Lesson Module Checklist

- Slides
- WB

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

- Lab 7 tested
- Lab X1 tested

- 9V backup battery for microphone
- Backup slides, CCC info, handouts on flash drive

**Student checklist**

1) Browse to the CIS 90 website Calendar page
   - http://simms-teach.com
   - Click CIS 90 link on left panel
   - Click Calendar link near top of content area
   - Locate today's lesson on the Calendar

2) Download the presentation slides for today's lesson for easier viewing

3) Click Enter virtual classroom to join CCC Confer session

4) Connect to Opus using Putty or ssh command

# Introductions and Credits

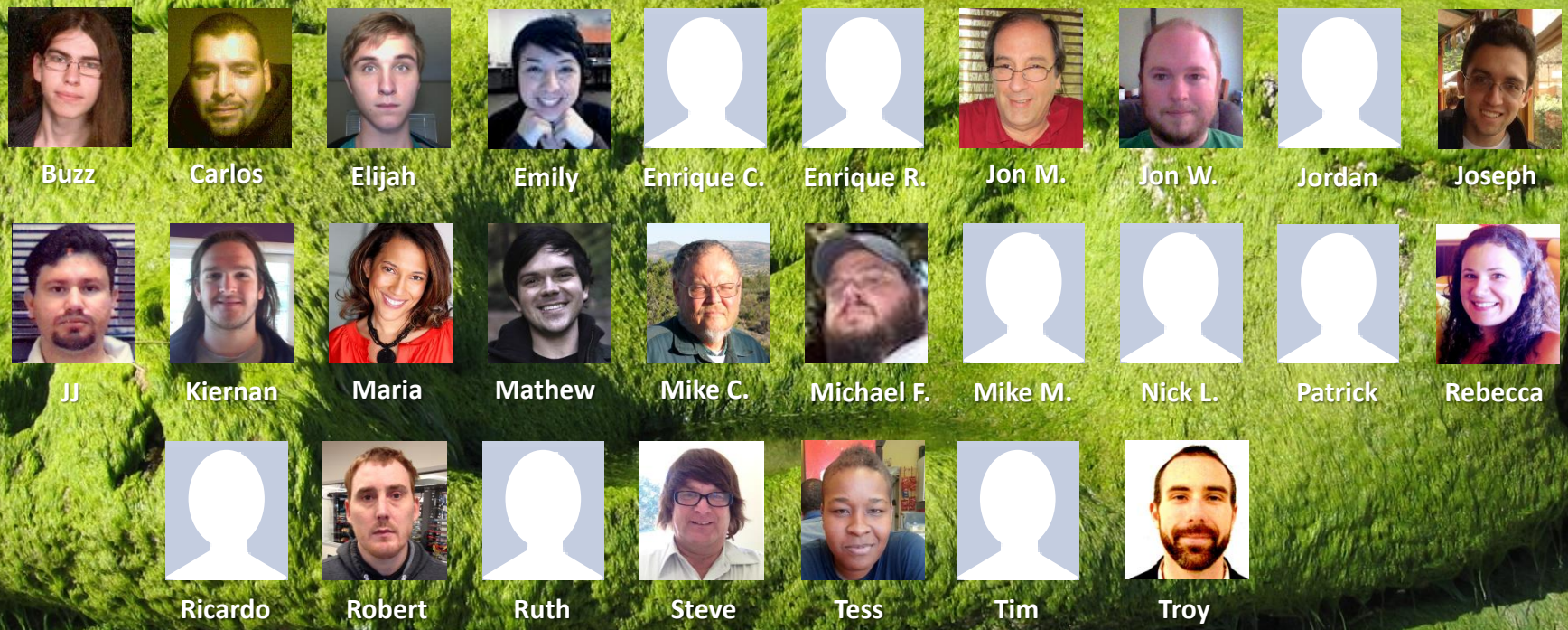
Jim Griffin
- Created this Linux course
- Created Opus and the CIS VLab
- Jim's site: http://cabrillo.edu/~jgriffin/

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/)

# CIS 90 - Lesson 8

Instructor: **Rich Simms**
Dial-in: **888-450-4821**
Passcode: **761867**




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

# Quiz

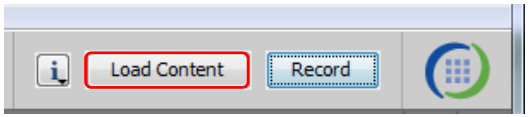
Please answer these questions **in the order** shown:

**See electronic white board**

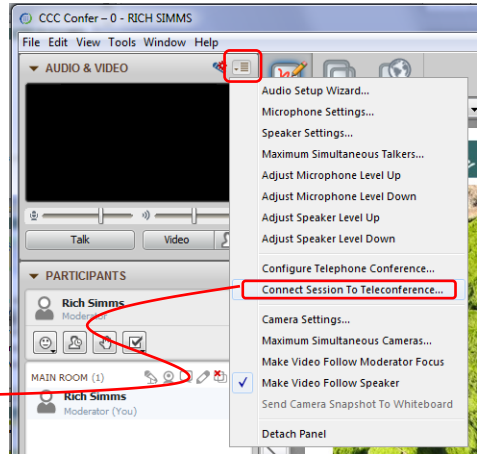
**email answers to: `risimms@cabrillo.edu`**

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

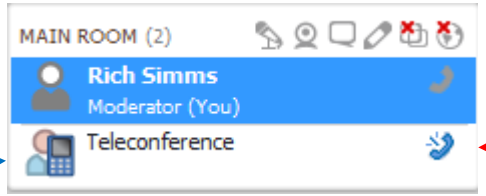

## [ ] Preload White Board with *cis*lesson??*-WB*



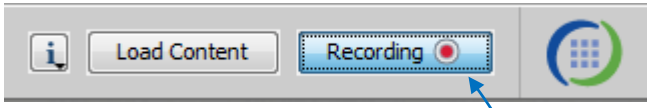

## [ ] Connect session to Teleconference




*Session now connected to teleconference*

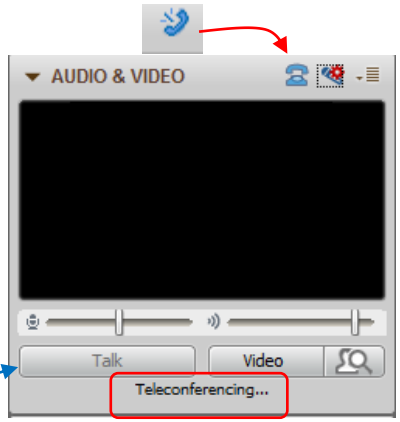



## [ ] Is recording on?




*Red dot means recording*

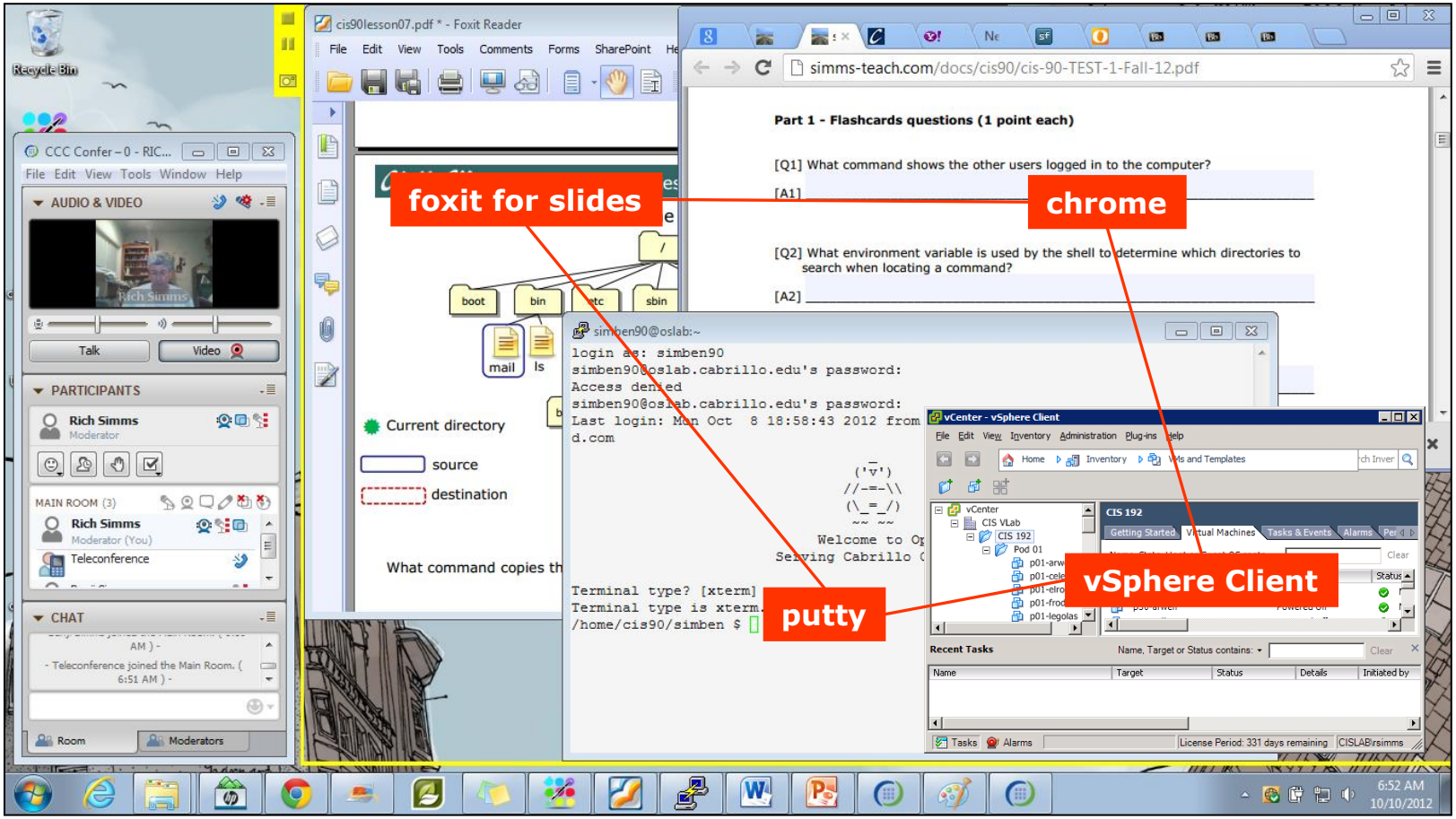
## [ ] Use teleconferencing, not mic




*Should be greyed out*

*Keep wireless mic tranmitter away from cell phone and podium if excess static occurs*

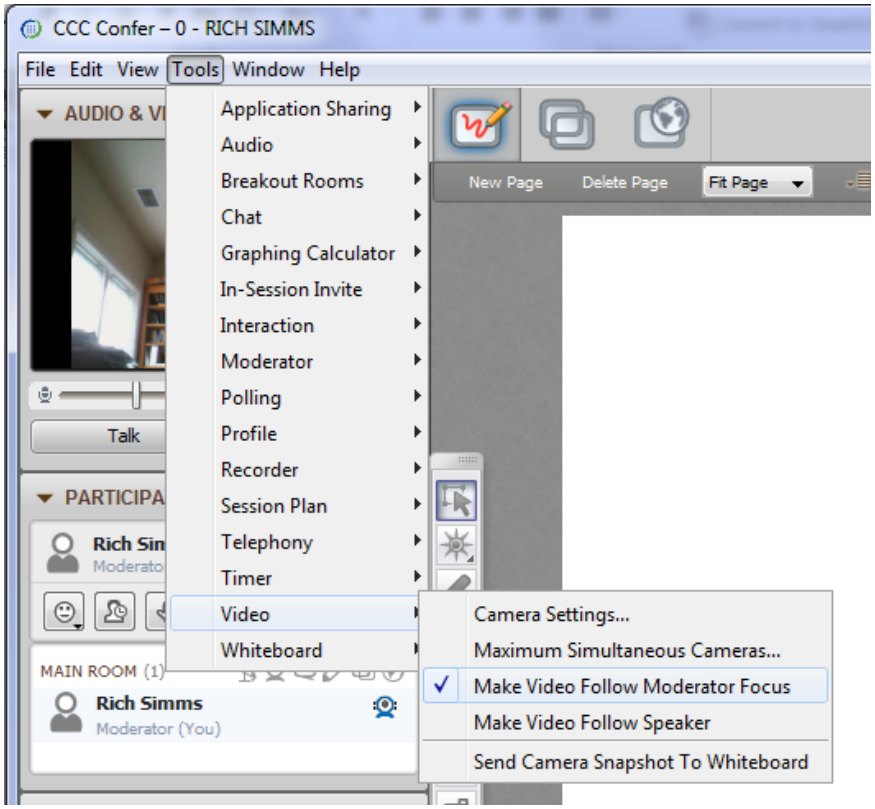
# [ ] layout and share apps



[ ] Video (webcam) optional

[ ] Follow moderator

[ ] Double-click on postages stamps

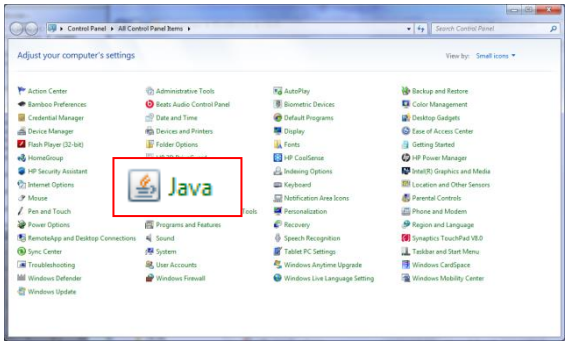


**Universal Fix for CCC Confer:**

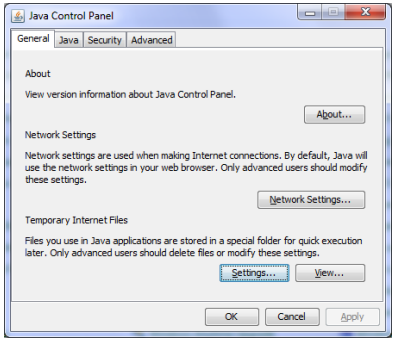
1) Shrink (500 MB) and delete Java cache
2) Uninstall and reinstall latest Java runtime



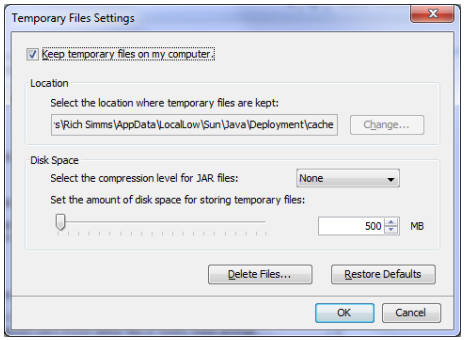
Control Panel (small icons)



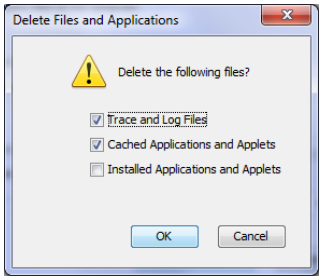

General Tab > Settings…




500MB cache size




Delete these




Google Java download

# Input/Output Processing

| Objectives | Agenda |
| --- | --- |
| • Identify the three open file descriptors an executing program is given when started.<br>• Be able to redirect input from files and output to files<br>• Define the terms pipe, filter, and tee<br>• Use pipes and tees to combine multiple commands<br>• Know how to use the following useful UNIX commands:<br>o find<br>o grep<br>o wc<br>o sort<br>o spell | • Quiz<br>• Questions<br>• Warmup<br>• Housekeeping<br>• Review<br>• File descriptors<br>• Pipelines<br>• New commands<br>• Tasks using pipelines |

# Questions

# Questions?

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

Lesson material?

Labs? Tests?

How this course works?

*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。<br>*He who asks a question is a fool for five minutes; he who does not ask a question remains a fool forever.* |

# Lab 5
# Post Mortem

**Moving Files**
Step 4 - no datecal in bin/ (x)
Step 11 - no better_town or what_am_i in edits/ (xxxx)

**Renaming Files**
Step 2 - Poems not renamed to poems (x)
Step 3 - proposal1 not renamed (x)

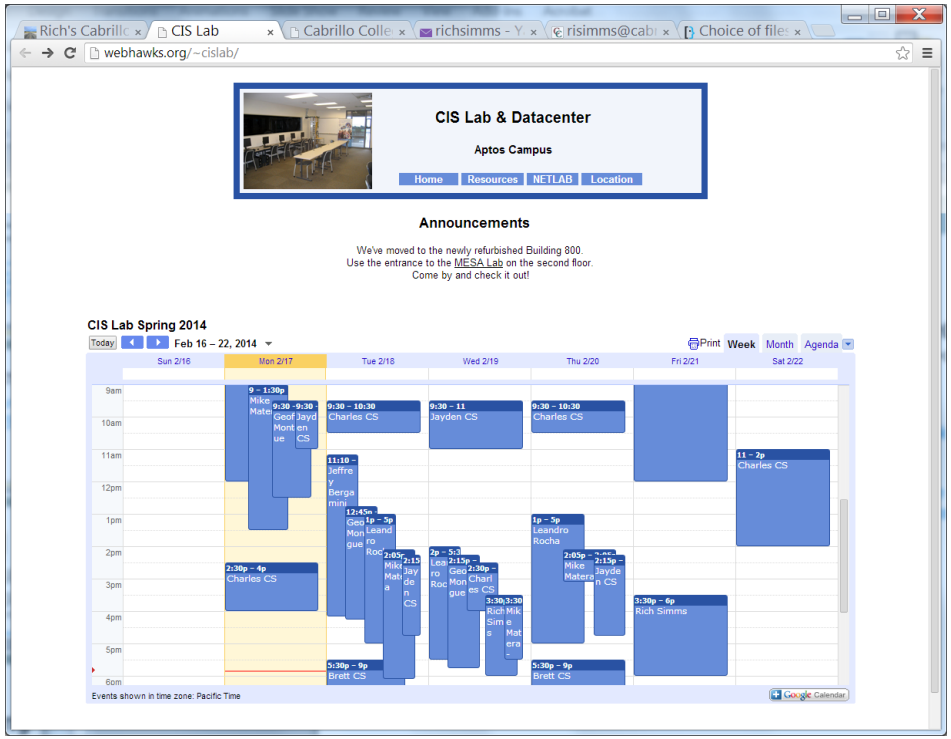
**Copying Files**
Step 2 - hosts not found in etc/ (x)
Step 3 - no graded labs in class/labs (xx)
Step 4 - sonnet6 not copied (xxx)

**Removing Files**
Step 2 - empty not removed (xx)
Step 6 - Lab2.0 not removed (x)
Step 7 - Lab2.1 not removed (x)
Step 8 - Sonnets not removed

**Linking Files**
Step 1 - bigfile not linked (xx)
Step 2 - motd not linked (xxx)

# CIS Lab Schedule

http://webhawks.org/~cislab/




Not submitting tests or lab work?

Would like some help?

Come to the CIS Lab to work with classmates, lab assistants and instructors on Lab assignments.

*Rich is in the lab Wednesdays and Fridays from 3:30 - 6:00 PM*

# Free CIS 90 Tutoring Available

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


Matt Smithey

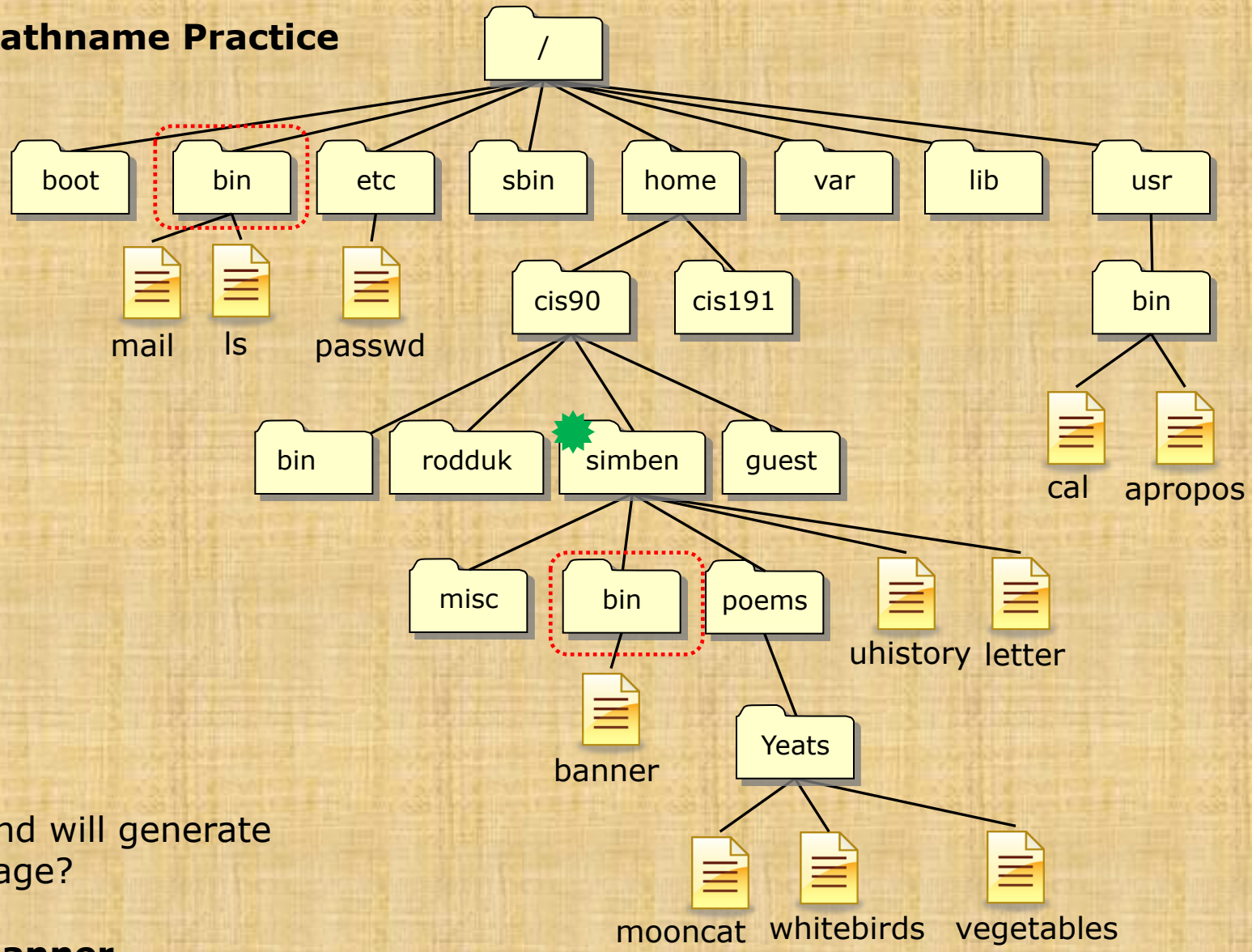
All students interested in tutoring in CIS 90, 172, and 81 classes need to come directly to the Tutorials Center to schedule, register and fill out some paperwork. This is just a one-time visit.

The tutoring will take place at the STEM center and they will log in and log out on a computer you have designated (I will figure out exactly what that means).

Matt is available M: 9:00-5:00, T: 9-11 and 2-5, Wed: 9-12 and Th: 9-11 and 3-5.

# Warmup

## File Tree Pathname Practice

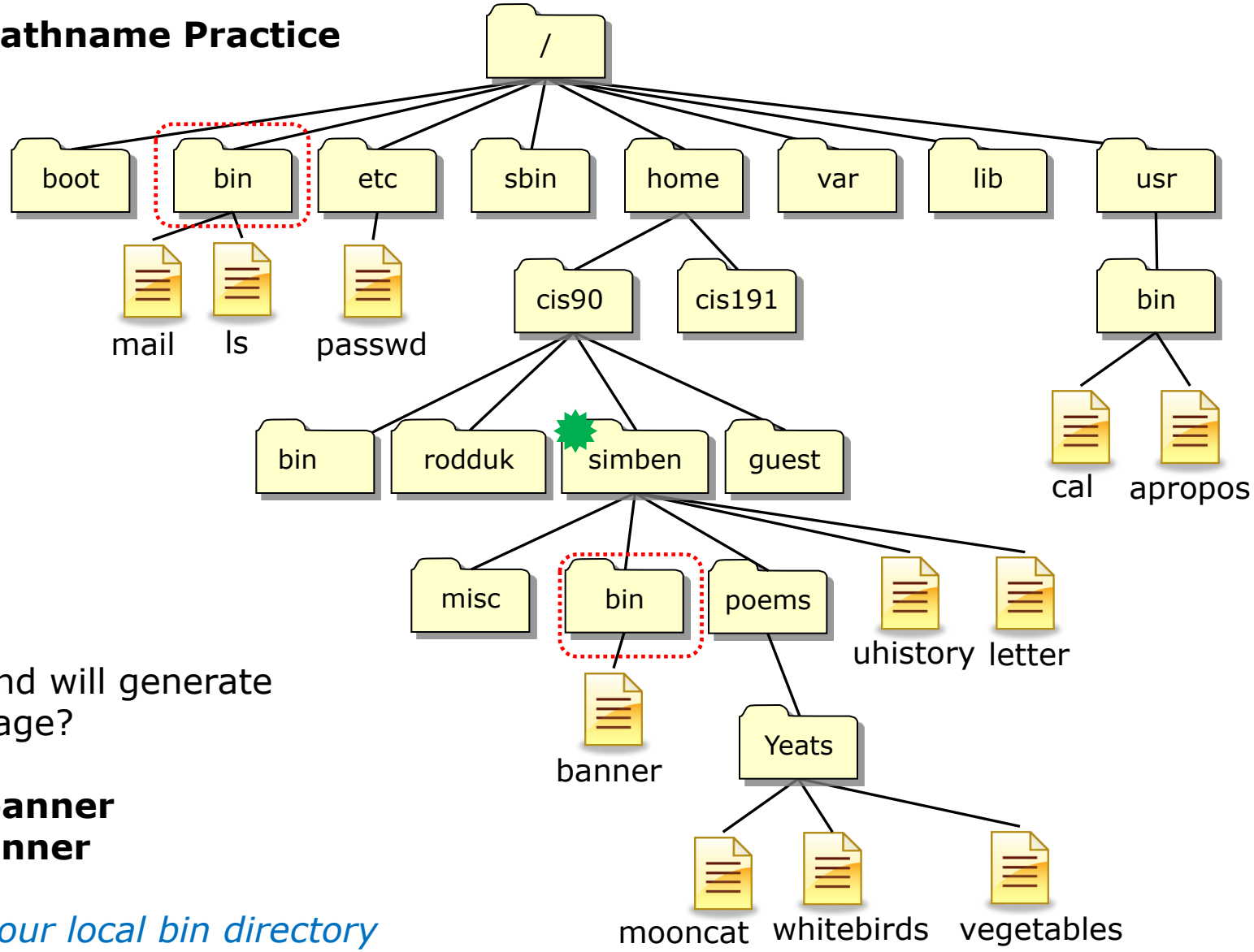



From ✸ how does Benji:

Which command will generate an error message?

**touch /bin/banner**
**touch bin/banner**

## File Tree Pathname Practice




From ✸ how does Benji:

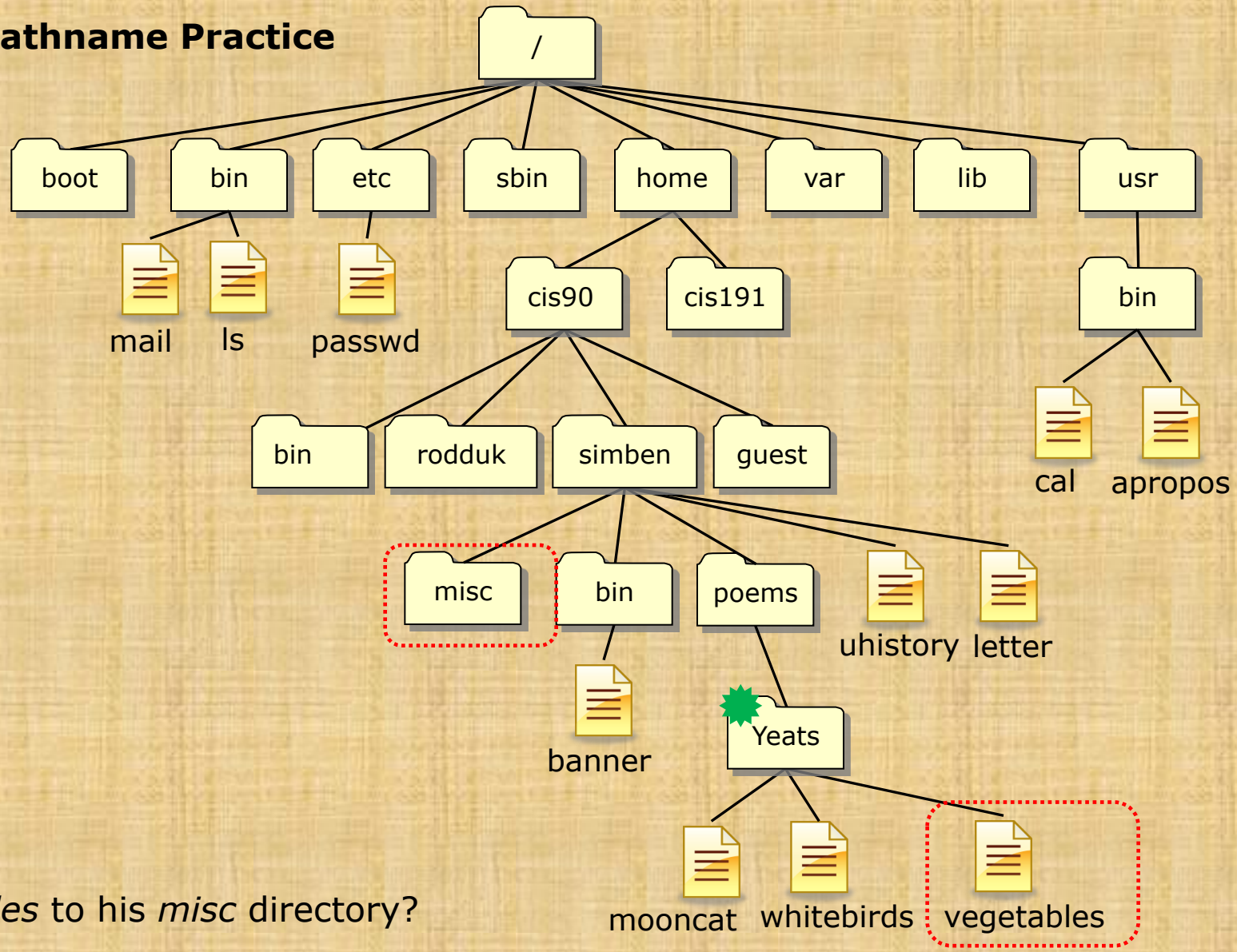
Which command will generate an error message?

**touch /bin/banner**
**touch bin/banner**

*banner is in your local bin directory*

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

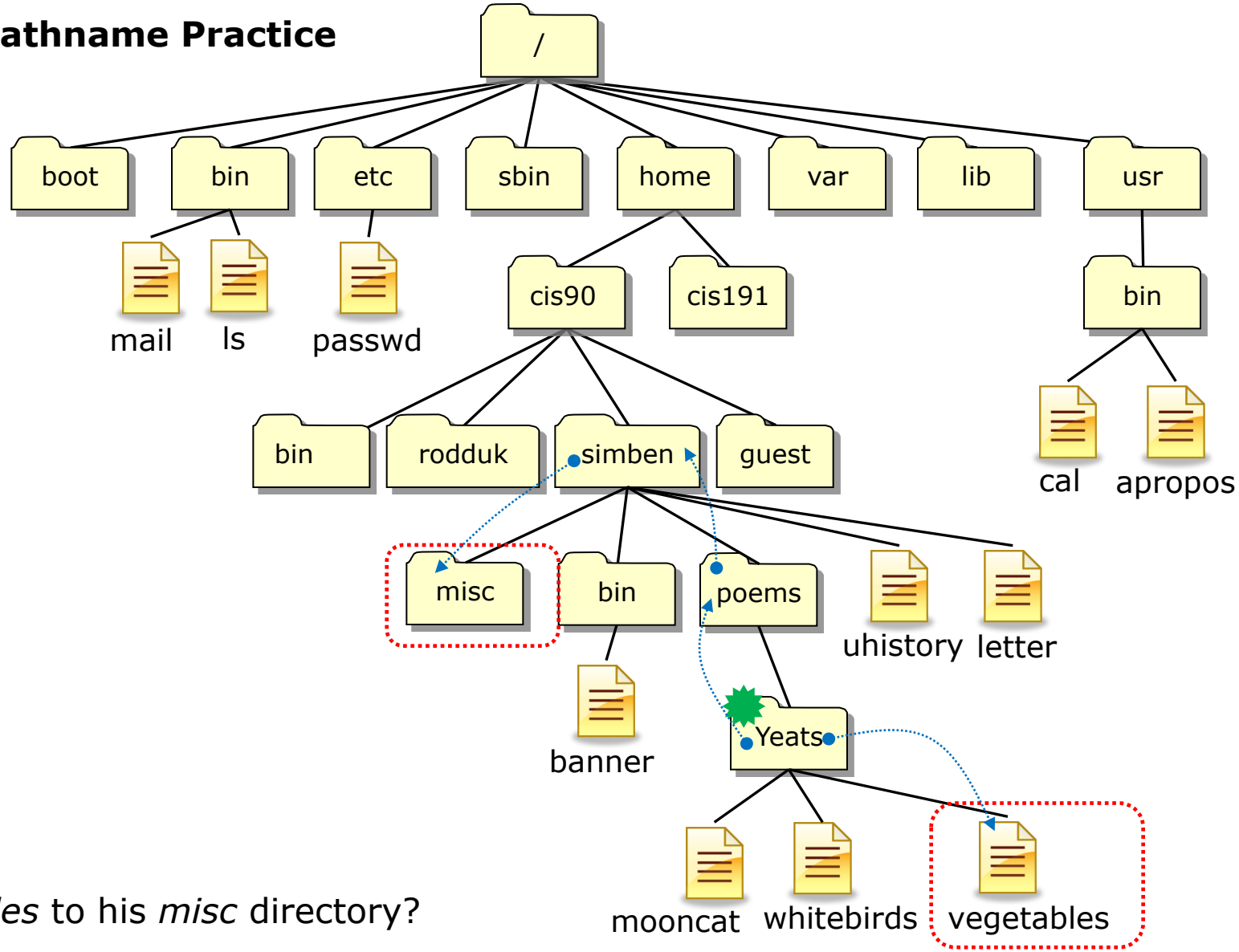
## File Tree Pathname Practice




From ✸ how does Benji:

Move *vegetables* to his *misc* directory?

## File Tree Pathname Practice

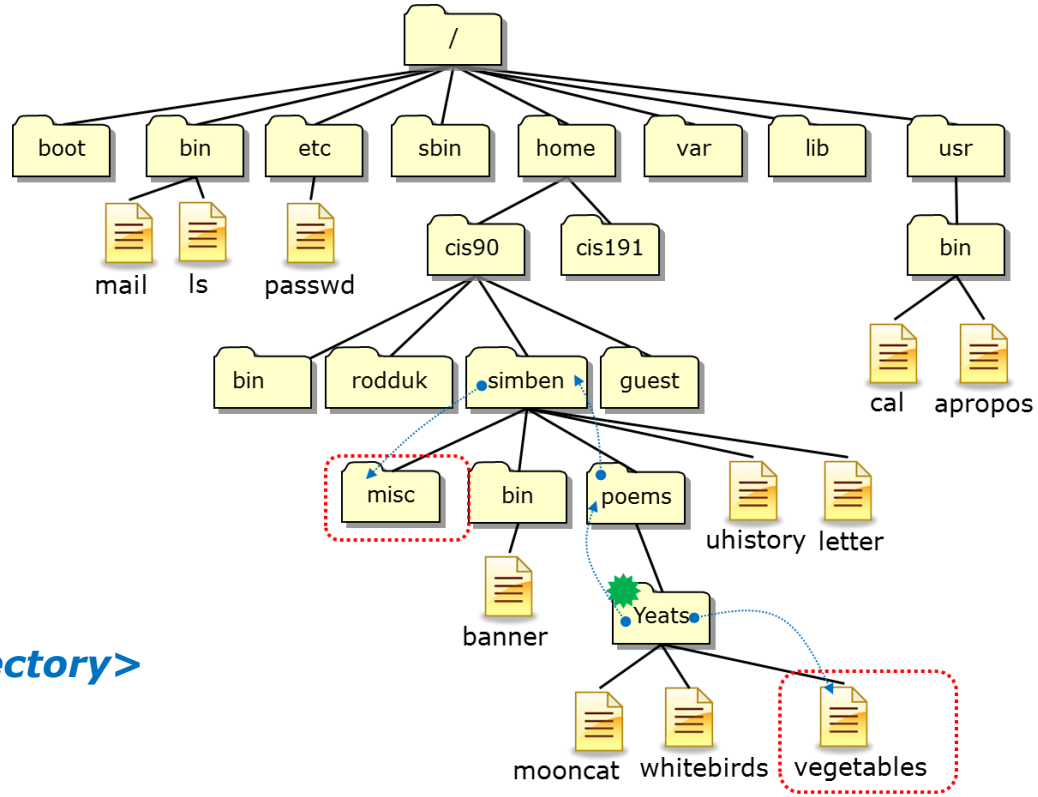



From ✸ how does Benji:

Move *vegetables* to his *misc* directory?

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

# *Other answers are also acceptable*




From ✸ how does Benji:

Move *vegetables* to his *misc* directory?

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

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

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

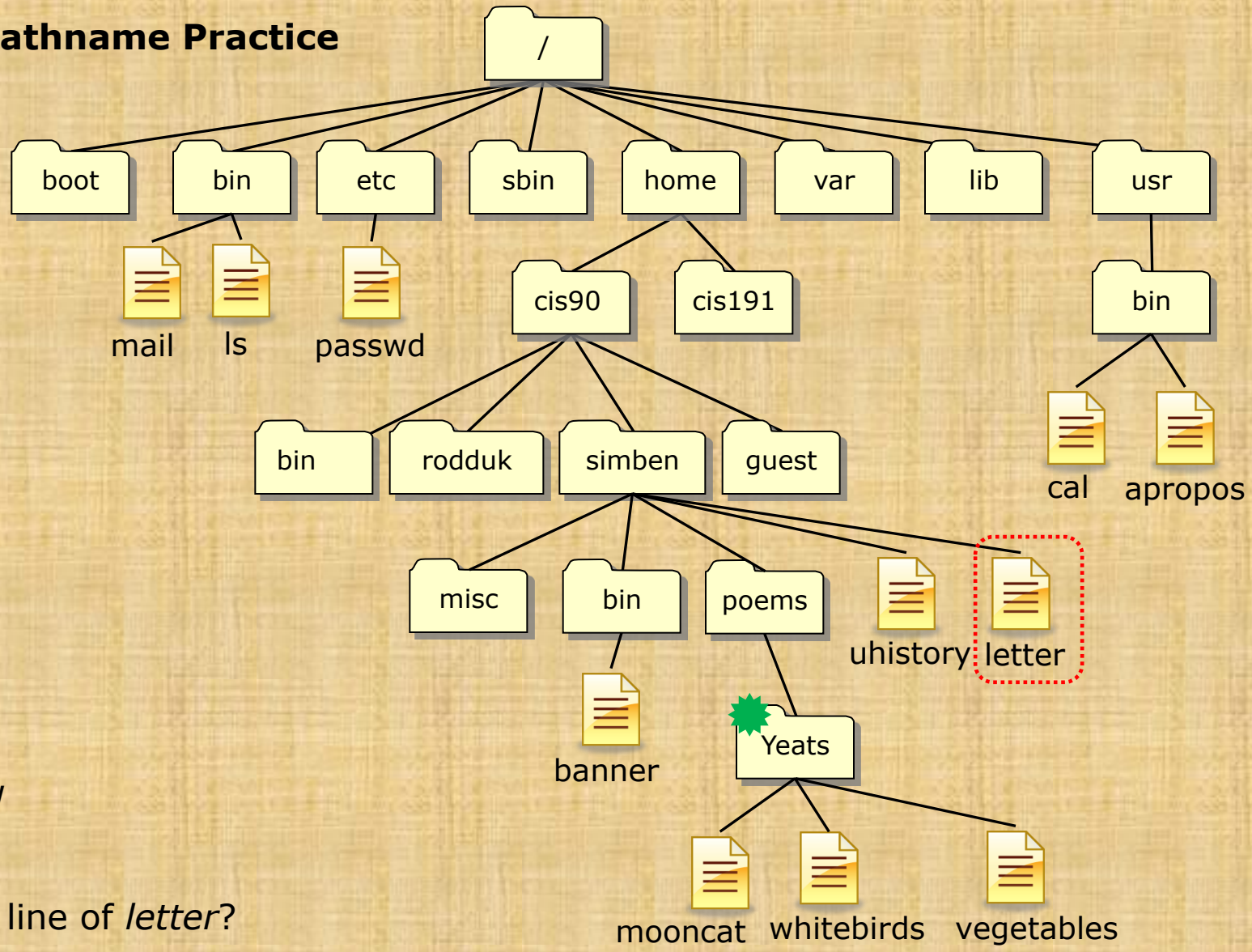
**mv /home/cis90/simben/poems/Yeats/vegetables ../../misc/**

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

**mv vegetables ~/misc/**

*All these answers are correct*

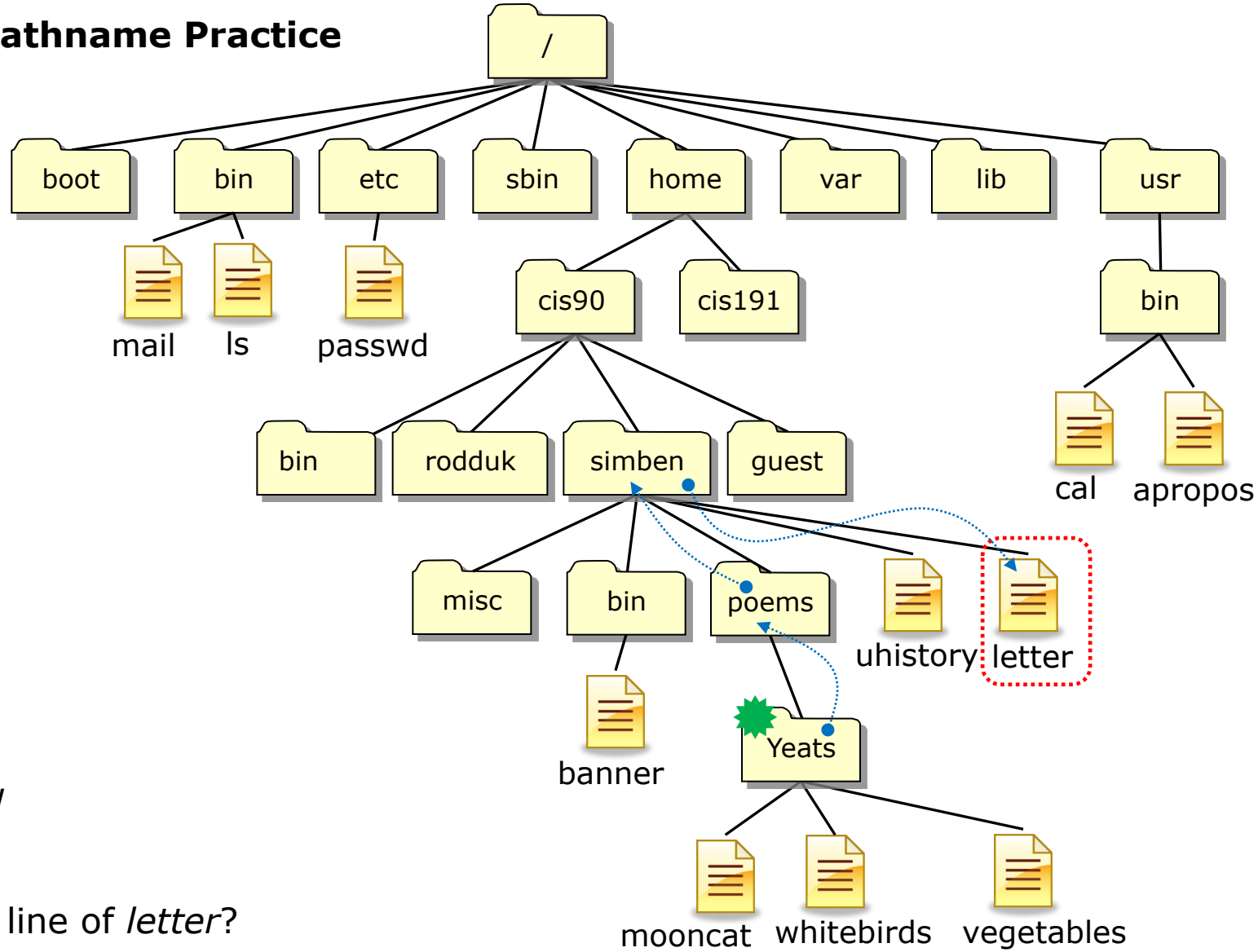
## File Tree Pathname Practice




From ✸ how does Benji:

Print the last line of *letter*?

## File Tree Pathname Practice

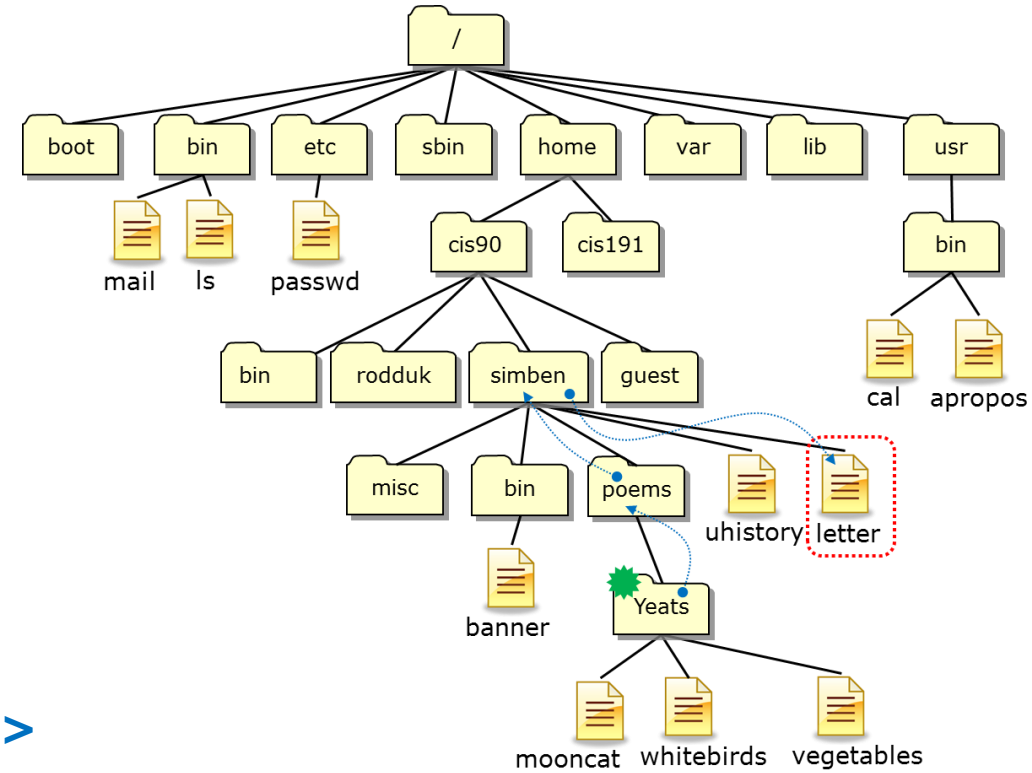



From ✸ how does Benji:

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>***

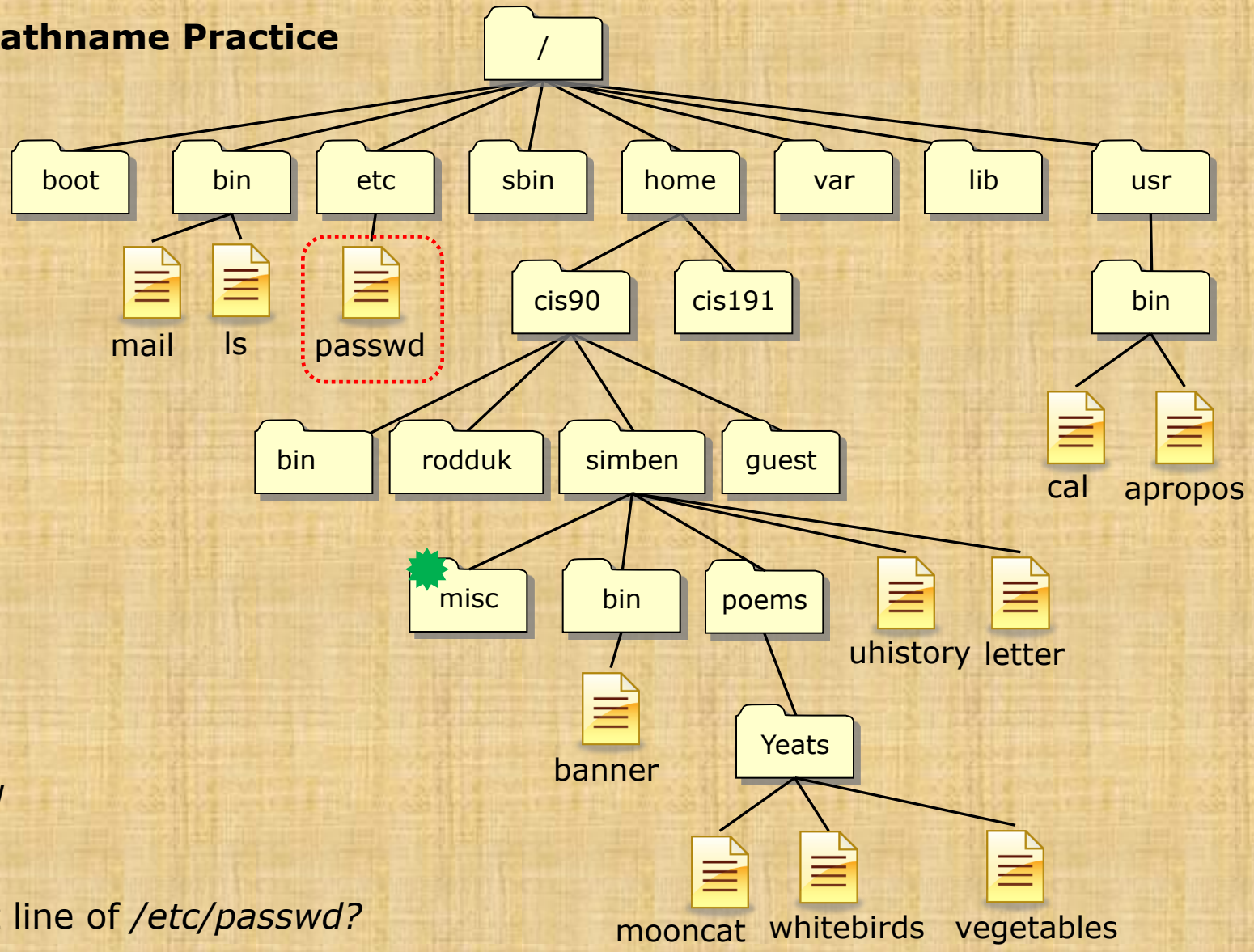
**tail -n1 ../../letter**

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

**tail -n1 ~/letter**

*All these answers are correct*

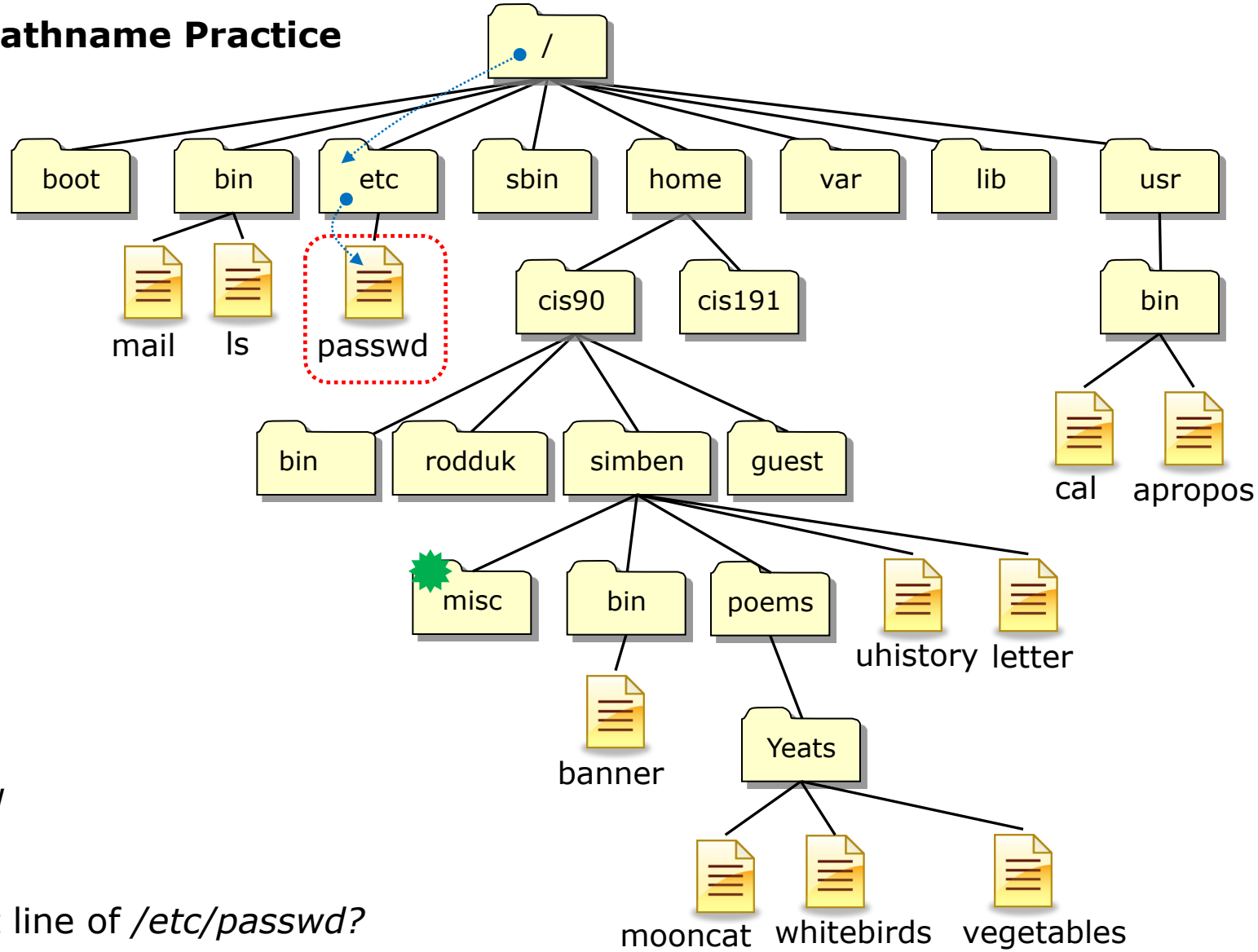
## File Tree Pathname Practice




From ✸ how does Benji:

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

## File Tree Pathname Practice

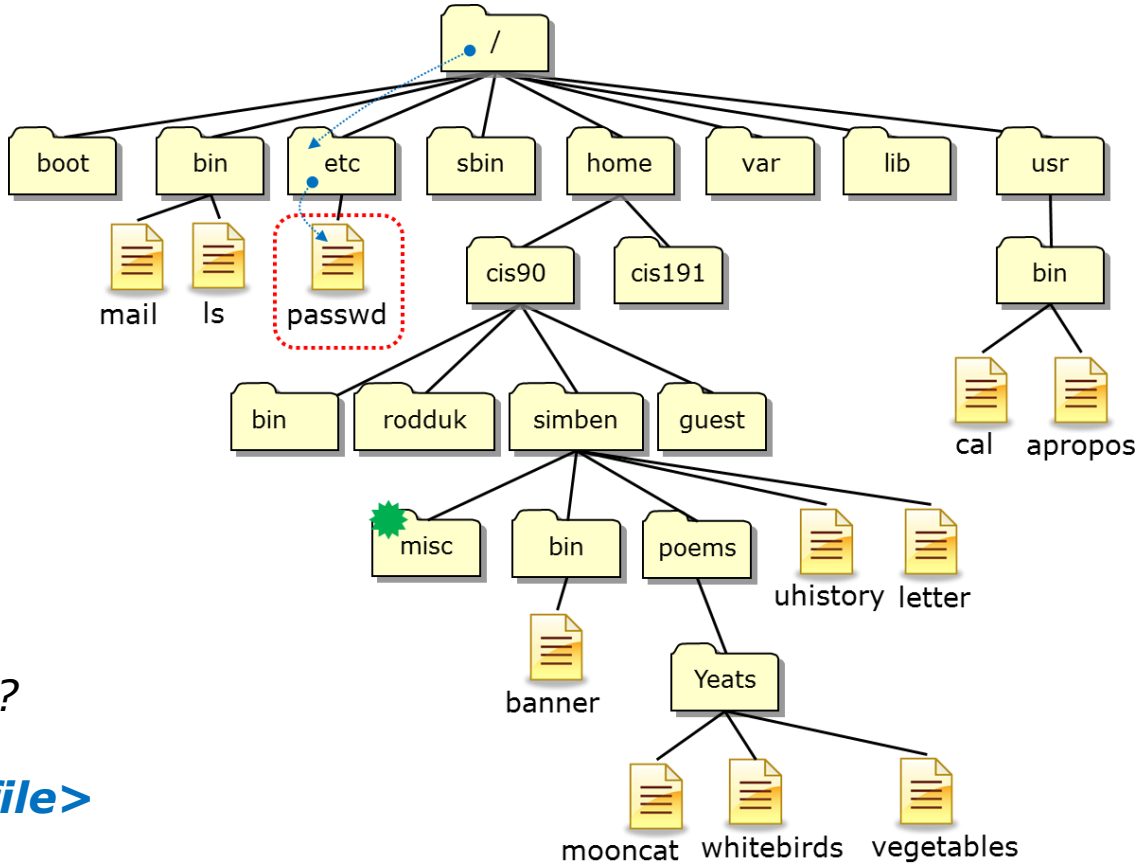



From ✸ how does Benji:

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

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

# *Other answers are also acceptable*




From ✸ how does Benji:

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

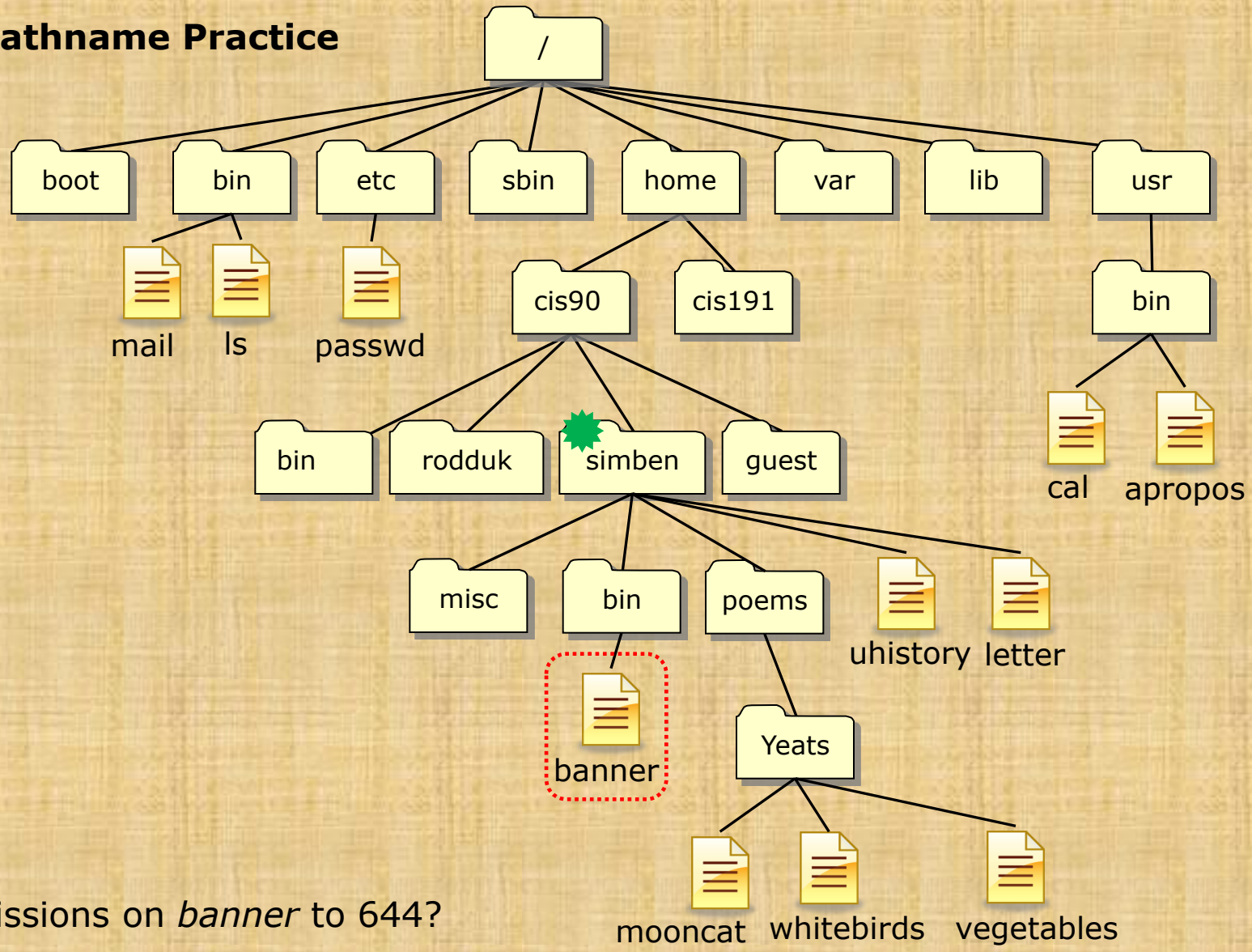
**head -n<number> <path-to-file>**

**head -n1 /etc/passwd**

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

*Both these answers are correct*

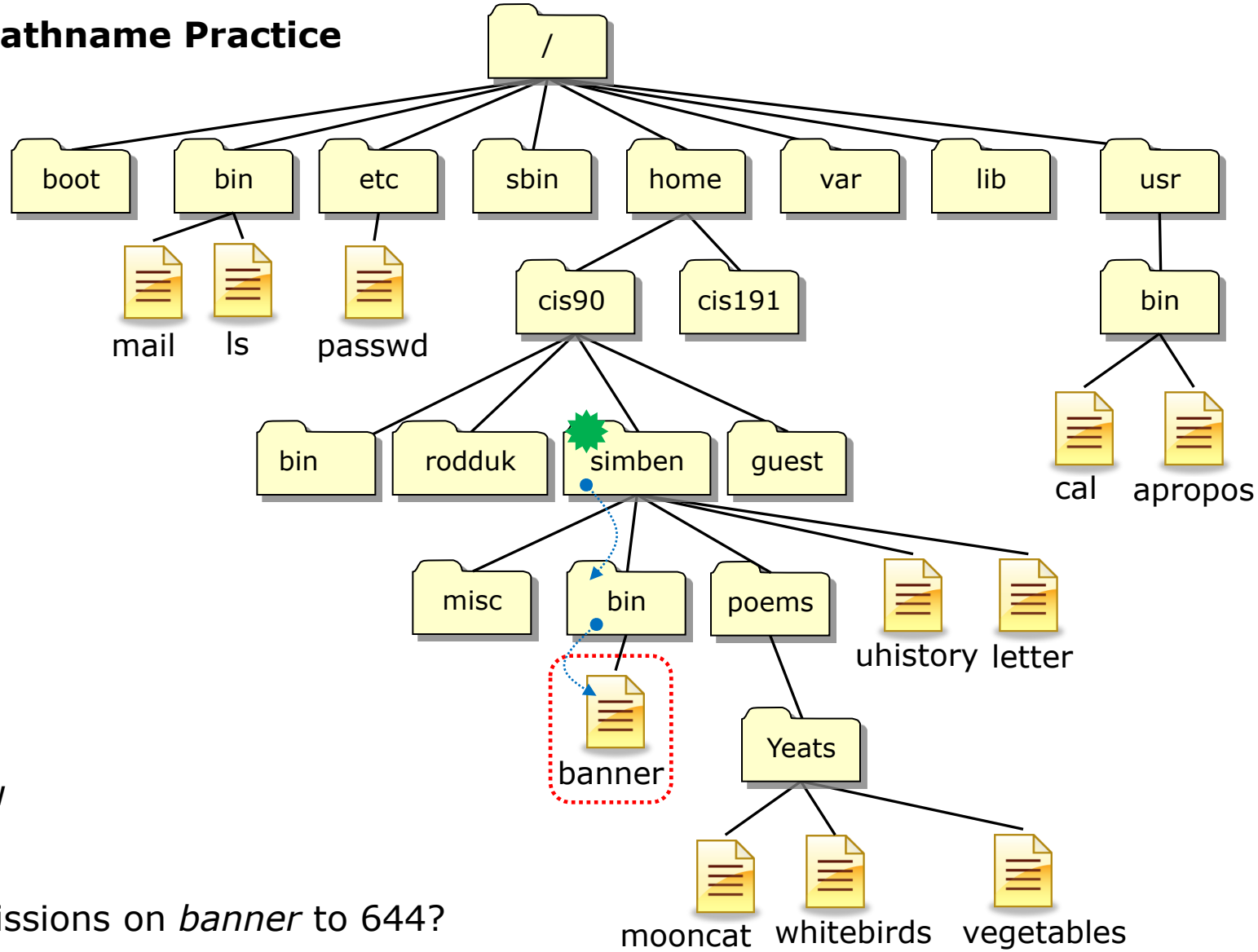
## File Tree Pathname Practice




From ✸ how does Benji:

Change permissions on *banner* to 644?

## File Tree Pathname Practice

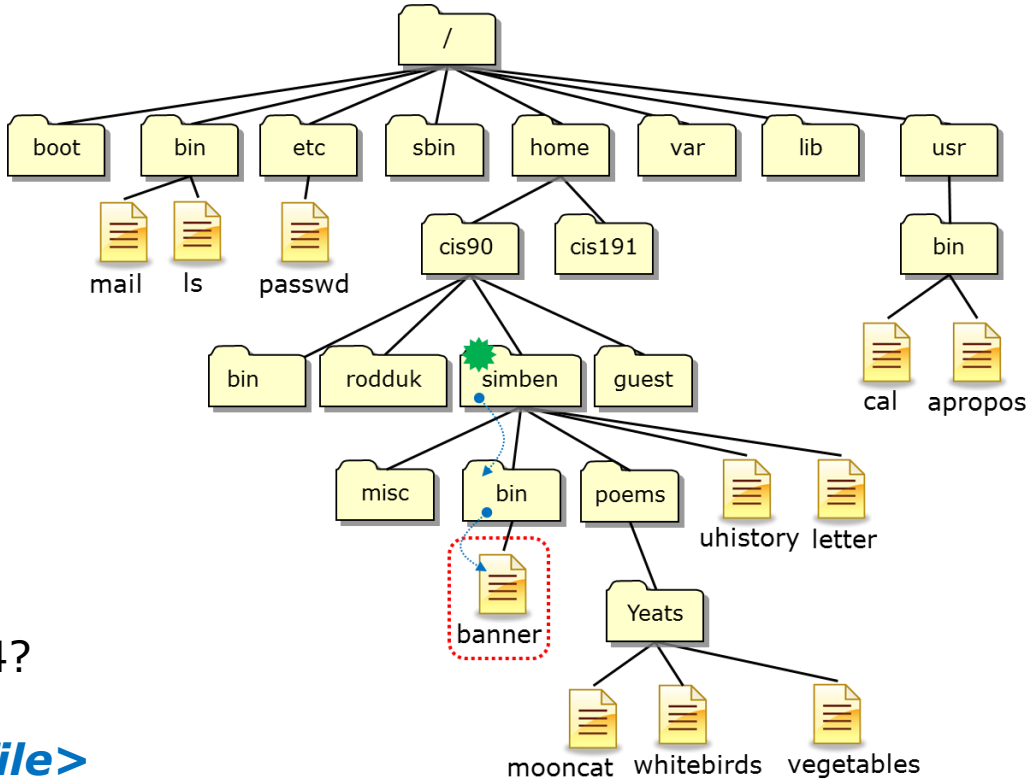



From ✹ how does Benji:

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?

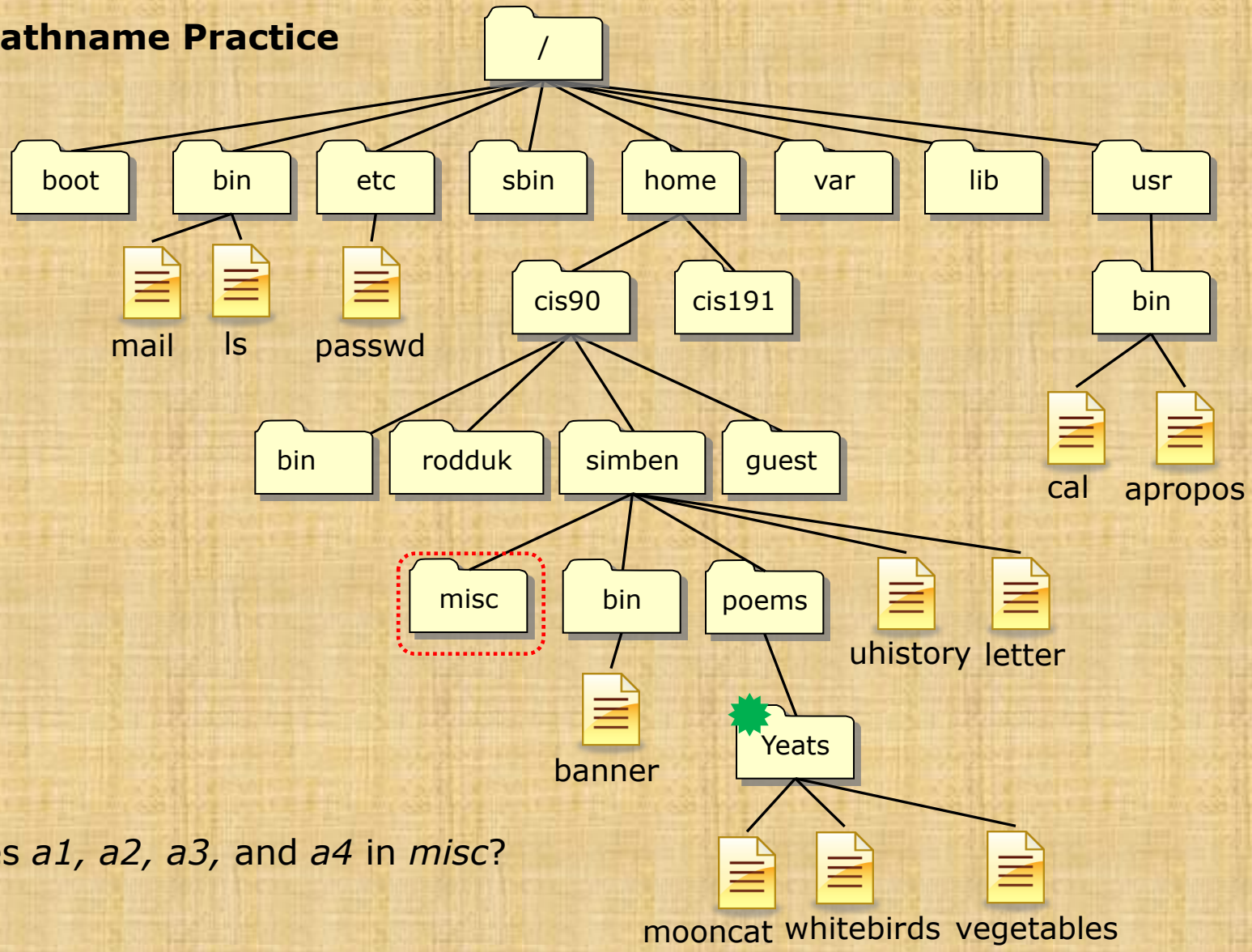
**chmod *<permissions> <path-to-file>***

**chmod 644 bin/banner**

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

*Both these answers are correct*

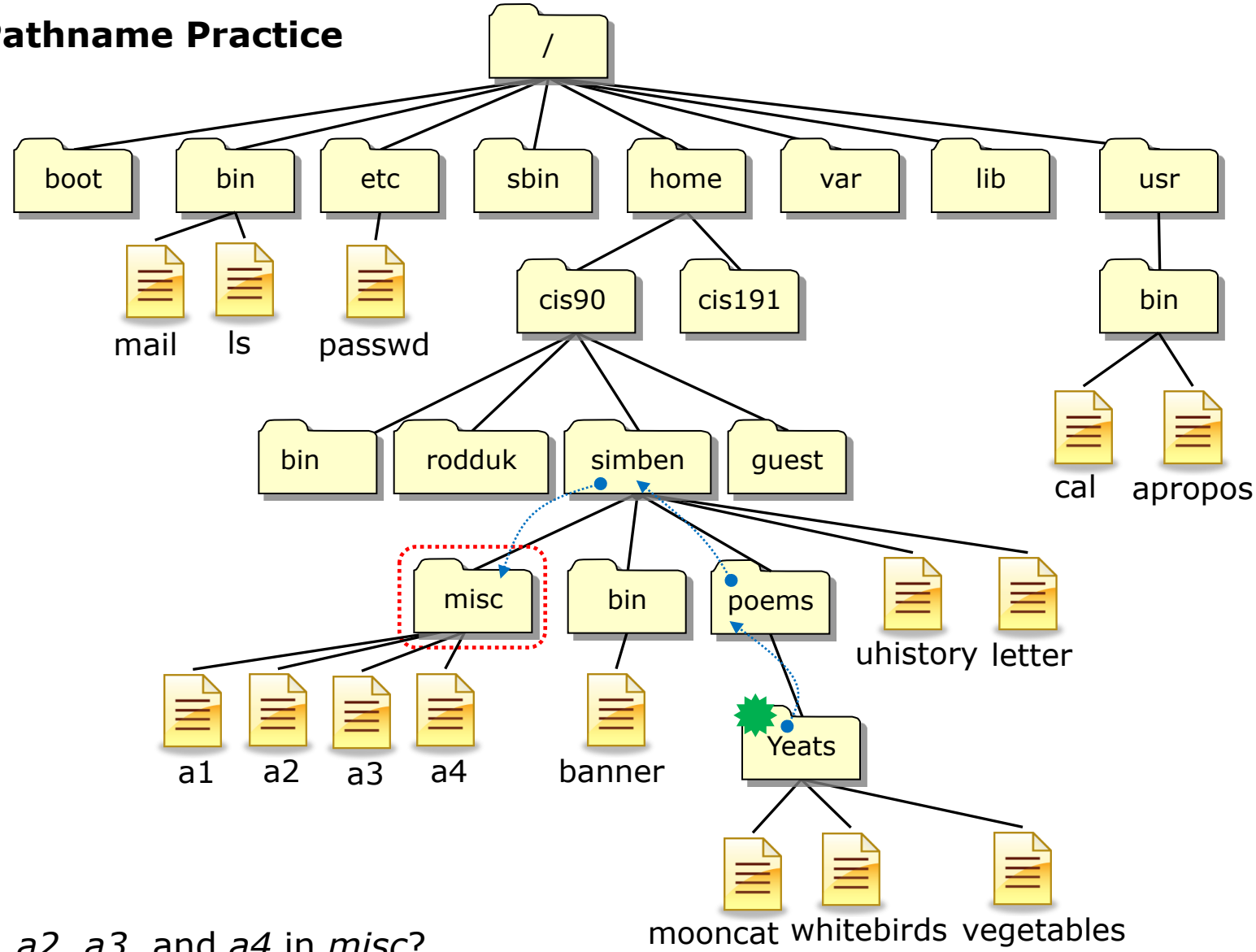
## File Tree Pathname Practice




From ✸ how does Benji:

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

## File Tree Pathname Practice

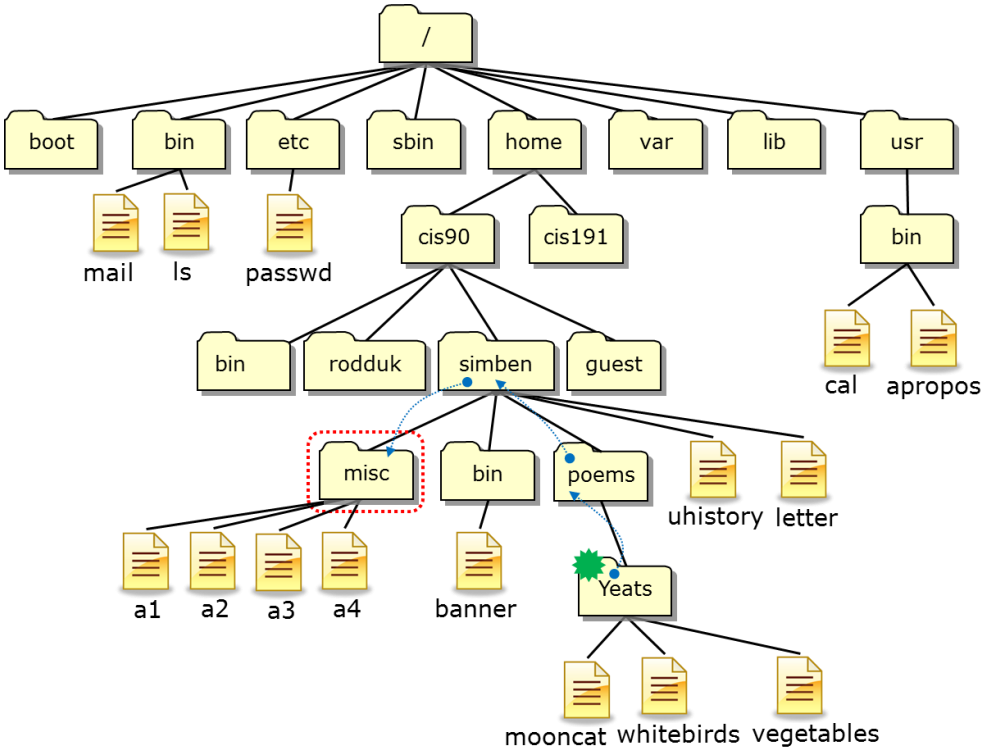



From ✸ how does Benji:

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**

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

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

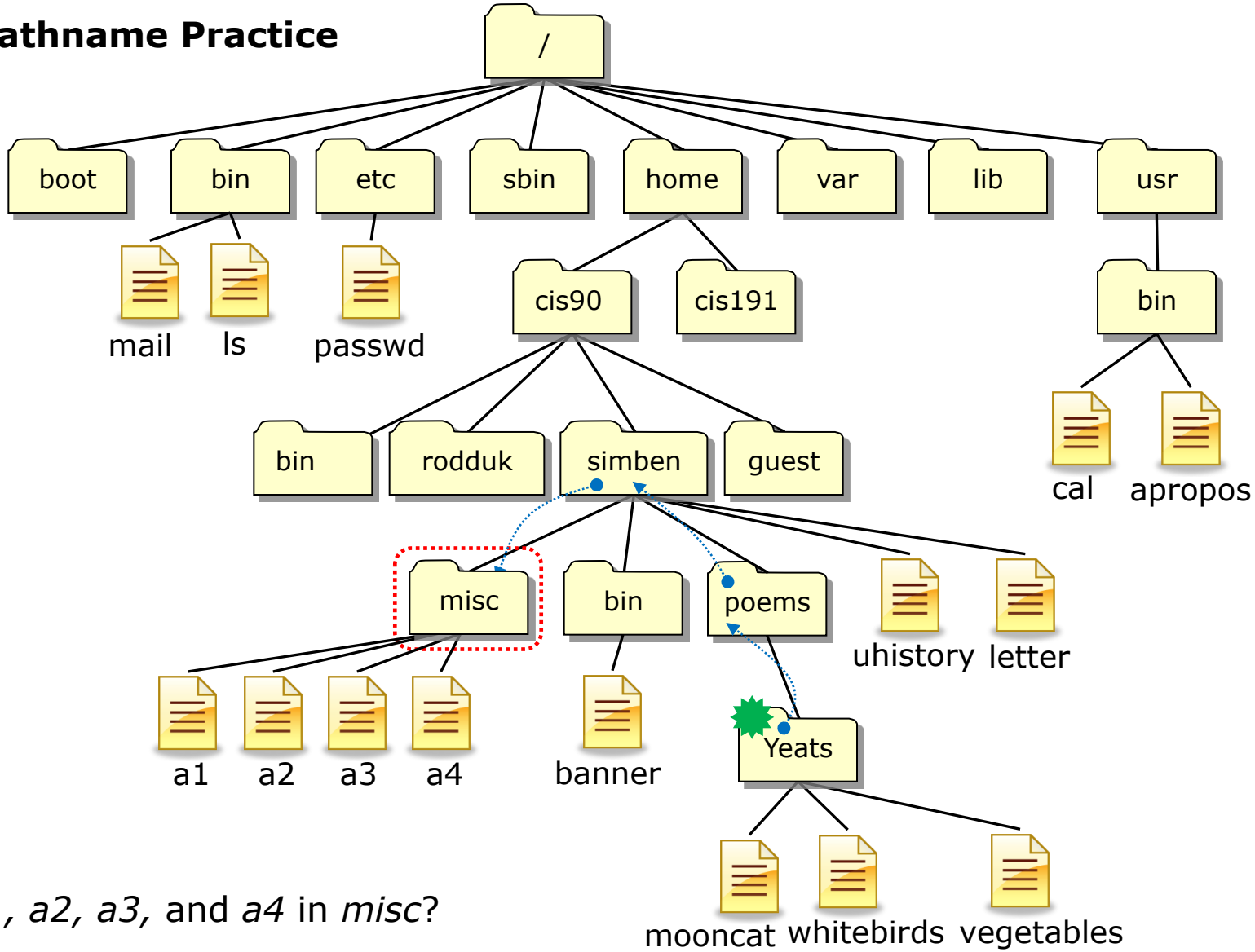


*All these answers are correct*

FYI
only

*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 permission:

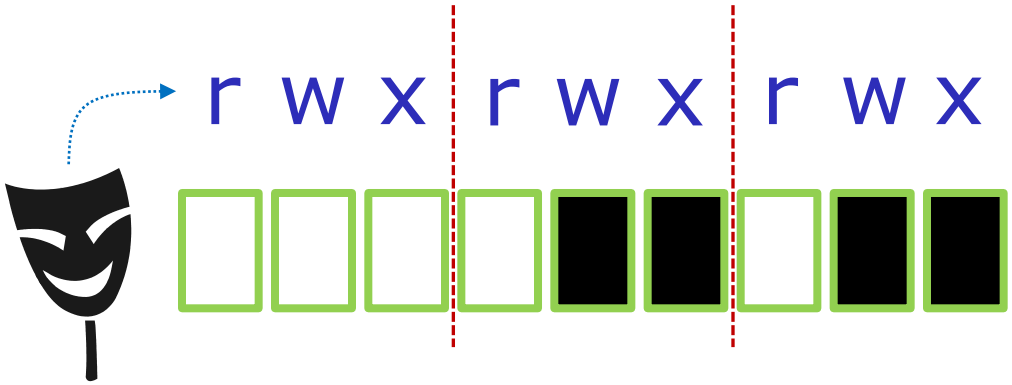
- For new files, start with **666**
- For new directories, start with **777**
- For file copies, start with **the permission on the source file**

## Case 1 – a new directory

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

## Case 1 – a new directory

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



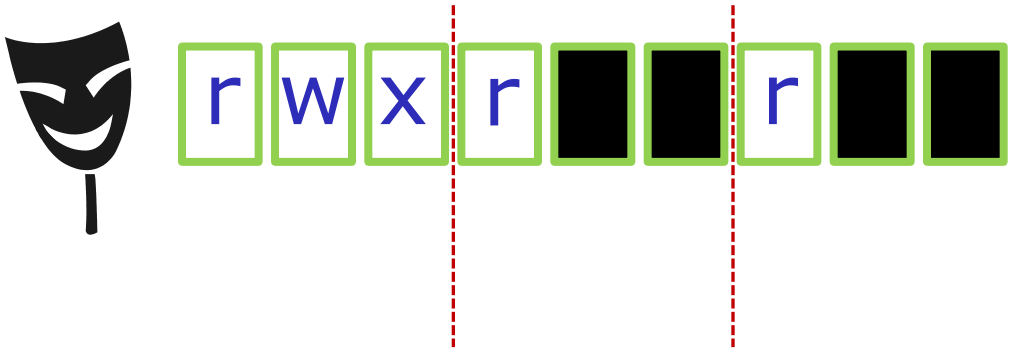

**starting point = 777 (new directory)**

**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 as shown here*

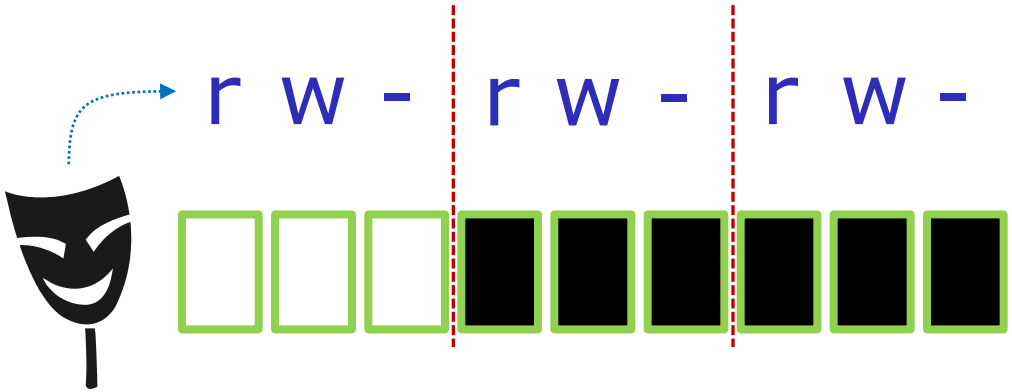
```
/home/cis90ol/simmsben $ umask 033
/home/cis90ol/simmsben $ mkdir brandnewdir
/home/cis90ol/simmsben $ ls -ld brandnewdir/
drwxr--r-- 2 simmsben cis90ol 4096 Apr 21 12:46 brandnewdir/
 7  4  4
```

## Case 2 – new file

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

## Case 2 – new file

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



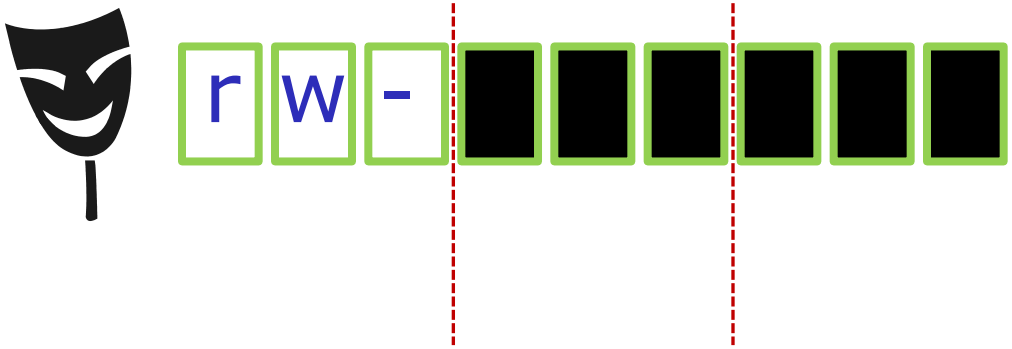

**starting point = 666 (new file)**

**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 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

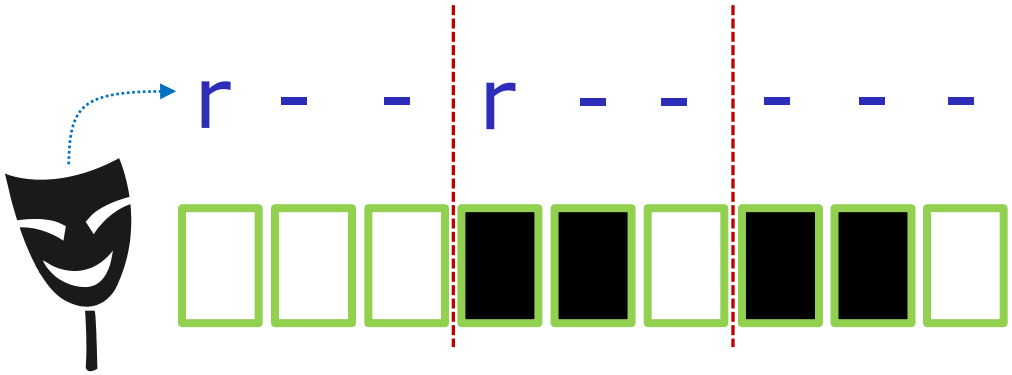
**If umask=066 and the *cinderella* file permissions are 440**
**What would the permissions be on *cinderella.bak* after:**

```
cp cinderella cinderella.bak
```

## 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)**

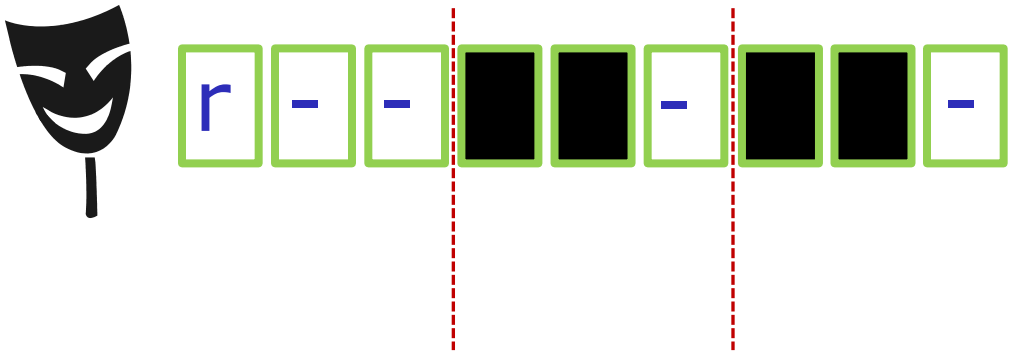
**umask setting of 066 strips these bits: --- rw- rw-**

*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 as shown here*

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

# Housekeeping

# Previous material and assignment

1. Lab 6 due 11:59PM

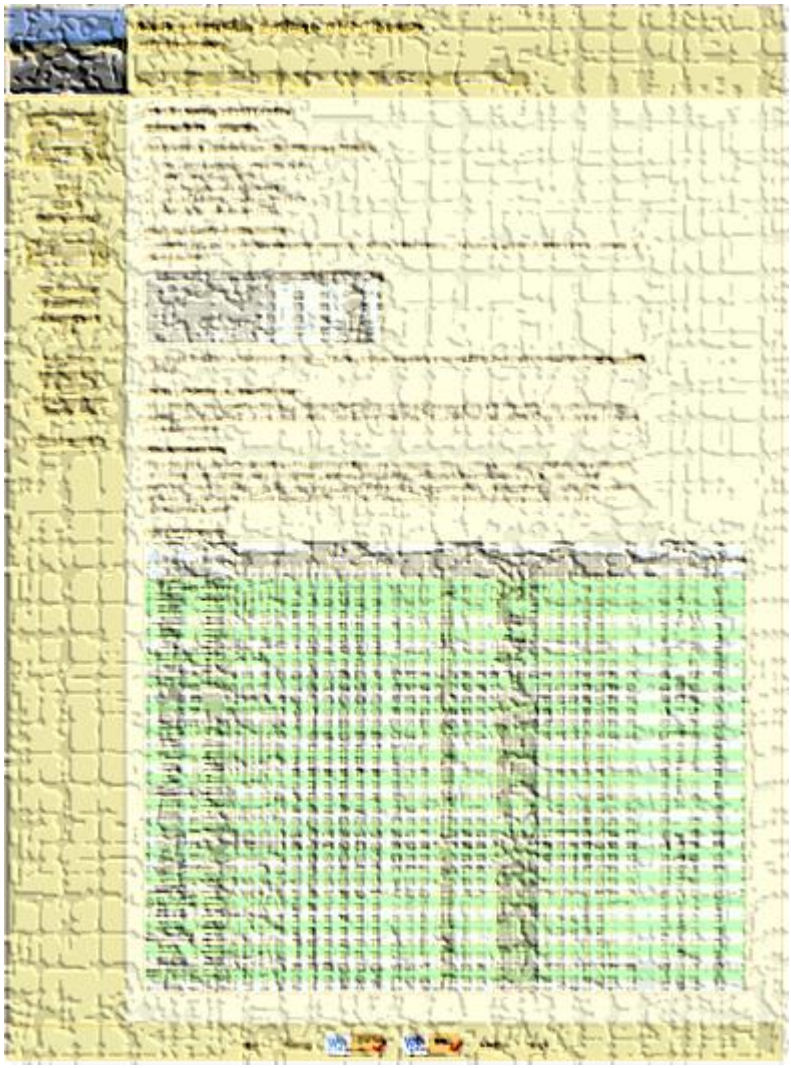
2. A **check6** script is available

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

3. Five more posts due 11:59PM

4. Early preview of Lab X2 is now available. This is recommended for anyone wanting more practice with pathnames.

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



# GRADES

- Check your progress on the Grades page
- If you haven't already, send me a student survey to get your LOR secret code name
- Graded labs & tests are placed in your home directories on Opus
- Answers to labs, tests and quizzes are in the */home/cis90/answers* directory on Opus

# Current Point Tally

As of 3/17/2014

| **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** |

| 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 |

alatar: 56% (122 of 215 points)
anborn: 82% (178 of 215 points)
aragorn: 90% (195 of 215 points)
arwen: 99% (213 of 215 points)
beregond: 0% (0 of 215 points)
bilbo: 58% (125 of 215 points)
celebrian: 100% (215 of 215 points)
dwalin: 96% (208 of 215 points)
eomer: 94% (203 of 215 points)
faramir: 94% (204 of 215 points)
frodo: 94% (204 of 215 points)
gwaihir: 107% (231 of 215 points)
ioreth: 97% (209 of 215 points)
legolas: 91% (196 of 215 points)
marhari: 60% (129 of 215 points)
orome: 81% (175 of 215 points)
pallando: 0% (0 of 215 points)
pippen: 76% (165 of 215 points)
quickbeam: 102% (221 of 215 points)
rian: 0% (0 of 215 points)
samwise: 83% (179 of 215 points)
shadowfax: 0% (0 of 215 points)
strider: 93% (201 of 215 points)
theoden: 60% (129 of 215 points)
treebeard: 108% (233 of 215 points)
tulkas: 84% (181 of 215 points)
ulmo: 74% (161 of 215 points)

*If you are not happy with your current standing contact the instructor ASAP*

# Jesse's checkgrades python script

http://oslab.cabrillo.edu/forum/viewtopic.php?f=31&t=773&p=2966

```
/home/cis90/simben $ checkgrades smeagol

Remember, your points may be zero simply because the
assignment has not been graded yet.

Quiz 1: You earned 3 points out of a possible 3.
Quiz 2: You earned 3 points out of a possible 3.
Quiz 3: You earned 3 points out of a possible 3.
Quiz 4: You earned 3 points out of a possible 3.


Forum Post 1: You earned 20 points out of a possible 20.

Lab 1: You earned 30 points out of a possible 30.
Lab 2: You earned 30 points out of a possible 30.
Lab 3: You earned 30 points out of a possible 30.
Lab 4: You earned 29 points out of a possible 30.


You've earned 15 points of extra credit.

You currently have a 109% grade in this class. (166 out of
152 possible points.)
```

*Use your LOR code name as an argument on the checkgrades command*

*Jesse is a CIS 90 Alumnus. He wrote this python script when taking the course. It mines data from the website to check how many of the available points have been earned so far.*

# sort command

# Tools for your toolbox

NEW

**sort** - sorts the lines in a file

# sort command

Basic syntax
(see man page for the rest of the story)

**sort** <*options*> <*filepath*>

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

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

# sort command

```
/home/cis90/simben $ cat misc/salad
orange
mango
banana
peach
apple
grapes
pear
apricot
kiwi
watermelon
pineapple
```

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

```
/home/cis90/simben $ sort misc/salad
apple
apricot
banana
grapes
kiwi
mango
orange
peach
pear
pineapple
watermelon
```

# 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*

# Input and Output

**Get the names file to use in the next module**

*return to home directory*

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

*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
```

/

# Input and Output

## File Descriptors

### 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 send the sorted lines to **stdout** (defaults to the terminal)*

# Input and Output

## File Descriptors

### sort command with no argument

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


*If no filename was specified **sort** will read from **stdin** (attached by default to the terminal keyboard)*

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

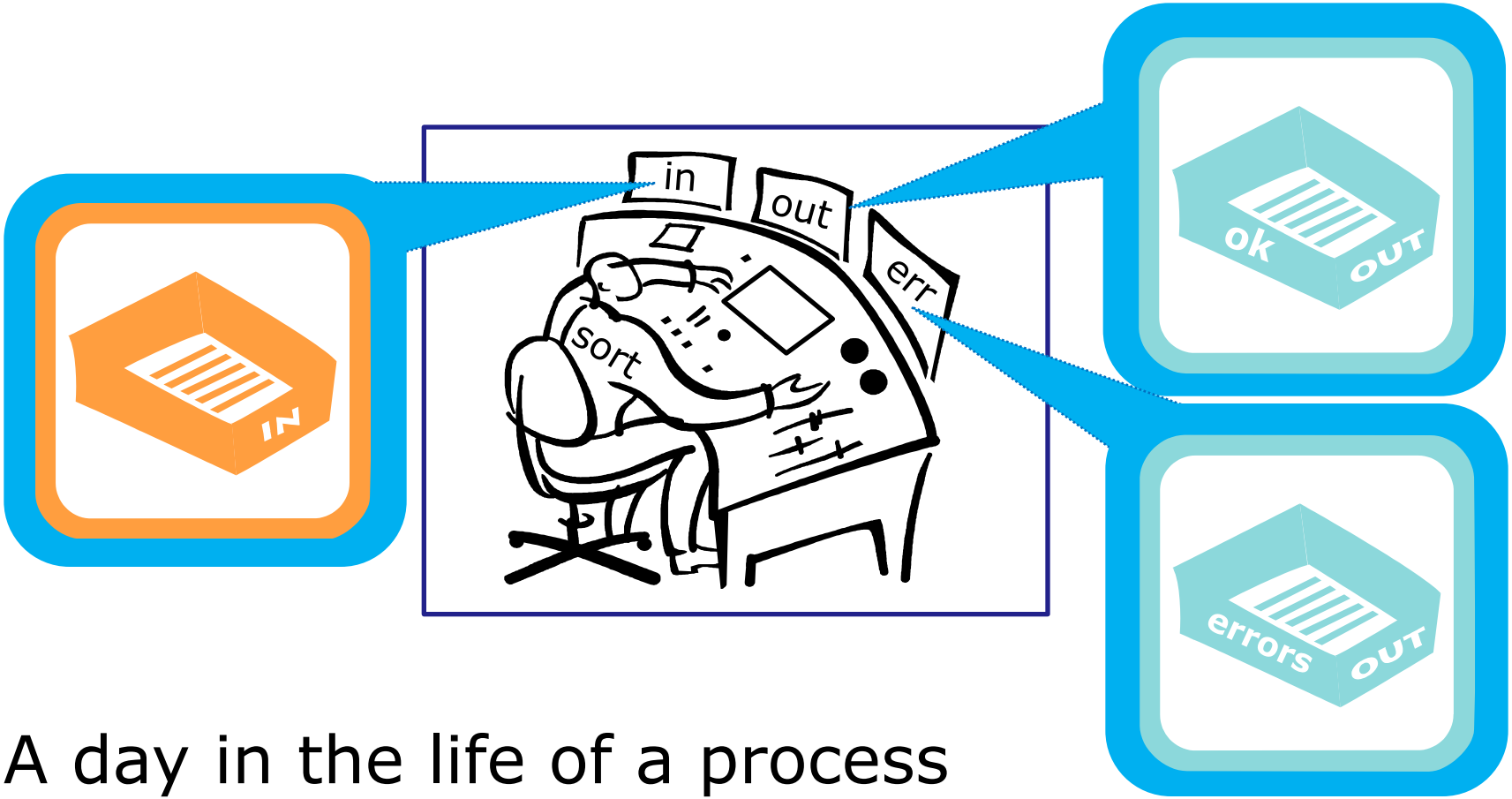
*After sort receives the EOF it sorts the lines and writes them to **stdout** (attached by defaults to terminal)*

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

*Lets visualize being the sort program and being loaded into memory and executing*

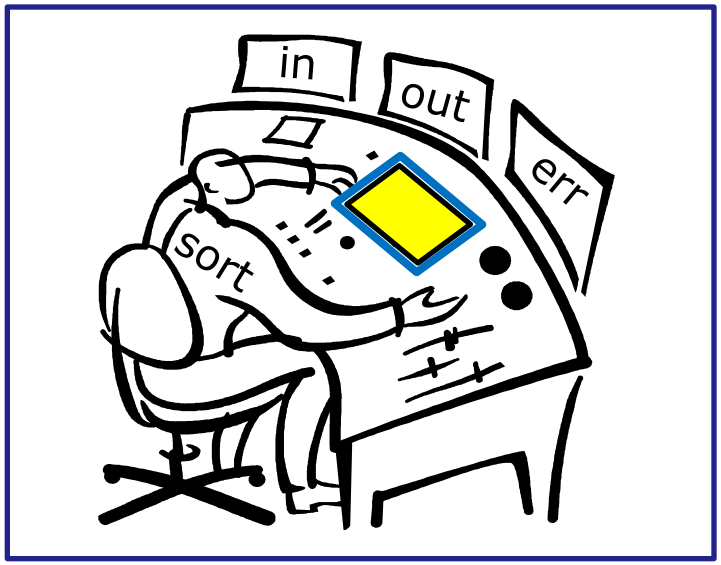

# A day in the life of a process

*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.*



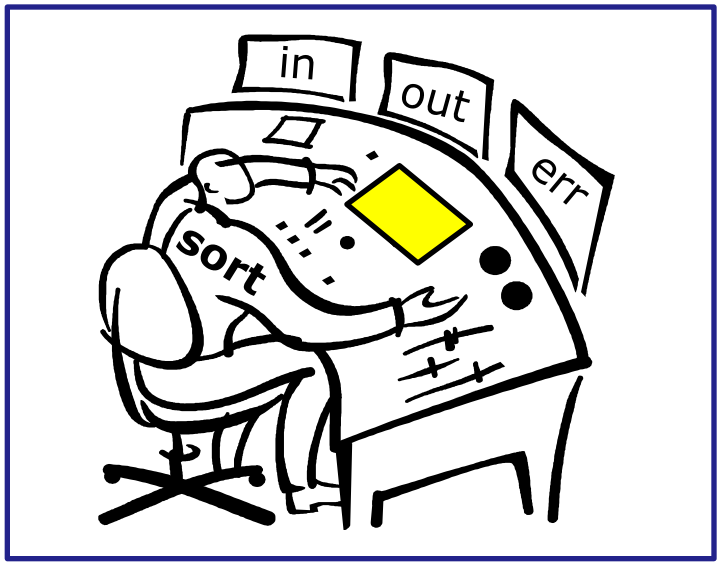

# A day in the life of a process

# sort process no argument

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

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

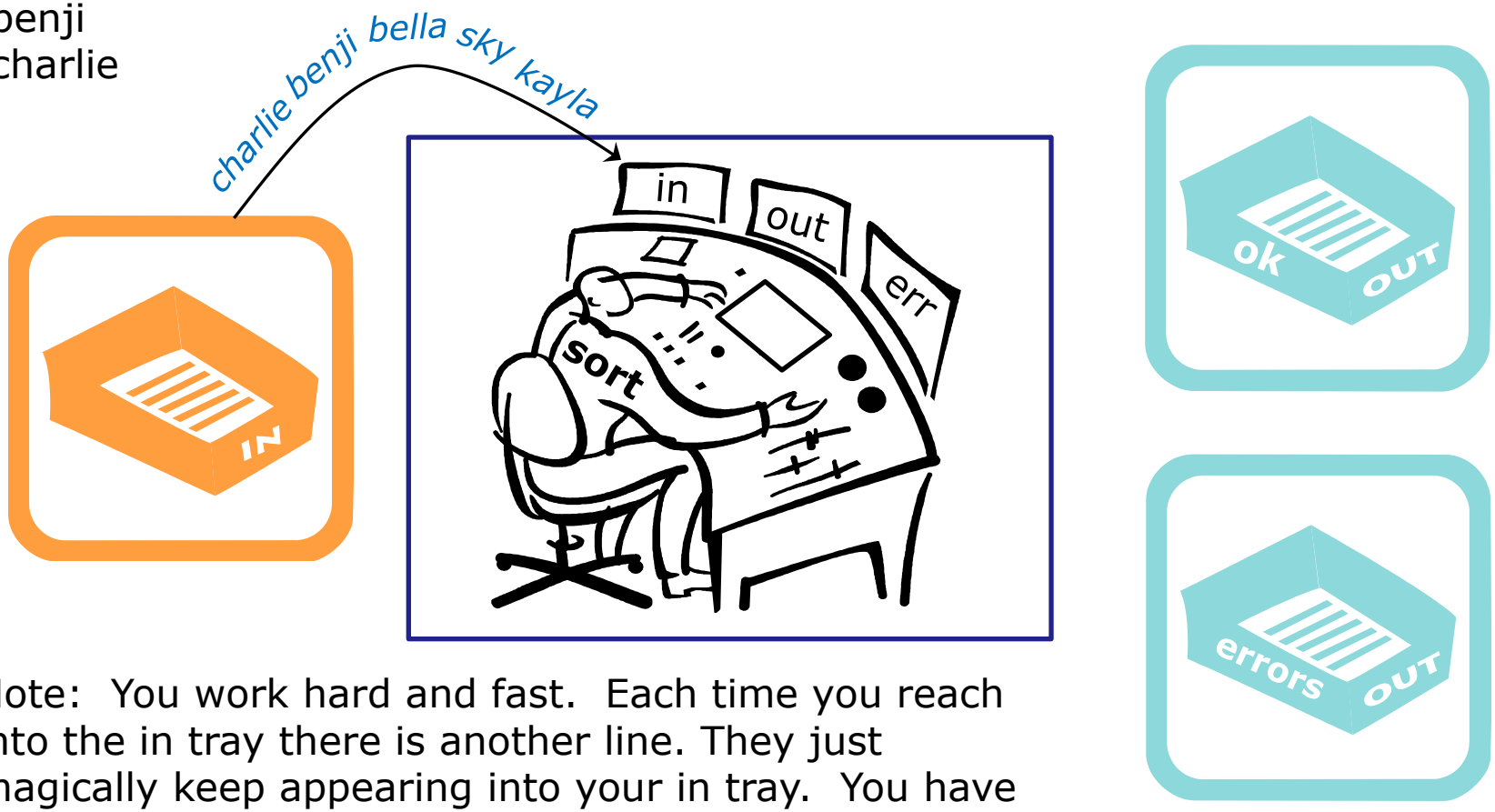
1. Prompt string is "/home/cis90/simben $ "
2. Parsing results is command=sort, no options, no arguments, no redirection
3. Search locates the sort program in /bin
4. Sort loaded into memory and execution begins



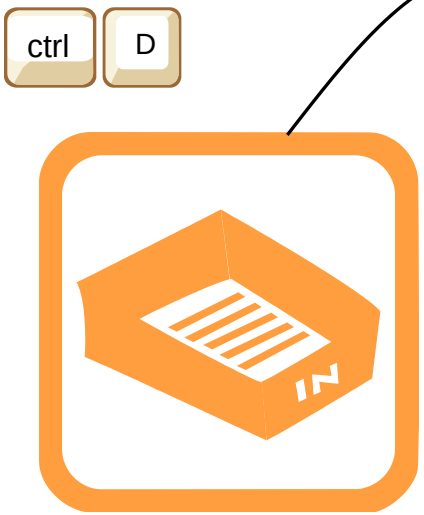

You (the sort process) check your instruction window and see that no options or arguments were passed to you to handle. You know (given your internal DNA code) that with no arguments you must look for lines to sort in your in tray, so you reach in to grab the first line to sort.

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




Note: 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
```



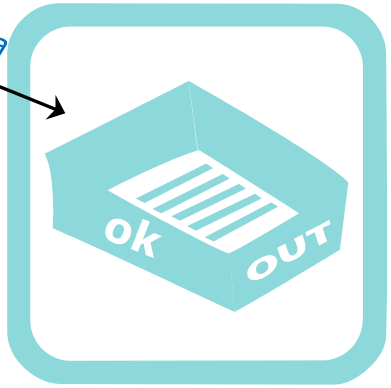




Then suddenly, when you reach for the next line, you find an EOF. You know (your internal DNA code) that this EOF means no more lines coming. You must sort what you have collected so far and place them, in order, into your out tray.

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 process bad argument

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

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

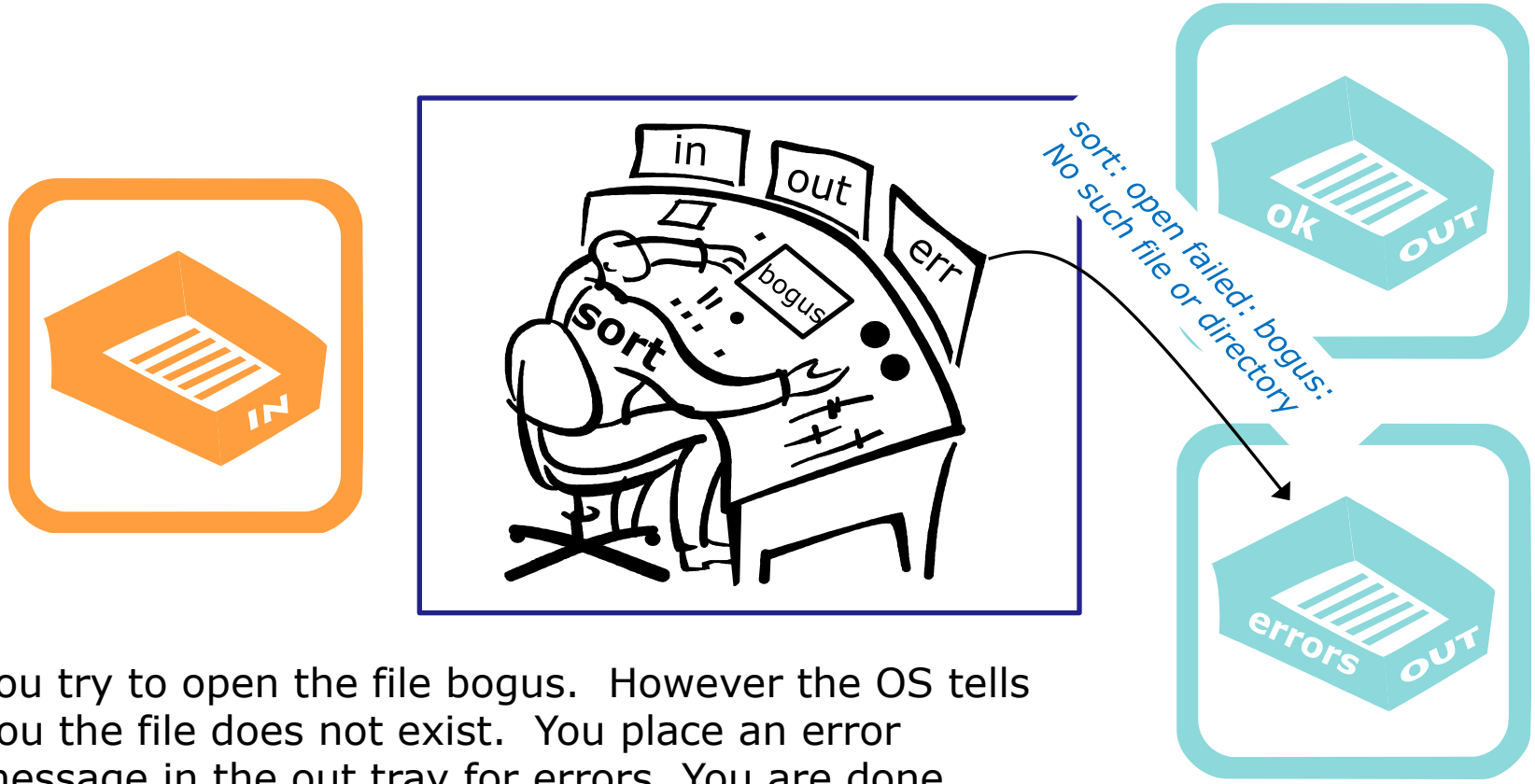
1. Prompt string is "/home/cis90/simben $ "
2. Parsing results is command=sort, no options, 1 argument="bogus", no redirection
3. Search locates the sort program in /bin
4. Sort loaded into memory and execution begins





You check the instruction window and notice the shell passed you one argument: "bogus". You know (your internal DNA code) that bogus is a file which should contain the lines to sort.

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




You try to open the file bogus. However the OS tells you the file does not exist. You place an error message in the out tray for errors. You are done.

# bringing it home

*Ok, lets make the visualization a little more realistic*

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

**stdout (1)**

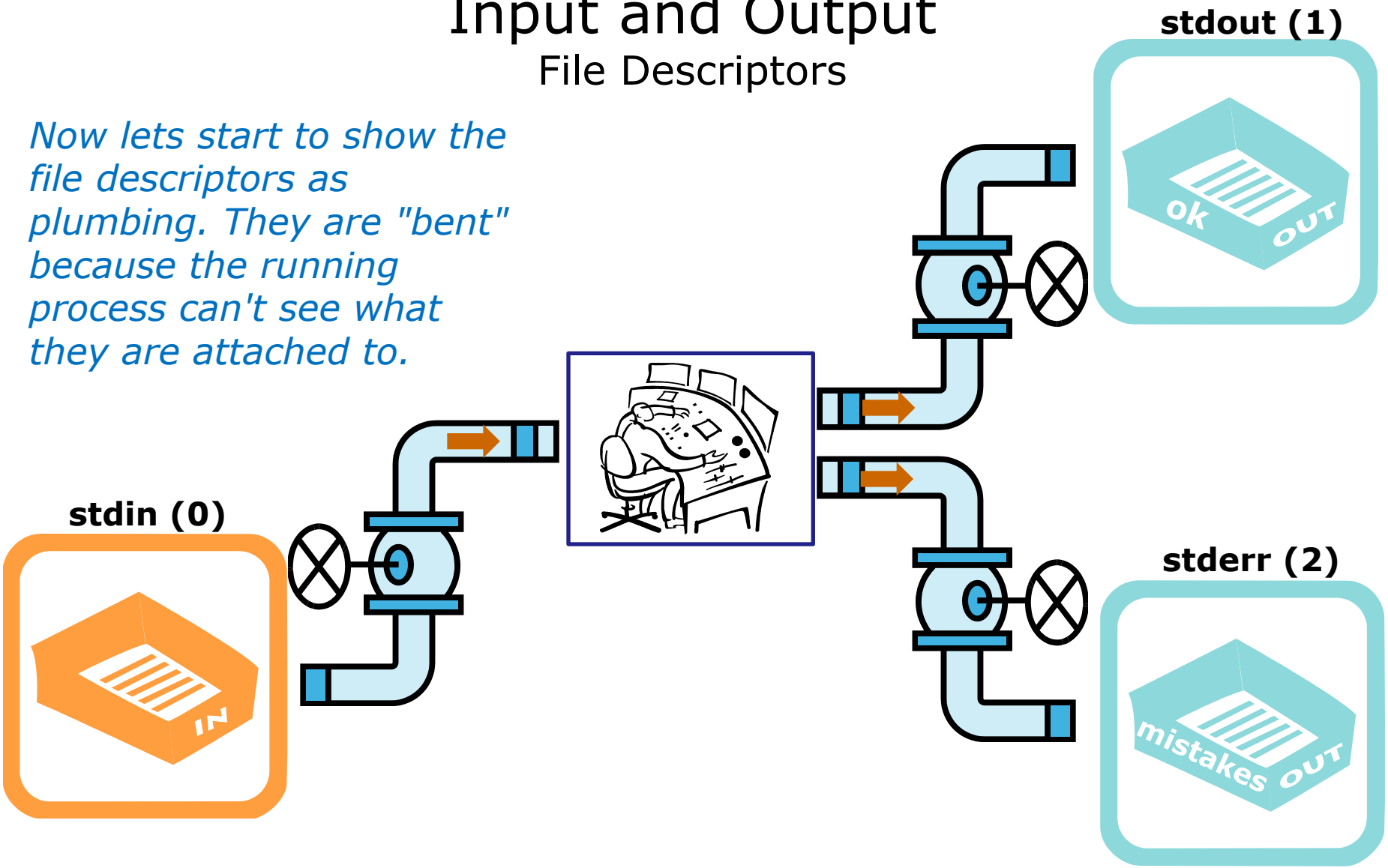

**stdin (0)**


**stderr (2)**

# Input and Output

## File Descriptors

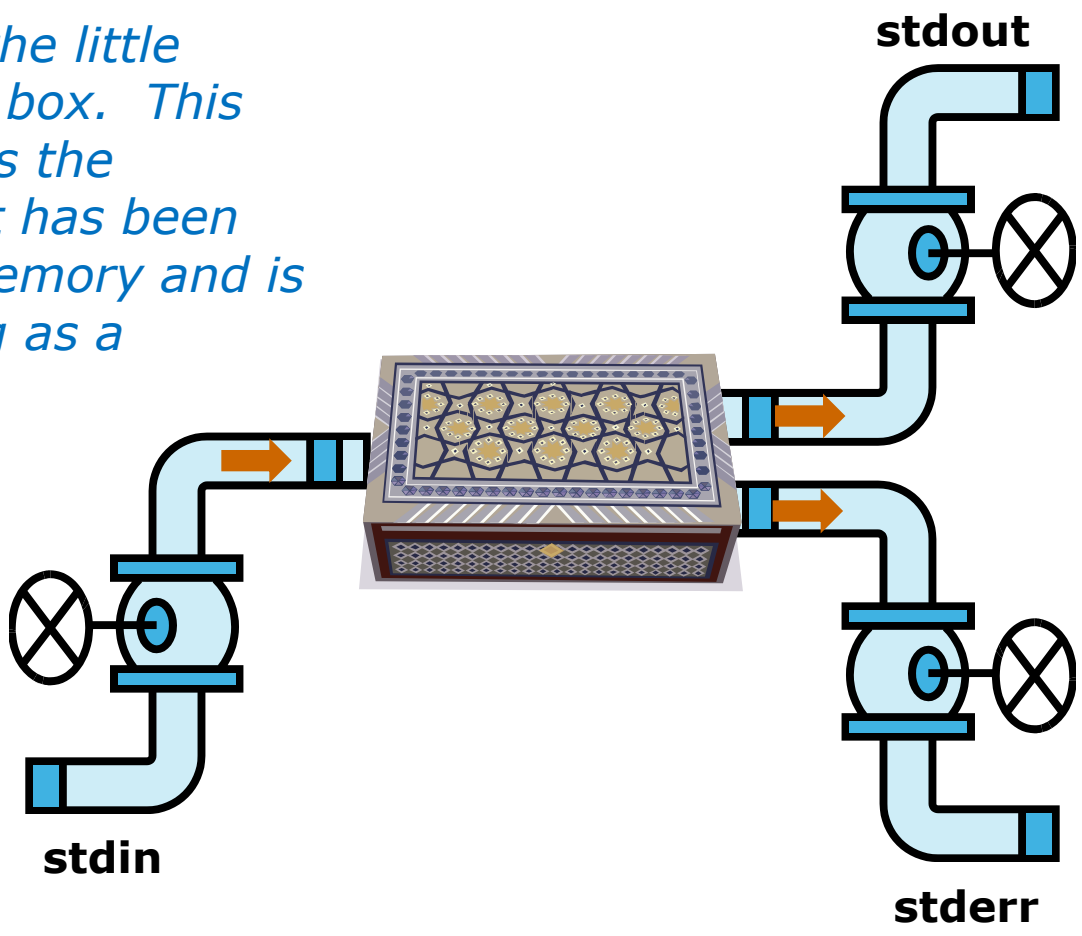
*Now lets start to show the file descriptors as plumbing. They are "bent" because the running process can't see what they are attached to.*



# Input and Output

## File Descriptors

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

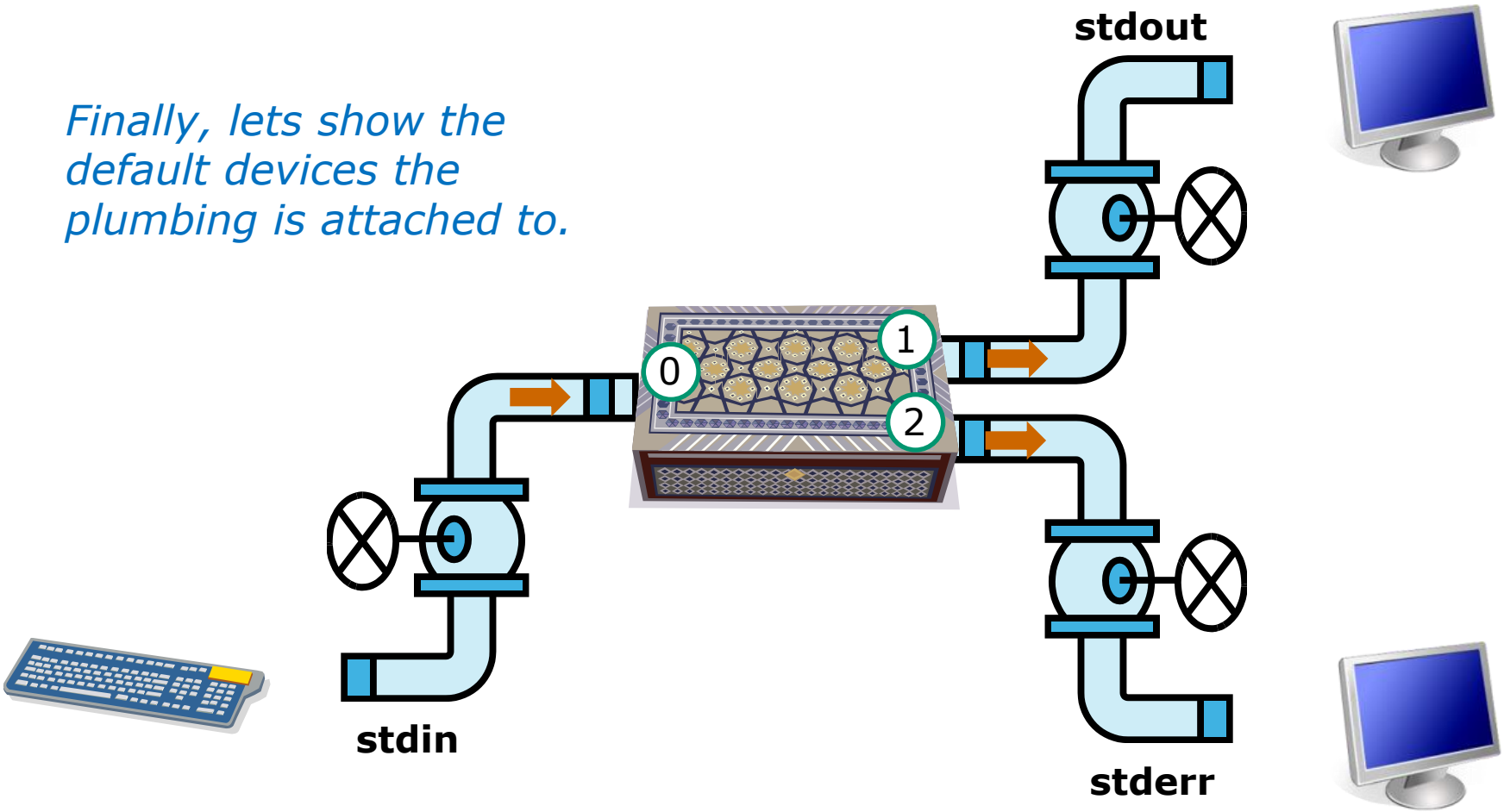


# Input and Output

## File Descriptors

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

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



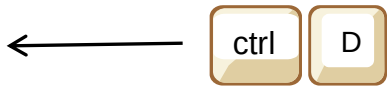

By default is attached to the user's terminal device (keyboard)

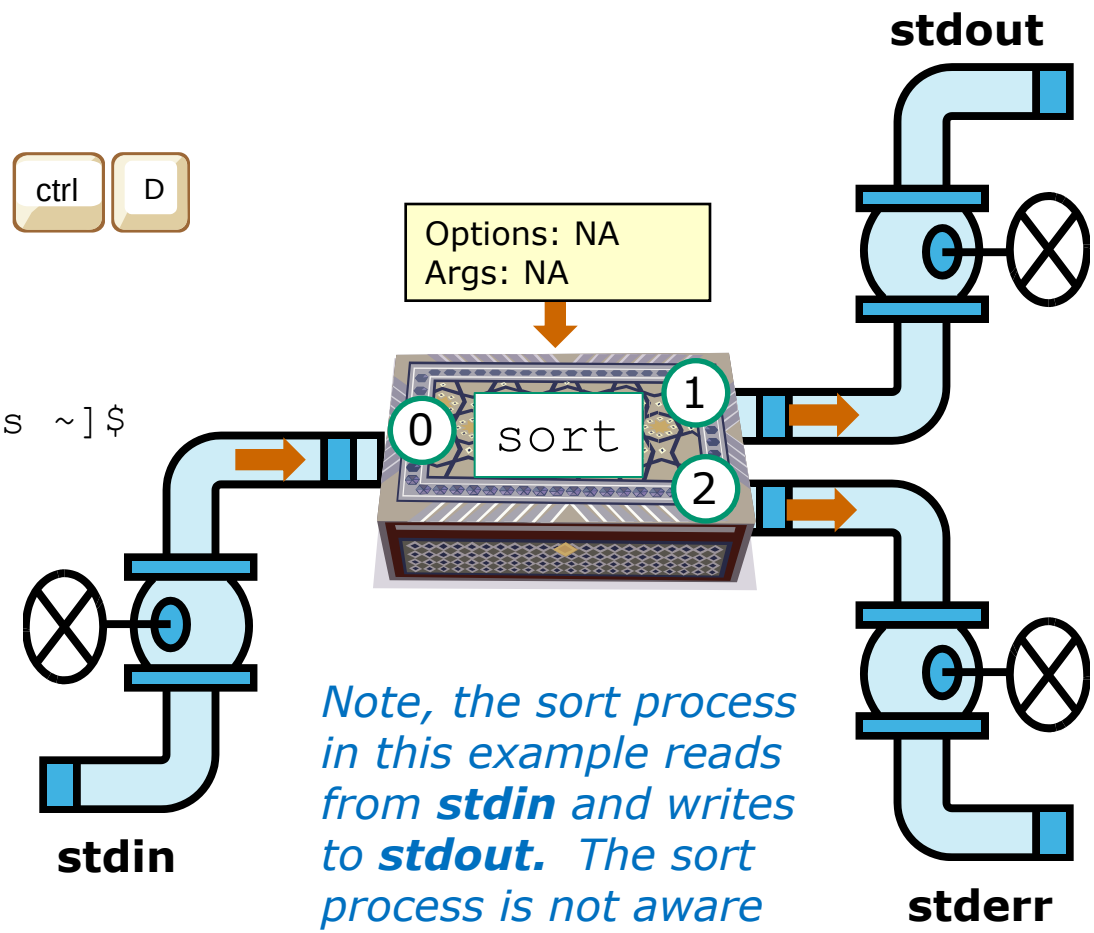
By default is attached to the user's terminal device (screen)

# Input and Output

## File Descriptors

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




```
benji
duke
homer
star
```

```
star
benji
duke
homer
```

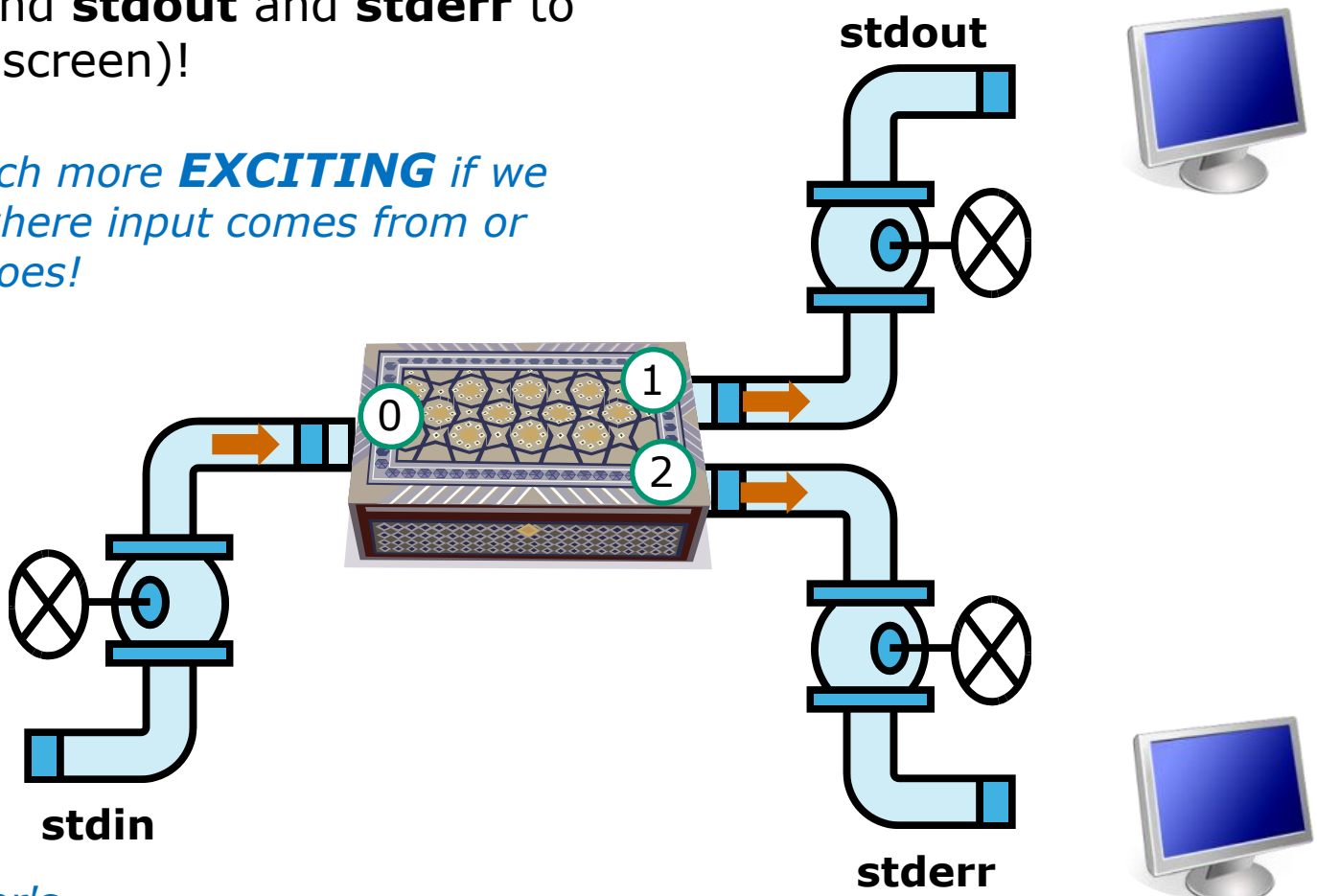
*Note, the sort process in this example reads from **stdin** and writes to **stdout.** The sort process is not aware what **stdin** or **stdout** are attached to*

# 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)*

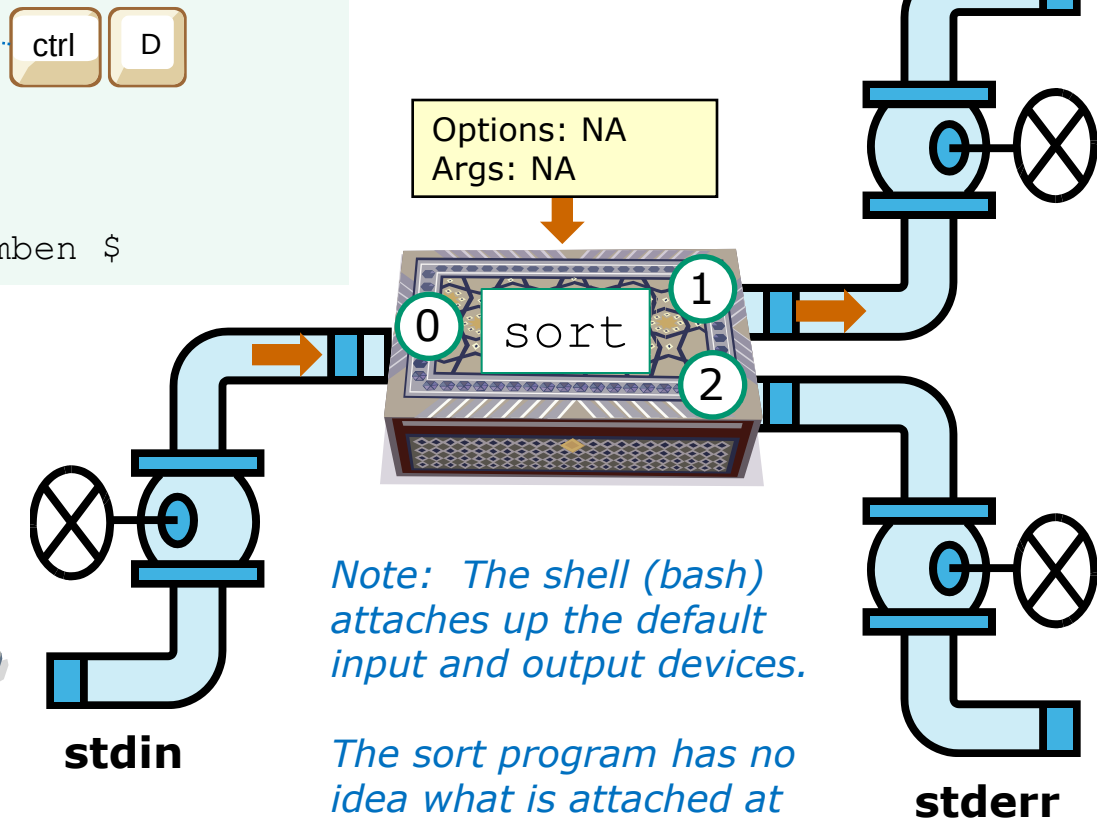

"End of File"

*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 $
```



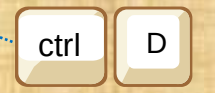

*Note: The shell (bash) attaches up the default input and output devices.*

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




"End of File"

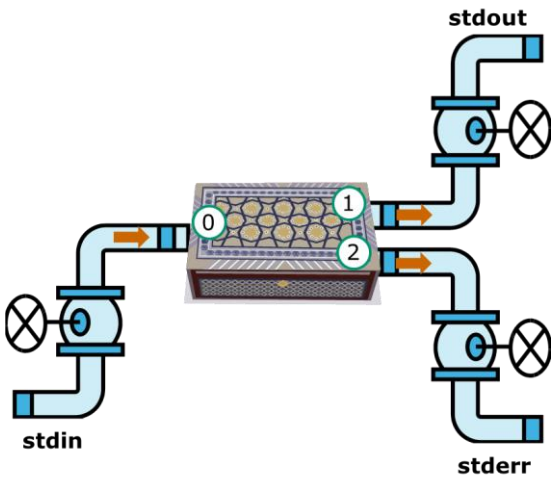
Written to **stdout**

*Now you try it*

# Input and Output

## File Redirection

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




~~**0**~~**<** ***filename***

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

~~**1**~~**>** ***filename***

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

**2>** ***filename***

*To redirect **stderr***

**>>** ***filename***

*To redirect **stdout** and append*

# No arguments, redirecting stdout

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

**stdout** has been redirected to the file *dogsinorder*

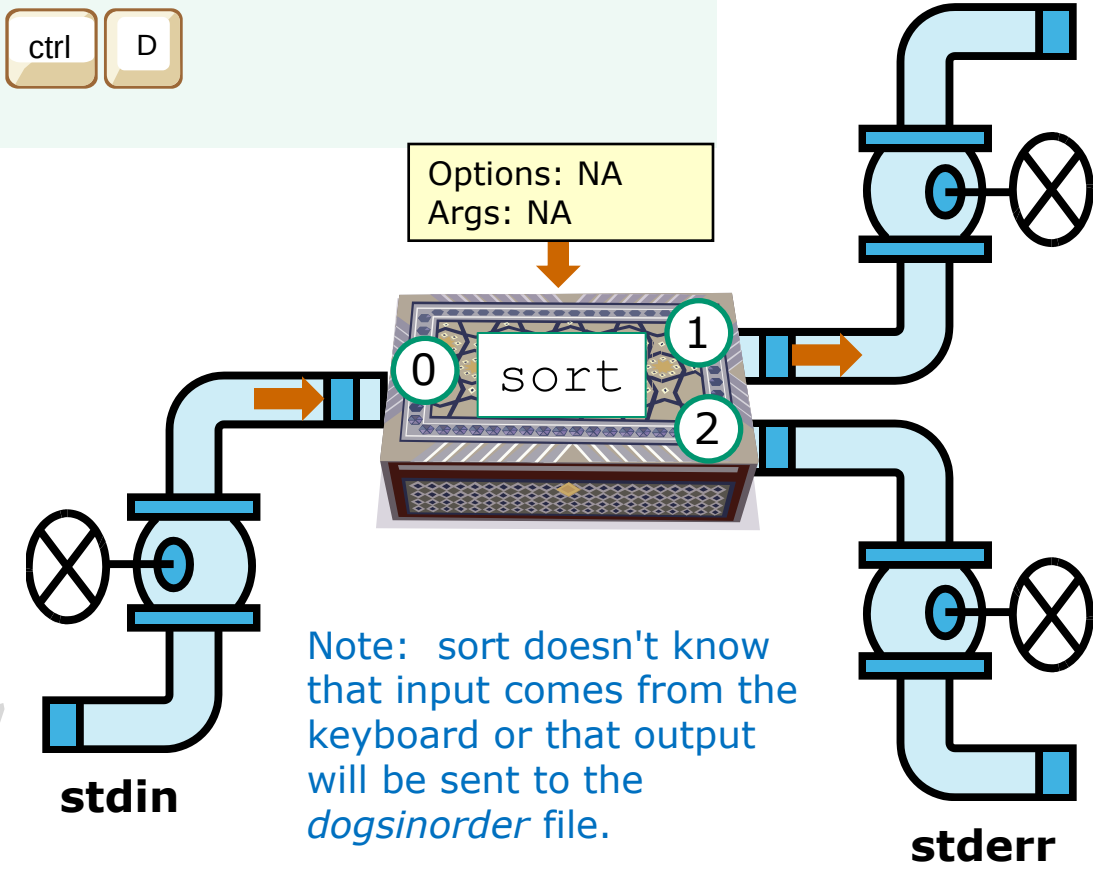
```
[simmsben@opus ~]$ sort > dogsinorder
duke
benji
star
homer
[simmsben@opus ~]$ cat dogsinorder
benji
duke
homer
star
[simmsben@opus ~]$
```

ctrl D

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
$
```





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

/dev/pts/0

duke
benji
star
homer

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**.

# 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 ~]$
```

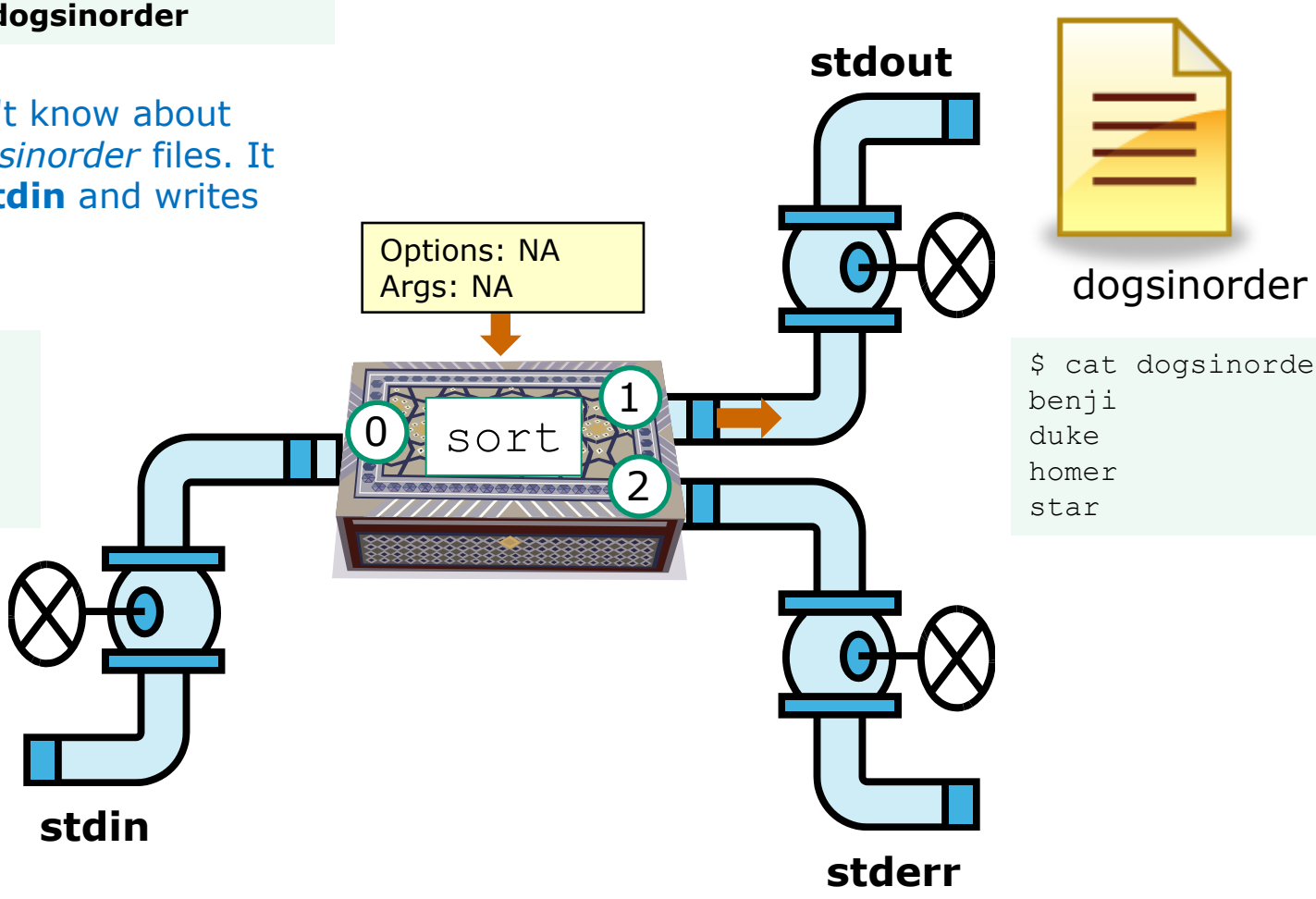
Note: The bash shell handles the command line parsing and redirection. The sort command has no idea what ***stdin*** or ***stdout*** are attached to.

# 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

# 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.

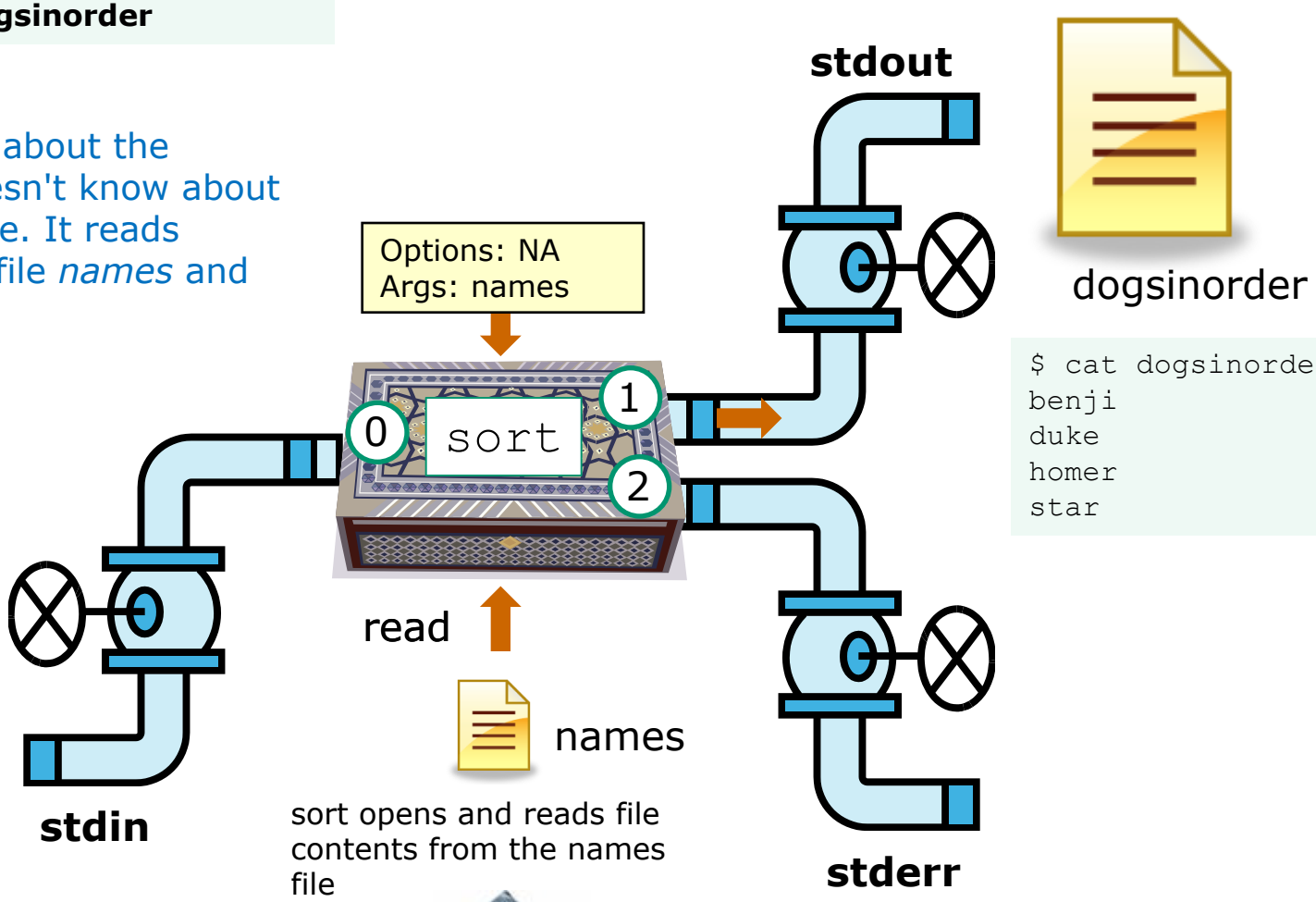
Корисне для наступного вікторини!

# 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!




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

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

# 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

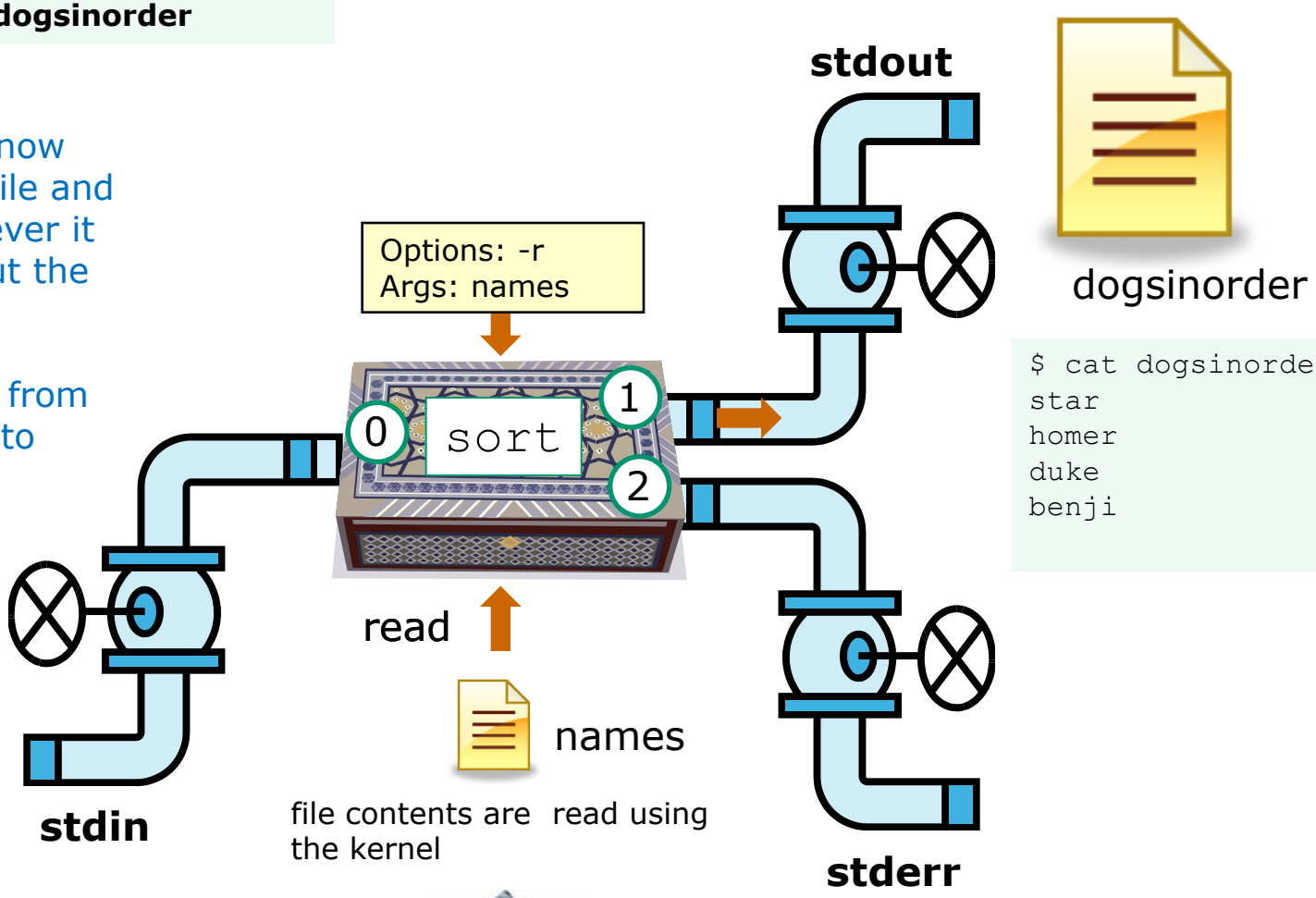
The shell opens the *dogsinorder* file. The sort process is not aware that output is redirected there.

# 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*

# More redirection examples

# 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*

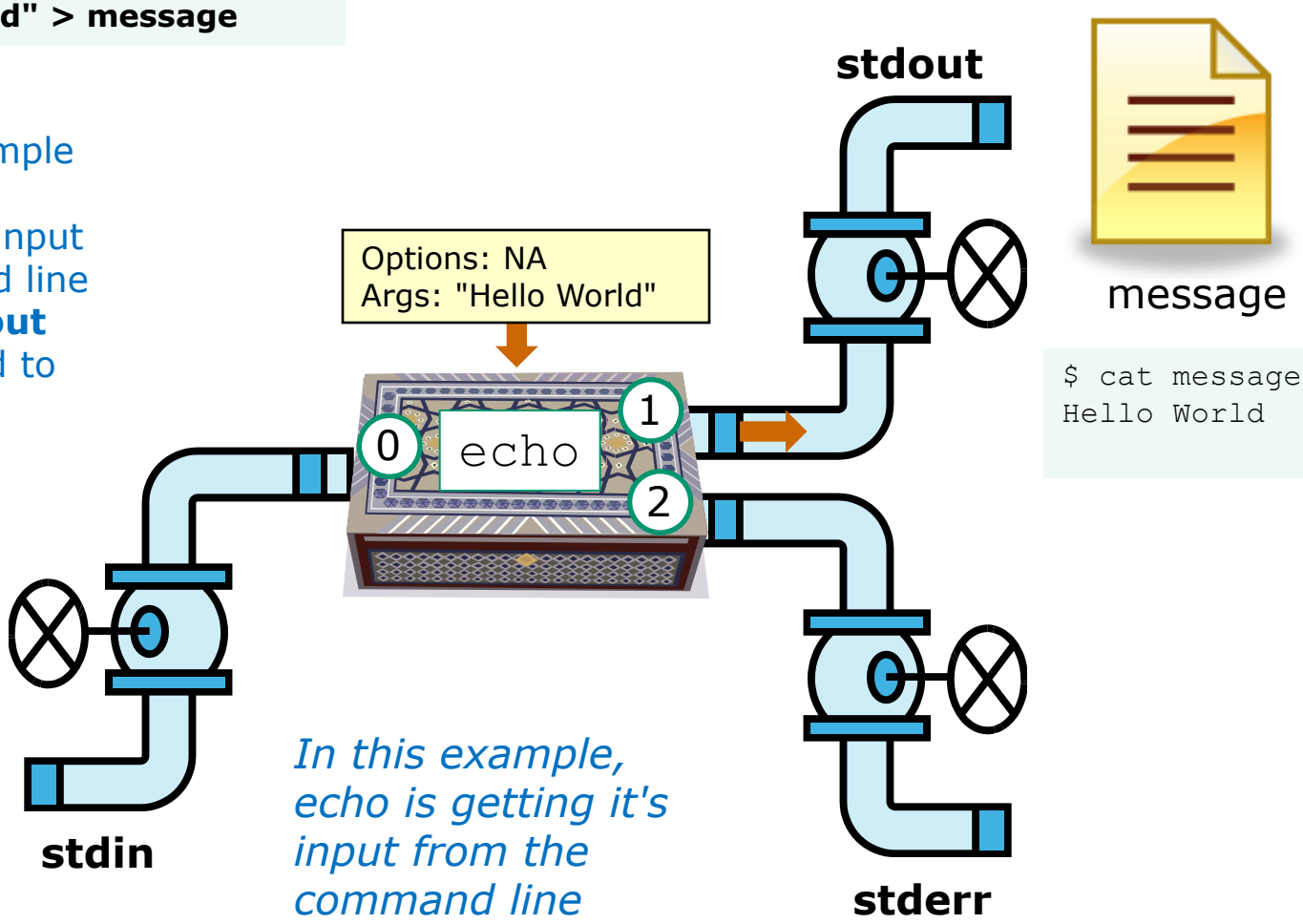
/dev/pts/1

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

# Input from the command line, redirecting stdout

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

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




```
$ cat message
Hello World
```

*In this example, echo is getting it's input from the command line*

# > (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!*

# Redirecting stdout and stderr

**Another example …**

```
[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*

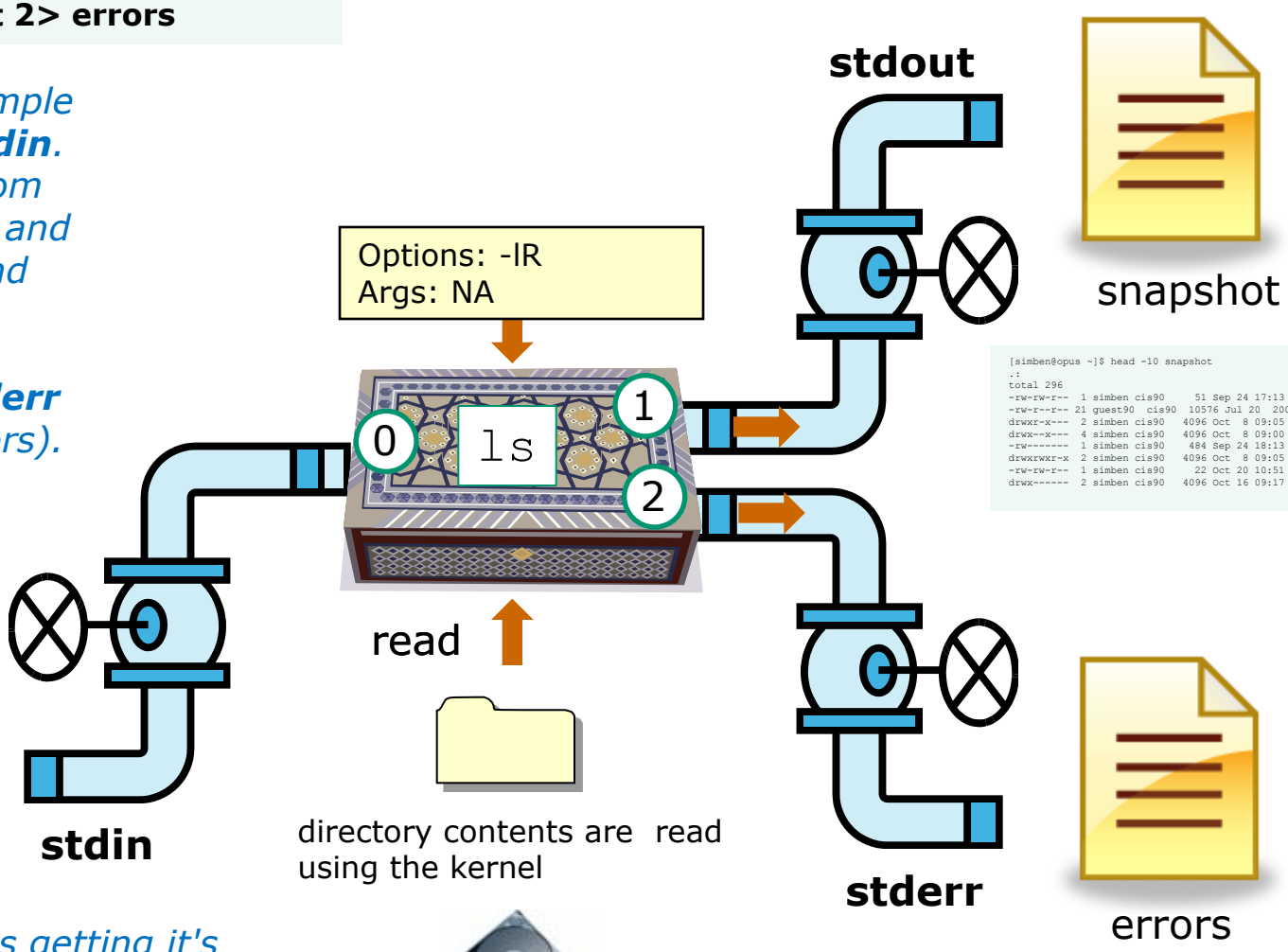
*> redirects **stdout** to file named snapshot*

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

# 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).*




```
[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
```

*In this example, ls is getting it's input from the OS*

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

# Redirecting stdin, stdout and stderr

## Using all three (<, > and 2>) on one command

```
[simben@opus ~]$ echo 2+2 > math
[simben@opus ~]$ bc < math
4
```

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

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

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

```
[simben@opus ~]$ bc < math > answers 2> errors
[simben@opus ~]$ cat answers
4
```

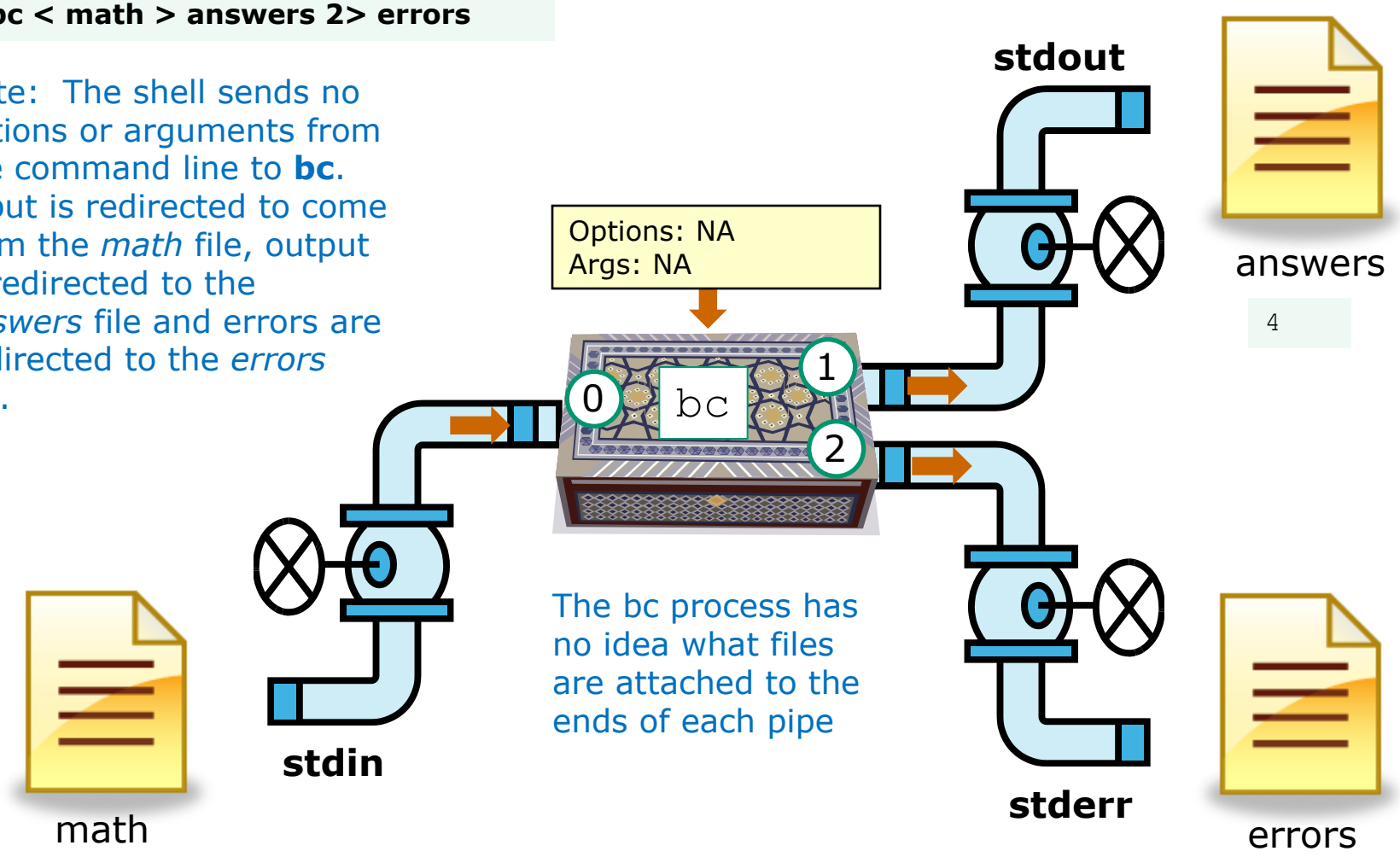
bc reads inputs from **stdin** (redirected to math), writes to **stdout** (redirected to *answers*) and writes errors to **stderr** (redirected to *errors*)

```
[simben@opus ~]$ cat errors
Runtime error (func=(main), adr=5): Divide by zero
```

# 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 bc process has no idea what files are attached to the ends of each pipe

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

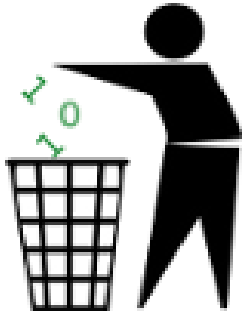
# The bit bucket

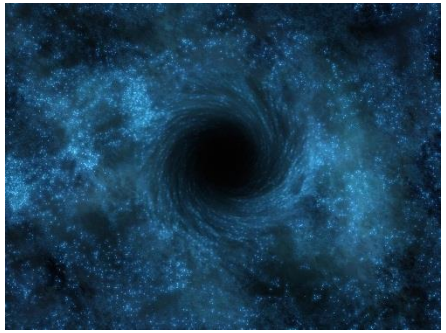
# /dev/null

# /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
```

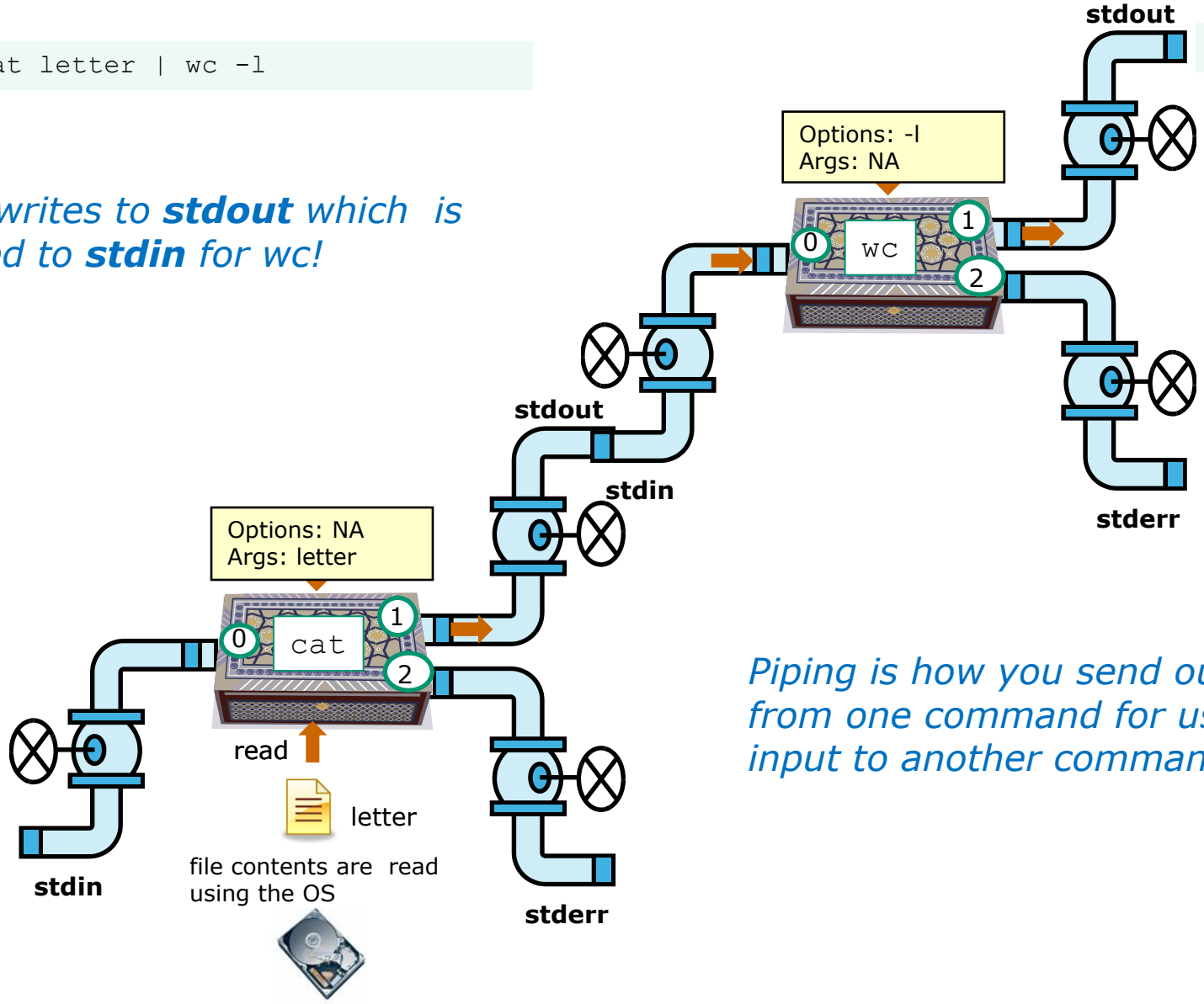
*Counting the lines in the letter file*

# 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*

# Pipeline example

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

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

*Counting, sorting and recording the currently logged in users*

# Why pipelines?

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

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

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

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

*who is logged in*

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

*who is logged in and sorted*

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

*who is logged in, sorted and counted*

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

*who is logged in, sorted, counted and saved in file named users*

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

# More Commands

# Tools for your toolbox

NEW **find** - Find file or content of a file

NEW **grep** - "Global Regular Expression Print"

**sort** - sort

NEW **spell** - spelling correction

**wc** - word count

**tee** - split output

NEW **cut** - cut fields from a line

# find command

# Find Command

Basic syntax
(see man page for the rest of the story)

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

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

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

# find command with no options or arguments

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

```
[simben@opus poems]$ find
.
./Blake
./Blake/tiger
./Blake/jerusalem
./Shakespeare
./Shakespeare/sonnet1
./Shakespeare/sonnet2
./Shakespeare/sonnet3
./Shakespeare/sonnet4
./Shakespeare/sonnet5
./Shakespeare/sonnet7
./Shakespeare/sonnet9
./Shakespeare/sonnet10
./Shakespeare/sonnet15
./Shakespeare/sonnet17
./Shakespeare/sonnet26
./Shakespeare/sonnet35
./Shakespeare/sonnet11
./Shakespeare/sonnet6
./Yeats
./Yeats/whitebirds
./Yeats/mooncat
./Yeats/old
./Anon
./Anon/ant
./Anon/nursery
./Anon/twister
[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
```

*Yuck! How annoying is this?*

*<snipped>*

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

# Redirecting find errors to the bit bucket

*redirecting stderr to the "bit bucket"*

```
[simben@opus ~]$ find  /home/cis90 -name sonnet6  2> /dev/null
/home/cis90/guest/Poems/Shakespeare/sonnet6
/home/cis90/simben/poems/Shakespeare/sonnet6
/home/cis90/stanlcha/poems/Shakespeare/sonnet6
/home/cis90/seatocol/poems/Shakespeare/sonnet6
/home/cis90/wrigholi/poems/Shakespeare/sonnet6
/home/cis90/dymesdia/poems/Shakespeare/sonnet6
/home/cis90/lyonsrob/poems/Shakespeare/sonnet6
/home/cis90/ybarrser/poems/Shakespeare/sonnet6
/home/cis90/ybarrser/poems/Sonnets/sonnet6
/home/cis90/valdemar/poems/Shakespeare/sonnet6
/home/cis90/elliokat/poems/Shakespeare/sonnet6
/home/cis90/jessuwes/poems/Shakespeare/sonnet6
/home/cis90/luisjus/poems/Shakespeare/sonnet6
/home/cis90/meyerjas/poems/Shakespeare/sonnet6
/home/cis90/bergelyl/sonnet6
/home/cis90/bergelyl/poems/Shakespeare/sonnet6
/home/cis90/gardnnic/poems/Shakespeare/sonnet6
/home/cis90/mohanchi/poems/Shakespeare/sonnet6
/home/cis90/whitfbob/poems/Shakespeare/sonnet6
/home/cis90/crivejoh/poems/Shakespeare/sonnet6
[simben@opus ~]$
```

*Ahhh … much better!*

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

*This is why we want a bit bucket*

# find command examples

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

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

```
[simben@opus ~]$ find / 2> /dev/null | wc -l
154033
```

Корисне для наступного вікторини!

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

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

*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

*start from "here"*

*specifies directories only*

*specifies only files whose names start with a B, S, Y or A*

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

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

# find command examples

*specifies only files whose names contain "town"*

*start from "here"*

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

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

# 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"*

# More filter commands

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

**cat** - concatenate

**grep** - "Global Regular Expression Print"

**sort** - sort

**spell** - spelling correction

**wc** - word count

**tee** - split output

**cut** - cut fields from a line

*Filters enable building useful pipelines*

# grep command

# grep command

Basic syntax
(see man page for the rest of the story)

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

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

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

Some other useful options:

- **-i** (case insensitive)
- **-w** (whole word)
- **-v** (does not contain)
- **-n** (show line number)

# grep for text string

*string to search for* — *files to search contents of*

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

*files that contain love*

*Looking for love in all the wrong places?*

*Find the word love in Shakespeare's sonnets*

# grep the output of a grep

*string to search for in the output of the previous command*

*string to search for*

*files to search contents of*

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

*Find all lines with both love and hate*

# grep using the -n (line number) option

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

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

*Show account in /etc/passwd for simben90*

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

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

*Found in line 52 of /etc/passwd*

*Same as before but include line number it was found on*

# grep using the -i (case insensitive) option

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

*Look for "so" in sonnet3, sonnet4 and sonnet5*

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

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

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

# grep using the -w (whole word) option

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

*Look for "so" in sonnet3, sonnet4 and sonnet5*

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

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

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

# grep recursively with the -R option

*Text string to search for*

*starting directory*
*(. is the current directory)*

*Search recursively*
*(all sub-directories )*

*discard permission errors*

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

*Search recursively for files containing "kind"*

# grep command

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

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

All configuration files are kept in /etc.

**Tasks**
- Can you use **grep** to find the Apache configuration file?
  *Hint: use the **-R** option to recursively search all sub-directories*

- What are the names of the files in Apache's document root directory on Opus?
  *Hint: Use the **ls** command on the document root directory*

# spell command

# spell command

Basic syntax
(see man page for the rest of the story)

**spell** <*filepath*>

**spell** <*filepath*> <*filepath*> …

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

# spell command

*Task: Run a spell check on the magna_cart file*

```
/home/cis90/simben $ cd docs
/home/cis90/simben/docs $ ls
magna_carta  MarkTwain  policy
/home/cis90/simben/docs $ spell magna_carta
Anjou
Arundel
Aymeric
Bergh
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
```

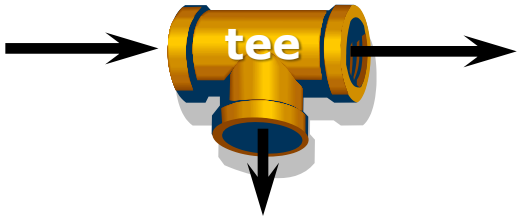
*The misspelled words from spell are piped to the tee command*

*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*

*output from **spell** command*




*input to **wc** command*

*redirected to mistakes file*

# cut command

# cut command

Basic syntax
(see man page for the rest of the story)

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

The **cut** command cuts a field from a line where each field is delimited by a delimiter (e.g. space, ":", etc.).

Use the **-c** option to cut by column numbers

# cut command

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

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

*This example shows how to cut the 7th field (the shell) from a line in /etc/passwd.*

*Each field in /etc/passwd is delimited by the":" character.*

# cut command

```
/home/cis90/simben $ ls -l letter
-rw-r--r--. 1 simben90 cis90 1044 Jul 20  2001 letter
/home/cis90/simben $ ls -l letter | cut -c 2-10
rw-r--r--
```

*Cutting columns 2 through 10 from a long listing letter file.*

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

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

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

*Which we can use to build a custom message*

# Pipeline Practice

# Class Exercise

## Pipeline Tasks

**Background**

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

**Task**

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

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

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

# Class Exercise

## Pipeline Tasks

**Background**

The **cut** command can cut a field out of a line of text where each field is 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?*

# Wrap up

New commands:

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

# Next Class

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

Lab 7

Quiz questions for next class:

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

# Backup

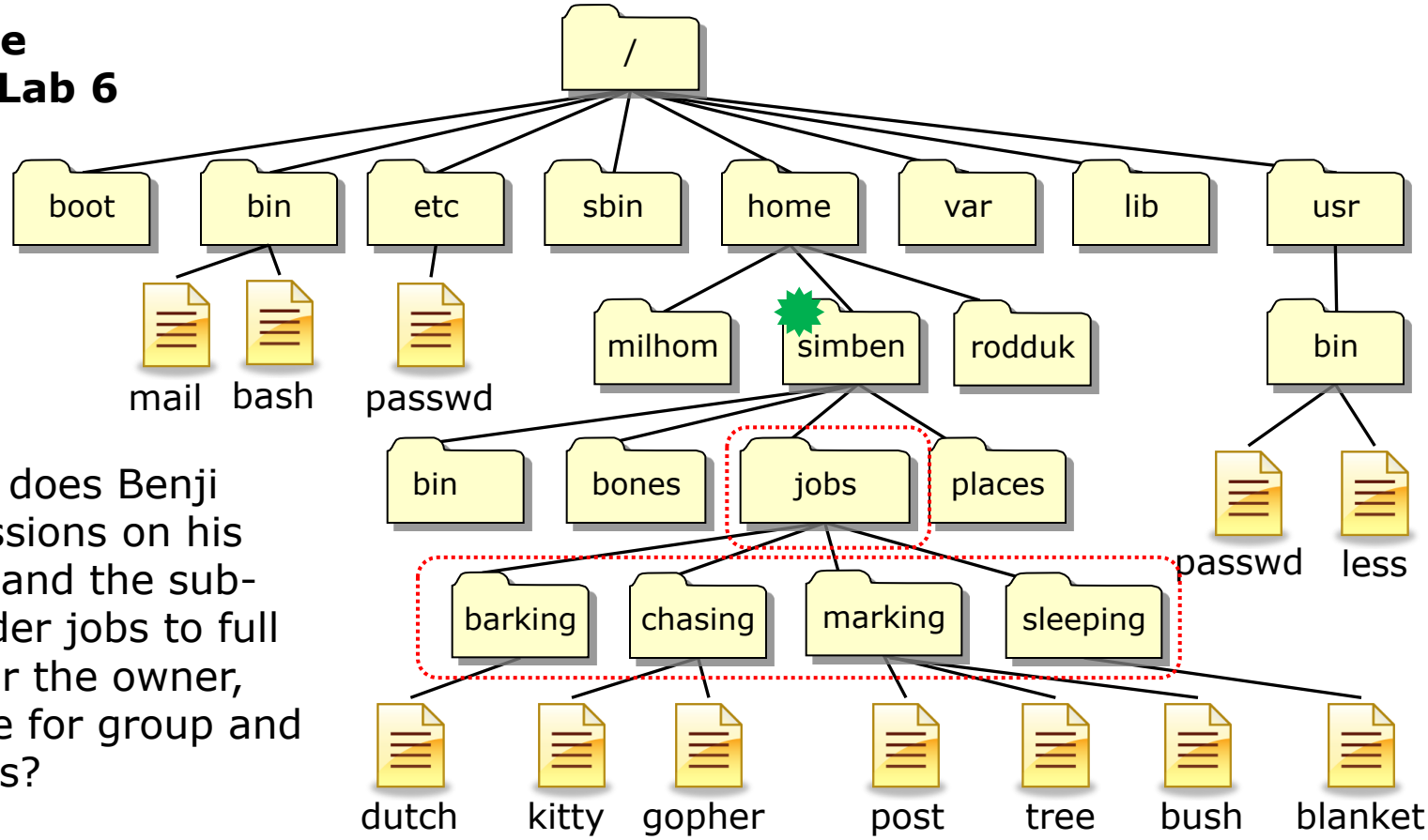
# Lab 6 Tips

```
/home/cis90/simben $ tree poems/
poems/
├── Anon
│   ├── ant
│   ├── nursery
│   └── twister
├── Blake
│   ├── jerusalem
│   └── tiger
├── Shakespeare
│   ├── sonnet1
│   ├── sonnet10
│   ├── sonnet11
│   ├── sonnet15
│   ├── sonnet17
│   ├── sonnet2
│   ├── sonnet26
│   ├── sonnet3
│   ├── sonnet35
│   ├── sonnet4
│   ├── sonnet5
│   ├── sonnet6
│   ├── sonnet7
│   └── sonnet9
└── Yeats
    ├── mooncat
    ├── old
    └── whitebirds

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

*One of the steps in Lab 6*

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

**An example related to Lab 6 Q9**

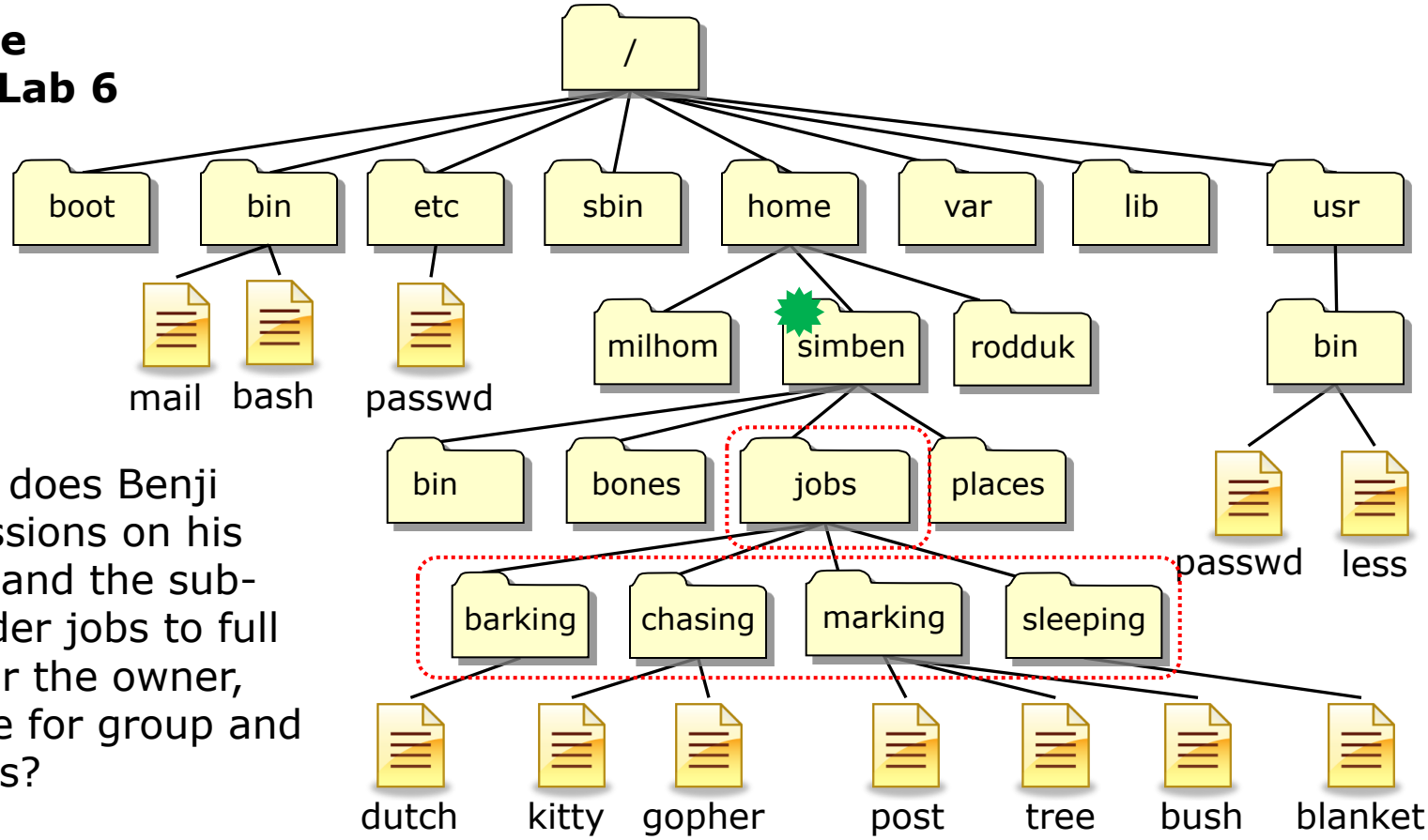



From ✸ how does Benji change permissions on his jobs directory and the sub-directories under jobs to full permissions for the owner, read & execute for group and none for others?

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

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

**An example related to Lab 6 Q9**

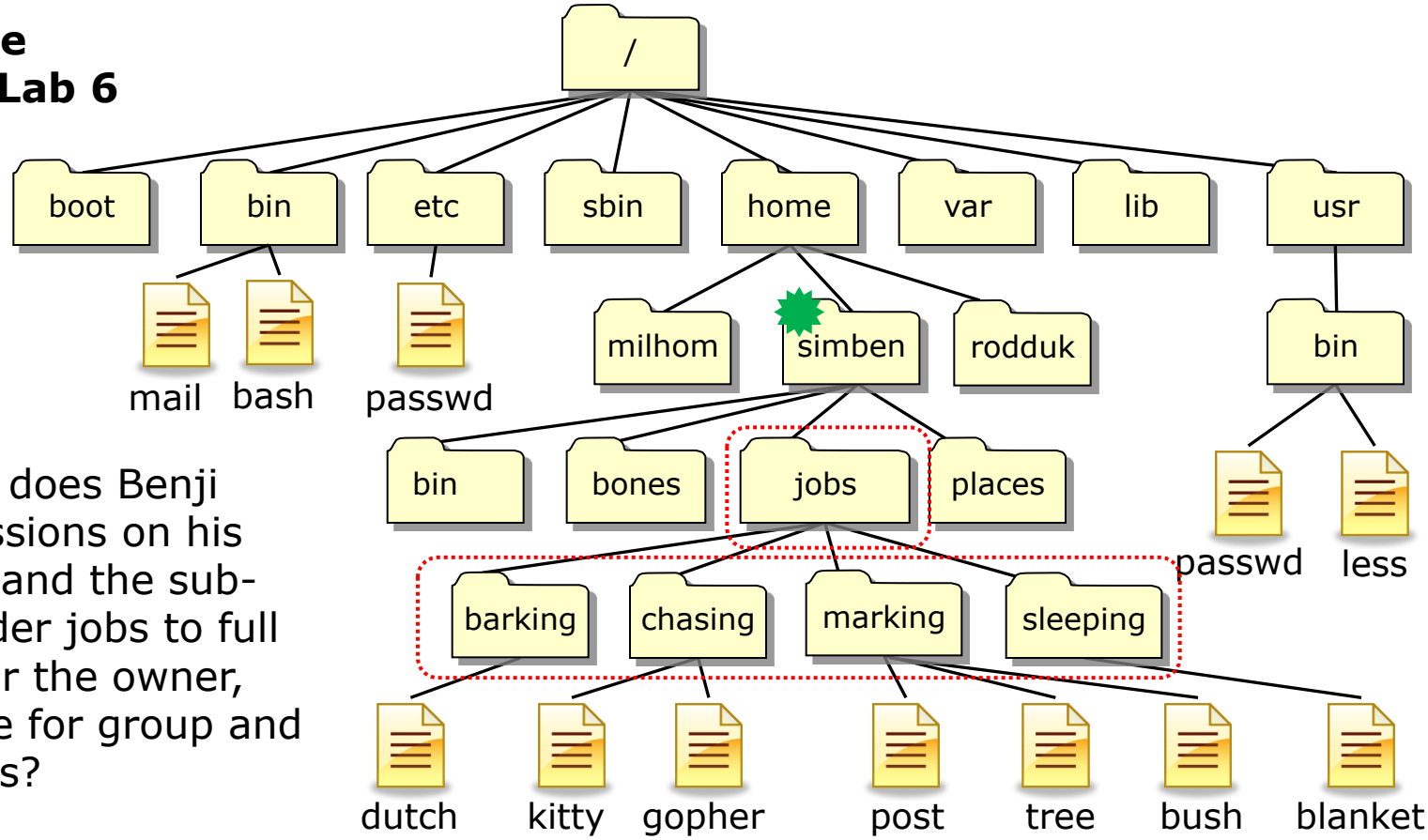



From ✸ how does Benji change permissions on his jobs directory and the sub-directories under jobs to full permissions for the owner, read & execute for group and none for others?

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

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

**An example related to Lab 6 Q9**

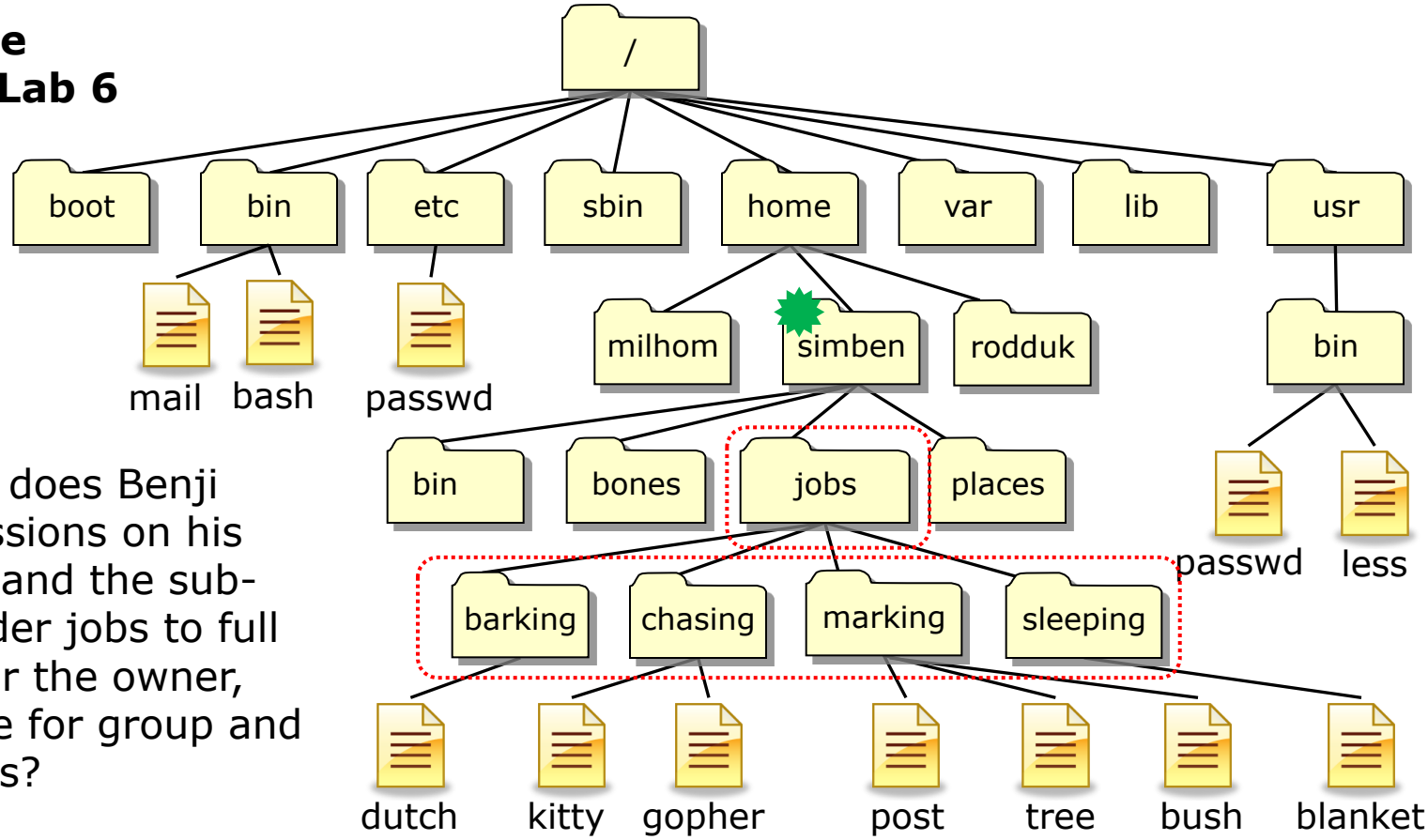



From ✸ how does Benji change permissions on his jobs directory and the sub-directories under jobs to full permissions for the owner, read & execute for group and none for others?

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

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

**An example related to Lab 6 Q9**

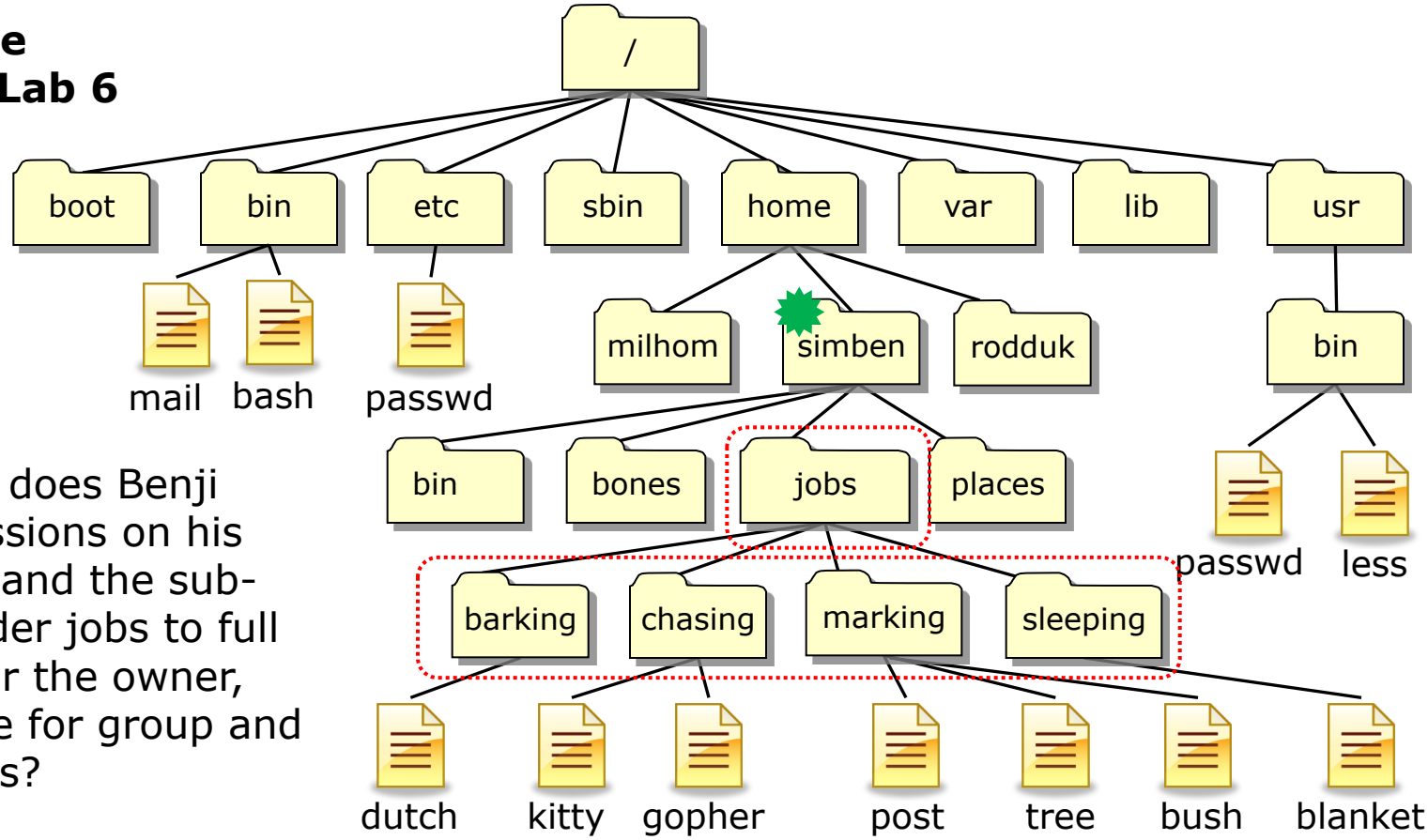



From ✸ how does Benji change permissions on his jobs directory and the sub-directories under jobs to full permissions for the owner, read & execute for group and none for others?

**chmod 750 jobs jobs/***

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

**An example related to Lab 6 Q9**




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

*The "elbow grease" method:*

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

*Both ways work, the choice is yours!*

*The "Linux guru" method:*

**chmod 750 jobs jobs/***

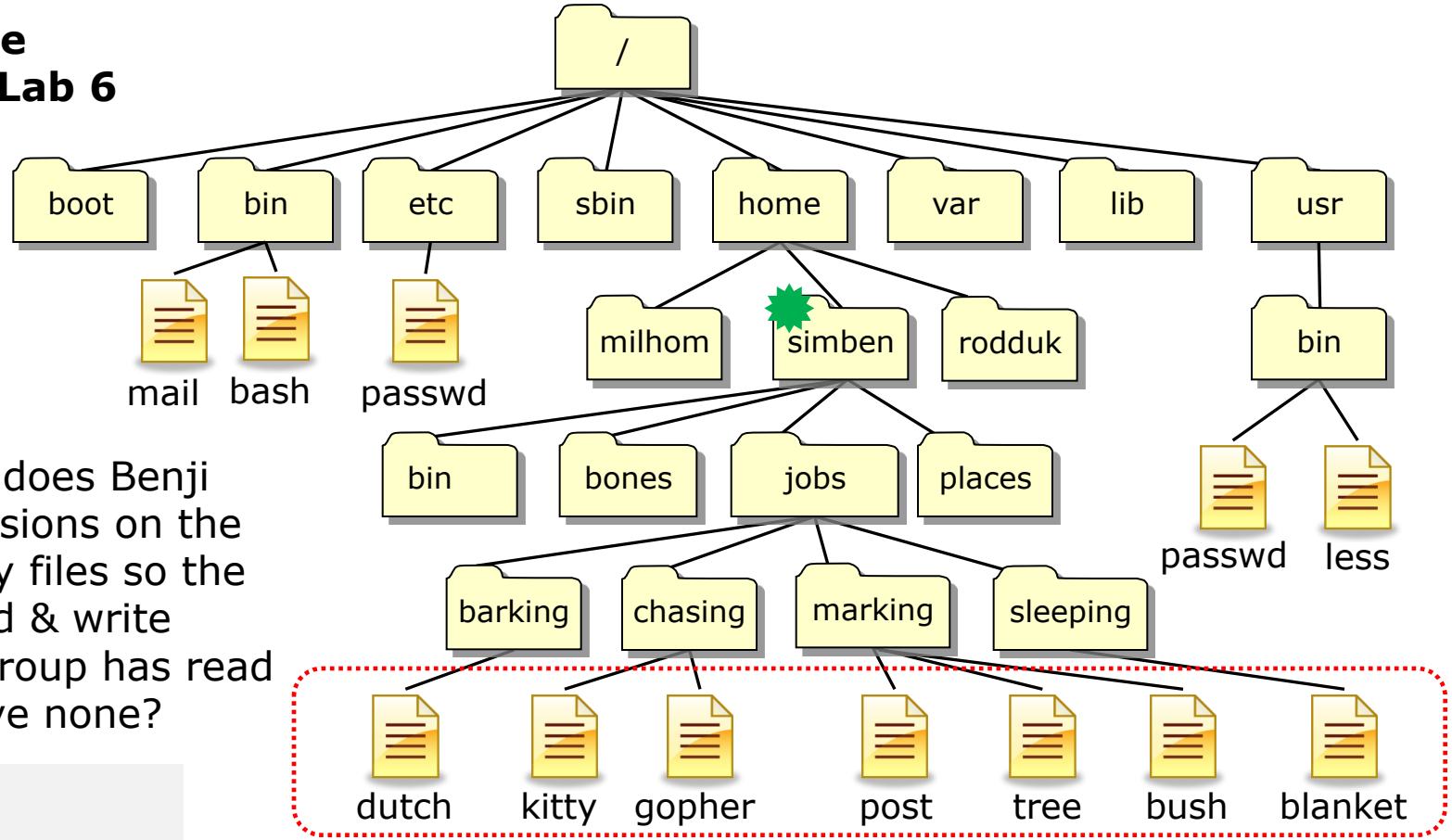
```
/home/cis90/simben $ tree poems/
poems/
├── Anon
│   ├── ant
│   ├── nursery
│   └── twister
├── Blake
│   ├── jerusalem
│   └── tiger
├── Shakespeare
│   ├── sonnet1
│   ├── sonnet10
│   ├── sonnet11
│   ├── sonnet15
│   ├── sonnet17
│   ├── sonnet2
│   ├── sonnet26
│   ├── sonnet3
│   ├── sonnet35
│   ├── sonnet4
│   ├── sonnet5
│   ├── sonnet6
│   ├── sonnet7
│   └── sonnet9
└── Yeats
    ├── mooncat
    ├── old
    └── whitebirds

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

*Another step in Lab 6*

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

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

**An example related to Lab 6 Q10**




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

**cd jobs**
**cd barking**
**chmod 640 dutch**
**cd ..**

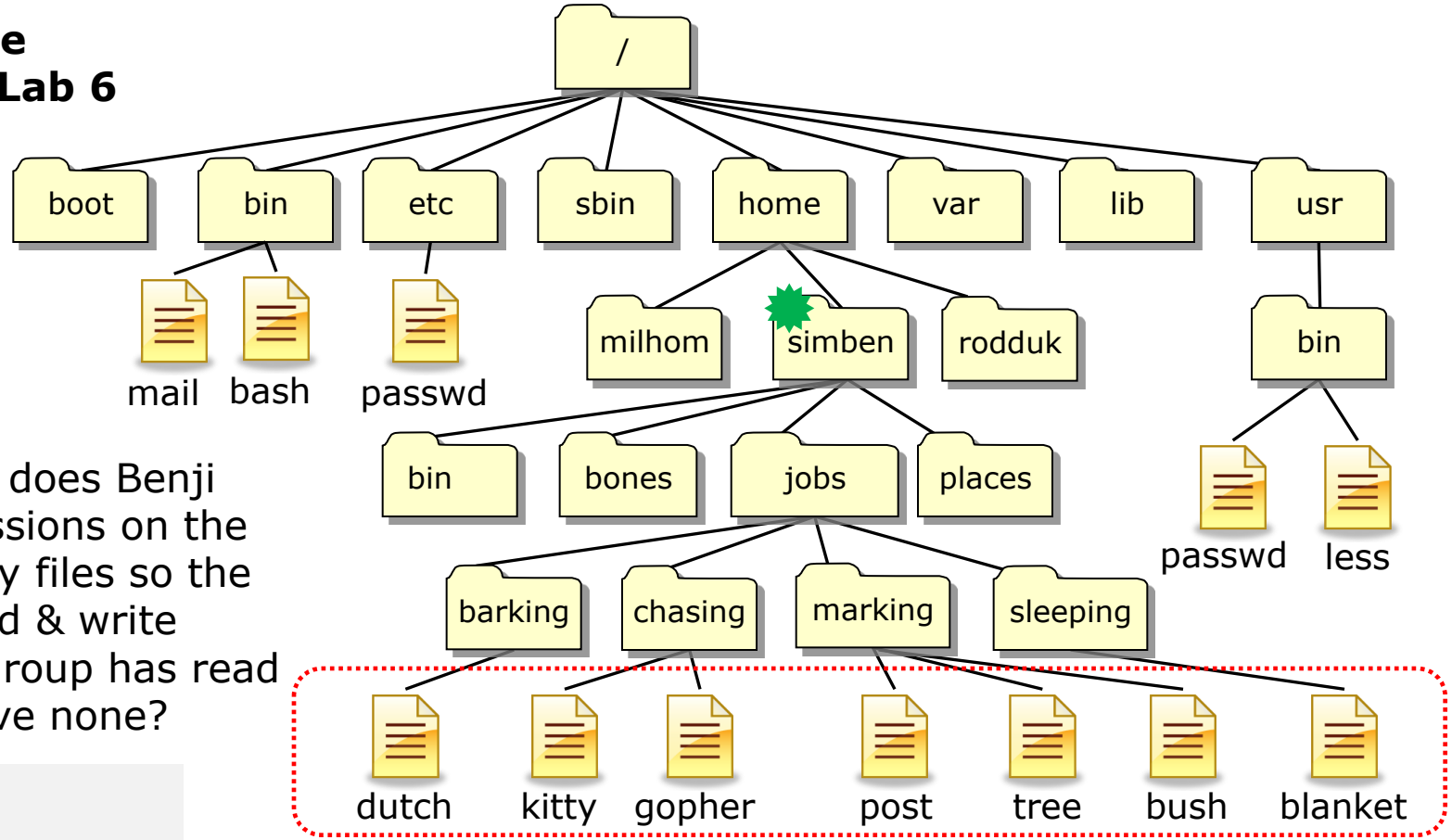
*The "elbow grease" method takes 16 commands*

**cd chasing**
**chmod 640 kitty**
**chmod 640 gopher**
**cd ..**

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

**cd sleeping**
**chmod 640 blanket**
**cd**

**An example related to Lab 6 Q10**




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

**cd jobs**
**cd barking**
**chmod 640 dutch**
**cd ..**

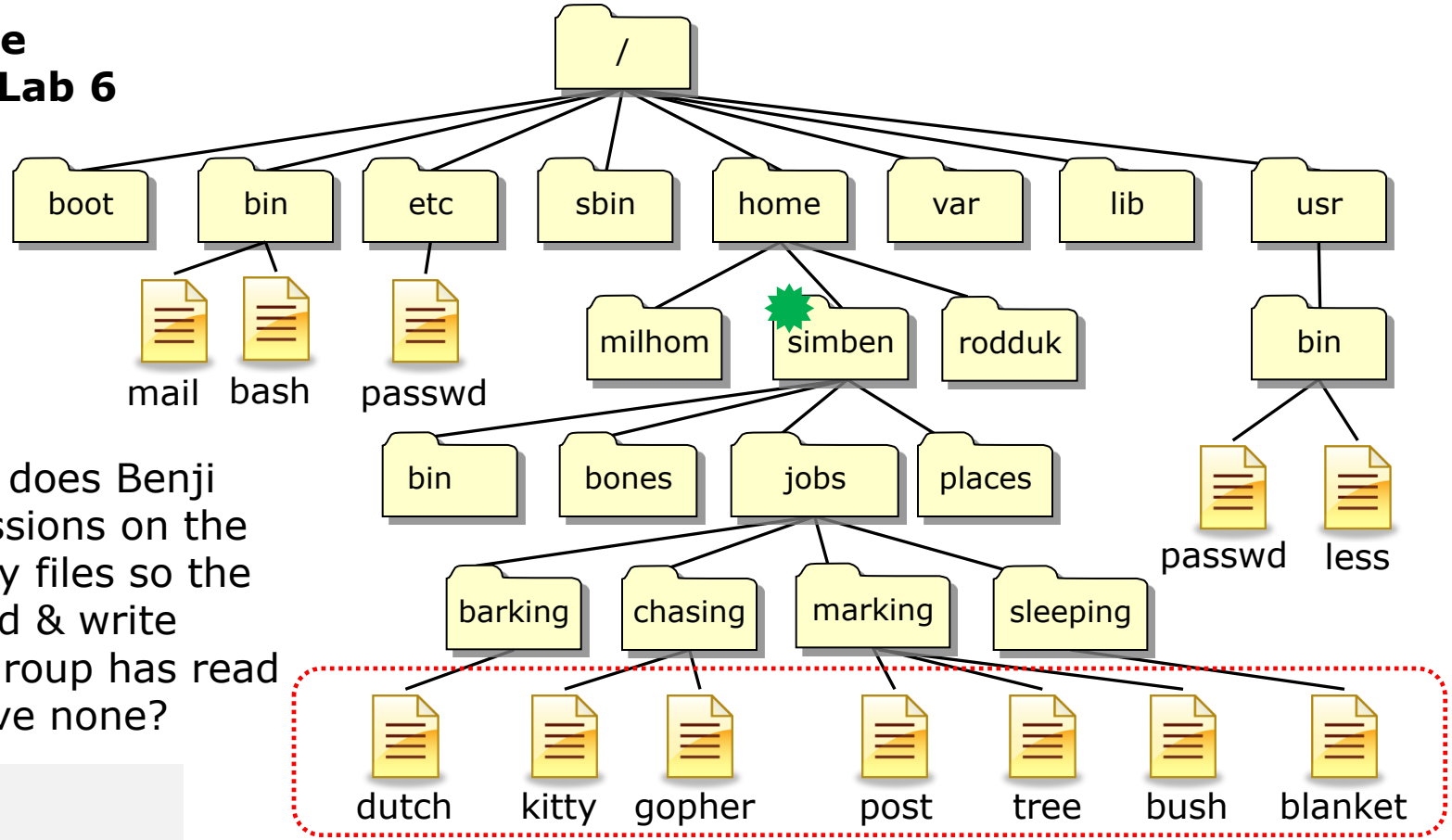
**cd chasing**
**chmod 640 kitty gopher**
**cd ..**

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

**cd sleeping**
**chmod 640 blanket**
**cd**

*Using multiple arguments on chmod: takes 13 commands*

**An example related to Lab 6 Q10**




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

**cd jobs**
**cd barking**
**chmod 640 ***
**cd ..**

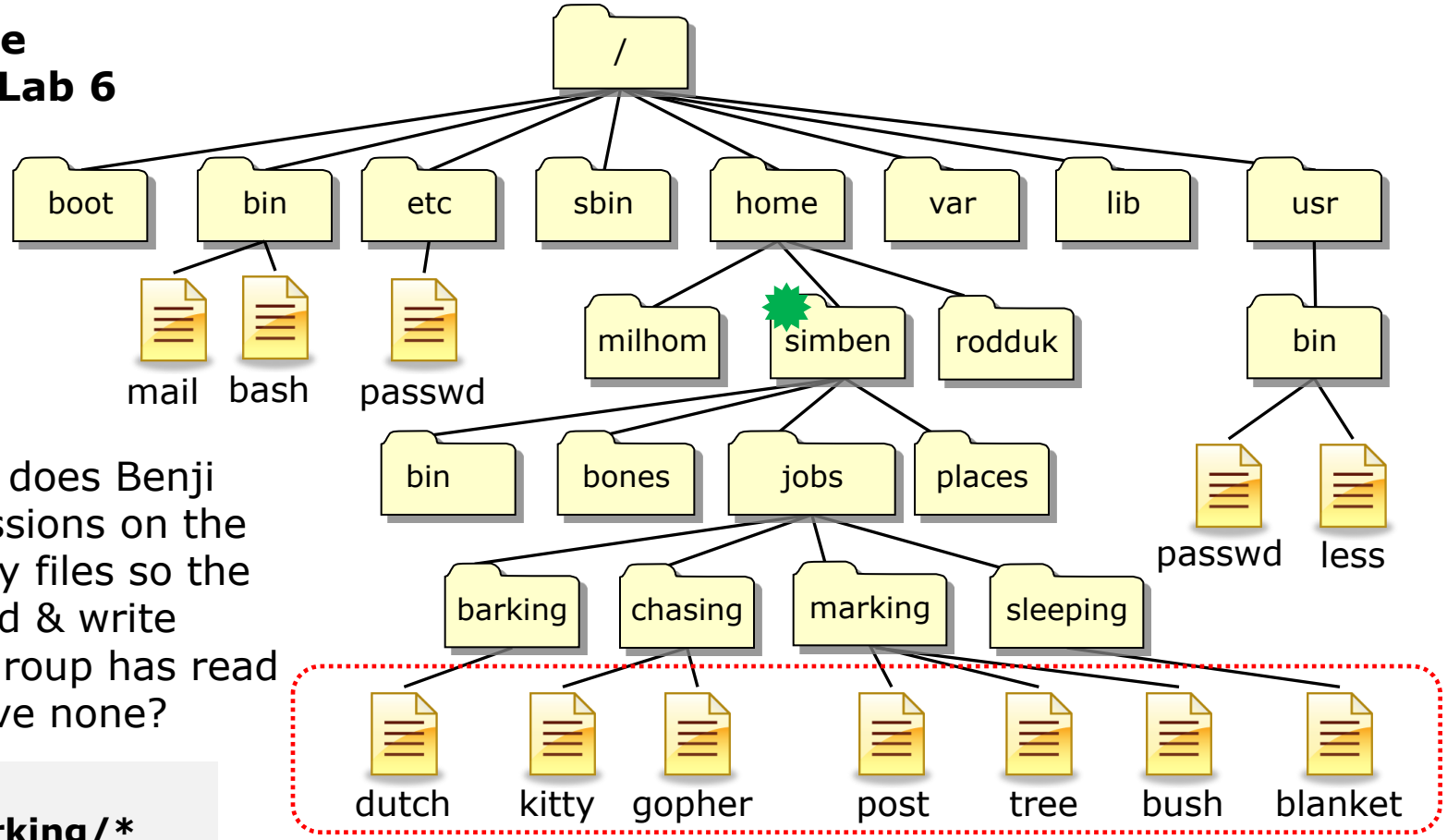
**cd chasing**
**chmod 640 ***
**cd ..**

**cd marking**
**chmod 640 ***
**cd ..**

**cd sleeping**
**chmod 640 ***
**cd**

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

**An example related to Lab 6 Q10**

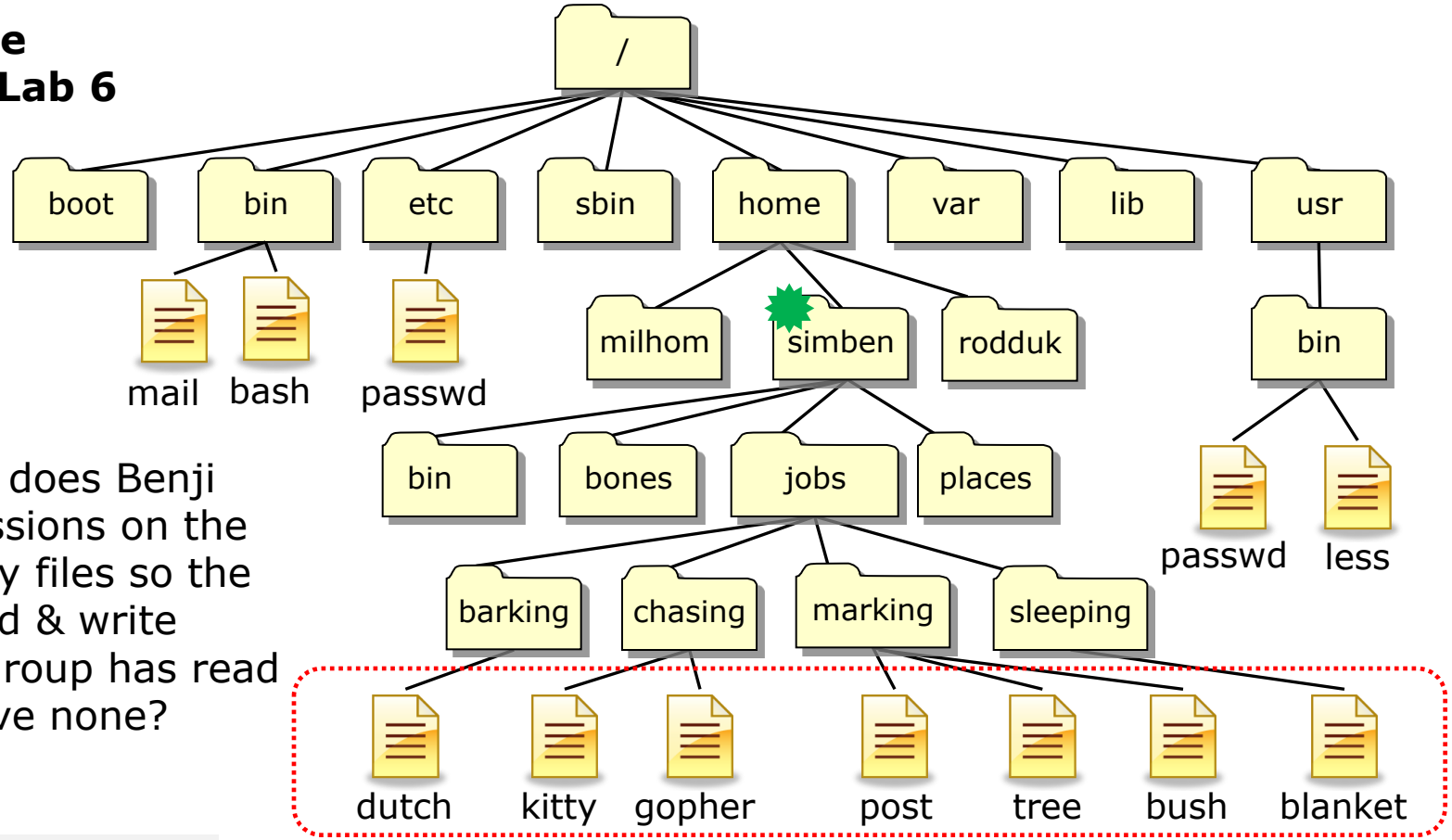



From ✸ how does Benji change permissions on the circled ordinary files so the owner has read & write permissions, group has read and others have none?

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

*Using relative paths and filename expansion characters takes 6 commands*

**An example related to Lab 6 Q10**

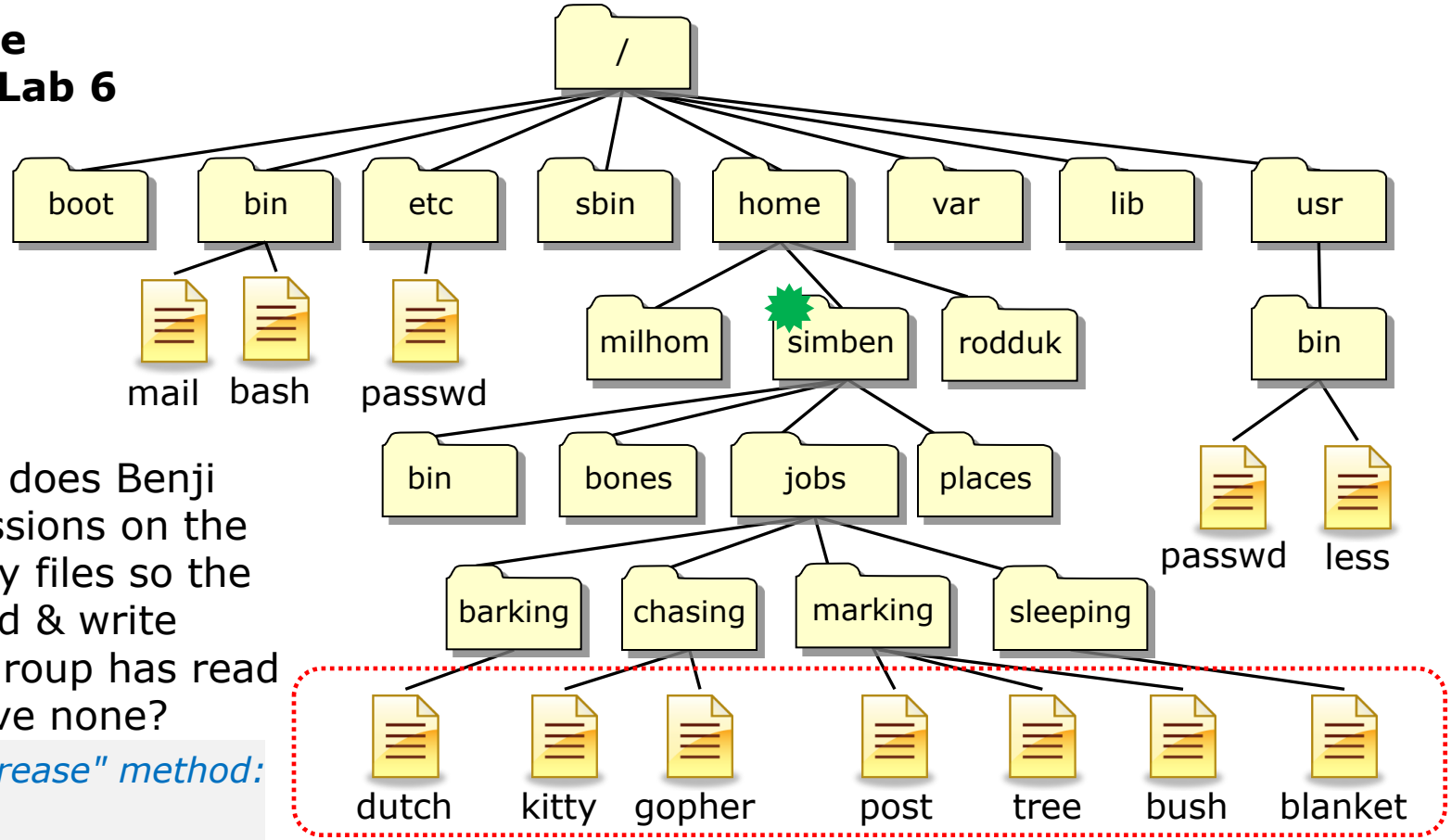



From ✸ how does Benji change permissions on the circled ordinary files so the owner has read & write permissions, group has read and others have none?

**chmod 640 jobs/*/***

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

**An example related to Lab 6 Q10**




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

*The "elbow grease" method:*

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

*Both ways work, the choice is yours!*

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

# Permissions Review

# File Permissions

## Binary

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

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

r (read) is the 4's column
w (write) is the 2's column
x (execute) is the 1's column

# File Permissions

*An example long listing*

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

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

*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
```

| user (owner) | | | group | | | others | | |
|---|---|---|---|---|---|---|---|---|
| r | w | - | r | - | - | r | - | - |
| read | write | execute | read | write | execute | read | write | execute |

<u>The permissions on letter:</u>

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

# Permissions

## "The rest of the story"

- Special Permissions
- ACLs
- Extended Attributes
- SELinux

FYI only

*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*

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.

**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/
```

*set sticky bit*

```
/home/cis90/simben $ chmod 1777 tempdir
/home/cis90/simben $ ls -ld tempdir/
drwxrwxrwt. 2 simben90 cis90 4096 Oct 16 15:25 tempdir/
```

*sticky bit set*

*green background with black text*

FYI only

# Special Permissions

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

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

*red background with gray text*

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

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

FYI only

# ACLs (Access Control Lists)

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

FYI only

# ACLs (Access Control Lists)

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

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

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

*Create a file and set permissions to 444*

*Use **getfacl** to show ACLs*

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

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

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

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

FYI only

# ACLs (Access Control Lists)

*Let's give special permissions to one user*

*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*

FYI only

# 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
```

!!

FYI only

# Extended File Attributes

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

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

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

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

FYI only

# Extended File Attributes

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

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

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

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

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

FYI only

# 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).

FYI only

# 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*

FYI only

# 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
```

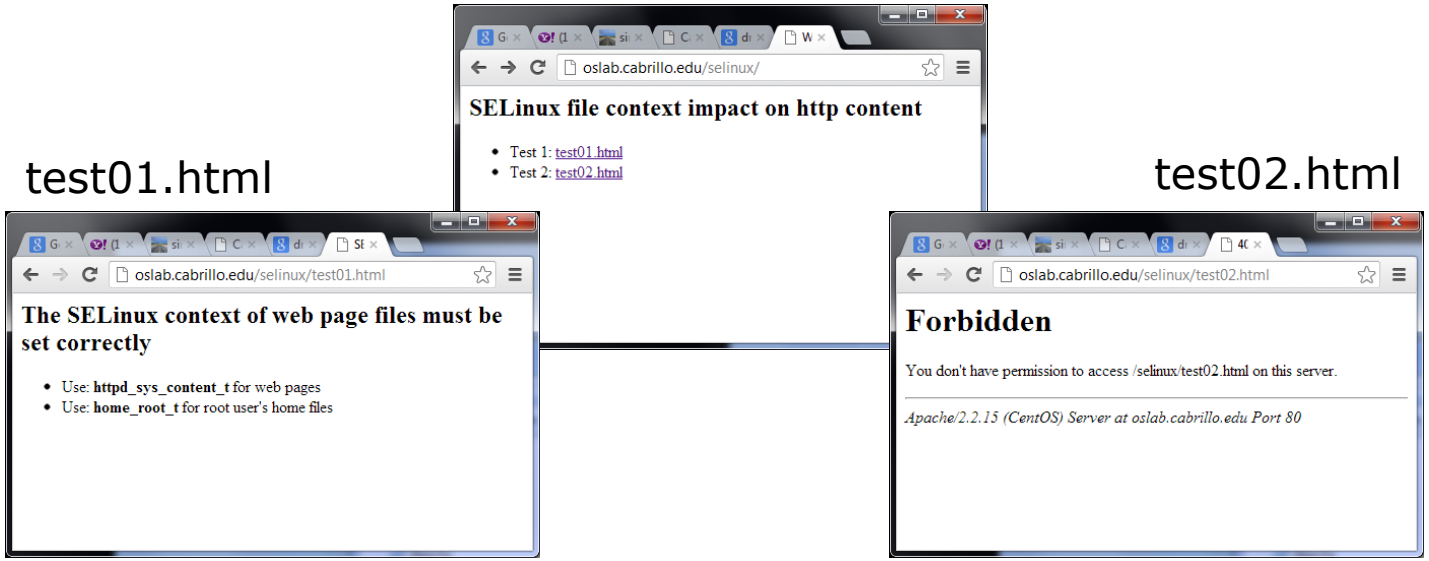
*Note, the root user's home files are not appropriate web content*

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]#
```




*type = httpd_sys_content_t*

*type = home_root_t*