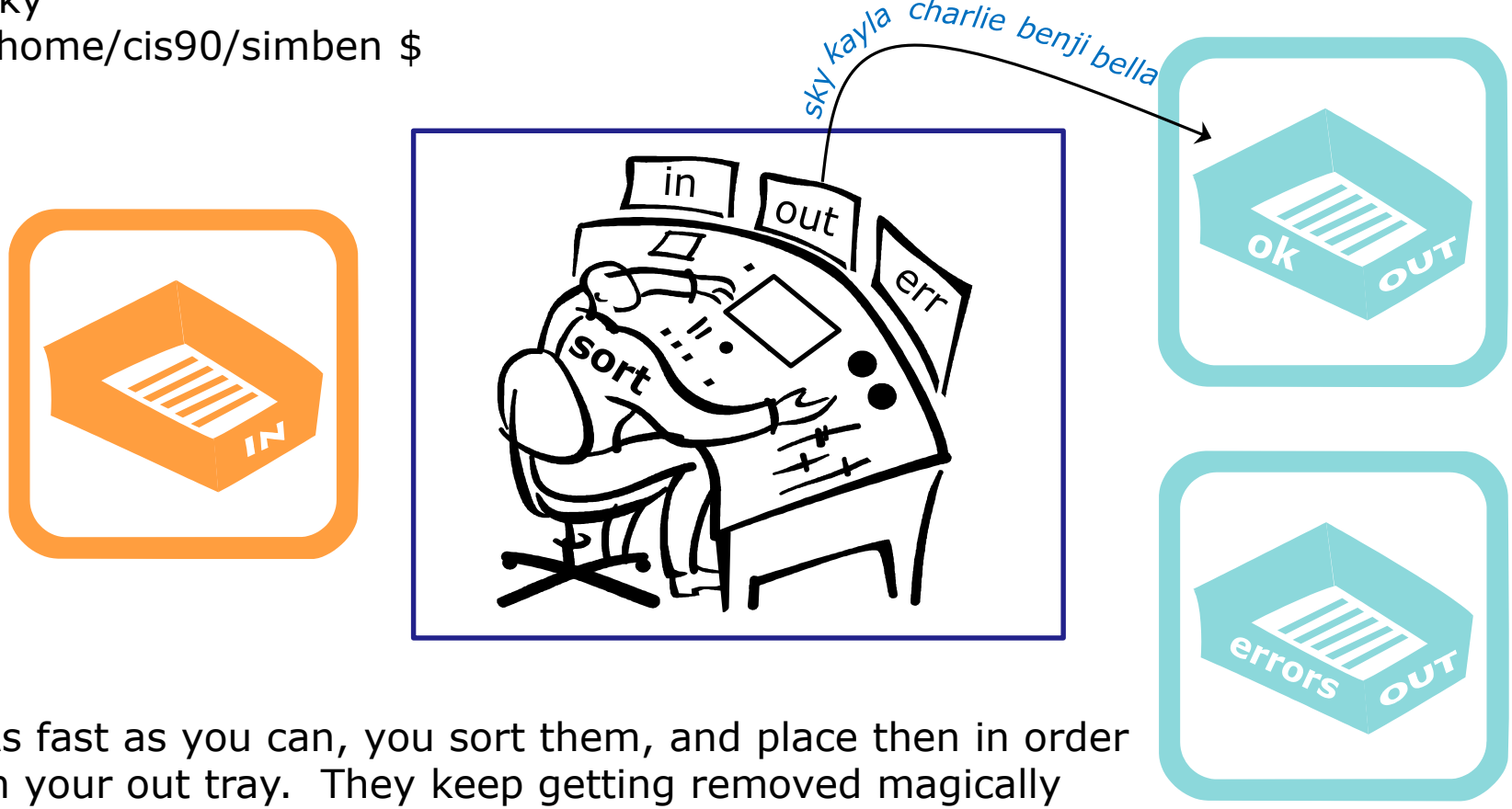
You work hard and fast. Each time you reach into the in tray there is another line! They just magically keep appearing into your in tray. You have no idea where they are coming from.

```
/home/cis90/simben $ sort
kayla
sky
bella
benji
charlie
```



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 $



As fast as you can, you sort them, and place then 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 <*bad filepath*>

No such file

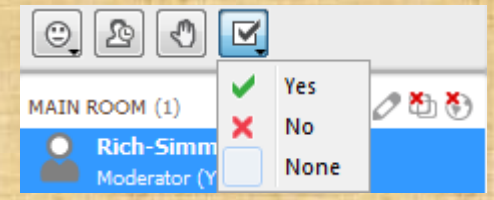
```
/home/cis90/simben $ sort bogus
sort: open failed: bogus: No such file or directory
/home/cis90/simben $
```

Activity

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 the file it receives as an argument and print an error message if the file does not exist



Give me a green Yes check if you get the same results

```
/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 $ "`

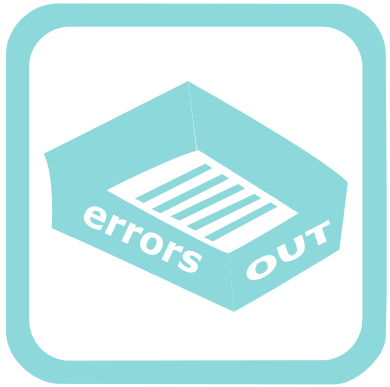
2. Parsing results:
   - command = `sort`
   - no options
   - 1 argument = `bogus`
   - no redirection

3. Search user's path and locate the sort program in */bin*

4. Sort command loaded into memory and execution begins

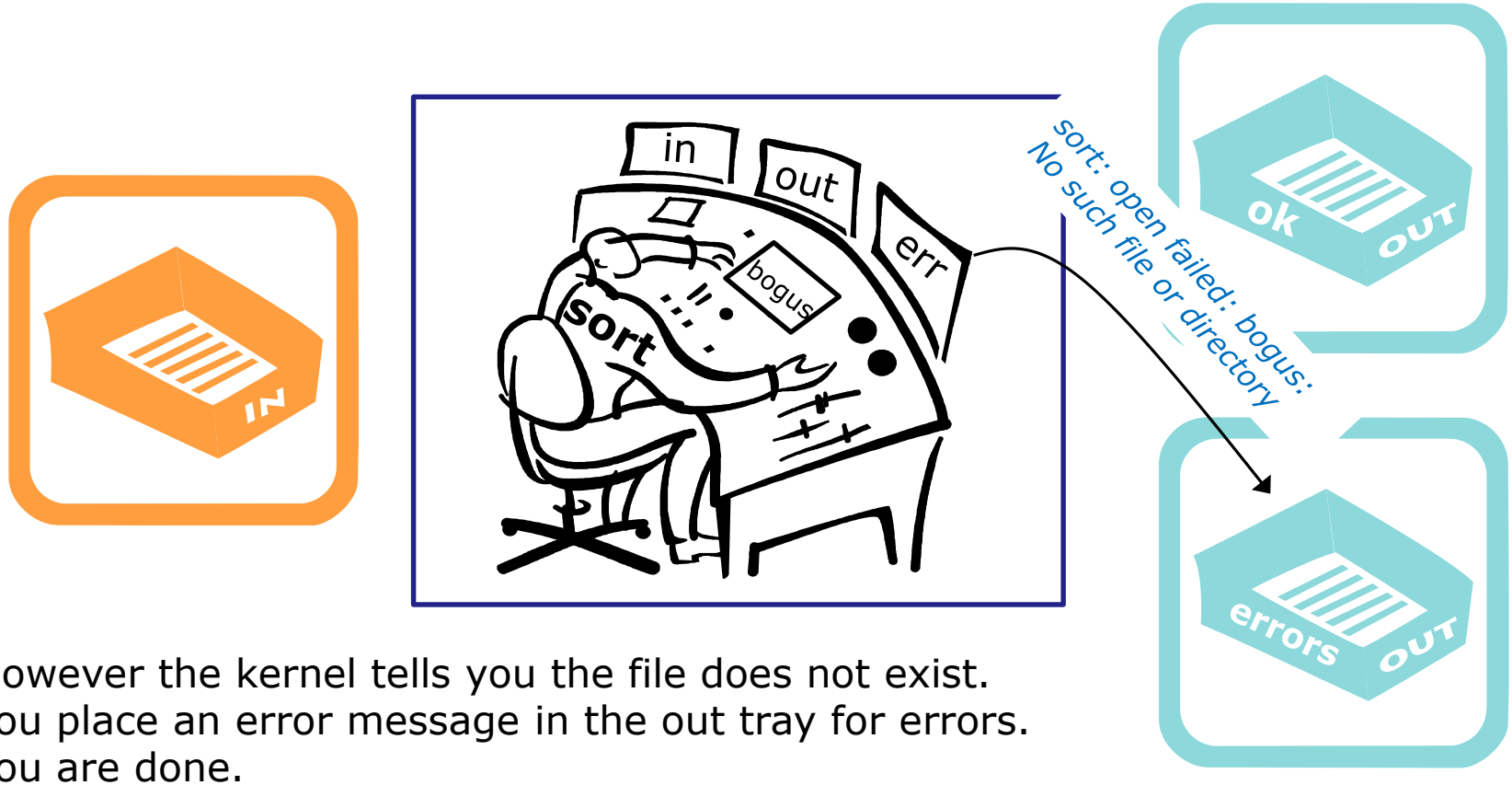
/home/cis90/simben $ **sort bogus**

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.

Bringing it home

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

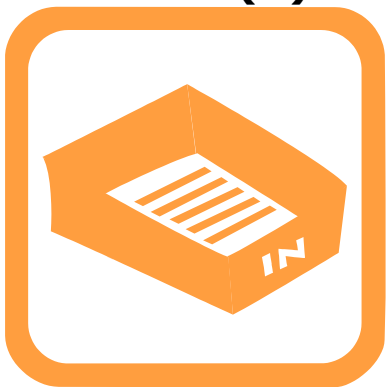
Standard Error (2)
defaults to the user's terminal screen

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)**.

stdout (1)

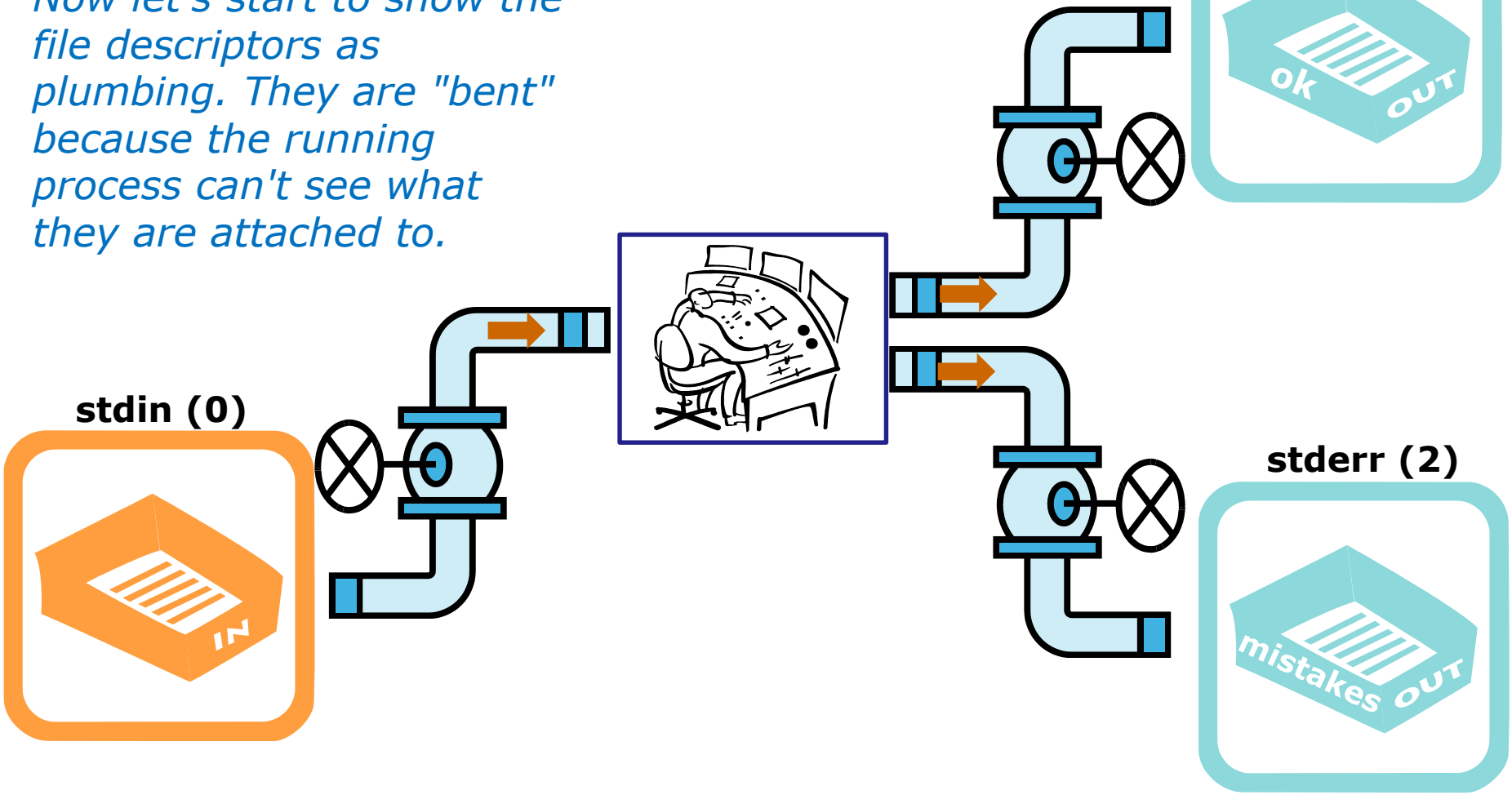
stdin (0)



stderr (2)

Input and Output
File Descriptors

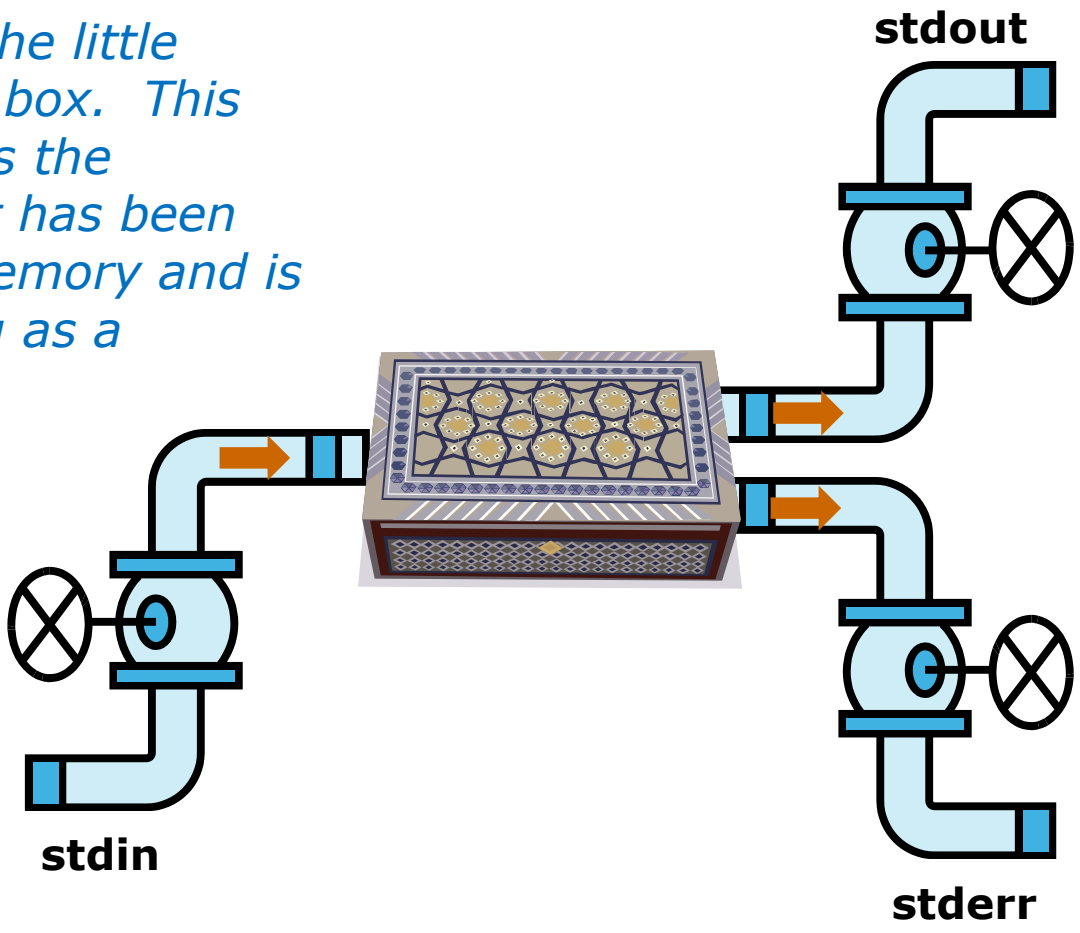
Now let's start to show the file descriptors as plumbing. They are "bent" because the running process can't see what they are attached to.



Input and Output

Loaded Process

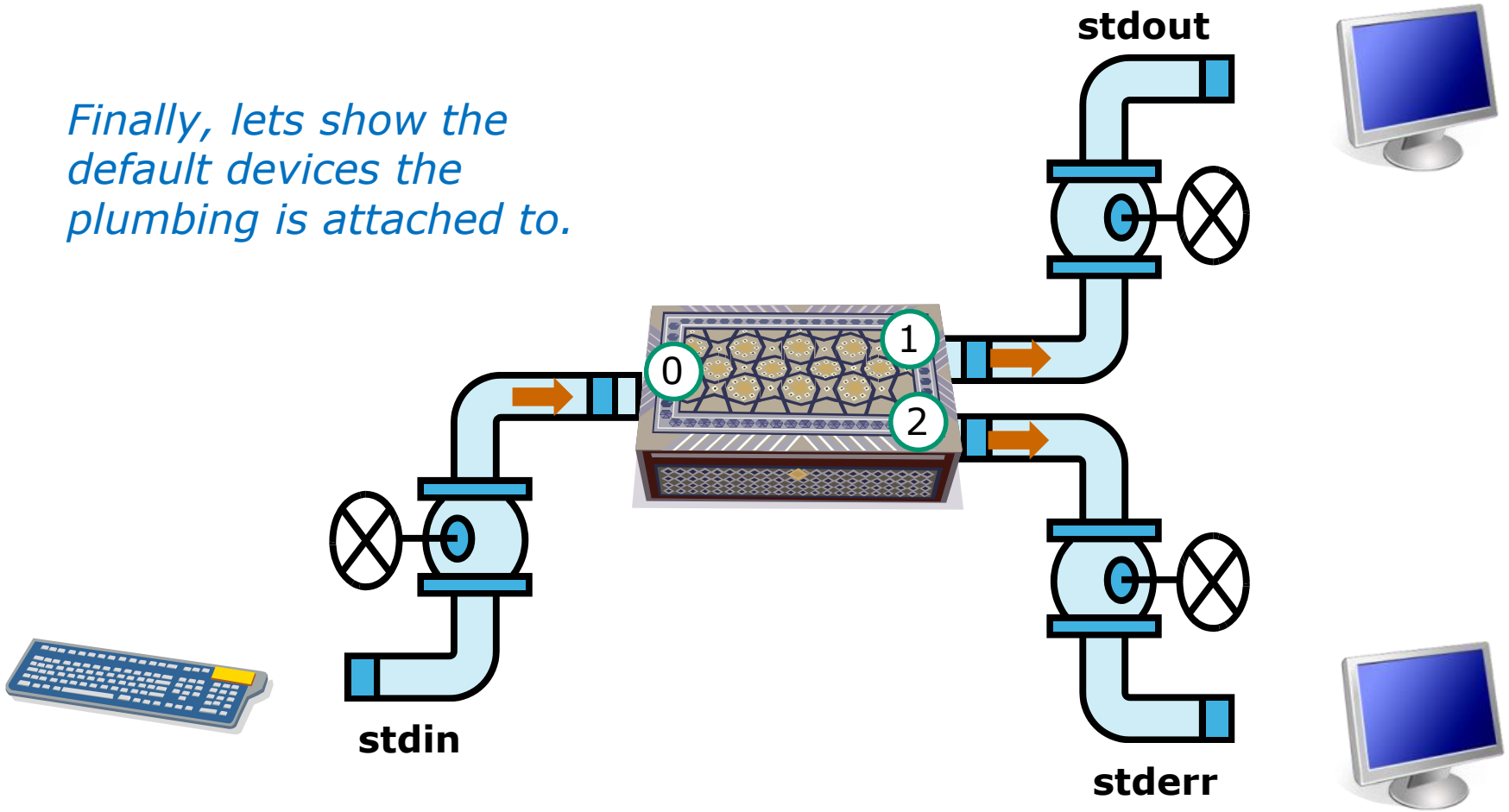
*Let's replace the little worker with a box. This box represents the **program** that has been loaded into memory and is now executing as a **process.***



Input and Output
Default I/O devices

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

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

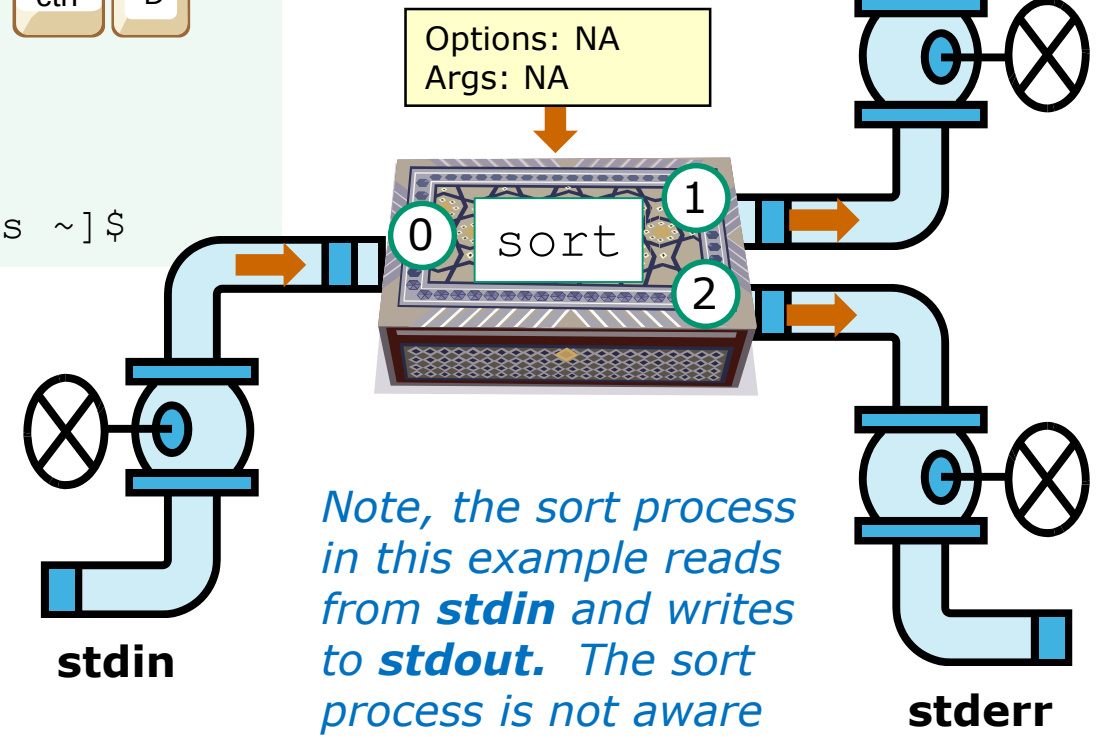


By default is attached to the user's terminal device (keyboard)

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

The sort example again with no arguments

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



```
benji
duke
homer
star
```

```
star
benji
duke
homer
```

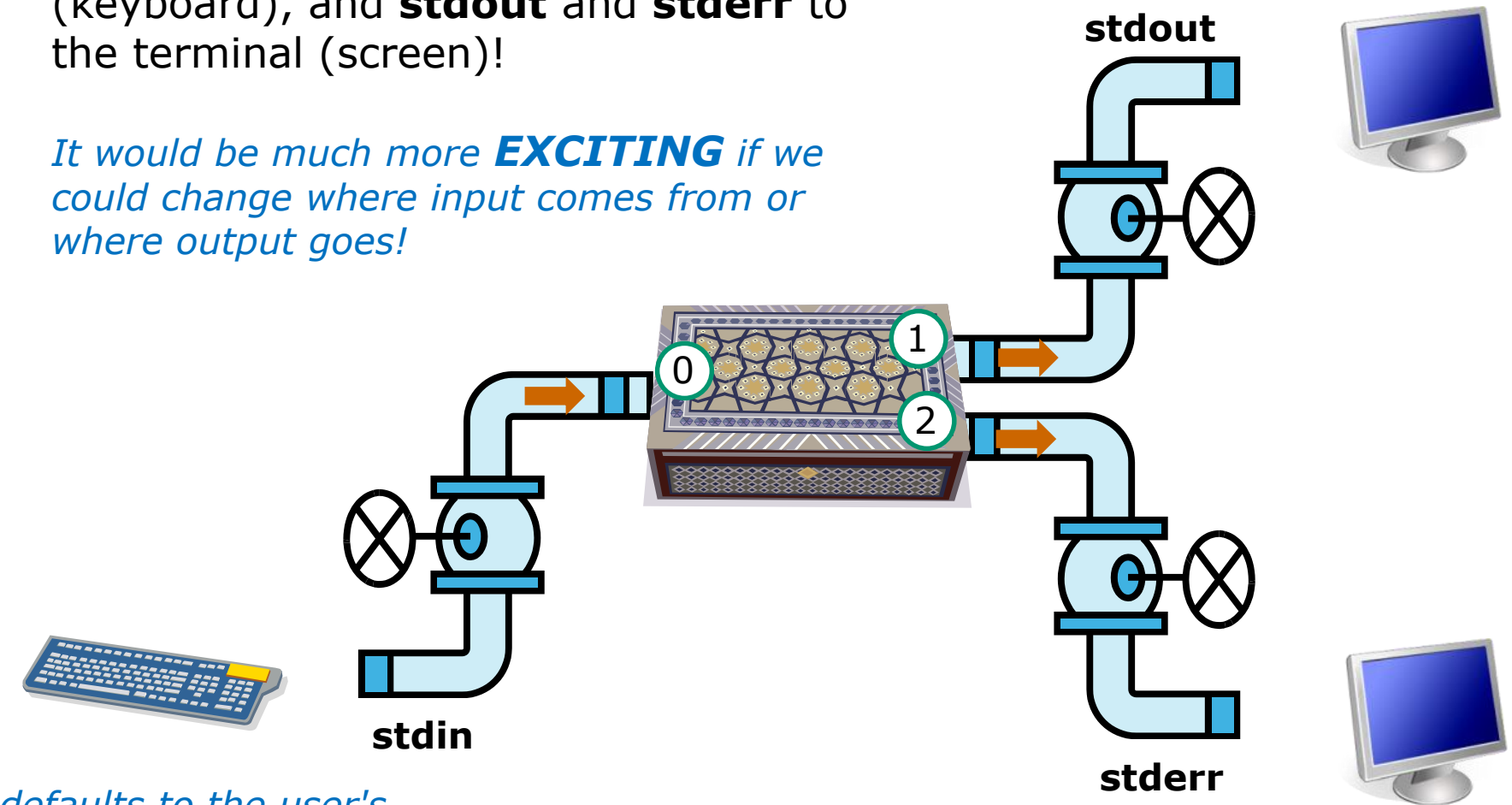
*Note, the sort process in this example reads from **stdin** and writes to **stdout.** The sort process is not aware what **stdin** or **stdout** are attached to*

File Redirection

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

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

defaults to the user's terminal screen



defaults to the user's terminal keyboard

defaults to the user's terminal screen

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 $
```

Input and Output
File Redirection

Read from **stdin**

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

*The sort program reads lines from **stdin** (attached to keyboard)*

"End of File"

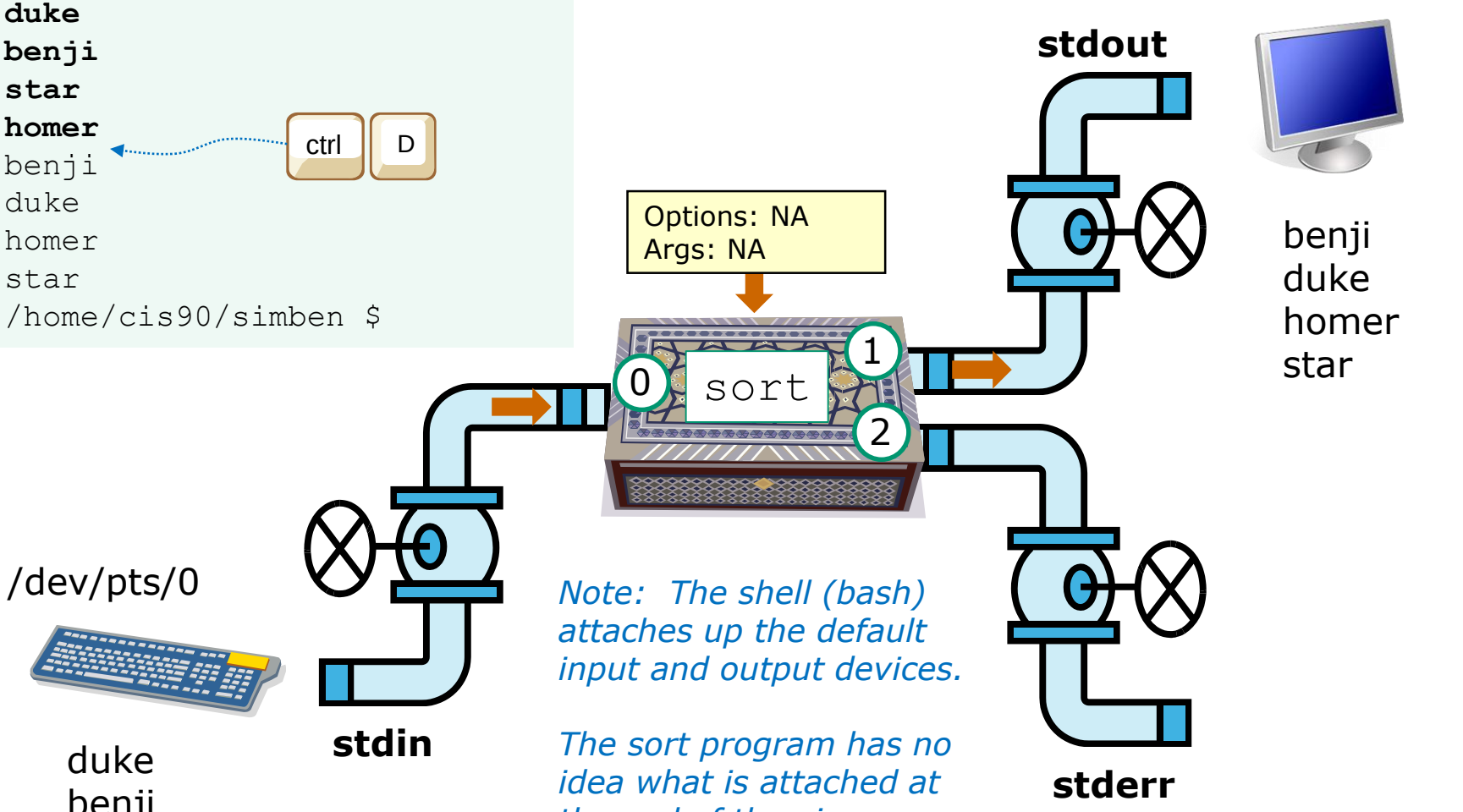
*After the EOF it performs the sort and writes to **stdout** (attached to terminal)*

Written to **stdout**

sort command (no arguments)

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

ctrl D



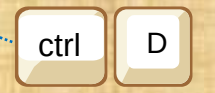
Note: The shell (bash) attaches up the default input and output devices.

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

Activity

Read from **stdin**

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



"End of File"

Written to **stdout**

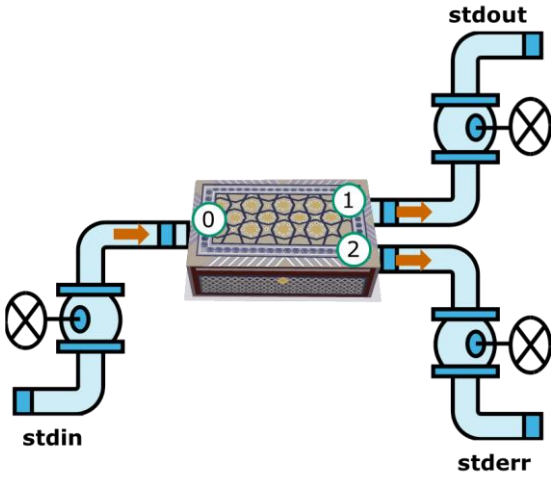
```
benji
duke
homer
star
/home/cis90/simben $
```

When YOU do this. What specific device is stdin and stdout attached to? Write your answer in the chat window.

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*

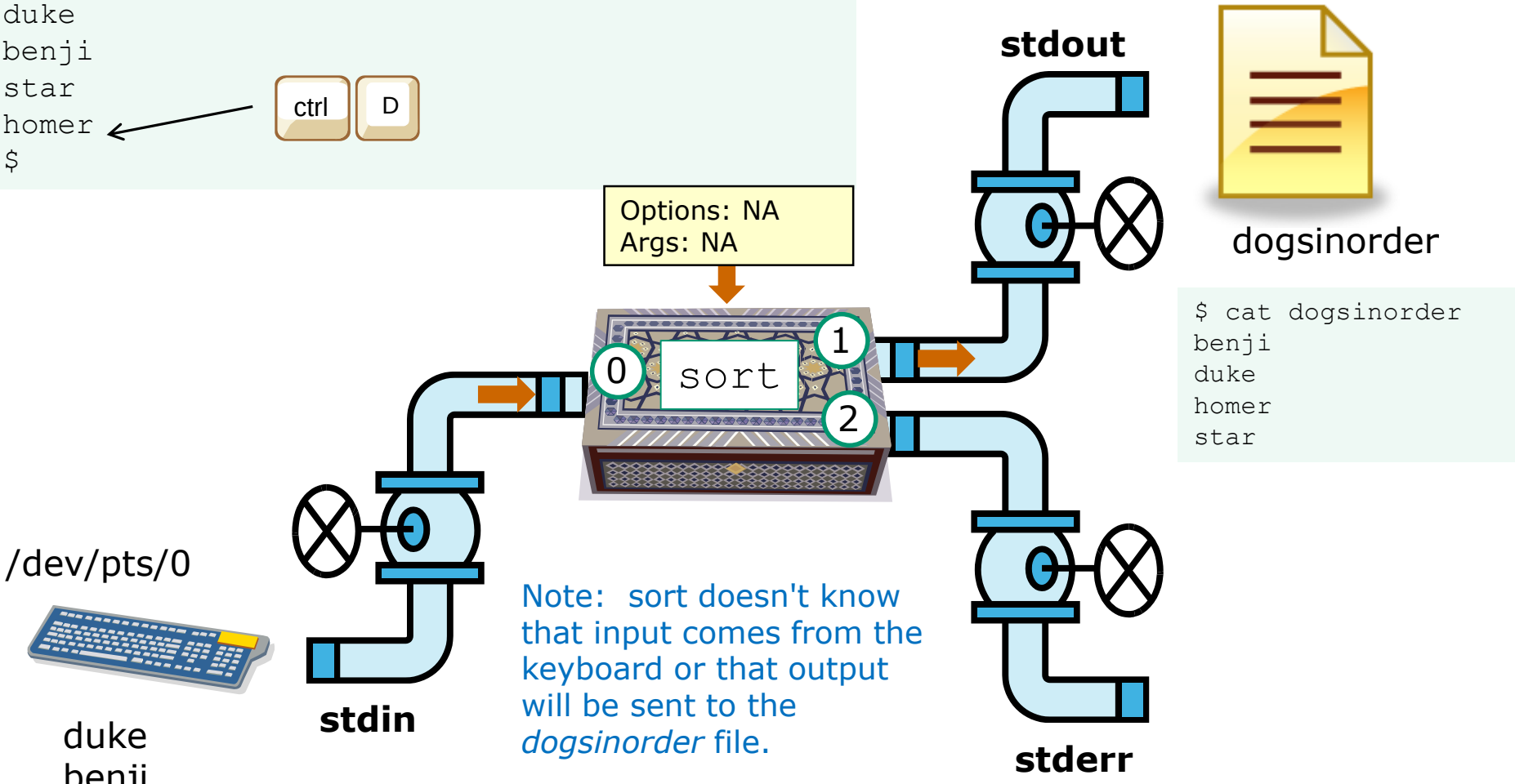
```
[simmsben@opus ~]$ sort > dogsinorder
duke
benji
star
homer
[simmsben@opus ~]$ cat dogsinorder
benji
duke
homer
star
[simmsben@opus ~]$
```

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

No arguments, redirecting stdout

```
$ sort > dogsinorder
duke
benji
star
homer
$
```

ctrl D



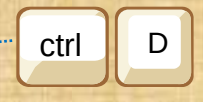

```
$ cat dogsinorder
benji
duke
homer
star
```

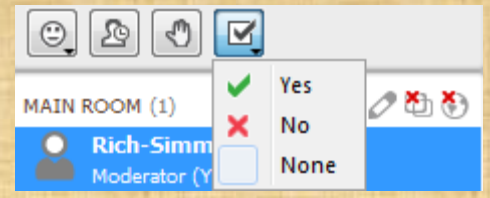
Note: sort doesn't know that input comes from the keyboard or that output will be sent to the *dogsinorder* file.

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

Now you try it

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





Give me a green Yes check if you get the same results

No arguments, redirecting stdin and stdout

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

input is redirected to come from the file *names*

output is redirected to the file *dogsinorder*

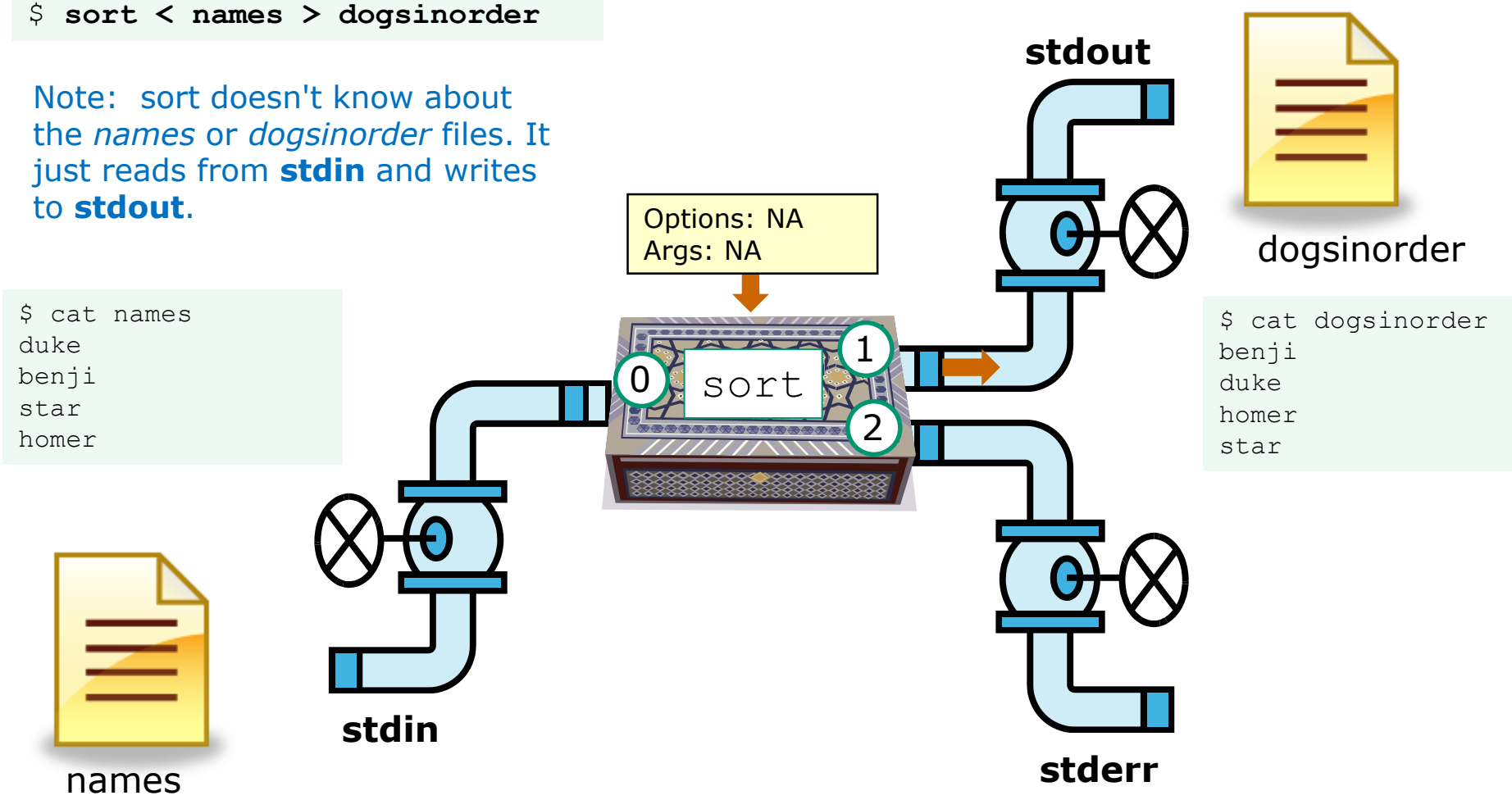
```
[simben@opus ~]$ cat dogsinorder
benji
duke
homer
star
[simben@opus ~]$
```

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
```

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

In this example, sort is getting it's input from **stdin**, which has been redirected to the *names* file

Now you try it

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


[simben@opus-ii ~]$ cat dogsinorder
benji
duke
homer
star
[simben@opus-ii ~]$
```

Does the **sort** program know that its input came from the *names* file or its output went to from the *dogsinorder* file?

Put your answer in the chat window.

One argument, redirecting stdout

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

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

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.

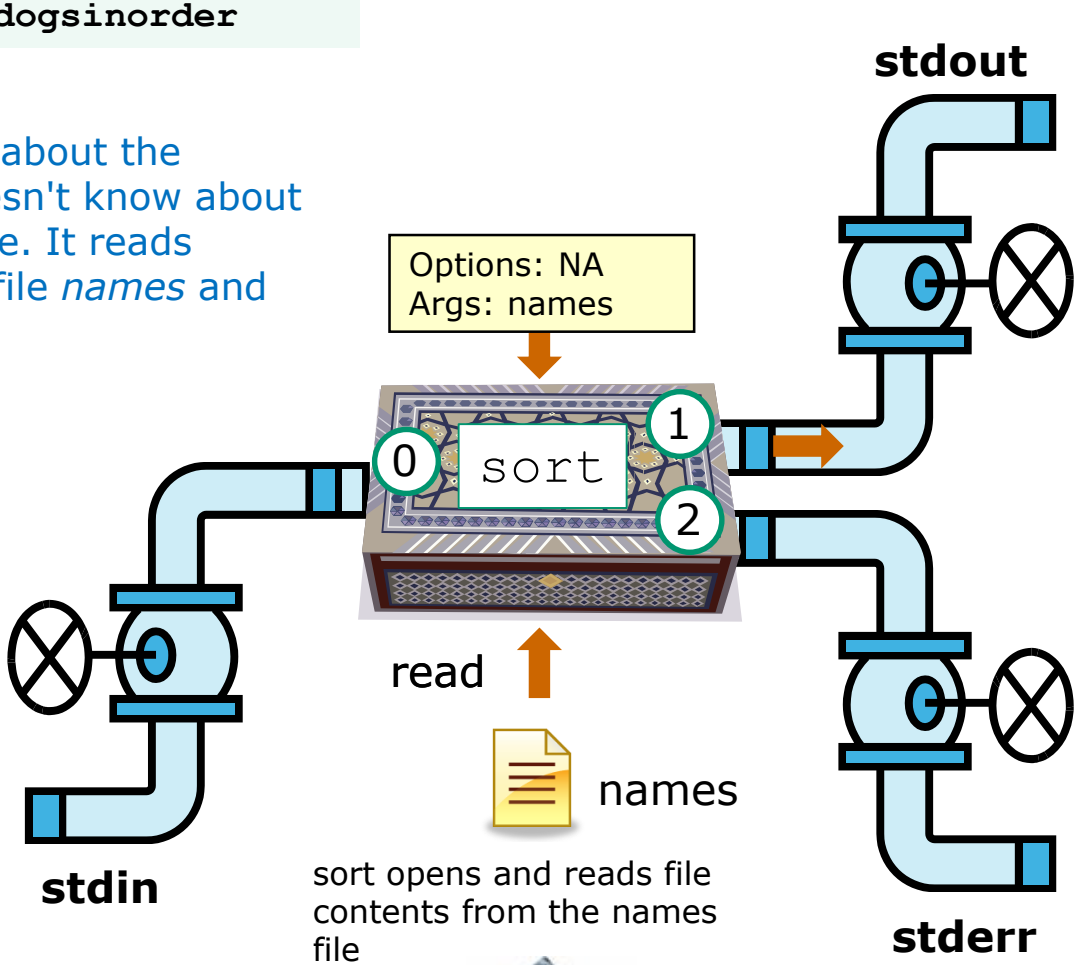
Корисне для наступного вікторини!

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!



dogsinorder

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

In this example, sort is getting it's input directly from the *names* file

Now you try it

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

Корисне для наступного вікторини!

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

Put your answer in the chat window

yes

One option, one argument, redirecting stdout

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

specifying an option (for reverse order)

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

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

This -r option does the sort in reverse order

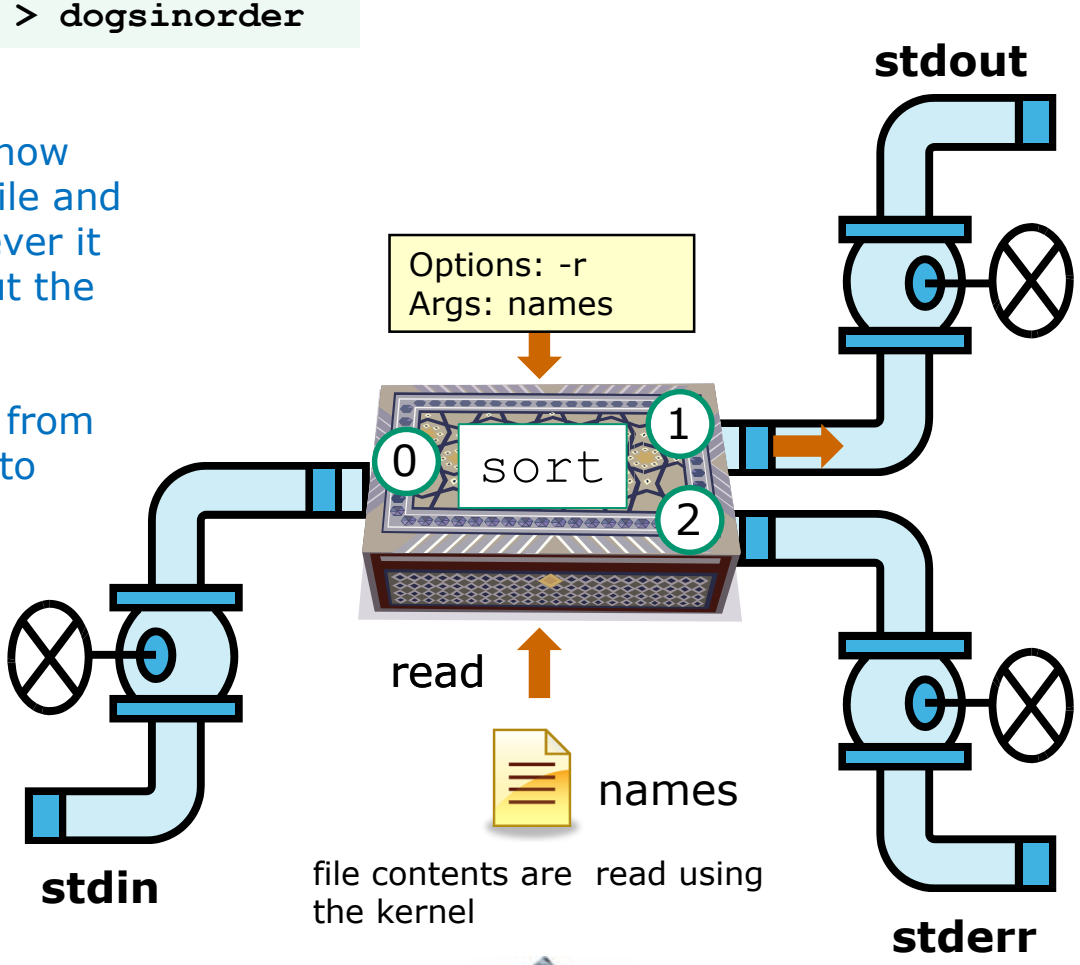
The shell opens the *dogsinorder* file. The sort process is not aware that output is redirected there.

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**.




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

In this example, sort is getting it's input directly from the names file

Now you try it

```
/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

no

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

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

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

Input from file, 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

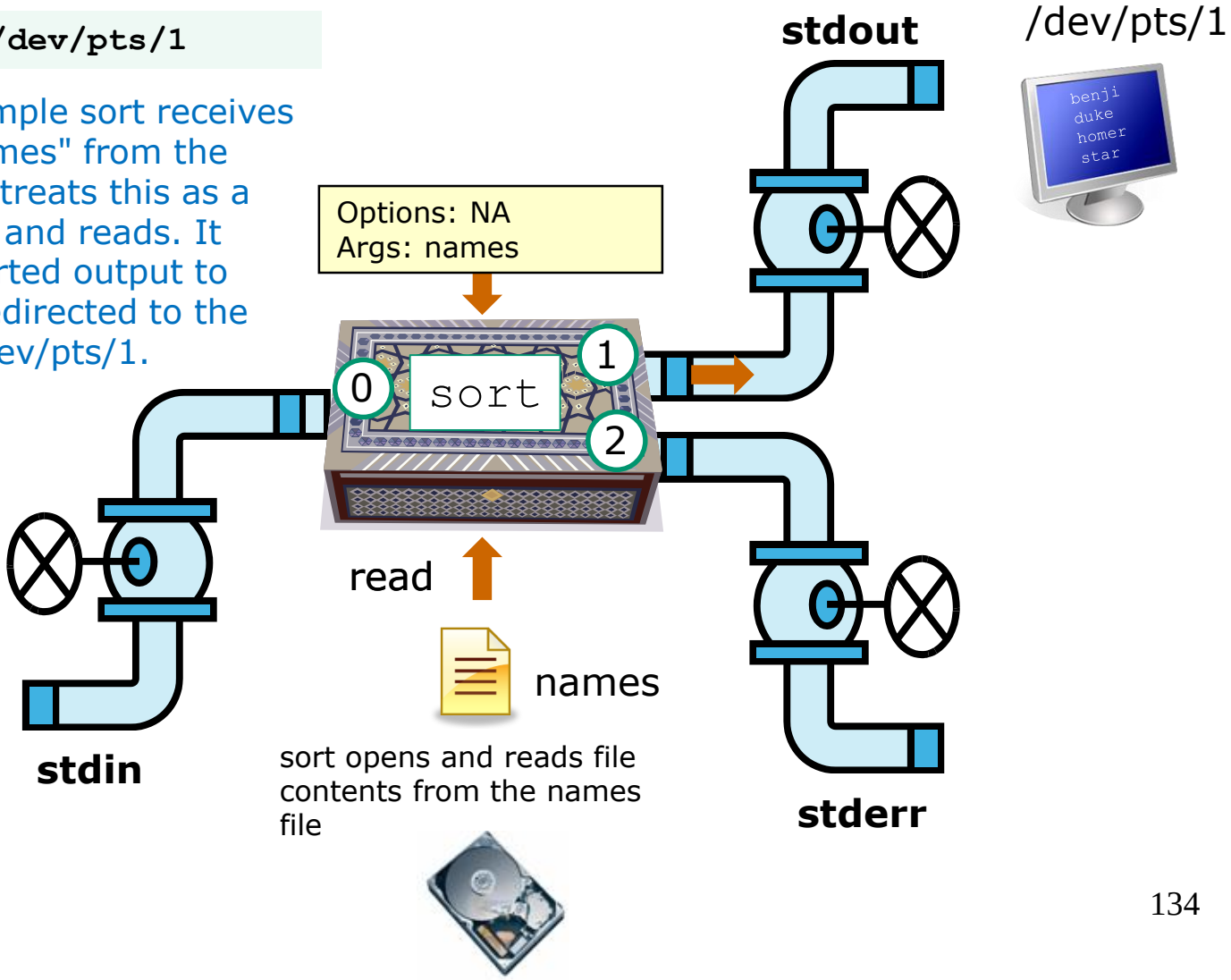
```
[simben@opus ~]$ tty
/dev/pts/1
[simben@opus ~]$ benji
duke
homer
star
```

Example 1 diagram

Input from stdin, redirecting stdout to another terminal device

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

Note: In this example sort receives the argument "names" from the command line. It treats this as a file which it opens and reads. It then writes the sorted output to **stdout** which is redirected to the terminal device /dev/pts/1.



Example 2

Input from the command line, redirecting stdout to file

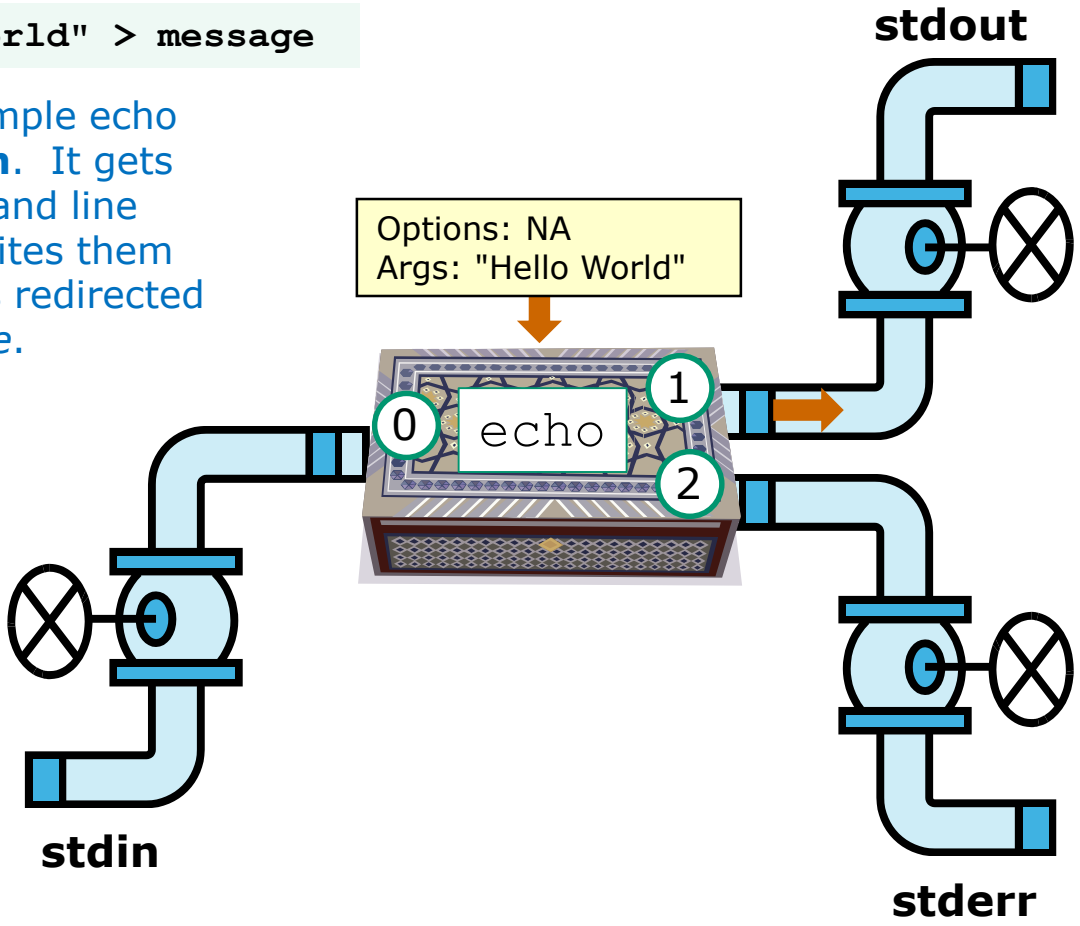
```
/home/cis90/simben $ echo "Hello World" > message
/home/cis90/simben $ cat message
Hello World
/home/cis90/simben $
```

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 as command line arguments and writes them to **stdout** which is redirected to the file *message*.



message

```
$ cat message
Hello World
```

Example 3

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 ~]$

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

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

*> redirects **stdout** to file named snapshot*

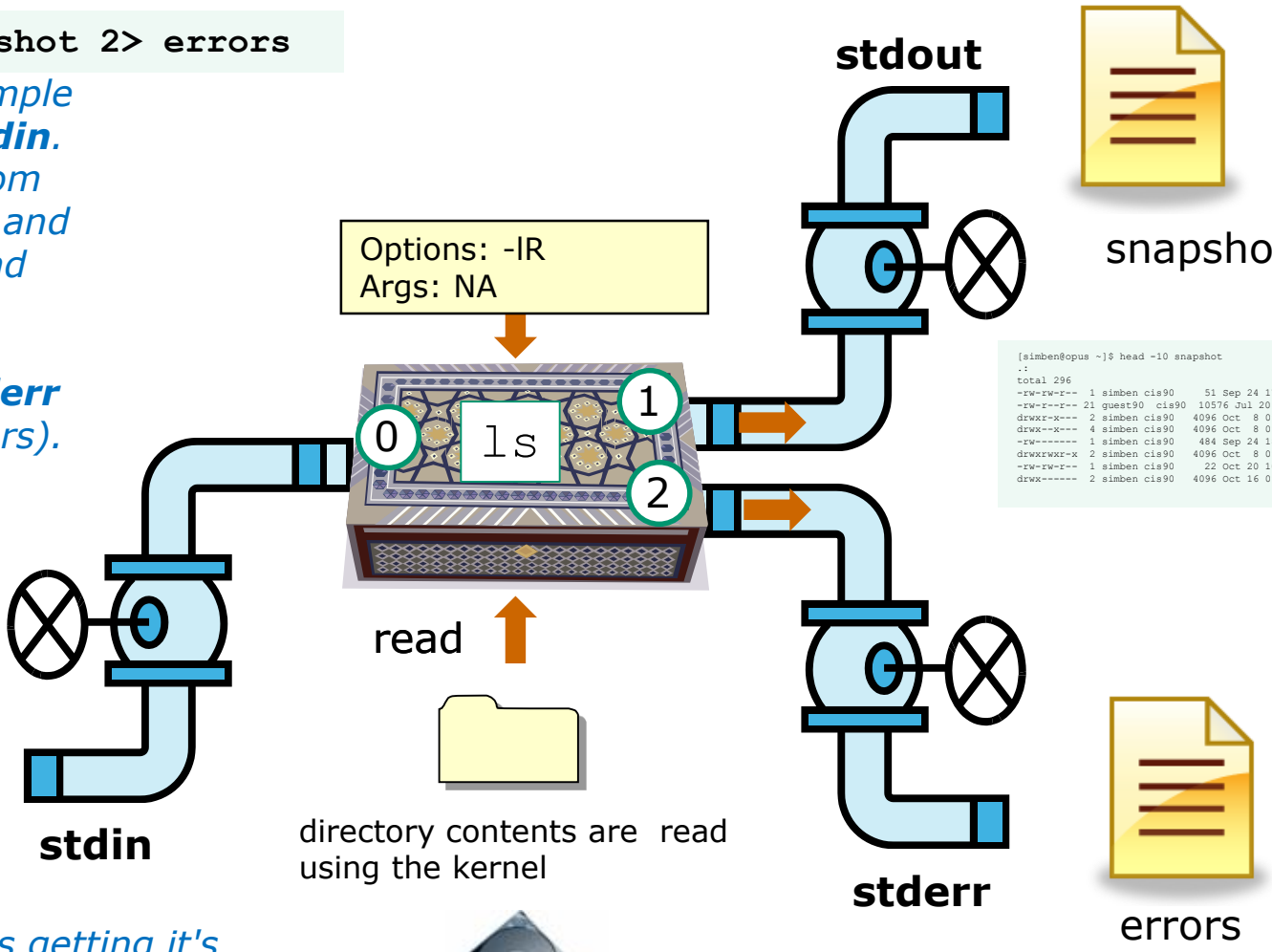
*2> redirects **stderr** to file named errors*

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 it's input from the OS

```
$ cat errors
ls: ./Hidden: Permission denied
```

Redirection Practice

Activity

```
/home/cis90/simben $ bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software
Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
2+2
4
4/0
Runtime error (func=(main), adr=5): Divide by zero
quit
/home/cis90/simben $
```

The bc command reads from stdin. It writes computed results to stdout and errors to stderr.

Activity

```
/home/cis90/simben $ echo 2+2 > math
/home/cis90/simben $ cat math
2+2
/home/cis90/simben $ bc < math
4
/home/cis90/simben $
```

Redirecting stdin to the math file

```
/home/cis90/simben $ echo 4/0 > math
/home/cis90/simben $ cat math
4/0
/home/cis90/simben $ bc < math
Runtime error (func=(main), adr=5): Divide by zero
/home/cis90/simben $
```

Redirecting stdin to the math file

Activity

```
/home/cis90/simben $ echo 2+2 > math
/home/cis90/simben $ echo 4/0 >> math
/home/cis90/simben $ cat math
2+2
4/0
/home/cis90/simben $ bc < math
4
Runtime error (func=(main), adr=5): Divide by zero
/home/cis90/simben $
```

Note that >> appends the output to stdout.

Activity

```
/home/cis90/simben $ cat math
2+2
4/0
/home/cis90/simben $ bc < math > answers 2> errors
/home/cis90/simben $
/home/cis90/simben $ cat answers
4
/home/cis90/simben $ cat errors
Runtime error (func=(main), adr=5): Divide by zero
/home/cis90/simben $
```

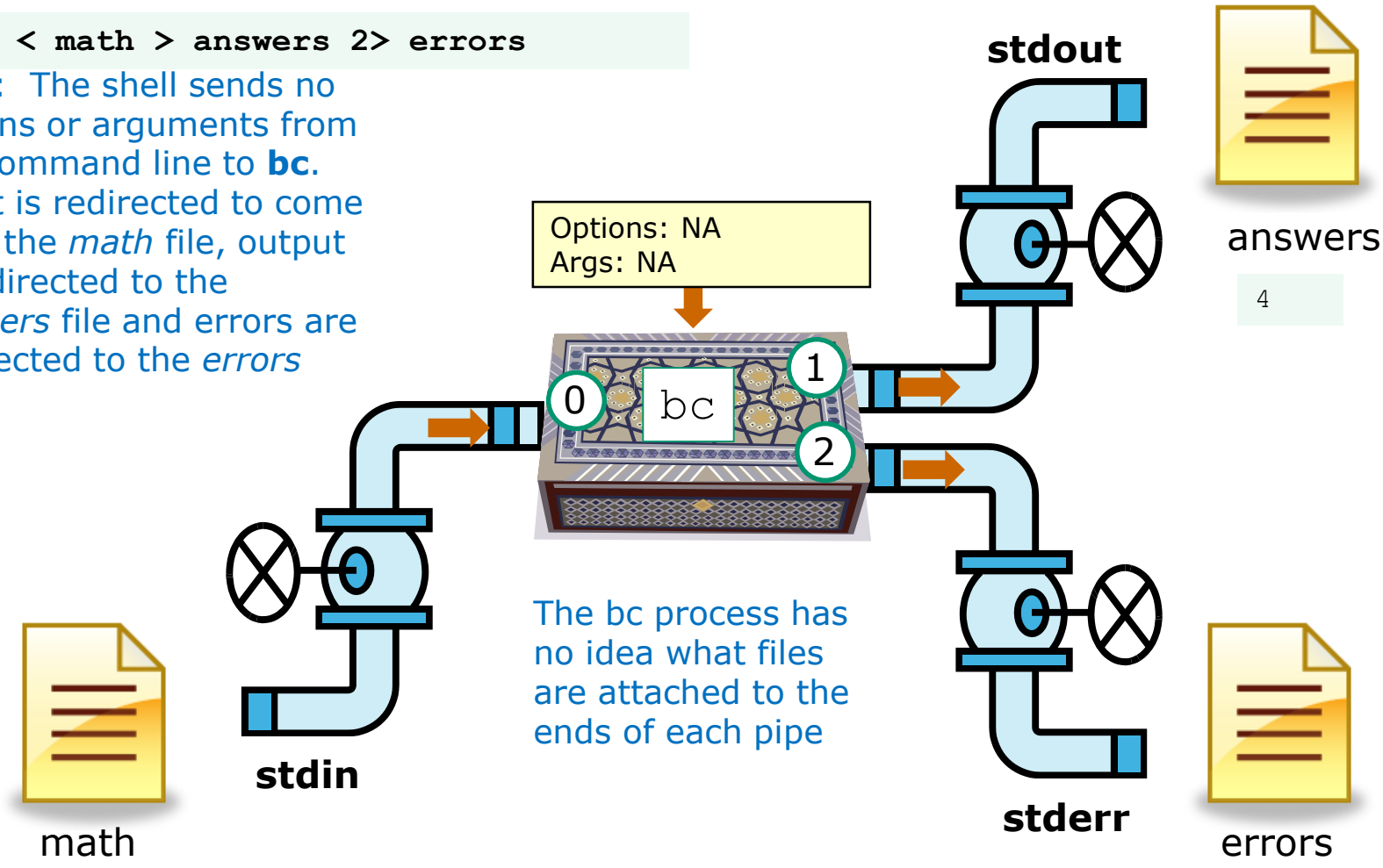
This time we redirect stdin, stdout and stderr!

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.

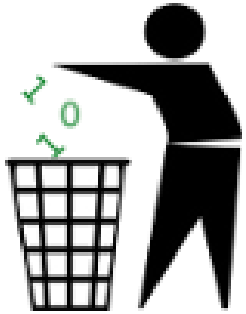


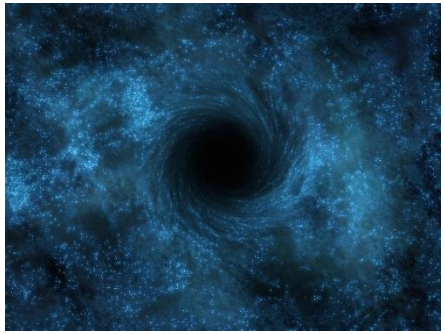
The bit bucket /dev/null

/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
```

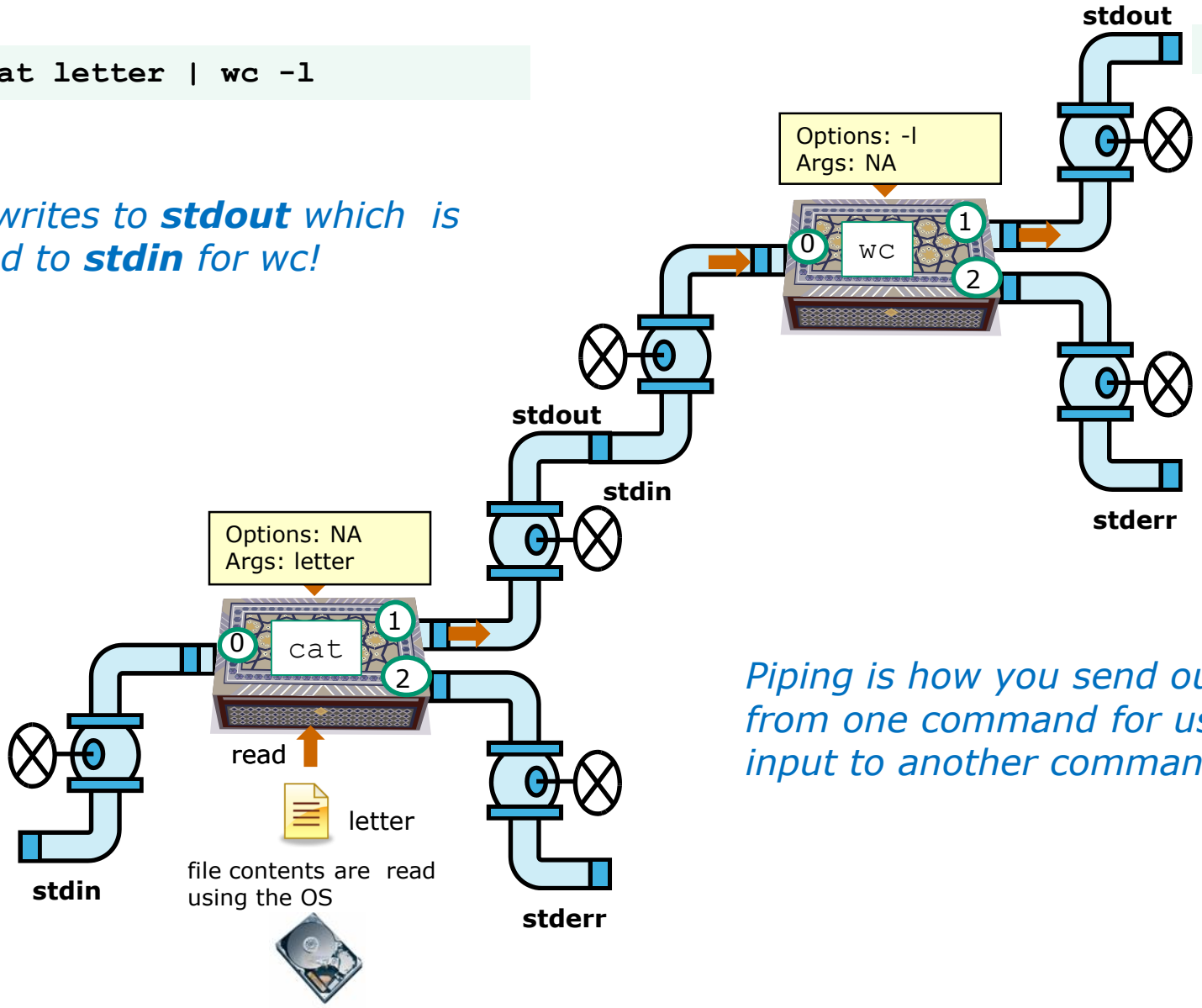
Counting the lines in the letter file

Counting lines in the letter file

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

```
28
```

*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
```

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
[simben@opus poems]$
```

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

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

<snipped>

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

Yuck! How annoying is this?

Redirecting find errors to the bit bucket

redirecting stderr to the "bit bucket"

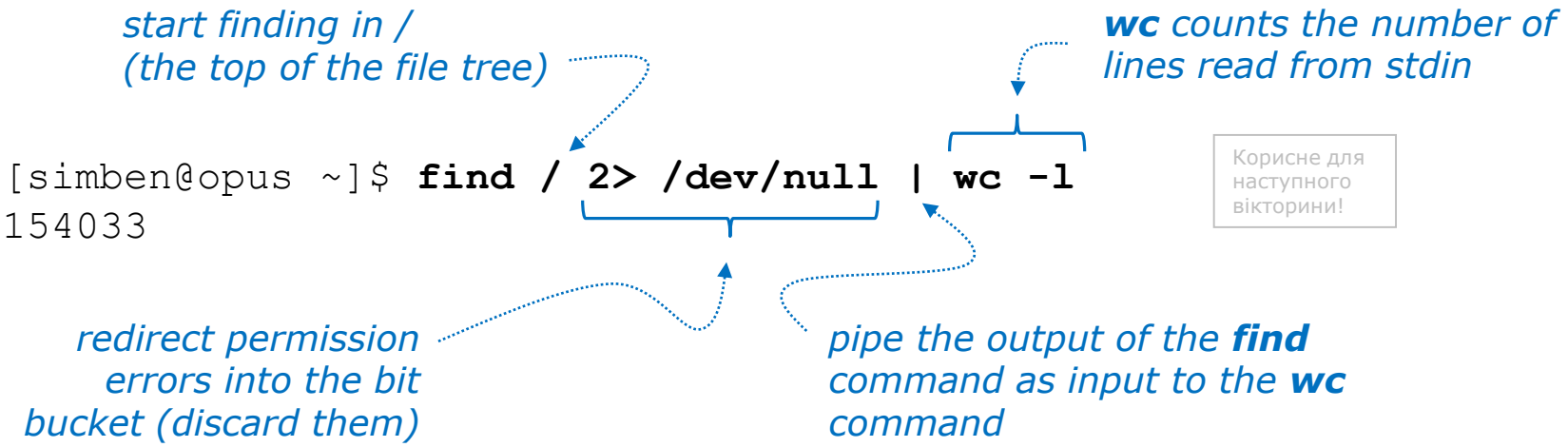
```
[simben@opus ~]$ find  /home/cis90 -name sonnet6  2> /dev/null
/home/cis90/guest/Poems/Shakespeare/sonnet6
/home/cis90/simben/poems/Shakespeare/sonnet6
/home/cis90/stanlcha/poems/Shakespeare/sonnet6
/home/cis90/seatocol/poems/Shakespeare/sonnet6
/home/cis90/wrigholi/poems/Shakespeare/sonnet6
/home/cis90/dymesdia/poems/Shakespeare/sonnet6
/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



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

The user that owns the files

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

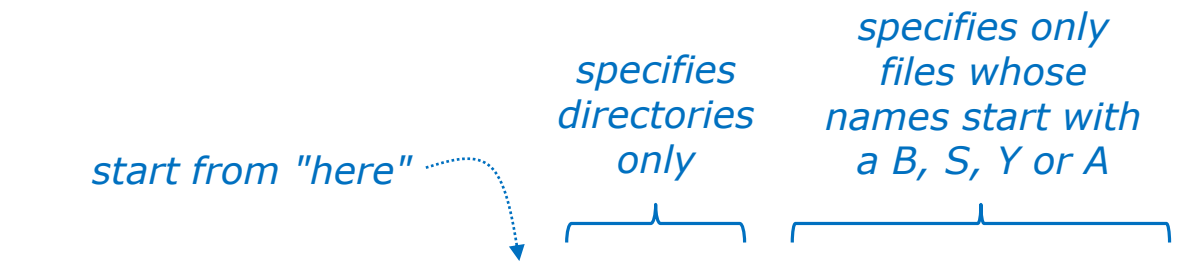
```
/home/cis90/simben $ find /home -type d -user milhom90 2> /dev/null
/home/turnin/cis90/milhom90
/home/cis90/milhom
/home/cis90/milhom/Hidden
/home/cis90/milhom/Lab2.0
/home/cis90/milhom/Miscellaneous
/home/cis90/milhom/bin
/home/cis90/milhom/Poems
/home/cis90/milhom/Poems/Shakespeare
/home/cis90/milhom/Poems/Yeats
/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




```
[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

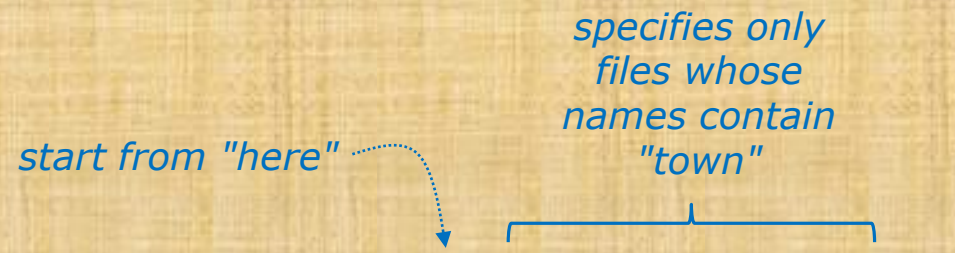
```
/home/cis90/simben $ find   -type f   -name '*k*'   -exec file {} \;
find: `./Hidden': Permission denied
./edits/spellk: ASCII English text
./.kshrc: ASCII text
./docs/MarkTwain: ASCII English text
./.ssh/known_hosts: ASCII text, with very long lines
/home/cis90/simben $
```

The {} are replaced by filenames as they are found

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

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

Now you try it




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

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

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 string "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,
```

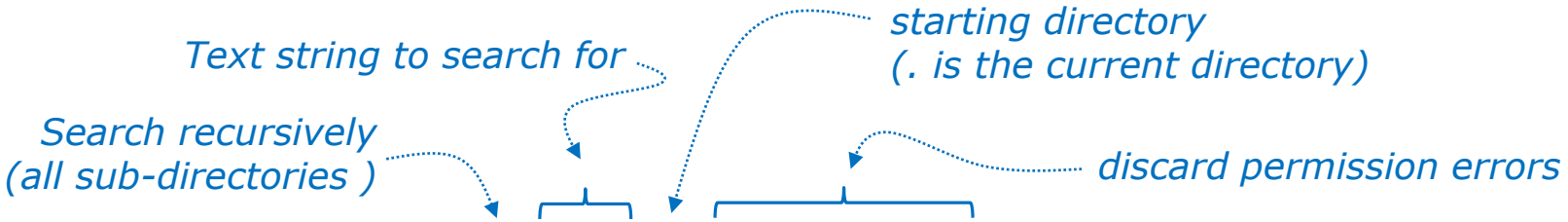
Look for "so" in sonnet3, sonnet4 and sonnet5

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




```
/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-II. Try it, you can browse to opus-ii.cis.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 GIF file in the Apache's document root directory on Opus-II?
  *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
Daubeny
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

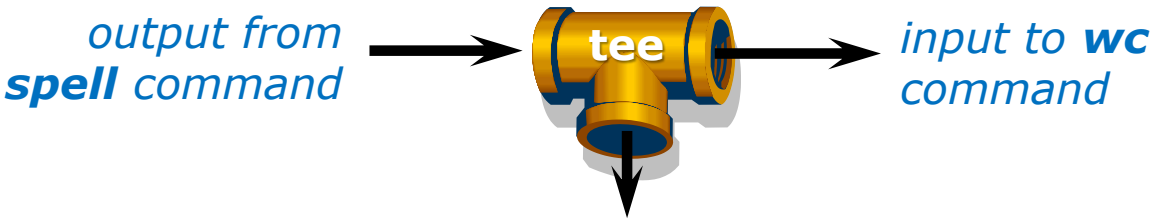
```
/home/cis90/simben $ head edits/spellk
Spell Check

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
1234567890123456789012345678901234567890123456789012345678901234567890
```

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
```

How many times did you log in on a Wednesday?
Write your answer in the chat window.

Class Exercise
Pipeline Tasks

Background
The **cut** command can cut a field out of a line of text where each field is delimitated 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?
Write your answer in the chat window.

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.

green background with black text

```
/home/cis90/simben $ ls -ld /tmp
drwxrwxrwt. 3 root root 4096 Oct 16 16:13 /tmp
```

```
/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

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.

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

ACLs (Access Control Lists)

Let's give special permissions to one user

modify

```
/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::---
```

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 $
```

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
```

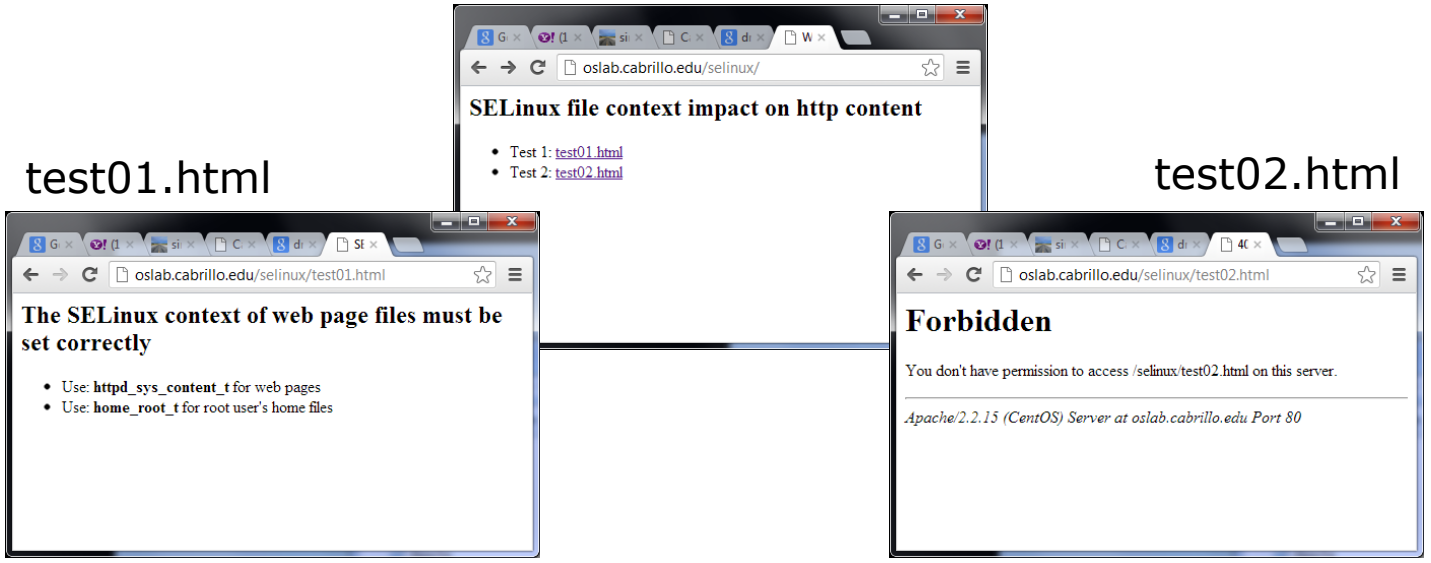
Note, the root user's home files are not appropriate web content

FYI only

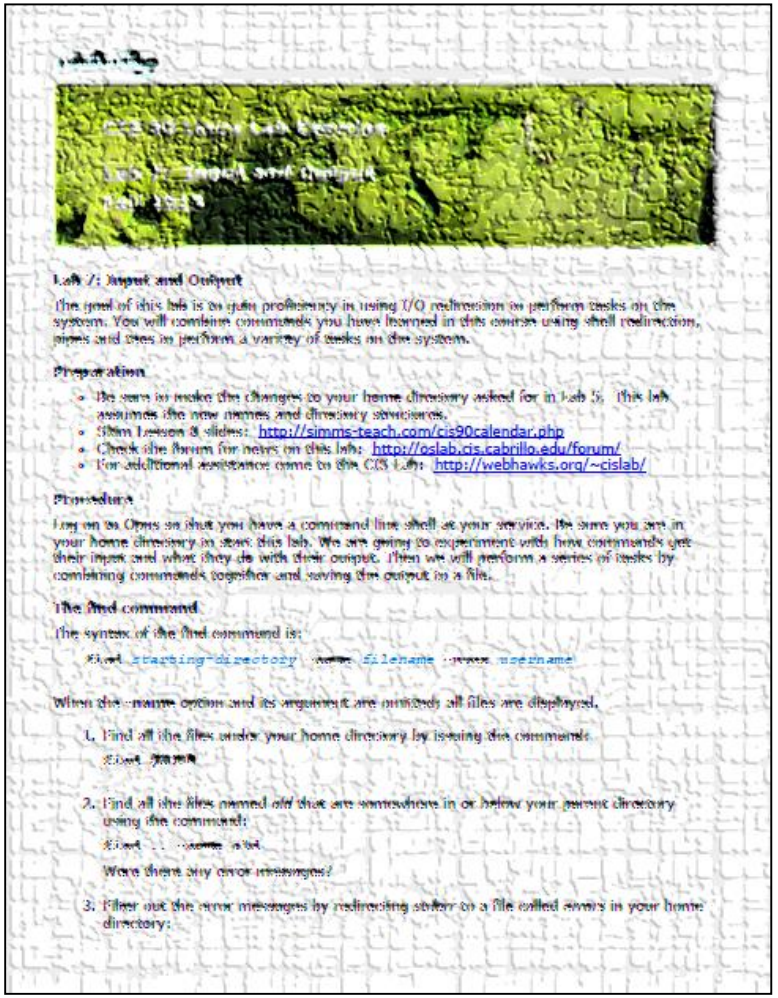
SELinux context

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]#
```



Assignment



Lab 7: Input and Output

The goal of this lab is to gain proficiency in using I/O redirection to perform tasks on the system. You will combine commands you have learned in this course using shell redirection, pipes and tees to perform a variety of tasks on the system.

Preparation

- Be sure to make the changes to your home directory asked for in Lab 5. This lab assumes the new names and directory structures.
- Slides & Lesson: http://simms-teach.com/cis90calendar.php
- Check the forum for news on this lab: http://oslab.cis.cabrillo.edu/forum/
- For additional assistance come to the CIS Lab: http://webhawks.org/~cislab/

Procedure

Log on to Opus so that you have a command line shell at your service. Be sure you are in your home directory as that is where this lab. We are going to experiment with how commands get their input and what they do with their output. Then we will perform a series of tasks by combining commands together and saving the output in a file.

The find command

The syntax of the find command is:

```
find starting-directory -name filename -exec cmdname
```

When the -name option and its argument are omitted all files are displayed.

1. Find all the files under your home directory by issuing the command:

   ```
   find ~
   ```

2. Find all the files named old that are somewhere in or below your current directory using the command:

   ```
   find . -name old
   ```

   Were there any error messages?

3. Filter out the error messages by redirecting stderr to a file called errors in your home directory:

Lab 7

If you get stuck please ask questions on the forum or ask the Lab Assistants in the CIS Lab.

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

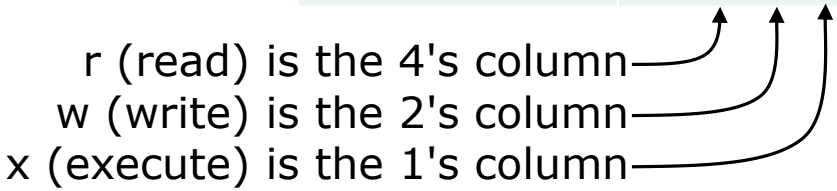
Permissions Review

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



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
```

*The **group***

*The **user***

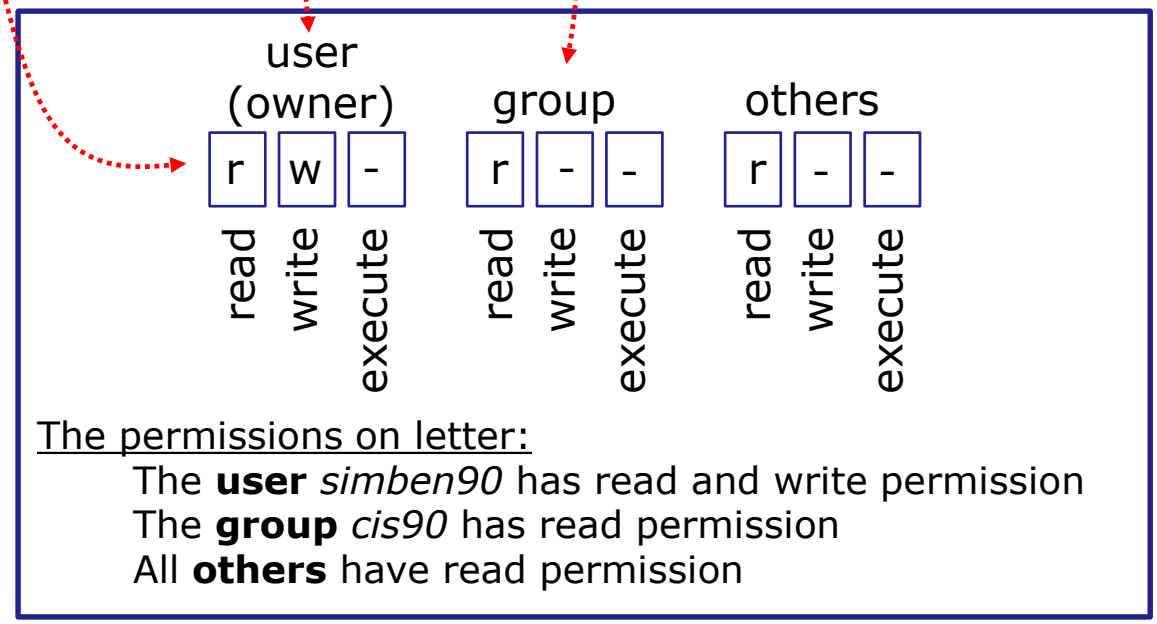
*Permissions that apply to **others***

*Permissions that apply to the **group***

*Permissions that apply to the **user***

File Permissions

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



The permissions on letter:

The **user** *simben90* has read and write permission
The **group** *cis90* has read permission
All **others** have read permission

Use long listings to show permissions

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 **group***

*The **user***

*Permissions that apply to **others***

*Permissions that apply to the **group***

*Permissions that apply to the **user***

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
```

*The **group***

*The **user***

*Permissions that apply to **others***

*Permissions that apply to the **group***

*Permissions that apply to the **user***

Does the simben90 user have execute permission on the letter file?
No

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 **group***

*The **user***

*Permissions that apply to **others***

*Permissions that apply to the **group***

*Permissions that apply to the **user***

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 **group***

*The **user***

*Permissions that apply to **others***

*Permissions that apply to the **group***

*Permissions that apply to the **user***

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

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 **group***

*The **user***

*Permissions that apply to **others***

*Permissions that apply to the **group***

*Permissions that apply to the **user***

Does the zamhum90 user have read 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 **group***

*The **user***

*Permissions that apply to **others***

*Permissions that apply to the **group***

*Permissions that apply to the **user***

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

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 **group***

*The **user***

*Permissions that apply to **others***

*Permissions that apply to the **group***

*Permissions that apply to the **user***

Does the smimat172 user have read 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 **group***

*The **user***

*Permissions that apply to **others***

*Permissions that apply to the **group***

*Permissions that apply to the **user***

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
```

*Using **numeric** permissions format*

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
```


*Using **mnemonic** permissions format*

Tools for managing permissions

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