Lesson Module Checklist

- Slides
- WB

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

- Lab tested and uploaded
- T2 mods made to Opus and Sun-Hwa-II
- Real test uploaded and permissions set

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

# Introductions and Credits

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

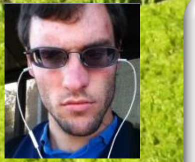
Rich Simms
- HP Alumnus
- Started teaching this course in 2008 when Jim went on sabbatical
- Rich’s site: http://simms-teach.com

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 10

Aaron

Andrew B.


Andrew C.

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

Arthur

Brian

Cory

Daniel

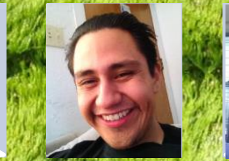
David G.

Dave L.

David P.

Debbie

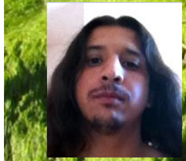
Edtson


Fidel

Humberto

Hunter

Imara


Ismael

Jessica

Joseph

Juliana

Lucie

Marc

Marty

Matt

Michael

Rochelle

Shawn

Tabitha

Taylor

Tyler

Will

Zachary

Zsolt

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

# Quiz

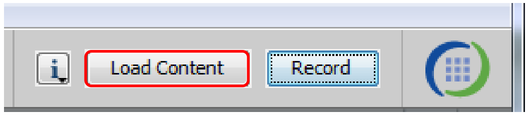
Please answer these questions **in the order** shown:

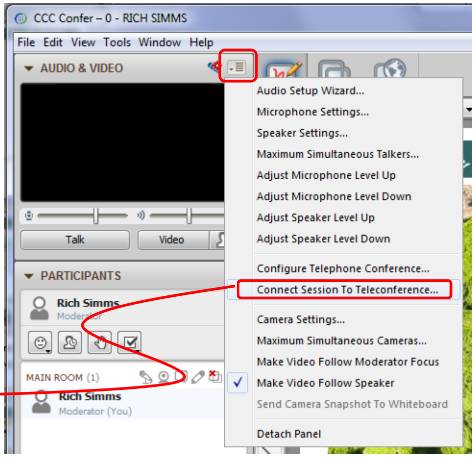
**No Quiz today … test instead**

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

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

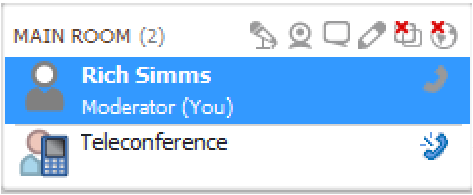
**[ ] Preload White Board with *cis*lesson??*-WB***



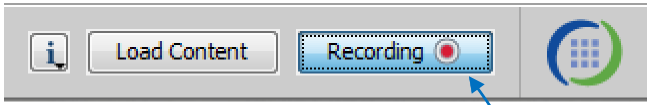

CCC Confer – 0 - RICH SIMMS
File Edit View Tools Window Help
AUDIO & VIDEO
Audio Setup Wizard...
Microphone Settings...
Speaker Settings...
Maximum Simultaneous Talkers...
Adjust Microphone Level Up
Adjust Microphone Level Down
Adjust Speaker Level Up
Adjust Speaker Level Down
Configure Telephone Conference...
Connect Session To Teleconference...
Camera Settings...
Maximum Simultaneous Cameras...
Make Video Follow Moderator Focus
✓ Make Video Follow Speaker
Send Camera Snapshot To Whiteboard
Detach Panel
Talk
Video
PARTICIPANTS
Rich Simms
Moderator
MAIN ROOM (1)
Rich Simms
Moderator (You)

**[ ] Connect session to Teleconference**

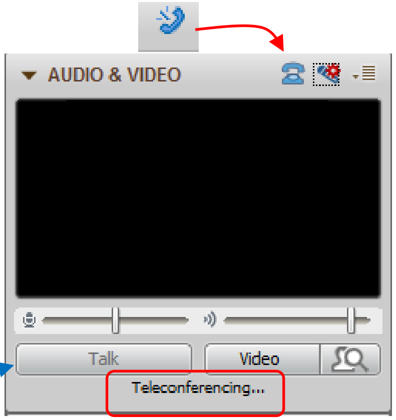
*Session now connected to teleconference*



**[ ] Is recording on?**



*Red dot means recording*

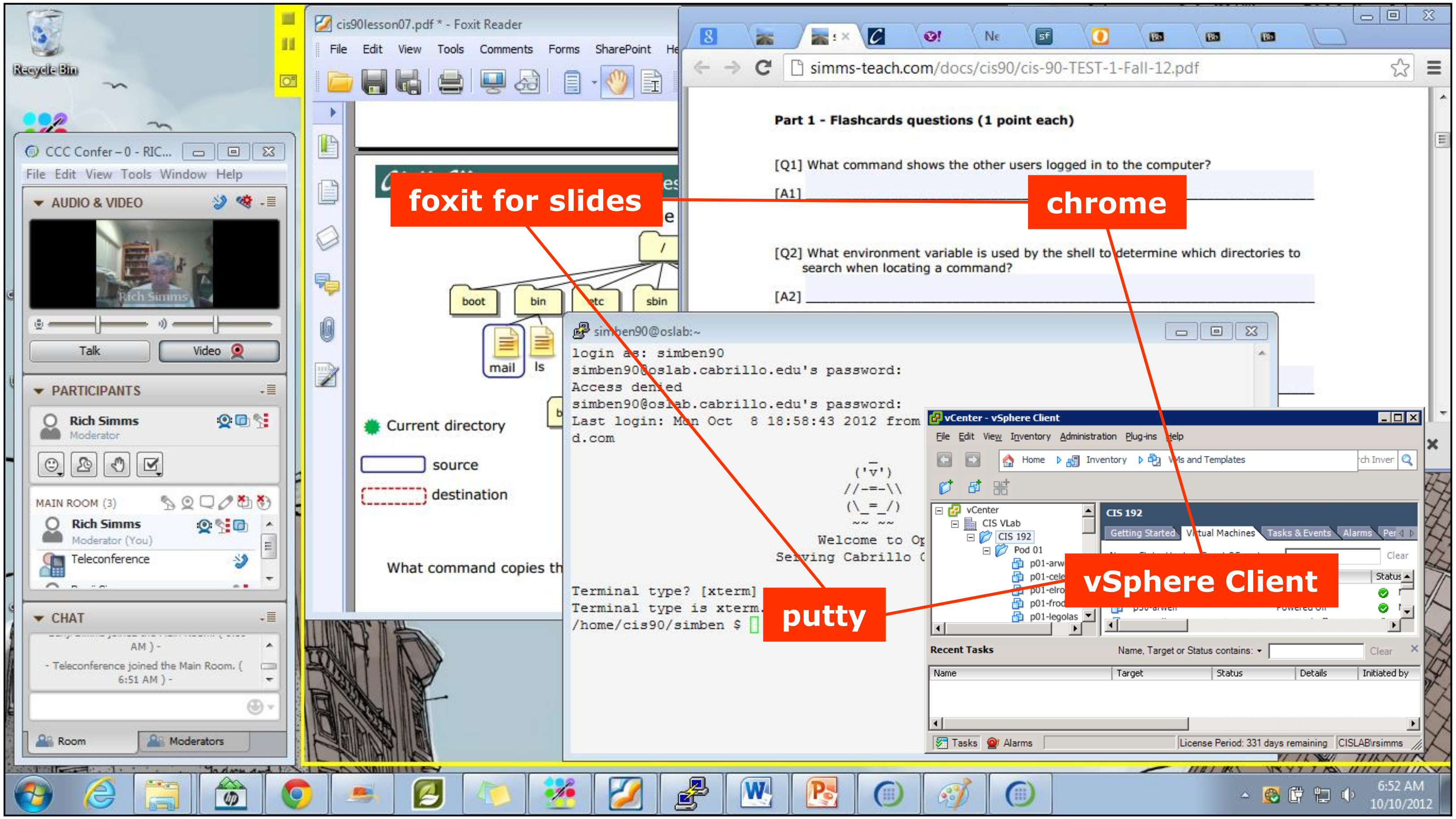


**[ ] Use teleconferencing, not mic**

*Should be greyed out*

[ ] **Video (webcam) optional**

[ ] **layout and share apps**



- Teleconference joined the Main Room. ( 6:51 AM ) -

simms-teach.com/docs/cis90/cis-90-TEST-1-Fall-12.pdf

**Part 1 - Flashcards questions (1 point each)**

[Q1] What command shows the other users logged in to the computer?

[A1] ____________________

[Q2] What environment variable is used by the shell to determine which directories to search when locating a command?

[A2] ____________________

Current directory

source

destination

What command copies th

simben90@oslab:~

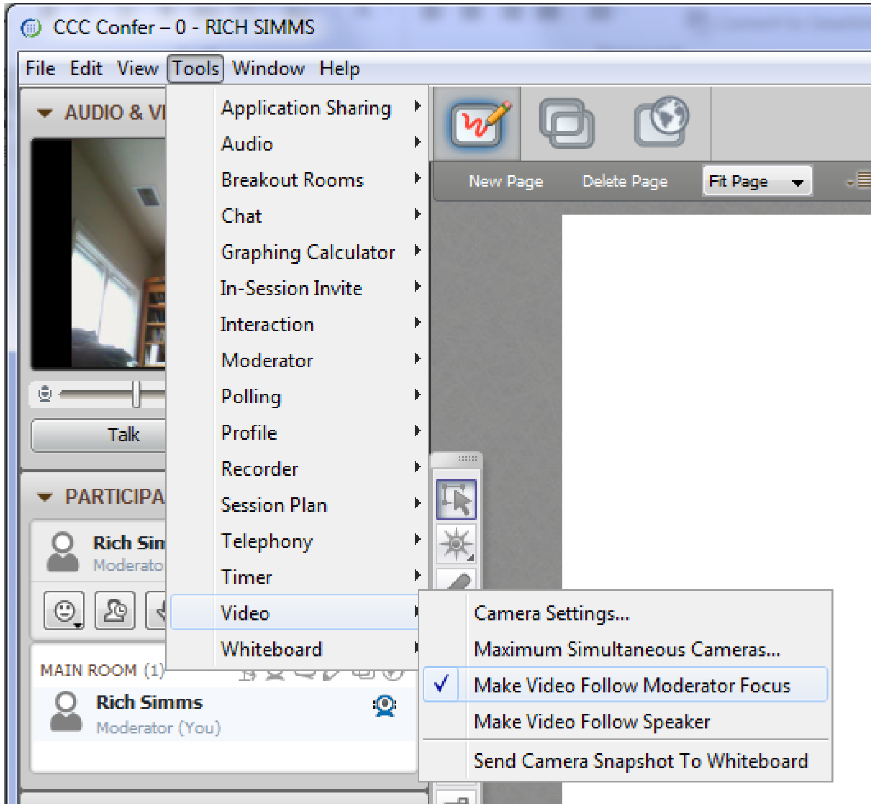
```
login as: simben90
simben90@oslab.cabrillo.edu's password:
Access denied
simben90@oslab.cabrillo.edu's password:
Last login: Mon Oct  8 18:58:43 2012 from
d.com

                                   ('v')
                                  //-=-\\
                                  (\_=_/)
                                   ~~ ~~
                                Welcome to Op
                             Serving Cabrillo C

Terminal type? [xterm]
Terminal type is xterm.
/home/cis90/simben $
```

CCC Confer

[ ] Video (webcam) optional

[ ] Follow moderator

[ ] Double-click on postages stamps



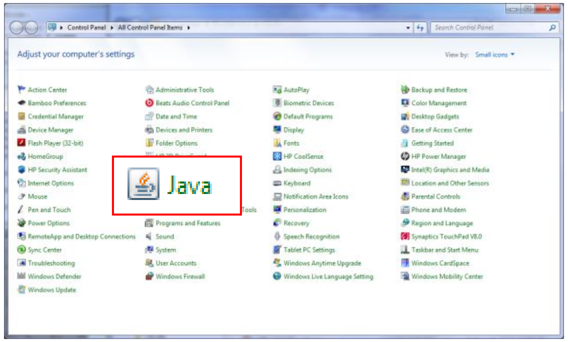
✓ Make Video Follow Moderator Focus

**Universal Fix for CCC Confer:**

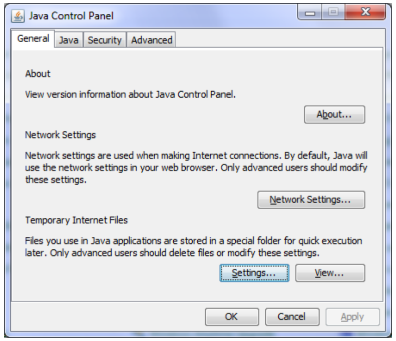
CCC Confer

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

Control Panel (small icons)



General Tab > Settings...



About

View version information about Java Control Panel.

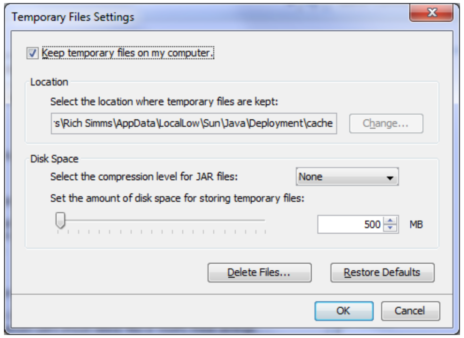
Network Settings

Network settings are used when making Internet connections. By default, Java will use the network settings in your web browser. Only advanced users should modify these settings.

Temporary Internet Files

Files you use in Java applications are stored in a special folder for quick execution later. Only advanced users should delete files or modify these settings.

500MB cache size



☑ Keep temporary files on my computer.

Location

Select the location where temporary files are kept:

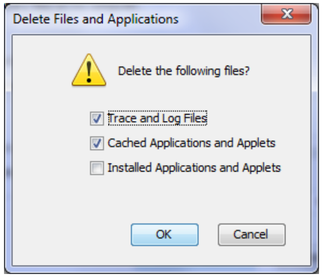
s\Rich Simms\AppData\LocalLow\Sun\Java\Deployment\cache

Disk Space

Select the compression level for JAR files: None

Set the amount of disk space for storing temporary files: 500 MB

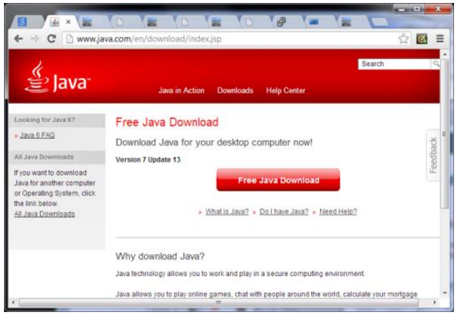
Delete these



Delete the following files?

☑ Trace and Log Files
☑ Cached Applications and Applets
□ Installed Applications and Applets

Google Java download



If you want to download Java for another computer or Operating System, click the link below.

Free Java Download

Download Java for your desktop computer now!

Version 7 Update 13

Why download Java?

Java technology allows you to work and play in a secure computing environment.

Java allows you to play online games, chat with people around the world, calculate your mortgage

# UNIX Processes

| Objectives | Agenda |
|---|---|
| • Know the process life cycle<br>• Interpret ps command output<br>• Run or schedule jobs to run in the background<br>• Send signals to processes<br>• Configure process load balancing | • Questions<br>• Housekeeping<br>• Process definition<br>• Process lifecycle<br>• Process information<br>• Job control<br>• Signals<br>• Load balancing<br>• Wrap up<br>• Test #2 |

# Questions

# Questions

Lesson material?

Labs?

How this course works?

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

| | |
|---|---|
| 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.* |

# umask Review

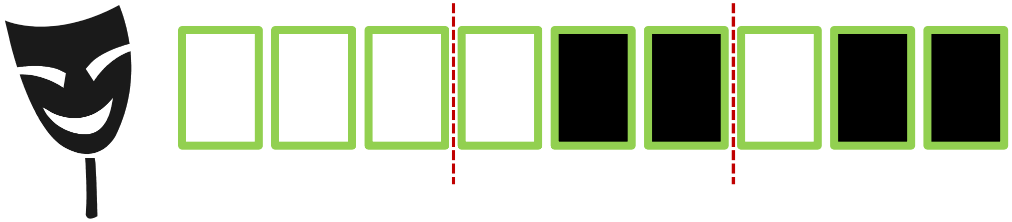
# umask summary

- Use the **umask** command to specify the permissions you want stripped from future new files and directories
- Does not change permissions on existing files

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

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

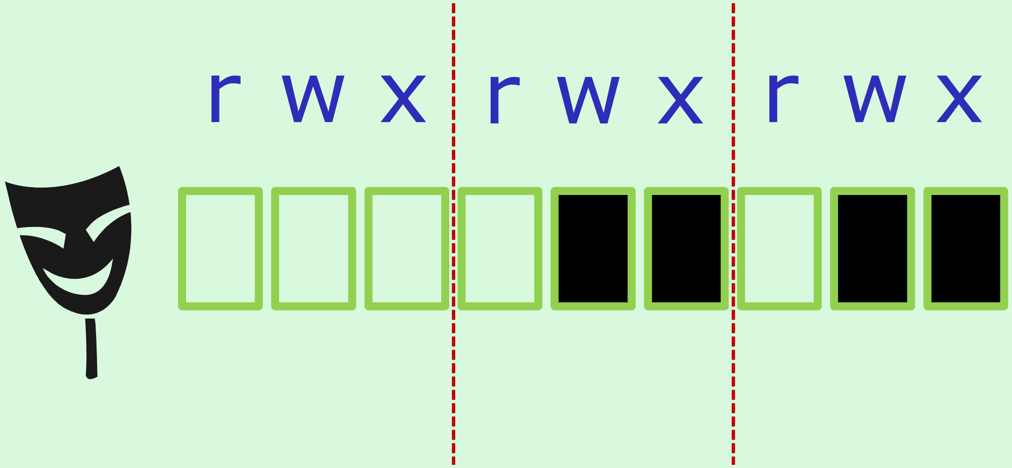
With a umask of 033 what permissions would a newly created directory have?



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

## Example 1 – new directory

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

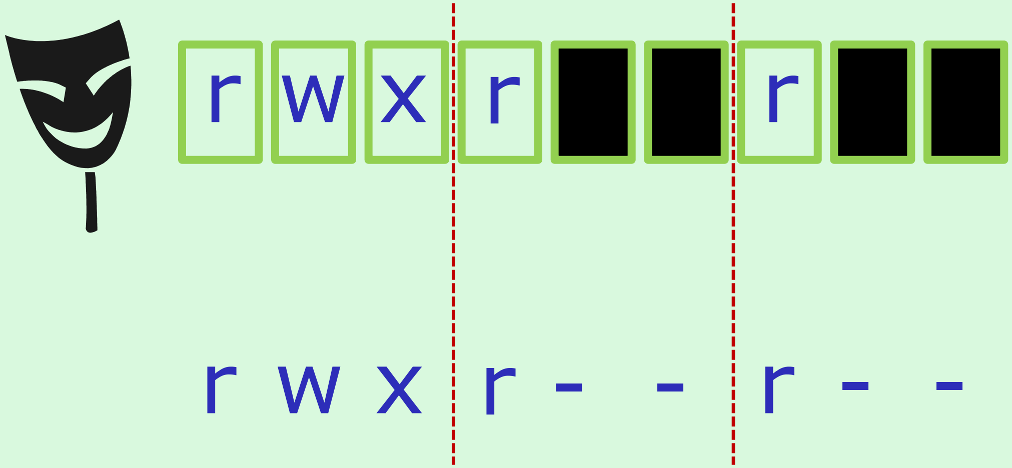


**starting point = 777**
**(new directory)**

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

## Example 1 – new directory

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



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

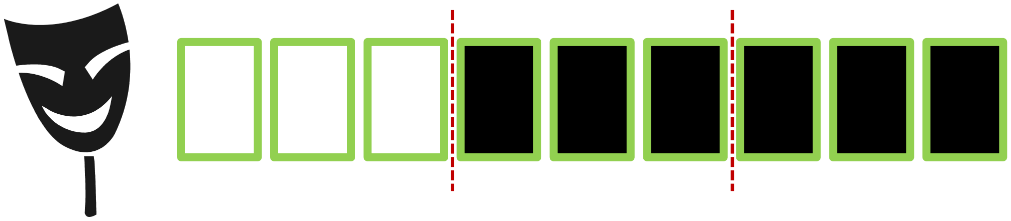
**umask setting of 033 strips these bits: --- -wx -wx**

**Answer: 744**

*Verify your answer on Opus:*

```
/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/
```

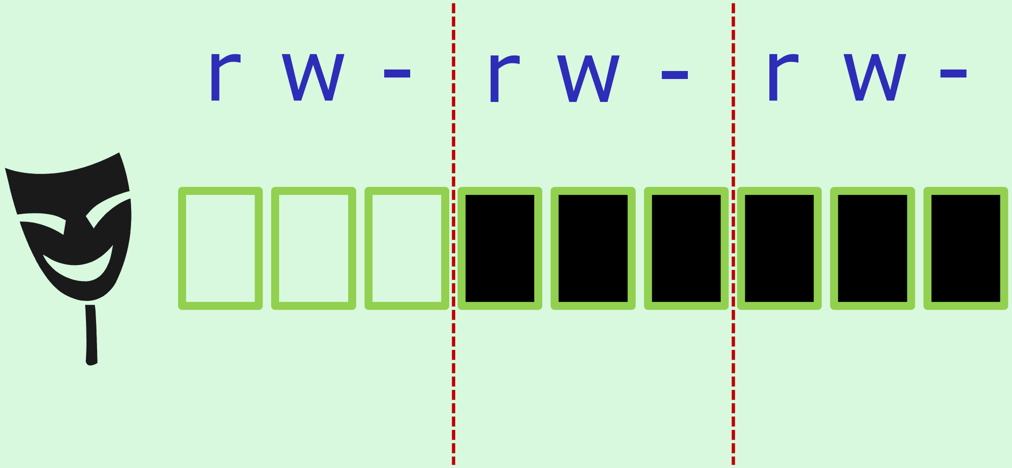
With a umask of 077 what permissions would a newly created file have?



From issuing **umask 077**

# Example 2 – new file

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



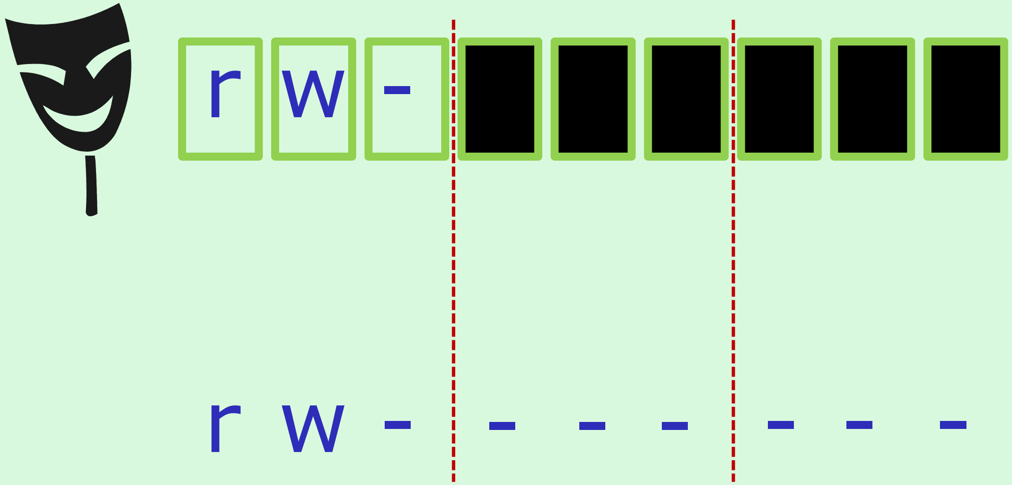
r w - r w - r w -

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

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

## Example 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

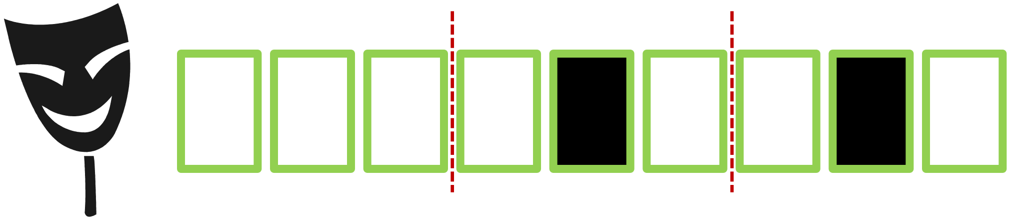
Answer: 600

*Verify your answer on Opus:*

```
/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
```

If umask=022 and *cinderella* file permissions=622

What would the permissions be on the file *cinderella.bak* after:
**cp  cinderella  cinderella.bak**

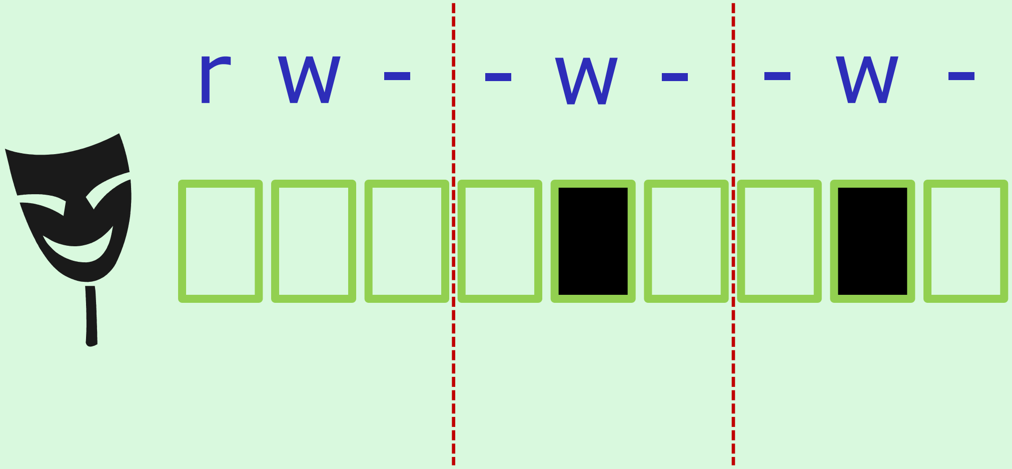


From issuing **umask 022**

## Example 2 – file copy

If umask=022 and the *cinderella* file permissions=622

What would the permissions be on the file *cinderella.bak* after:
**cp cinderella cinderella.bak**



r w - - w - - w -

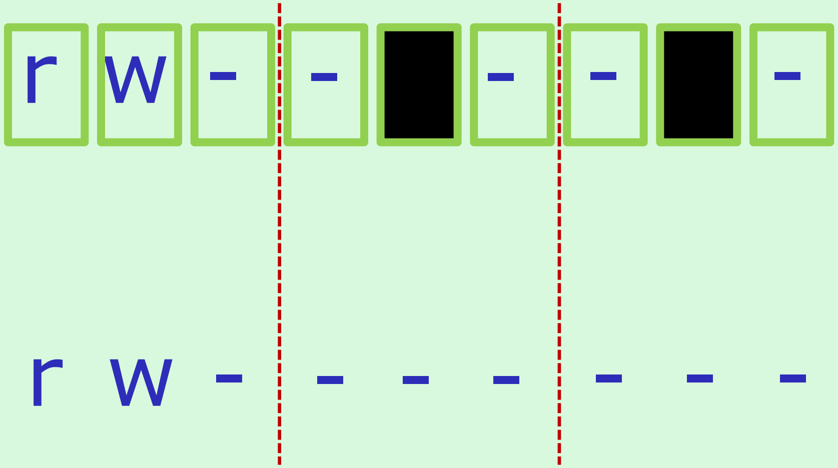
**starting point = 622
(source file permissions)**

**umask setting of 022 strips
these bits: --- -w- -w-**

## Example 2 – file copy

If umask=022 and the *cinderella* file permissions=622

What would the permissions be on the file *cinderella.bak* after:
**cp  cinderella  cinderella.bak**



r w - | - ■ - | - ■ -

r w - | - - - | - - -

**starting point = 622 (source file permissions)**

**umask setting of 022 strips these bits: --- -w- -w-**

**Answer: 600**

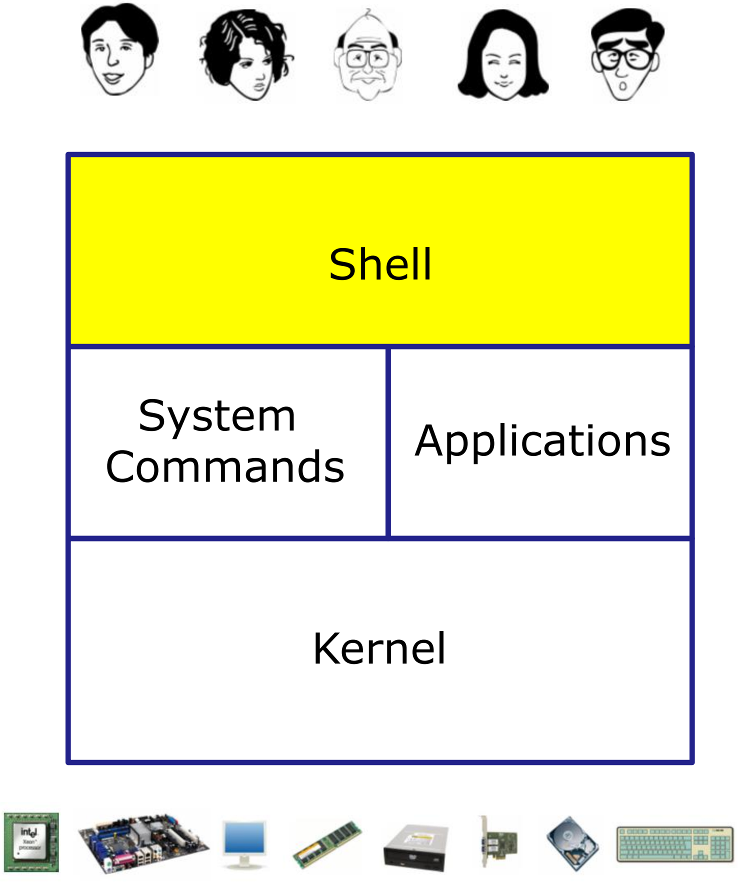
*Verify your answer on Opus:*

```
/home/cis90ol/simmsben $ touch cinderella
/home/cis90ol/simmsben $ chmod 622 cinderella
/home/cis90ol/simmsben $ umask 022
/home/cis90ol/simmsben $ cp cinderella cinderella.bak
/home/cis90ol/simmsben $ ls -l cinderella.bak
-rw------- 1 simmsben cis90ol 0 Apr 21 12:53 cinderella.bak
```

# FYI

## shell debugging and {}

# The Shell **Parse** Step



1) **Prompt** for a command
2) **Parse** (interpret metacharacters, expand file names and dissect command line into options and arguments)
3) **Search** for program (along the path)
4) **Execute** program by loading into memory (becomes a process), hookup input and outputs, and pass along command line options and arguments.
5) **Nap** (wait till process is done)
6) **Repeat**

# Important Concept to Understand

- It’s a **team effort** between the **shell** and the **command** to process what a user types after the prompt
- The shell does the initial work during the **parse step** and provides a list of options and arguments to the command
- The command may not see everything the user actually typed in

# FYI **set –x, set +x**

FYI only

```
/home/cis90/rodduk $ set -x
+ set -x
++ echo -ne '\033]0;rodduk@opus:~'

/home/cis90/rodduk $ type /bin/pi*
+ type /bin/ping /bin/ping6
/bin/ping is /bin/ping
/bin/ping6 is /bin/ping6
++ echo -ne '\033]0;rodduk@opus:~'

/home/cis90/rodduk $ type -af /usr/bin/p[ek]*[ct] 2> /dev/null
+ type -af /usr/bin/perlcc /usr/bin/perldoc /usr/bin/pkcs11_inspect
/usr/bin/perlcc is /usr/bin/perlcc
/usr/bin/perldoc is /usr/bin/perldoc
/usr/bin/pkcs11_inspect is /usr/bin/pkcs11_inspect
++ echo -ne '\033]0;rodduk@opus:~'

/home/cis90/rodduk $ set +x
+ set +x
/home/cis90/rodduk $
```

*Enable shell debugging*

*Shows what arguments are actually passed to the command being run*

*Disable shell debugging*

# FYI **set –x, set +x**

FYI only

```
/home/cis90/rodduk $ set -x
+ set -x
++ echo -ne '\033]0;rodduk@opus:~'
```

*Enable shell debugging*

```
/home/cis90/rodduk $ find . -name '$LOGNAME'
+ find . -name '$LOGNAME'
find: ./Hidden: Permission denied
find: ./testdir: Permission denied
++ echo -ne '\033]0;rodduk@opus:~'

/home/cis90/rodduk $ find . -name "$LOGNAME"
+ find . -name rodduk
find: ./Hidden: Permission denied
./rodduk
find: ./testdir: Permission denied
++ echo -ne '\033]0;rodduk@opus:~'
```

*Shows variables in double (weak) quotes get expanded, while those in single (strong) quotes do not*

```
/home/cis90/rodduk $ set +x
+ set +x
/home/cis90/rodduk $
```

*Disable shell debugging*

# FYI **set –x, set +x**

FYI only

```
/home/cis90/milhom $ set -x
++ printf '\033]0;%s@%s:%s\007' milhom90 oslab '~'

/home/cis90/milhom $ find . -name *treat*
+ find . -name treat1
find: `./Hidden': Permission denied
./treat1
++ printf '\033]0;%s@%s:%s\007' milhom90 oslab '~'

/home/cis90/milhom $ find . -name *trick*
+ find . -name '*trick*'
find: `./Hidden': Permission denied
./Miscellaneous/.trick6
./Poems/Shakespeare/.trick3
./Poems/Yeats/.trick2
./Poems/.trick5
./Poems/Blake/.trick4
./.ssh/.trick1
++ printf '\033]0;%s@%s:%s\007' milhom90 oslab '~'

/home/cis90/milhom $ set +x
+ set +x
/home/cis90/milhom $
```

*Enable shell debugging*

*Shows how filename expansion metacharacters are expanded or not depending on whether a match was found!*

*Disable shell debugging*

# FYI using {}

FYI only

*The braces {} are filename expansion metacharacters*

```
/home/cis90/simben $ mkdir fast
/home/cis90/simben $ ls fast
/home/cis90/simben $ touch fast/file{1,2,3,4,5}
/home/cis90/simben $ ls fast
file1  file2  file3  file4  file5
```

*Short hand for specifying multiple filenames at once*

```
/home/cis90/simben $ set -x
++ echo -ne '\033]0;simben90@opus:~'

/home/cis90/simben $ touch fast/file{1,2,3,4,5}
+ touch fast/file1 fast/file2 fast/file3 fast/file4 fast/file5
++ echo -ne '\033]0;simben90@opus:~'
```

*Showing how bash did the expansion above*

# Housekeeping

# Housekeeping

1. Lab 8 is due next week

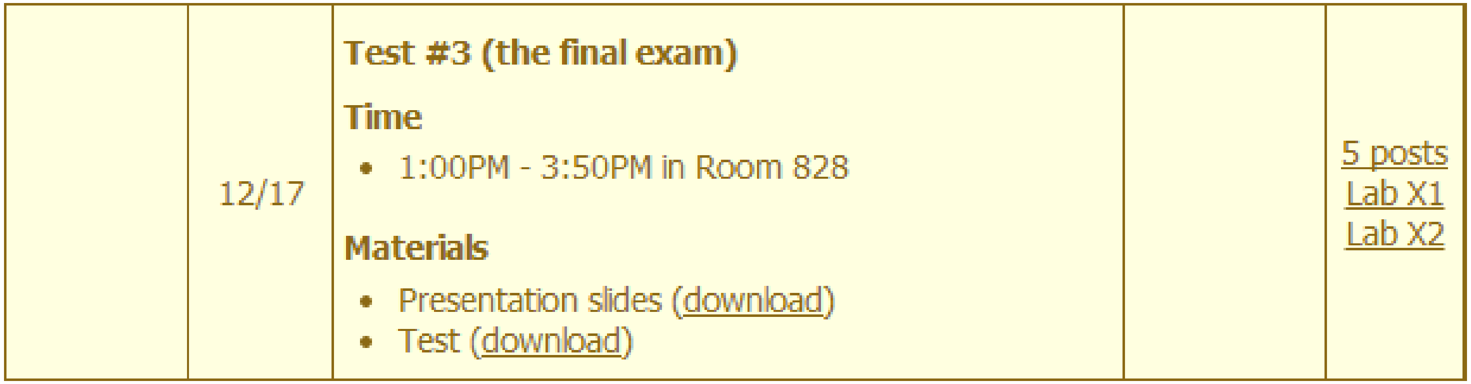
2. Don't wait till the last minute to submit Lab 8 because if you make a mistake with the **at** command it will be too late to fix it!

3. Test 2 during the last part of class today
   - Open book, notes, computer
   - Closed mouths (don't ask for or give assistance to others)
   - Email it the filled-in PDF file to me at the end of class
   - If you would like more time you can email me your final version no later than 11:59AM tonight.

# Final Exam

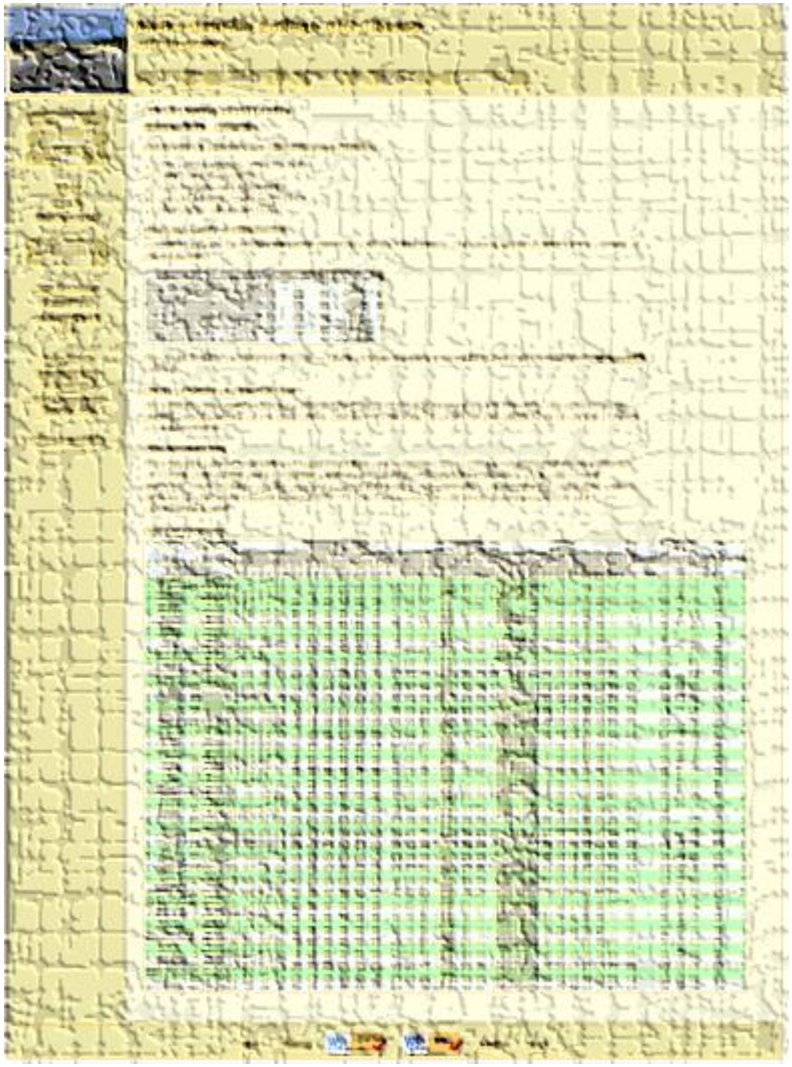
Test #3 (final exam)

- Must be face-to-face or proctored (not online using CCC Confer).
- We will be in room 828 on campus.



| | | | | |
|---|---|---|---|---|
| | 12/17 | **Test #3 (the final exam)**<br>**Time**<br>• 1:00PM - 3:50PM in Room 828<br>**Materials**<br>• Presentation slides (download)<br>• Test (download) | | 5 posts<br>Lab X1<br>Lab X2 |

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 11/4/2013

| **Points that could have been earned:** | |
|---|---|
| 7 quizzes: | 21 points |
| 7 labs: | 210 points |
| 1 test: | 30 points |
| 2 forum quarters: | 40 points |
| **Total:** | **301 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 |

adaldrida: 98% (295 of 301 points)
anborn: 0% (0 of 301 points)
aragorn: 97% (292 of 301 points)
arwen: 85% (257 of 301 points)
balrog: 49% (150 of 301 points)
barliman: 1% (4 of 301 points)
beregond: 73% (221 of 301 points)
boromir: 2% (9 of 301 points)
celebrian: 83% (251 of 301 points)
dori: 48% (146 of 301 points)
dwalin: 87% (262 of 301 points)
elrond: 95% (288 of 301 points)
eomer: 82% (249 of 301 points)
faramir: 100% (301 of 301 points)
frodo: 96% (289 of 301 points)
gimli: 94% (284 of 301 points)
goldberry: 105% (317 of 301 points)

huan: 40% (122 of 301 points)
ingold: 98% (297 of 301 points)
ioreth: 71% (215 of 301 points)
legolas: 76% (229 of 301 points)
marhari: 100% (304 of 301 points)
pallando: 103% (311 of 301 points)
pippen: 94% (283 of 301 points)
quickbeam: 34% (105 of 301 points)
samwise: 83% (251 of 301 points)
sauron: 101% (306 of 301 points)
shadowfax: 73% (220 of 301 points)
strider: 87% (264 of 301 points)
theoden: 101% (305 of 301 points)
treebeard: 91% (274 of 301 points)
tulkas: 97% (294 of 301 points)
ulmo: 64% (194 of 301 points)

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

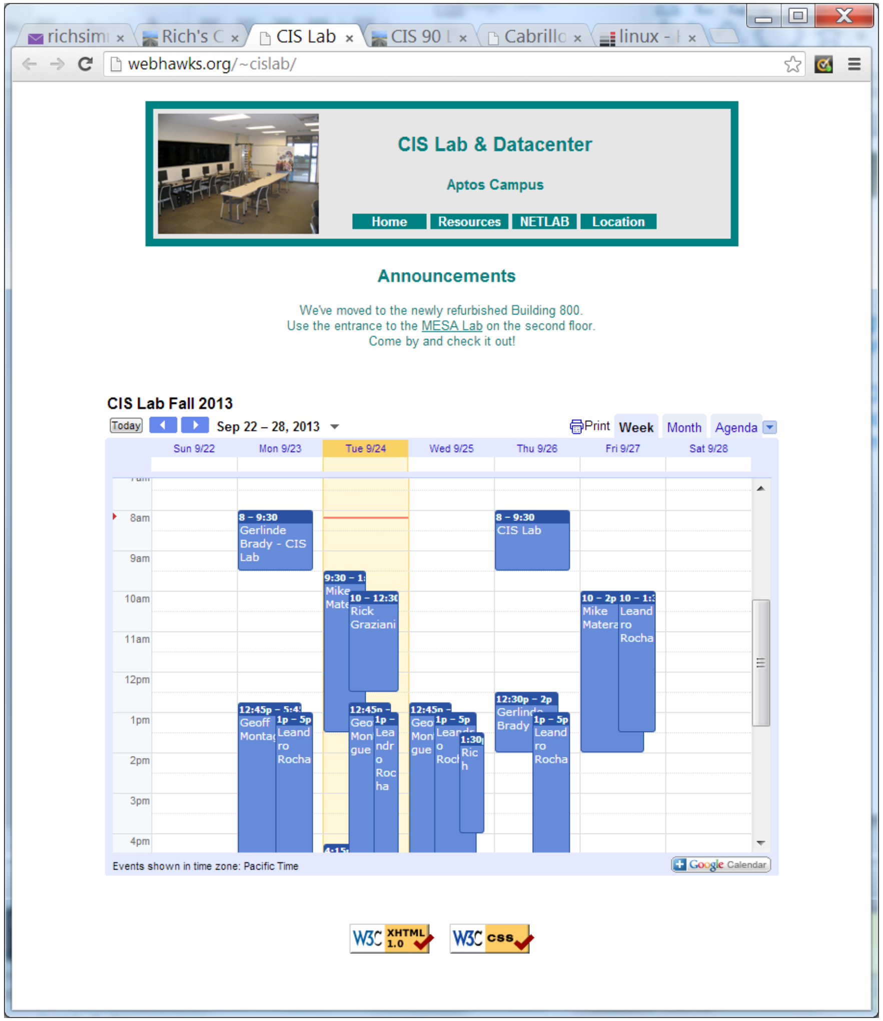
# Jesse's checkgrades python script

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.*



CIS Lab & Datacenter

Aptos Campus

Announcements

We've moved to the newly refurbished Building 800.
Use the entrance to the MESA Lab on the second floor.
Come by and check it out!

CIS Lab Fall 2013

Sep 22 – 28, 2013

Sun 9/22 | Mon 9/23 | Tue 9/24 | Wed 9/25 | Thu 9/26 | Fri 9/27 | Sat 9/28

8 – 9:30 Gerlinde Brady - CIS Lab; 9:30 – 1 Mike Matera; 10 – 12:30 Rick Graziani; 8 – 9:30 CIS Lab; 10 – 2p Mike Matera; 10 – 1: Leandro Rocha; 12:45p – 5:4 Geoff Montague; 1p – 5p Leandro Rocha; 12:45p – Geoff Montague; 1p – Leandro Rocha; 12:30p – 2p Gerlinde Brady; 1:30 Rich

Events shown in time zone: Pacific Time

Google Calendar

W3C XHTML 1.0 W3C CSS

CIS Lab Schedule
http://webhawks.org/~cislab/

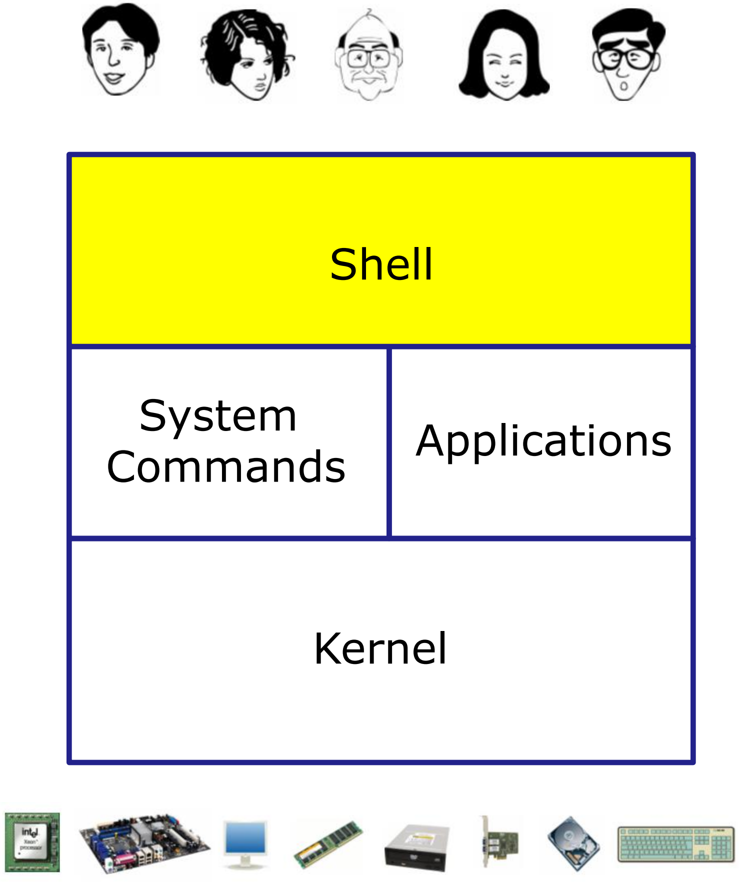
*Work on assignments together with other classmates*

*Get help from instructors and student lab assistants*

*MESA grants requires logging help sessions with MESA funded student assistants*

# Process Definition

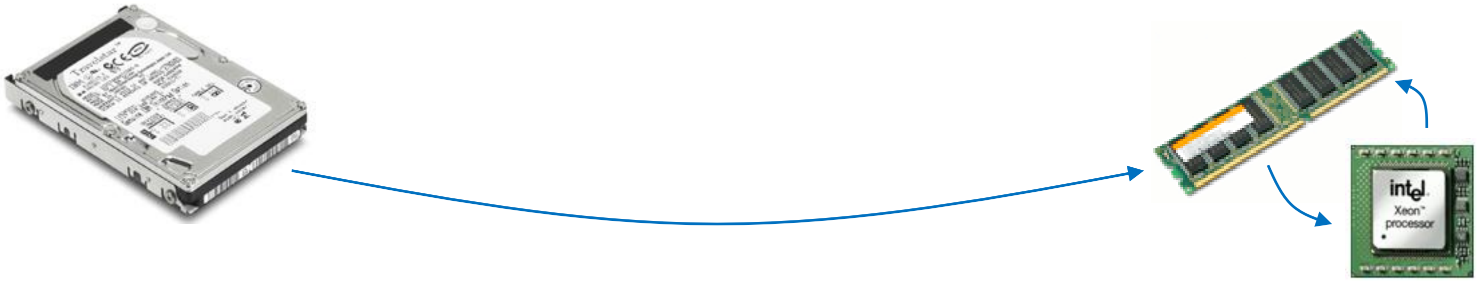
# The Shell **Execute** Step



1) **Prompt** for a command
2) **Parse** (interpret metacharacters, expand file names and dissect command line into options and arguments)
3) **Search** for program (along the path)
4) **Execute** program by loading it into memory (as a process) and providing it with the parsed options/arguments. In addition hook up all inputs and outputs (stdin, stdout and stderr)
5) **Nap** (wait till process is done)
6) **Repeat**

# Definition of a process

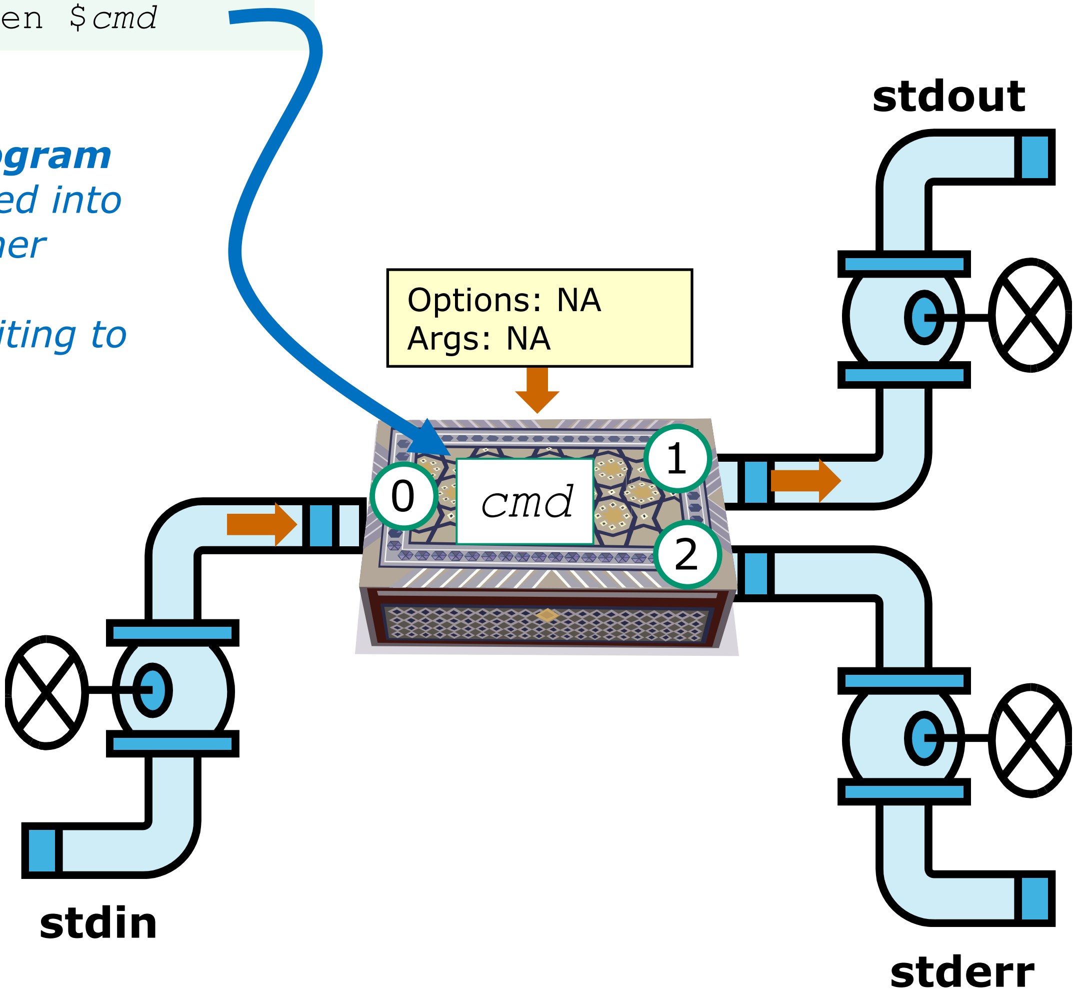
*A* ***process*** *is a* ***program*** *that has been copied (loaded) into memory by the kernel and is either running (executing instructions) or waiting to run.*



# Program to process

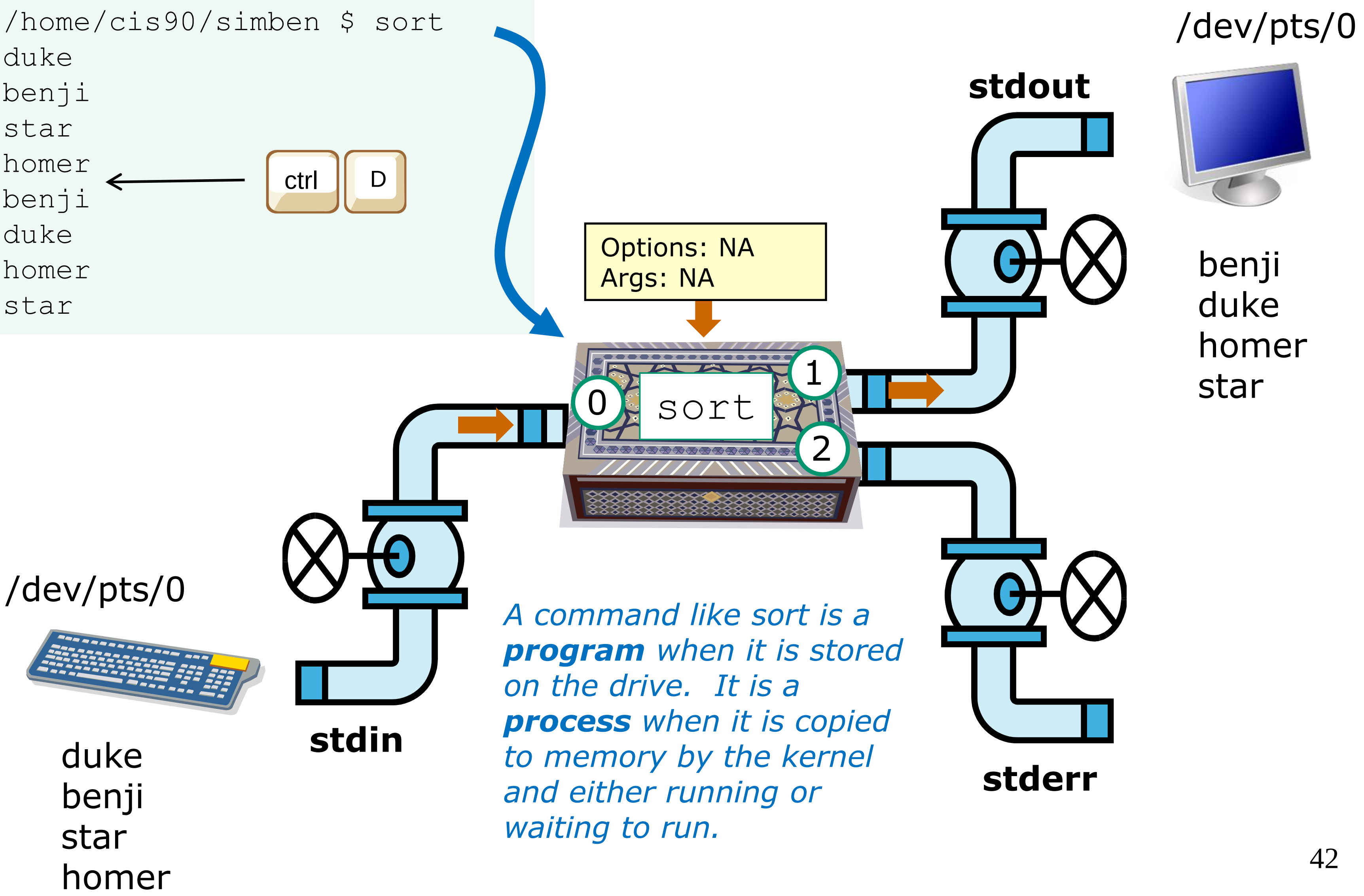
```
/home/cis90/simben $cmd
```

*A **process** is a **program** that has been loaded into memory and is either running (executing instructions) or waiting to run*

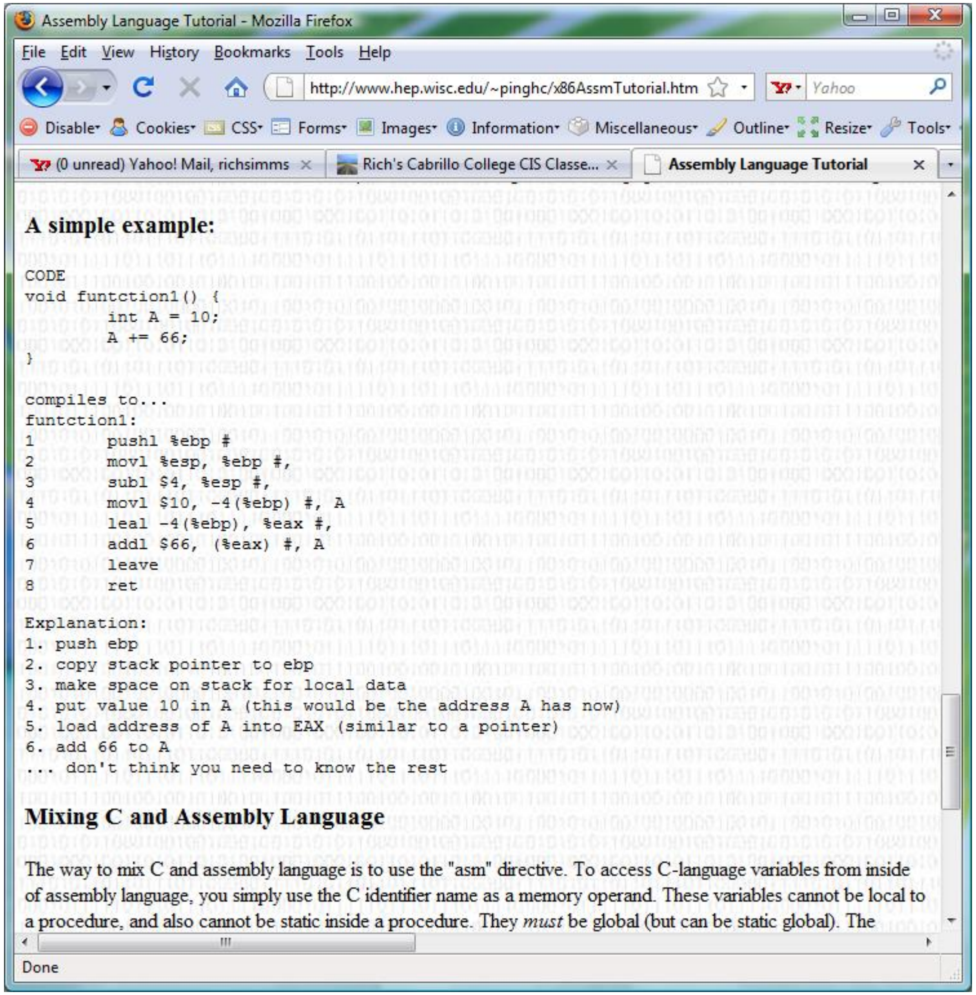


# Example program to process: sort command

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



*A command like sort is a* ***program*** *when it is stored on the drive. It is a* ***process*** *when it is copied to memory by the kernel and either running or waiting to run.*


Assembly Language Tutorial - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.hep.wisc.edu/~pinghc/x86AssmTutorial.htm
Yahoo
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools
(0 unread) Yahoo! Mail, richsimms
Rich's Cabrillo College CIS Classe...
Assembly Language Tutorial
Done

## A simple example:

```
CODE
void funtction1() {
        int A = 10;
        A += 66;
}

compiles to...
funtction1:
1       pushl %ebp #
2       movl %esp, %ebp #,
3       subl $4, %esp #,
4       movl $10, -4(%ebp) #, A
5       leal -4(%ebp), %eax #,
6       addl $66, (%eax) #, A
7       leave
8       ret

Explanation:
1. push ebp
2. copy stack pointer to ebp
3. make space on stack for local data
4. put value 10 in A (this would be the address A has now)
5. load address of A into EAX (similar to a pointer)
6. add 66 to A
... don't think you need to know the rest
```

## Mixing C and Assembly Language

The way to mix C and assembly language is to use the "asm" directive. To access C-language variables from inside of assembly language, you simply use the C identifier name as a memory operand. These variables cannot be local to a procedure, and also cannot be static inside a procedure. They *must* be global (but can be static global). The

*Most programs are written in the C language*

*The  C compiler translates the C code into binary machine code instructions the CPU can execute.*

http://www.hep.wisc.edu/~pinghc/x86AssmTutorial.htm

# Example program to process: sort command

```
[rsimms@opus ~]$ type sort
sort is /bin/sort
```

*Use **type** to find where the sort program is located*

*Use **file** to see sort is a binary executable*

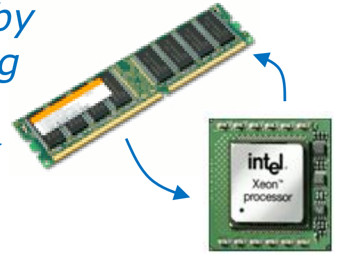
```
[rsimms@opus ~]$ file /bin/sort
/bin/sort: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux
2.6.9, dynamically linked (uses shared libs), for GNU/Linux 2.6.9, stripped
[rsimms@opus ~]$
```

```
[rsimms@opus ~]$ xxd /bin/sort | more

0000000: 7f45 4c46 0101 0100 0000 0000 0000 0000  .ELF............
0000010: 0200 0300 0100 0000 e093 0408 3400 0000  ............4...
0000020: 2cdb 0000 0000 0000 3400 2000 0800 2800  ,.......4. ...(.
0000030: 1f00 1e00 0600 0000 3400 0000 3480 0408  ........4...4...
0000040: 3480 0408 0001 0000 0001 0000 0500 0000  4...............
0000050: 0400 0000 0300 0000 3401 0000 3481 0408  ........4...4...
0000060: 3481 0408 1300 0000 1300 0000 0400 0000  4...............
```

< *snipped* >

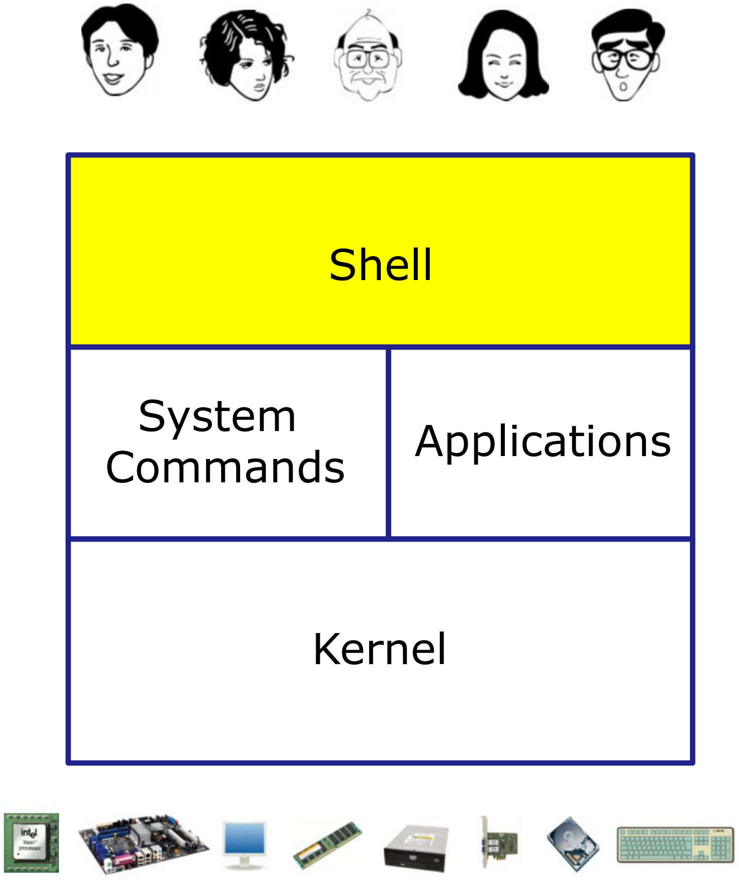
*Use **xxd** to produce a hexadecimal dump of the sort file*

*A command like **sort** is a **program** when it is stored on the drive. It is a **process** when it is copied to memory by the kernel and either running or waiting to run by the CPU*



# Process Life Cycle

# The Shell **Execute** Step

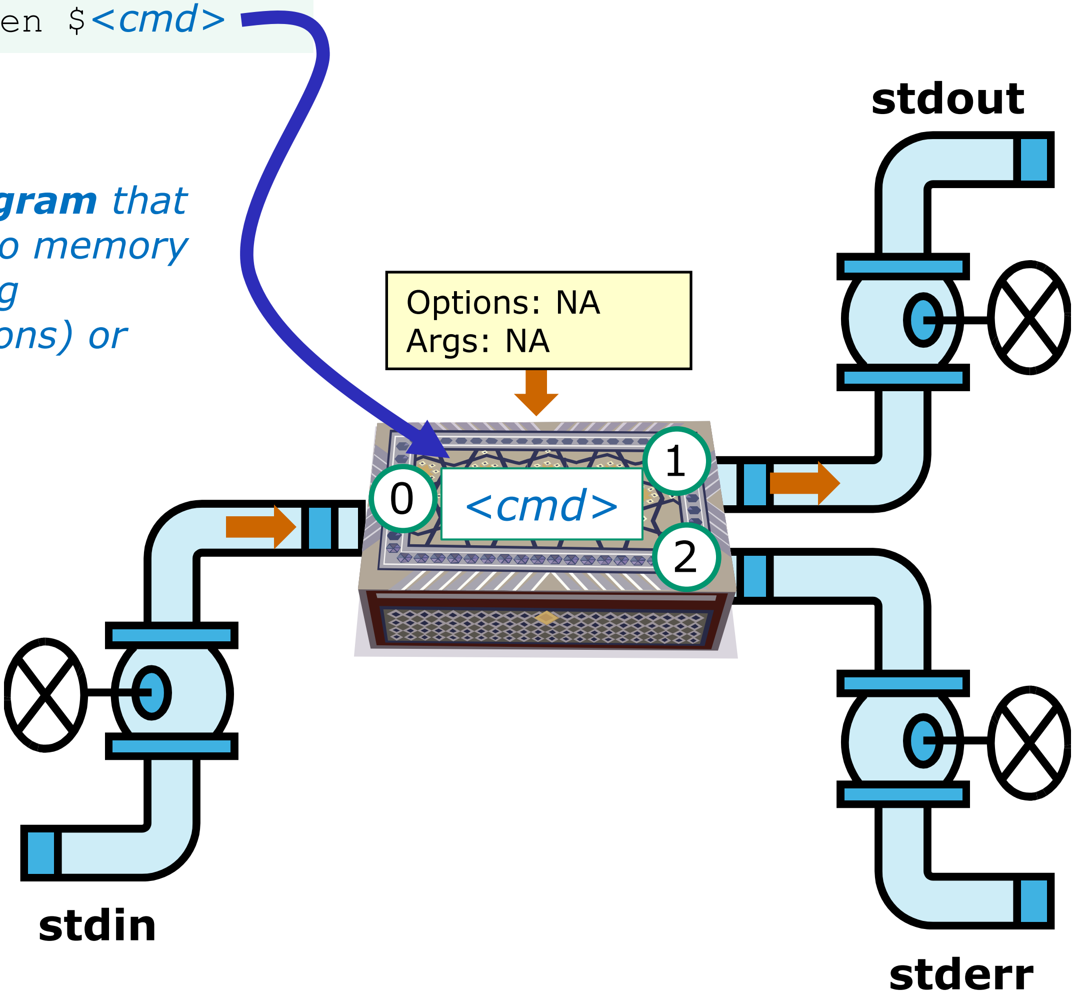


1) **Prompt** for a command
2) **Parse** (interpret metacharacters, expand file names and dissect command line into options and arguments)
3) **Search** for program (along the path)
4) **Execute** program by loading it into memory (as a process) and providing it with the parsed options/arguments. In addition hook up all inputs and outputs (stdin, stdout and stderr)
5) **Nap** (wait till process is done)
6) **Repeat**

# Executing a command <*cmd*>

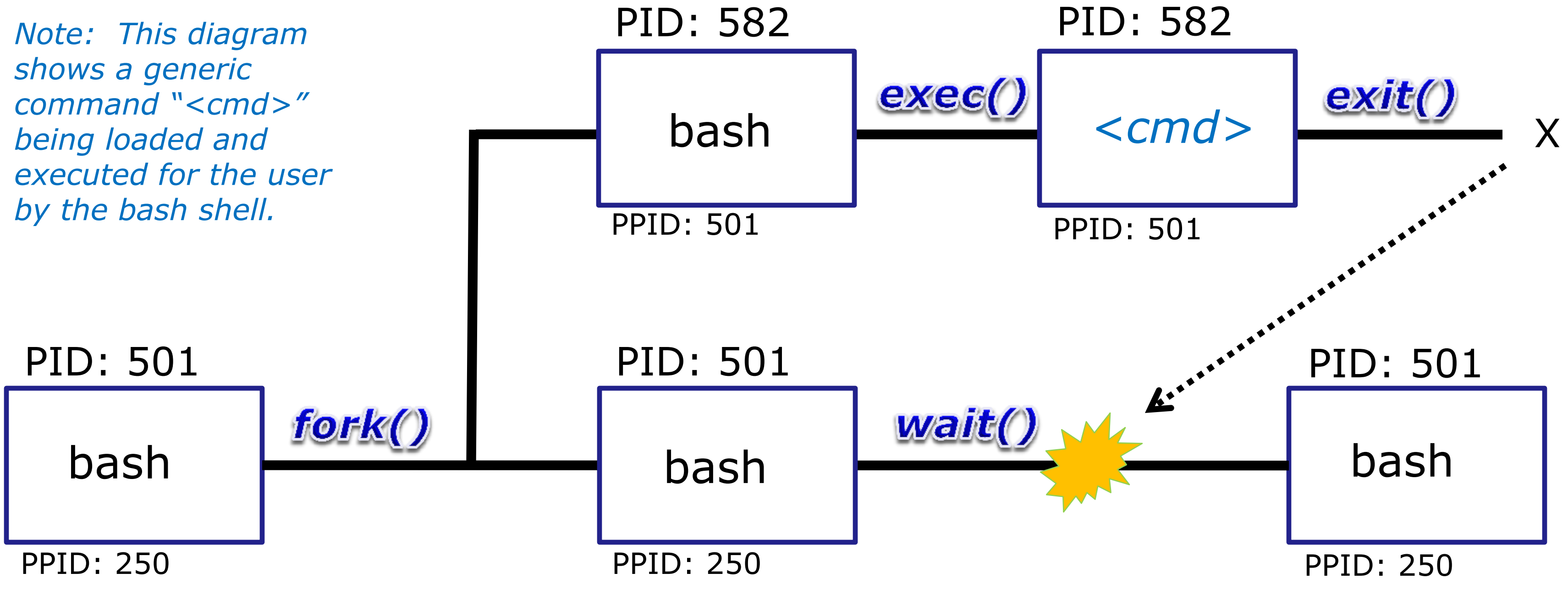
```
/home/cis90/simben $<cmd>
```

*A **process** is a **program** that has been loaded into memory and is either running (executing instructions) or waiting to run*



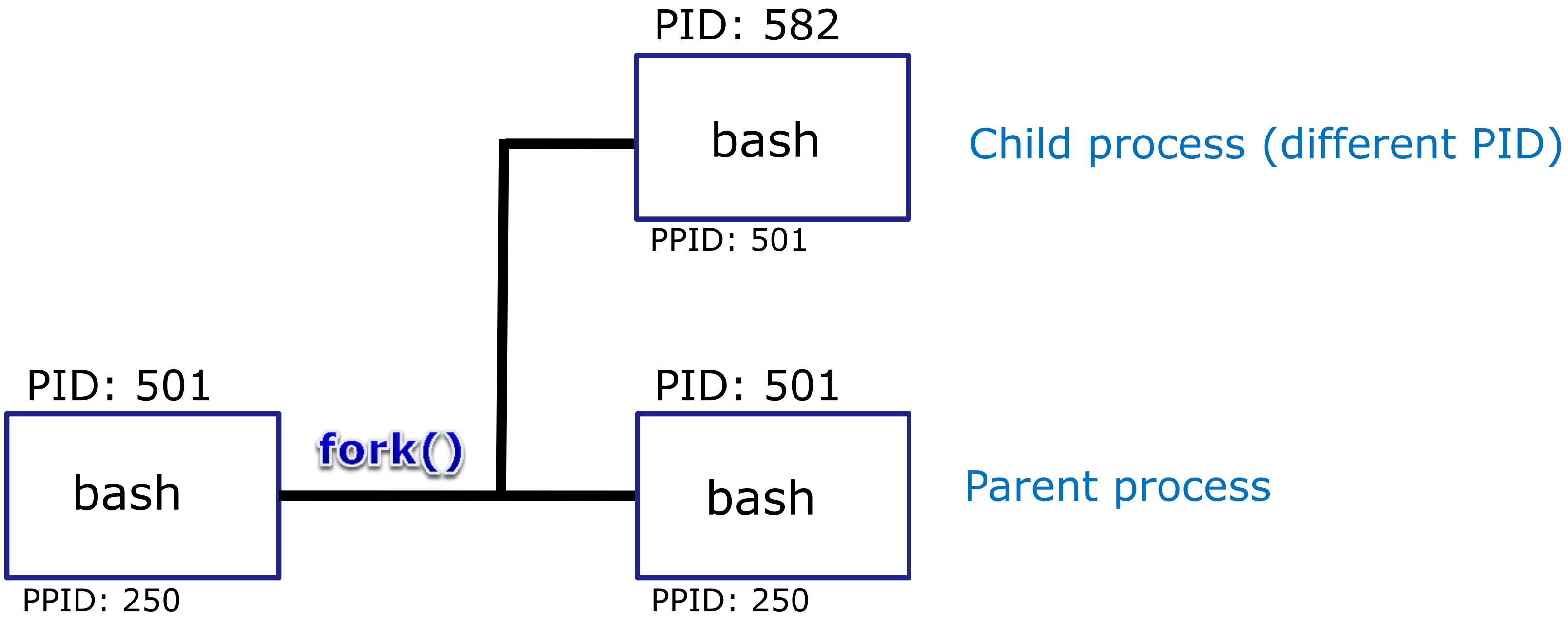
# Process Lifecycle

*Note: This diagram shows a generic command "<cmd>" being loaded and executed for the user by the bash shell.*



*A process uses system calls (e.g.* ***fork*****,** ***exec*****,** ***wait*****,** ***exit****)*
*to request services from the kernel*

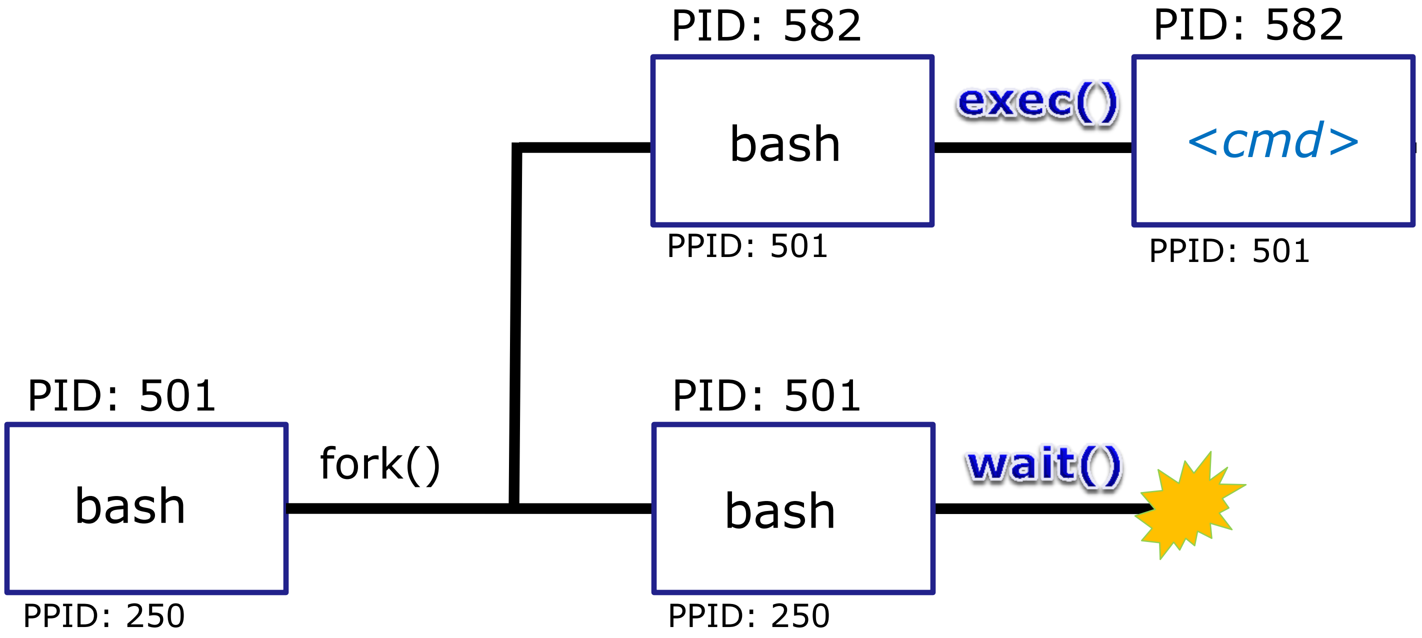
# Process Lifecycle – fork child process



1) The first step in executing a command is to create a new child process

- This is done by the **parent** process (bash) making a copy of itself using the ***fork*** system call.
- The new **child** process is a duplicate of the **parent** but it has a different PID.

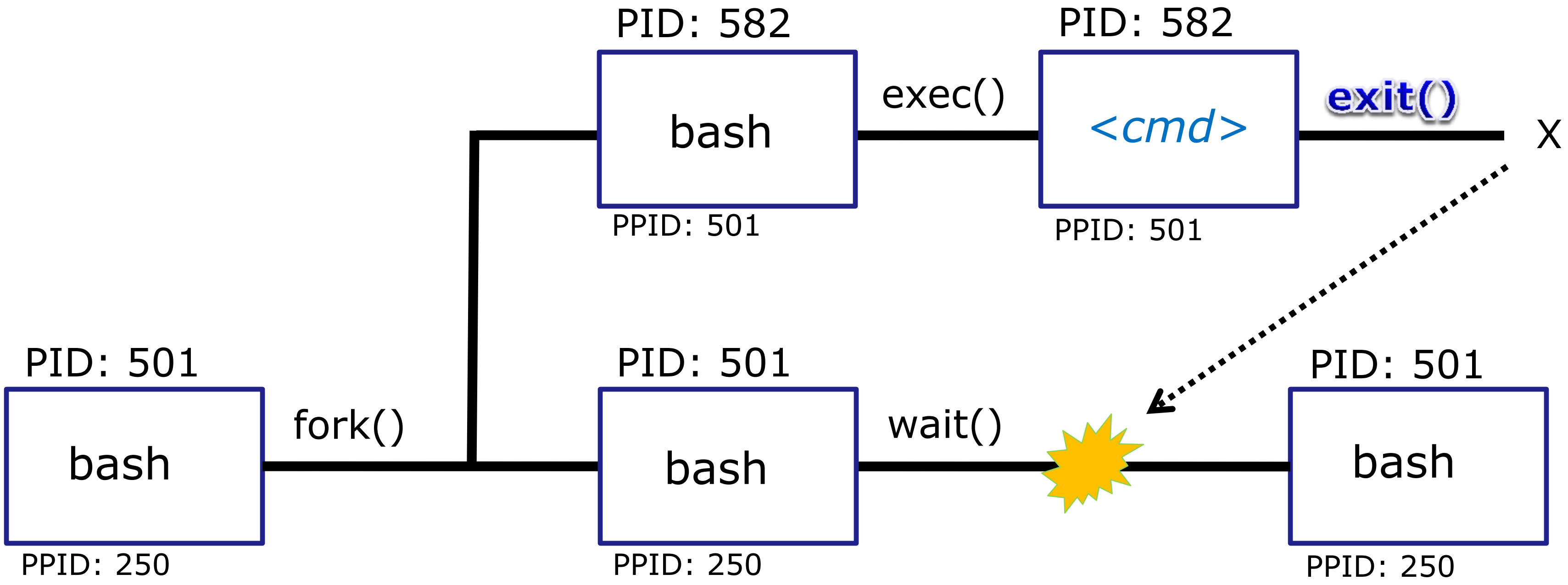
# Process Lifecycle



2) The next step is to load the command into the new child process

- An **exec** system call is issued to overlay the **child** process with the instructions of the requested command. The new instructions then are executed.
- The **parent** process issues the **wait** system call and goes to sleep.

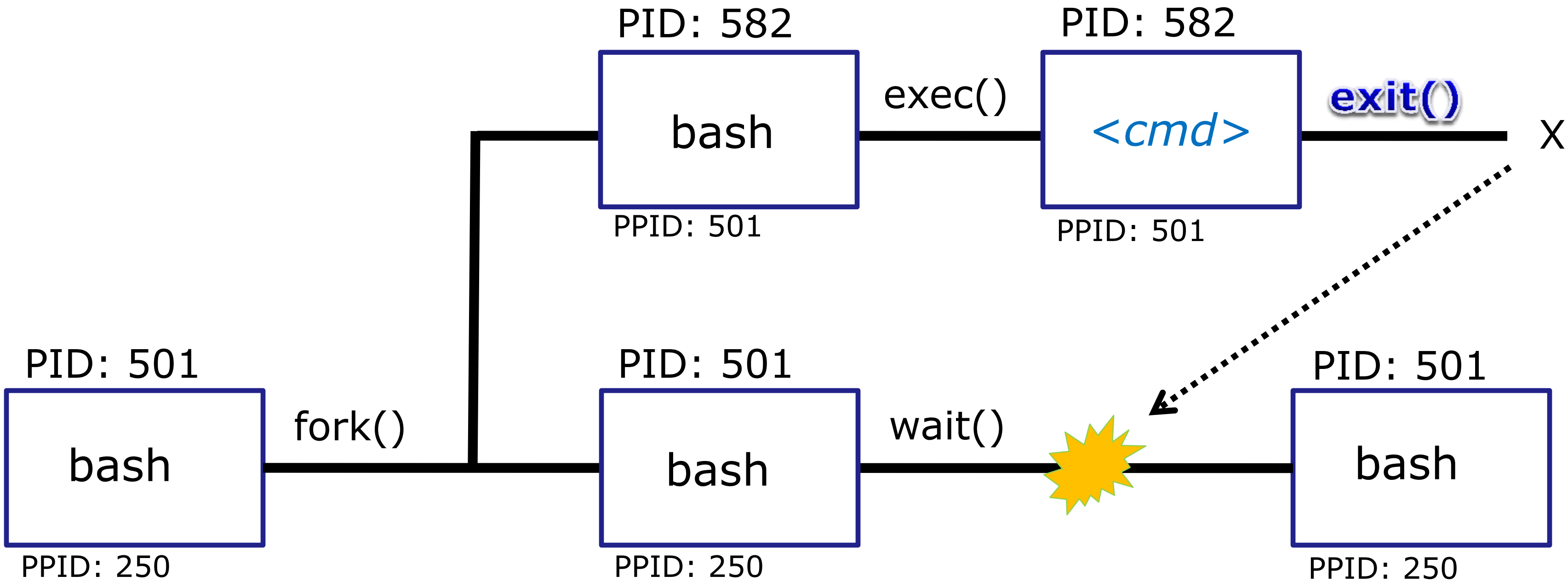
# Process Lifecycle



3) The final step is to terminate the new child process after it has finished

- When the **child** process finishes executing the instructions it issues the **exit** system call. At this point it gives up all its resources and becomes a **zombie**.
- The **parent** is woken up. Once the **parent** has informed the kernel it has finished working with the **child**, the **child** process is killed and removed from the process table.

# Process Lifecycle



*Note: If the **parent** process were to die before the **child**, the zombie will become an **orphan**.*

*Fortunately the init process will adopt any orphaned **zombies!***

# Process Information

# ps command

# Process Information

| Information | Description |
|---|---|
| PID | Process Identification Number, a unique number identifying the process |
| PPID | Parent PID, the PID of the parent process (like … in the file hierarchy) |
| UID | The user running the process |
| TTY | The terminal that the process's stdin and stdout are connected to |
| S | The status of the process: S=Sleeping, R=Running, T=Stopped, Z=Zombie |
| PRI | Process priority |
| SZ | Process size |
| CMD | The name of the process (the command being run) |
| C | The CPU utilization of the process |
| WCHAN | Waiting channel  (name of kernel function in which the process is sleeping) |
| F | Flags (1=forked but didn't exit, 4=used superuser privileges) |
| TIME | Cumulative CPU time |
| NI | Nice value |

*Just a few of the types of information kept on a process.*

*Use **man ps** to see a lot more.*

# **ps** command

```
[rsimms@opus ~]$ ps
PID TTY          TIME CMD
 6204 pts/6    00:00:00 bash
 6285 pts/6    00:00:00 ps
[rsimms@opus ~]$
```

*Show just my processes. Note **bash** was started for me when I logged into my terminal session. **ps** is showing because it is running the instant this output is printed.*

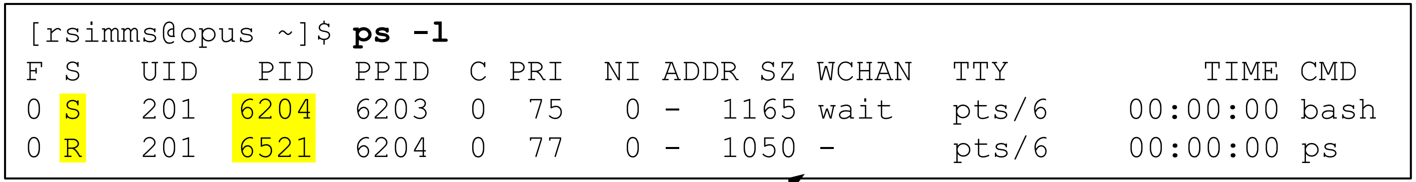
# **ps** command with **-u** option

```
[rsimms@opus ~]$ cat /etc/passwd | grep Marcos
valdemar:x:1200:103:Marcos Valdebenito:/home/cis90/valdemar:/bin/bash


[rsimms@opus ~]$ ps -u 1200
  PID TTY          TIME CMD
 5971 ?        00:00:00 sshd
 5972 pts/5    00:00:00 bash


[rsimms@opus ~]$ ps -u dymesdia
PID TTY          TIME CMD
 6418 ?        00:00:00 sshd
 6419 pts/1    00:00:00 bash


[rsimms@opus ~]$ ps -u rsimms
  PID TTY          TIME CMD
 5368 ?        00:00:00 sshd
 5369 pts/0    00:00:00 bash
 6173 pts/0    00:00:00 man
 6176 pts/0    00:00:00 sh
 6177 pts/0    00:00:00 sh
 6182 pts/0    00:00:00 less
 6203 ?        00:00:00 sshd
 6204 pts/6    00:00:00 bash
 6510 pts/6    00:00:00 ps
```

*Use the **-u** (user) option to look at processes owned by a specific user*

# **ps** command with **-l** option

*Use **–l** (long format) to show additional process information*



```
[rsimms@opus ~]$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   201  6204  6203  0  75   0 -  1165 wait   pts/6    00:00:00 bash
0 R   201  6521  6204  0  77   0 -  1050 -      pts/6    00:00:00 ps
```

*6204 is sleeping*
*6521 is running*

*Running or sleeping*

*User ID*

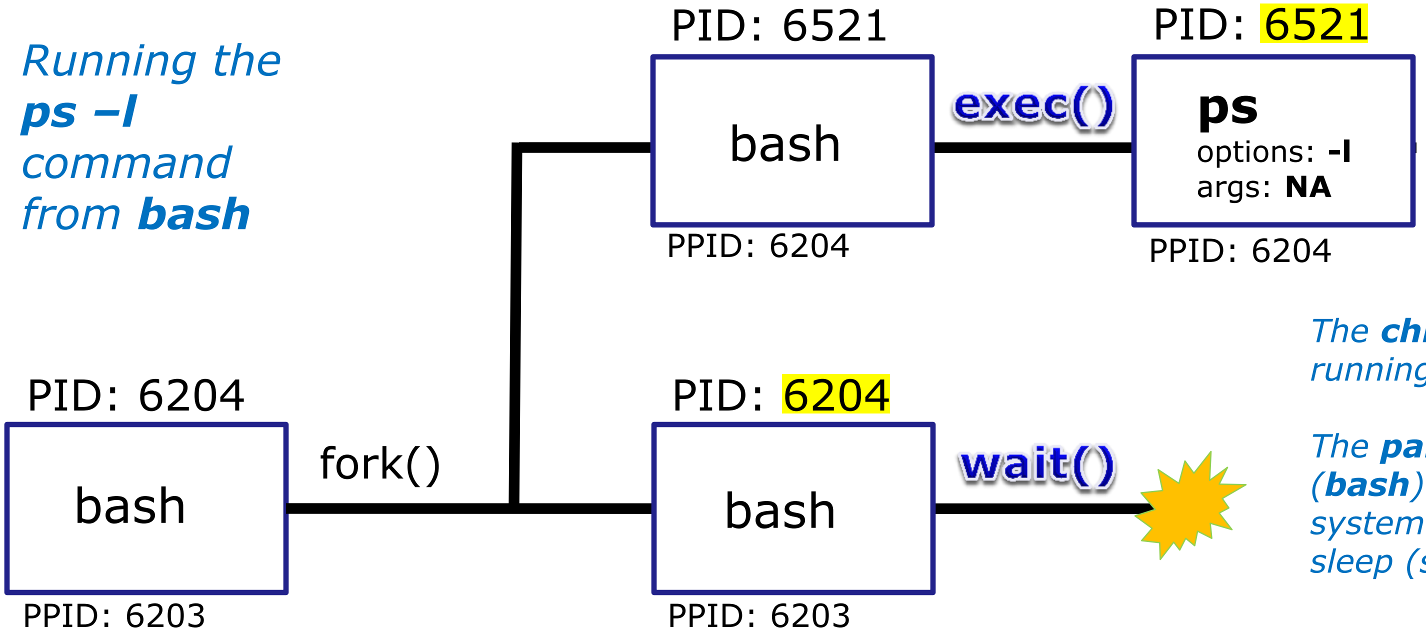
*Process ID*

*Parent Process ID*

*Size in 1K blocks*

# Deep Dive View of **ps -l** command

*Running the* ***ps –l*** *command from* ***bash***



*The **child** process (**ps**) is running (status=R)*

*The **parent** process (**bash**) issues the wait system call and goes to sleep (status=S)*

*6204 is sleeping*
*6521 is running*

```
[rsimms@opus ~]$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   201  6204  6203  0  75   0 -  1165 wait   pts/6    00:00:00 bash
0 R   201  6521  6204  0  77   0 -  1050 -      pts/6    00:00:00 ps
```

An **exec** system call is issued to overlay the **child** process with the instructions of the requested command.  The new instructions then are executed.

# **ps** command with -**ef** options (page 1)

```
[rsimms@opus ~]$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Sep10 ?        00:00:05 init [3]
root         2     1  0 Sep10 ?        00:00:00 [migration/0]
root         3     1  0 Sep10 ?        00:00:00 [ksoftirqd/0]
root         4     1  0 Sep10 ?        00:00:00 [watchdog/0]
root         5     1  0 Sep10 ?        00:00:02 [migration/1]
root         6     1  0 Sep10 ?        00:00:00 [ksoftirqd/1]
root         7     1  0 Sep10 ?        00:00:00 [watchdog/1]
root         8     1  0 Sep10 ?        00:00:00 [events/0]
root         9     1  0 Sep10 ?        00:00:00 [events/1]
root        10     1  0 Sep10 ?        00:00:00 [khelper]
root        11     1  0 Sep10 ?        00:00:00 [kthread]
root        15    11  0 Sep10 ?        00:00:00 [kblockd/0]
root        16    11  0 Sep10 ?        00:00:00 [kblockd/1]
root        17    11  0 Sep10 ?        00:00:00 [kacpid]
root       109    11  0 Sep10 ?        00:00:00 [cqueue/0]
root       110    11  0 Sep10 ?        00:00:00 [cqueue/1]
root       113    11  0 Sep10 ?        00:00:00 [khubd]
root       115    11  0 Sep10 ?        00:00:00 [kseriod]
root       181    11  0 Sep10 ?        00:00:00 [pdflush]
root       182    11  0 Sep10 ?        00:00:07 [pdflush]
root       183    11  0 Sep10 ?        00:00:01 [kswapd0]
root       184    11  0 Sep10 ?        00:00:00 [aio/0]
root       185    11  0 Sep10 ?        00:00:00 [aio/1]
root       341    11  0 Sep10 ?        00:00:00 [kpsmoused]
root       371    11  0 Sep10 ?        00:00:00 [ata/0]
```

*Use **-ef** option to see everything with full format*

# **ps** command with -**ef** options (page 2)

```
root       372    11  0 Sep10 ?        00:00:00 [ata/1]
root       373    11  0 Sep10 ?        00:00:00 [ata_aux]
root       377    11  0 Sep10 ?        00:00:00 [scsi_eh_0]
root       378    11  0 Sep10 ?        00:00:00 [scsi_eh_1]
root       379    11  0 Sep10 ?        00:01:25 [kjournald]
root       412    11  0 Sep10 ?        00:00:00 [kauditd]
root       446     1  0 Sep10 ?        00:00:00 /sbin/udevd -d
root       869    11  0 Sep10 ?        00:00:01 [kedac]
root      1420    11  0 Sep10 ?        00:00:00 [kmpathd/0]
root      1421    11  0 Sep10 ?        00:00:00 [kmpathd/1]
root      2082     1  0 Sep10 ?        00:00:05 /usr/sbin/restorecond
root      2098     1  0 Sep10 ?        00:00:11 auditd
root      2100  2098  0 Sep10 ?        00:00:05 /sbin/audispd
root      2120     1  0 Sep10 ?        00:00:23 syslogd -m 0
root      2123     1  0 Sep10 ?        00:00:00 klogd -x
root      2160     1  0 Sep10 ?        00:00:20 mcstransd
rpc       2183     1  0 Sep10 ?        00:00:00 portmap
root      2201     1  0 Sep10 ?        00:01:18 /usr/bin/python -E /usr/sbin/setroub
rpcuser   2227     1  0 Sep10 ?        00:00:00 rpc.statd
root      2275     1  0 Sep10 ?        00:00:00 rpc.idmapd
root      2345     1  0 Sep10 ?        00:00:00 /usr/bin/vmnet-bridge -d /var/run/vm
root      2364     1  0 Sep10 ?        00:00:00 /usr/bin/vmnet-natd -d /var/run/vmne
dbus      2383     1  0 Sep10 ?        00:00:15 dbus-daemon --system
root      2434     1  0 Sep10 ?        00:00:51 pcscd
root      2472     1  0 Sep10 ?        00:00:00 /usr/bin/hidd --server
root      2493     1  0 Sep10 ?        00:00:02 automount
```

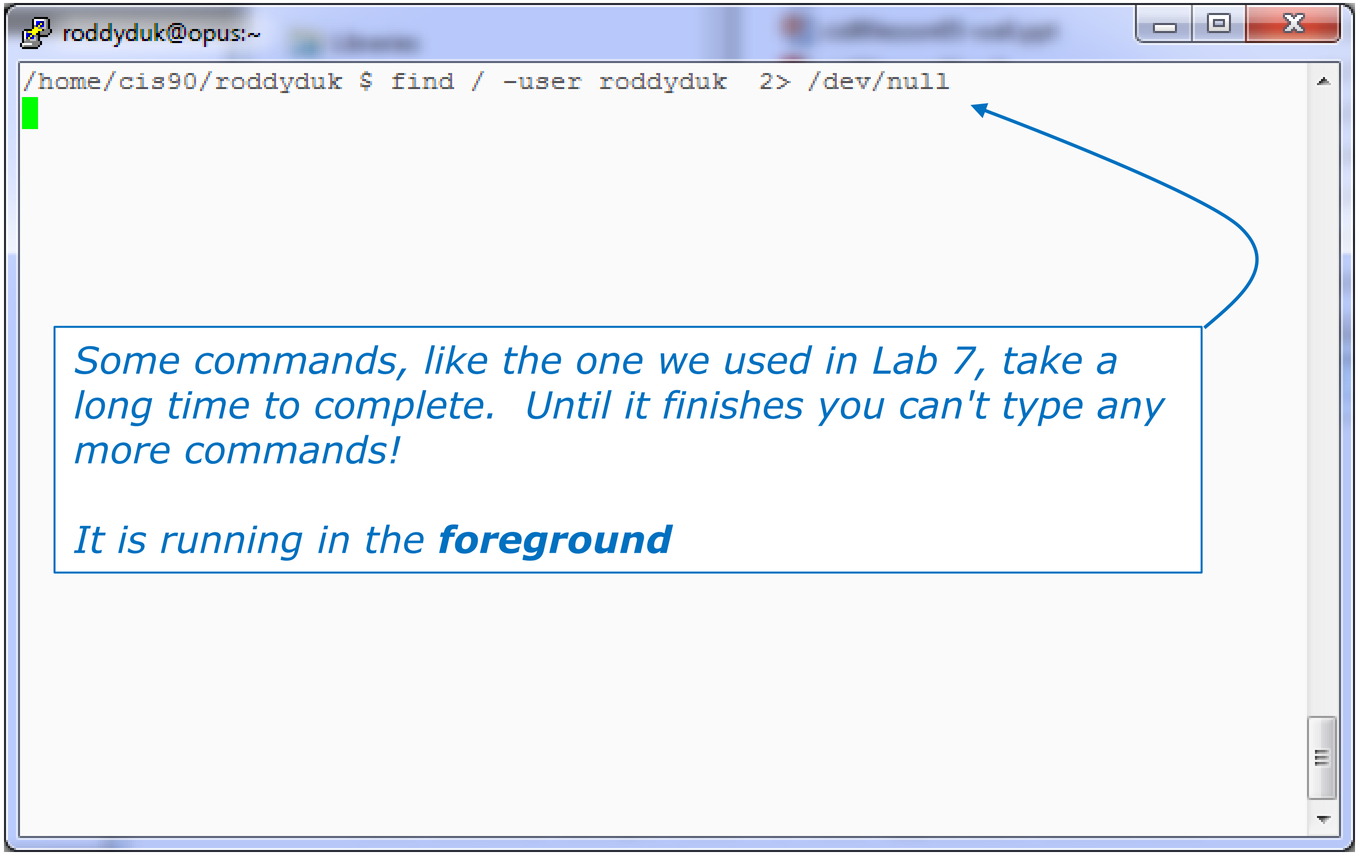
# **ps** command with -**ef** options (page 3)

```
root      2534     1  0 Sep10 ?        00:00:00 ./hpiod
root      2539     1  0 Sep10 ?        00:00:00 python ./hpssd.py
root      2556     1  0 Sep10 ?        00:00:00 cupsd
root      2575     1  0 Sep10 ?        00:00:11 /usr/sbin/sshd
root      2600     1  0 Sep10 ?        00:00:01 sendmail: accepting connections
smmsp     2609     1  0 Sep10 ?        00:00:00 sendmail: Queue runner@01:00:00 for
root      2626     1  0 Sep10 ?        00:00:00 crond
xfs       2662     1  0 Sep10 ?        00:00:00 xfs -droppriv -daemon
root      2693     1  0 Sep10 ?        00:00:00 /usr/sbin/atd
root      2710     1  0 Sep10 ?        00:00:00 rhnsd --interval 240
root      2743     1  0 Sep10 ?        00:01:33 /usr/bin/python -tt /usr/sbin/yum-up
root      2745     1  0 Sep10 ?        00:00:00 /usr/libexec/gam_server
root      2749     1  0 Sep10 ?        00:00:00 /usr/bin/vmnet-netifup -d /var/run/v
root      2758     1  0 Sep10 ?        00:00:00 /usr/bin/vmnet-netifup -d /var/run/v
root      2768     1  0 Sep10 ?        00:00:00 /usr/bin/vmnet-netifup -d /var/run/v
root      2827     1  0 Sep10 ?        00:00:00 /usr/bin/vmnet-dhcpd -cf /etc/vmware
root      2858     1  0 Sep10 ?        00:00:00 /usr/bin/vmnet-dhcpd -cf /etc/vmware
root      2859     1  0 Sep10 ?        00:00:00 /usr/bin/vmnet-dhcpd -cf /etc/vmware
68        2875     1  0 Sep10 ?        00:00:01 hald
root      2876  2875  0 Sep10 ?        00:00:00 hald-runner
68        2883  2876  0 Sep10 ?        00:00:00 hald-addon-acpi: listening on acpid
68        2886  2876  0 Sep10 ?        00:00:00 hald-addon-keyboard: listening on /d
68        2890  2876  0 Sep10 ?        00:00:00 hald-addon-keyboard: listening on /d
root      2898  2876  0 Sep10 ?        00:02:46 hald-addon-storage: polling /dev/hda
root      2944     1  0 Sep10 ?        00:00:00 /usr/sbin/smartd -q never
root      2949     1  0 Sep10 tty2     00:00:00 /sbin/mingetty tty2
```

# **ps** command with -**ef** options (page 4)

```
root      2950     1  0 Sep10 tty3     00:00:00 /sbin/mingetty tty3
root      5365  2575  0 08:19 ?        00:00:00 sshd: rsimms [priv]
rsimms    5368  5365  0 08:19 ?        00:00:00 sshd: rsimms@pts/0
rsimms    5369  5368  0 08:19 pts/0    00:00:00 -bash
root      5969  2575  0 10:14 ?        00:00:00 sshd: valdemar [priv]
valdemar  5971  5969  0 10:14 ?        00:00:00 sshd: valdemar@pts/5
valdemar  5972  5971  0 10:14 pts/5    00:00:00 -bash
rsimms    6173  5369  0 10:36 pts/0    00:00:00 man ps
rsimms    6176  6173  0 10:36 pts/0    00:00:00 sh -c (cd /usr/share/man && (echo ".
rsimms    6177  6176  0 10:36 pts/0    00:00:00 sh -c (cd /usr/share/man && (echo ".
rsimms    6182  6177  0 10:36 pts/0    00:00:00 /usr/bin/less -is
root      6200  2575  0 10:37 ?        00:00:00 sshd: rsimms [priv]
rsimms    6203  6200  0 10:37 ?        00:00:00 sshd: rsimms@pts/6
rsimms    6204  6203  0 10:37 pts/6    00:00:00 -bash
root      6408  2575  0 11:07 ?        00:00:00 sshd: dymesdia [priv]
dymesdia  6418  6408  0 11:08 ?        00:00:00 sshd: dymesdia@pts/1
dymesdia  6419  6418  0 11:08 pts/1    00:00:00 -bash
rsimms    6524  6204  0 11:15 pts/6    00:00:00 ps -ef
lyonsrob 12891     1  0 Oct01 ?        00:00:00 SCREEN
lyonsrob 12892 12891  0 Oct01 pts/3    00:00:00 /bin/bash
root     29218     1  0 Oct15 tty1     00:00:00 /sbin/mingetty tty1
[rsimms@opus ~]$
```

# Job Control



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

*Some commands, like the one we used in Lab 7, take a long time to complete.  Until it finishes you can't type any more commands!*

*It is running in the* ***foreground***

# Job Control

## A feature of the bash shell

Foreground processes

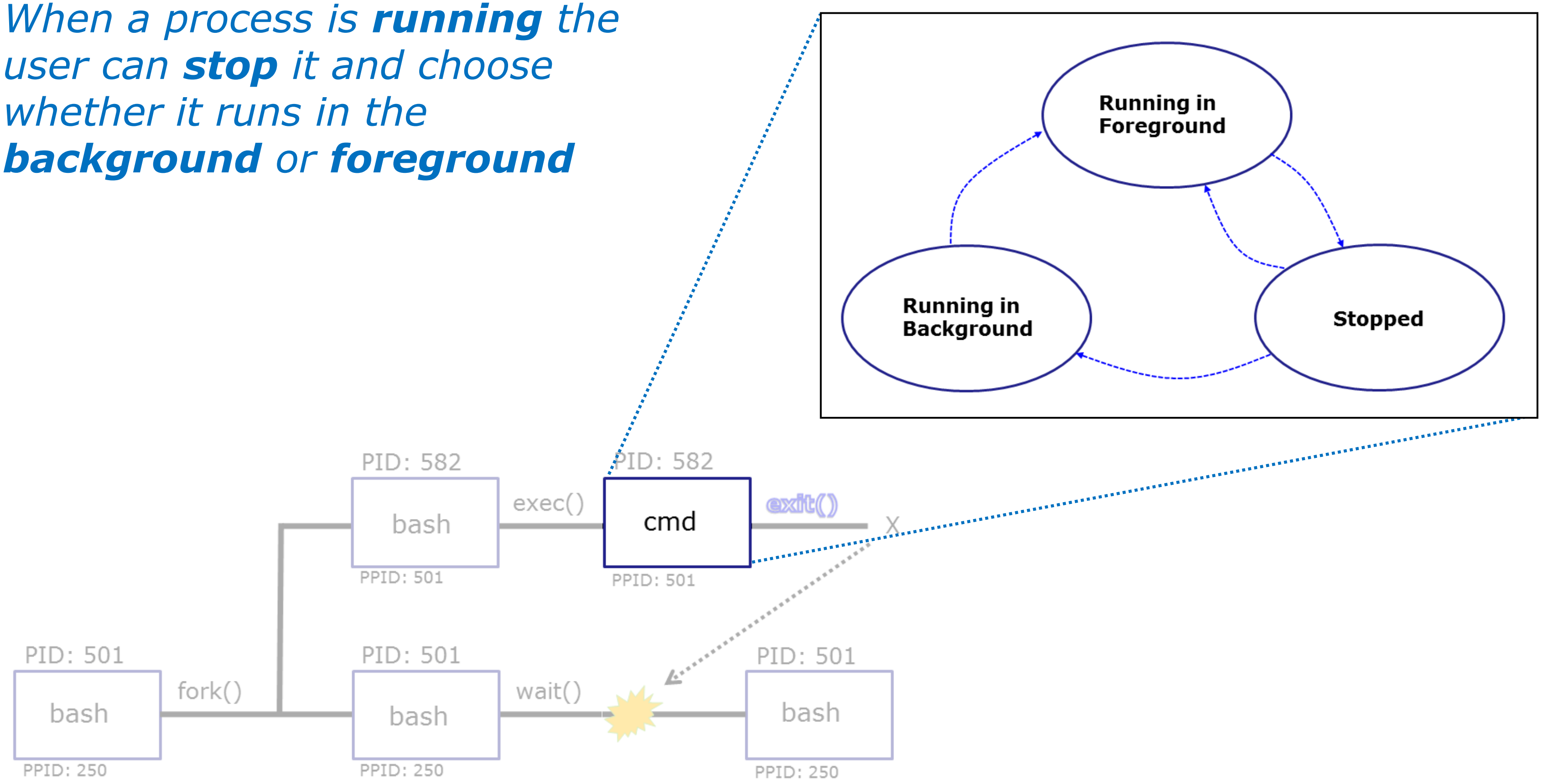
- Processes that receive their input and write their output to the terminal.
- The parent shell waits on these processes to die.

Background Processes

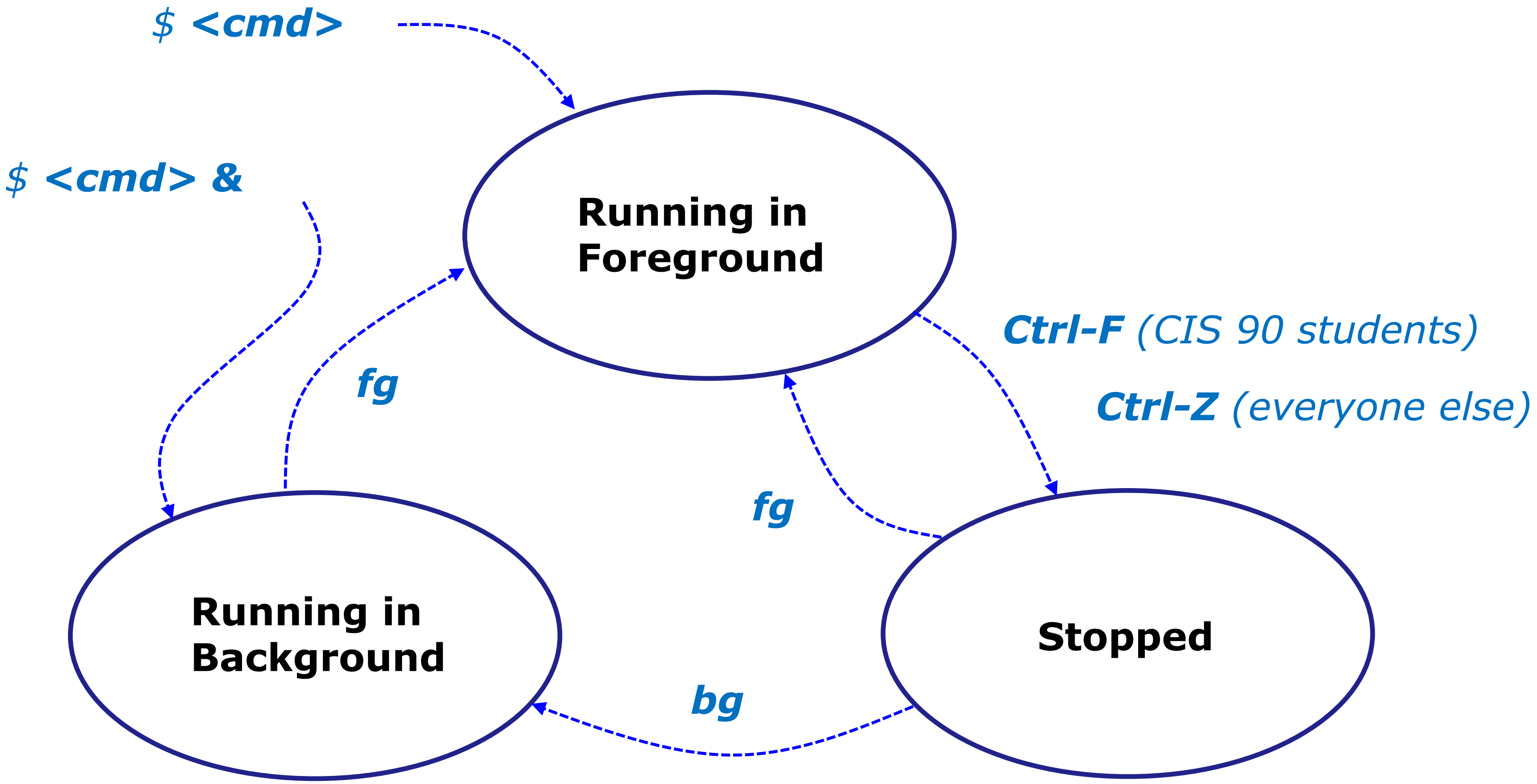
- Processes that do not get their input from a user keyboard.
- The parent shell does not wait on these processes; it re-prompts the user for next command.

# Job Control
## A feature of the bash shell

*When a process is **running** the user can **stop** it and choose whether it runs in the **background** or **foreground***



# Job Control
## A feature of the bash shell



*Use the **jobs** command to view stopped and background jobs*

# Job Control
## Suspending and Resuming

**Ctrl-F**

- Stops (suspends) a foreground process by sending it a "TTY Stop" (SIGTSTP) signal

*Note, CIS 90 students will be using Ctrl-F which has been configured in their shell environment. Normally Ctrl-Z is used.*

**bg**

- resumes the currently suspended process and runs it in the background

# Job Control

## Keyboard customization for CIS 90

**Ctrl-Z** or **Ctrl-F**

- To send a SIGTSTP signal from the keyboard
- Stops (suspends) a foreground process

*CIS 90 accounts use **Ctrl-F***

```
/home/cis90/simben $ stty -a
speed 38400 baud; rows 26; columns 78; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; swtch = <undef>; start = ^Q; stop = ^S; susp = ^F; rprnt = ^R;
werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
```

*Other Opus accounts use **Ctrl-Z***

```
[rsimms@opus ~]$ stty -a
speed 38400 baud; rows 39; columns 84; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>;
swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V; flush = ^O; min = 1; time = 0;
```

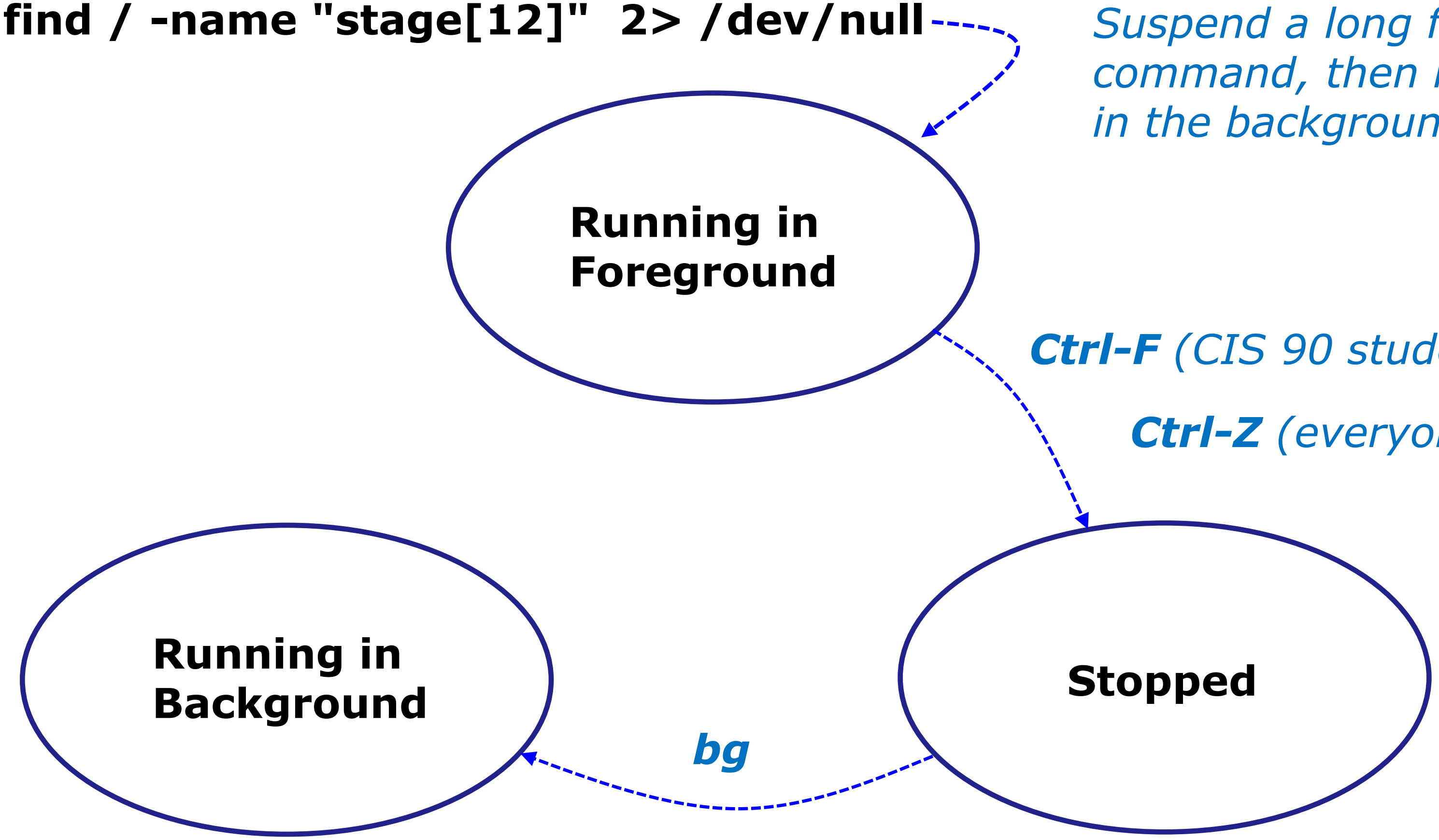
*The bash shell environment for the CIS 90 accounts was customized to use a different keystroke for sending a SIGTSTP signal*

# Job Control

## Example - suspending a **find** command

```
$ find / -name "stage[12]"  2> /dev/null
```

*Suspend a long find command, then resume it in the background*



Running in Foreground
Stopped
Running in Background
***Ctrl-F*** *(CIS 90 students)*
***Ctrl-Z*** *(everyone else)*
***bg***

# Job Control

## Example - suspending a **find** command

```
[rsimms@opus ~]$ find / -name "stage[12]"  2> /dev/null

[1]+  Stopped                 find / -name "stage[12]" 2> /dev/null
[rsimms@opus ~]$ bg
[1]+ find / -name "stage[12]" 2> /dev/null &
[rsimms@opus ~]$
```

*Notice, we can type more commands again after the find command was stopped*

***Ctrl-F*** *(CIS 90 accounts) or* ***Ctrl-Z*** *(other accounts) is tapped to suspend the find command*

```
[rsimms@opus ~]$ ps -l -u rsimms
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
5 S   201 25055 25044  0  75   0 -  2481 stext  ?        00:00:00 sshd
0 S   201 25056 25055  0  78   0 -  1168 -      pts/3    00:00:00 bash
5 S   201 25087 25084  0  75   0 -  2481 stext  ?        00:00:00 sshd
0 S   201 25088 25087  0  75   0 -  1168 wait   pts/4    00:00:00 bash
0 T   201 25124 25056  2  78   0 -  1098 finish pts/3    00:00:00 find
0 R   201 25127 25088  0  77   0 -  1065 -      pts/4    00:00:00 ps
```

*Process ID 25124 (find) is stopped (status =T)*

# Job Control

## Example - suspending a **find** command

```
[rsimms@opus ~]$ find / -name "stage[12]"  2> /dev/null
/boot/grub/stage1
/boot/grub/stage2
/usr/share/grub/i386-redhat/stage1
/usr/share/grub/i386-redhat/stage2

[1]+  Stopped                 find / -name "stage[12]" 2> /dev/null
[rsimms@opus ~]$ bg
[1]+ find / -name "stage[12]" 2> /dev/null &
[rsimms@opus ~]$
```

***bg** resumes the find command in the background*

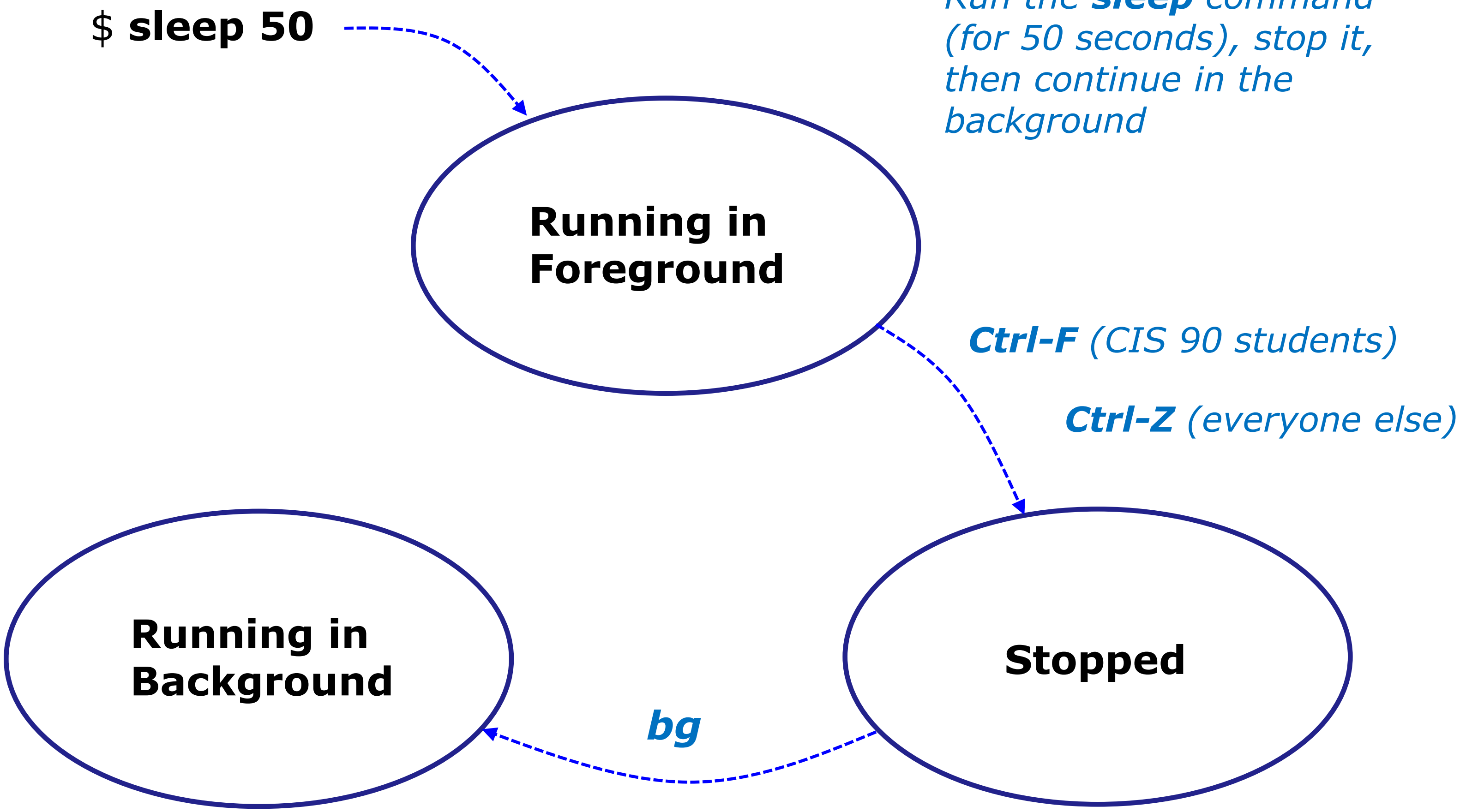
```
[rsimms@opus ~]$ ps -l -u rsimms
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
5 S   201 25055 25044  0  75   0 -  2481 stext  ?        00:00:00 sshd
0 S   201 25056 25055  0  75   0 -  1168 -      pts/3    00:00:00 bash
5 S   201 25087 25084  0  75   0 -  2481 stext  ?        00:00:00 sshd
0 S   201 25088 25087  0  75   0 -  1168 wait   pts/4    00:00:00 bash
0 R   201 25124 25056  1  78   0 -  1099 -      pts/3    00:00:00 find
0 R   201 25129 25088  0  77   0 -  1065 -      pts/4    00:00:00 ps
```

*Process ID 25124 (find) is running (status=R)*

# Job Control

## Example - suspending a **sleep** command

*Run the **sleep** command (for 50 seconds), stop it, then continue in the background*



$ **sleep 50**

**Running in Foreground**

***Ctrl-F*** *(CIS 90 students)*

***Ctrl-Z*** *(everyone else)*

**Stopped**

***bg***

**Running in Background**

# Job Control

## Example - suspending a **sleep** command

```
[rsimms@opus ~]$ sleep 50

[1]+  Stopped                 sleep 50
[rsimms@opus ~]$
```

***Ctrl-F** (CIS 90 accounts) or **Ctrl-Z** (other accounts) is tapped while sleep is running*

```
[rsimms@opus ~]$ ps -l  -u rsimms
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
5 S   201 25055 25044  0  75   0 -  2481 stext  ?        00:00:00 sshd
0 S   201 25056 25055  0  76   0 -  1168 -      pts/3    00:00:00 bash
5 S   201 25087 25084  0  75   0 -  2481 stext  ?        00:00:00 sshd
0 S   201 25088 25087  0  75   0 -  1168 wait   pts/4    00:00:00 bash
0 T   201 25389 25056  0  76   0 -   929 finish pts/3    00:00:00 sleep
0 R   201 25391 25088  0  77   0 -  1065 -      pts/4    00:00:00 ps
```

*PID 25389 (sleep) is stopped*

# Job Control

## Example - suspending a **sleep** command

```
[rsimms@opus ~]$ sleep 50

[1]+  Stopped                 sleep 50
[rsimms@opus ~]$ bg
[1]+ sleep 50 &
```

***bg** resumes the sleep command and it finishes*

*PID 25389 is sleeping and no longer stopped (status=S)*

```
[rsimms@opus ~]$ ps -l  -u rsimms
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
5 S   201 25055 25044  0  75   0 -  2481 stext  ?        00:00:00 sshd
0 S   201 25056 25055  0  75   0 -  1168 -      pts/3    00:00:00 bash
5 R   201 25087 25084  0  81   0 -  2481 stext  ?        00:00:00 sshd
0 S   201 25088 25087  0  75   0 -  1168 wait   pts/4    00:00:00 bash
0 S   201 25389 25056  0  75   0 -   929 322807 pts/3    00:00:00 sleep
0 R   201 25394 25088  0  77   0 -  1065 -      pts/4    00:00:00 ps
[rsimms@opus ~]$
```

# Job Control
## Additional Control Options

**&**

- Append to a command to run it in the background

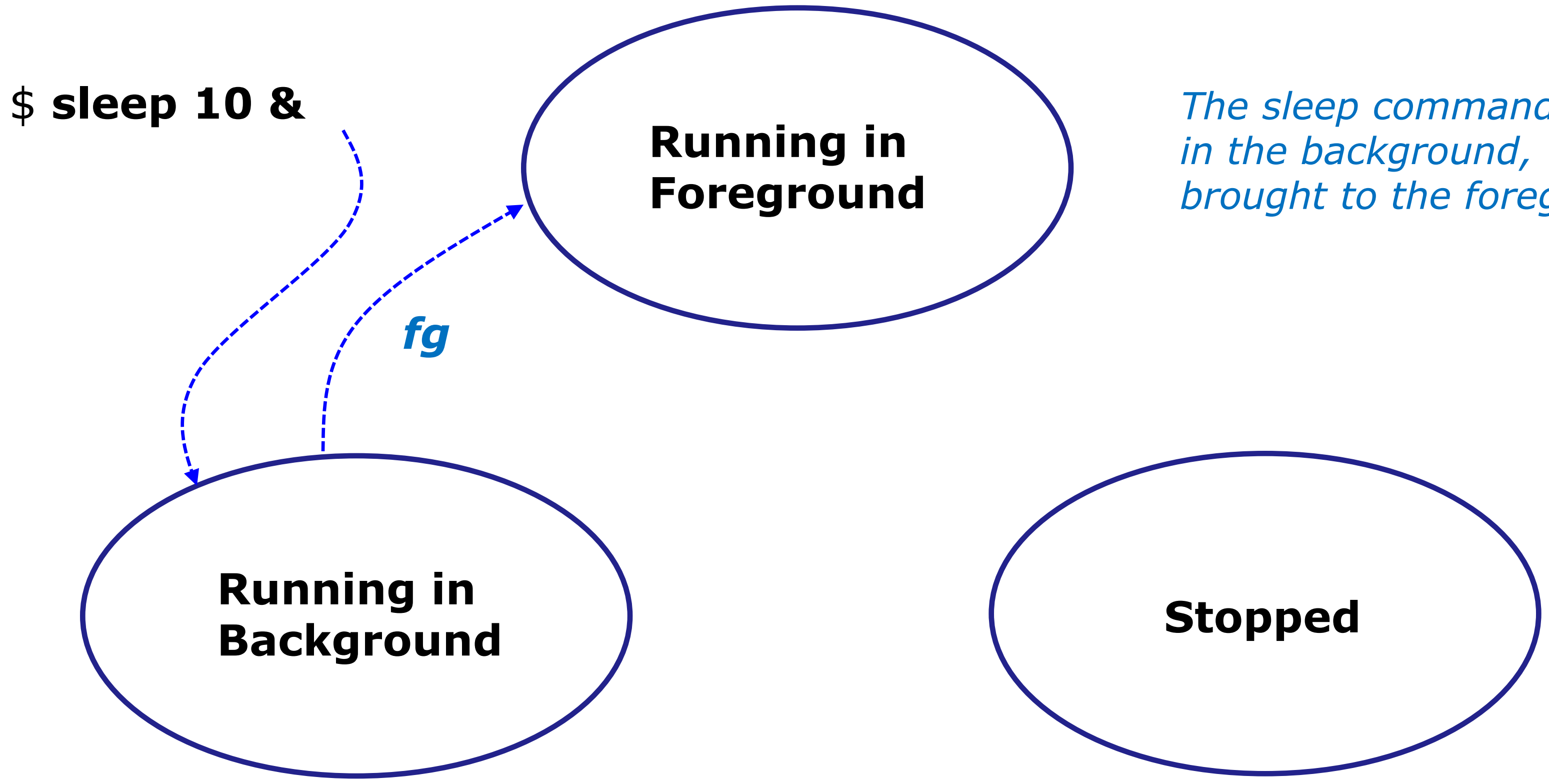
**fg**

- Brings the most recent background process to the foreground

**jobs**

- Lists all background jobs

# Job Control
## Example



*The sleep command is started in the background, then brought to the foreground*

# Job Control
## Example

*The **&** has **sleep** run in the background and jobs shows the  shows it as the one and only background job*

```
[rsimms@opus ~]$ sleep 10 &
[1] 7761
[rsimms@opus ~]$ jobs
[1]+  Running                 sleep 10 &
[rsimms@opus ~]$ fg
sleep 10
```

*After **fg**, sleep now runs in the foreground. The prompt is gone.  Need to wait until **sleep** finishes for prompt to return.*

```
[rsimms@opus ~]$
[rsimms@opus ~]$
```

***&** is often used when running GUI tools like **firefox** or **wireshark** from the command line.  This allows you to keep using the terminal for more commands while those applications run.*

# Signals

# Signals

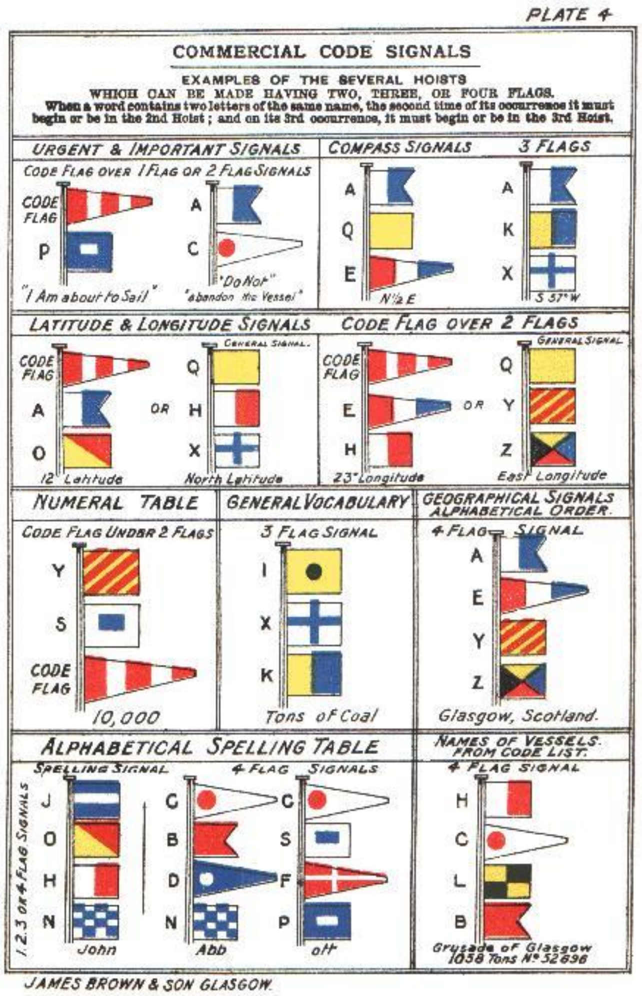
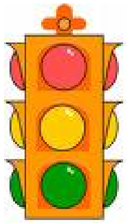
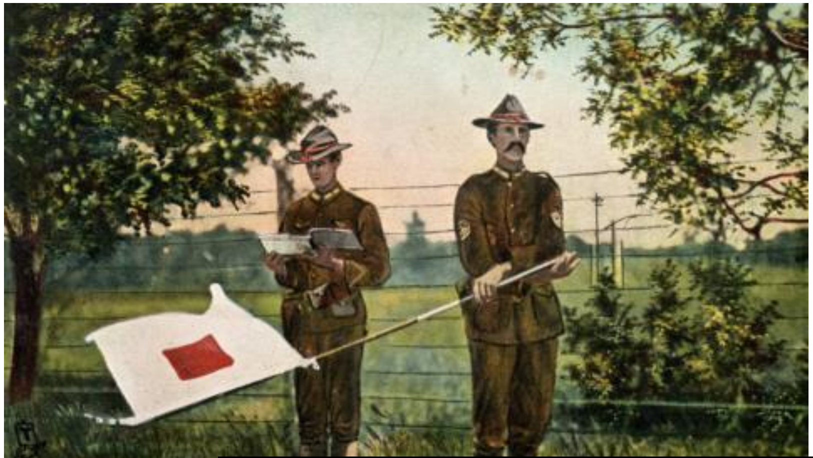
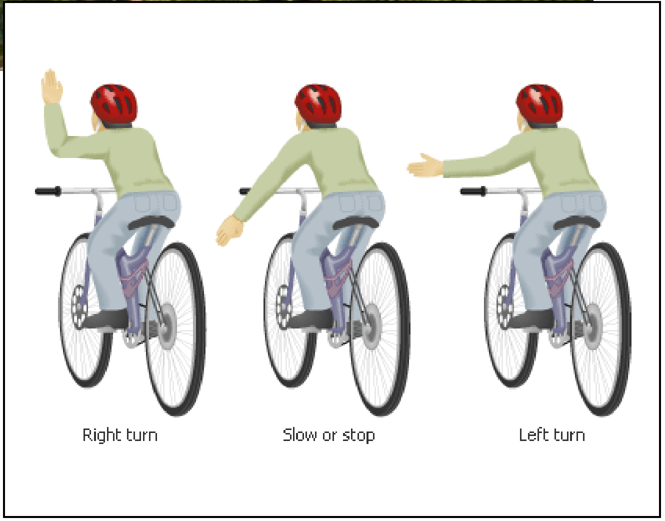


PLATE 4
COMMERCIAL CODE SIGNALS
EXAMPLES OF THE SEVERAL HOISTS
WHICH CAN BE MADE HAVING TWO, THREE, OR FOUR FLAGS.
When a word contains two letters of the same name, the second time of its occurrence it must begin or be in the 2nd Hoist; and on its 3rd occurrence, it must begin or be in the 3rd Hoist.
URGENT & IMPORTANT SIGNALS
CODE FLAG OVER 1 FLAG OR 2 FLAG SIGNALS
CODE FLAG
P
"I Am about to Sail"
A
C
"Do Not" "abandon the Vessel"
COMPASS SIGNALS
3 FLAGS
A
Q
E
N½E
A
K
X
S 57° W
LATITUDE & LONGITUDE SIGNALS
GENERAL SIGNAL
CODE FLAG
A
O
12° Latitude
OR
Q
H
X
North Latitude
CODE FLAG OVER 2 FLAGS
GENERAL SIGNAL
CODE FLAG
E
H
23° Longitude
OR
Q
Y
Z
East Longitude
NUMERAL TABLE
CODE FLAG UNDER 2 FLAGS
Y
S
CODE FLAG
10,000
GENERAL VOCABULARY
3 FLAG SIGNAL
I
X
K
Tons of Coal
GEOGRAPHICAL SIGNALS ALPHABETICAL ORDER
4 FLAG SIGNAL
A
E
Y
Z
Glasgow, Scotland.
ALPHABETICAL SPELLING TABLE
SPELLING SIGNAL
1.2.3 OR 4 FLAG SIGNALS
J
O
H
N
John
4 FLAG SIGNALS
C
B
D
N
Abb
C
S
F
P
off
NAMES OF VESSELS FROM CODE LIST
4 FLAG SIGNAL
H
C
L
B
Grusade of Glasgow 1058 Tons Nº 32896
JAMES BROWN & SON GLASGOW



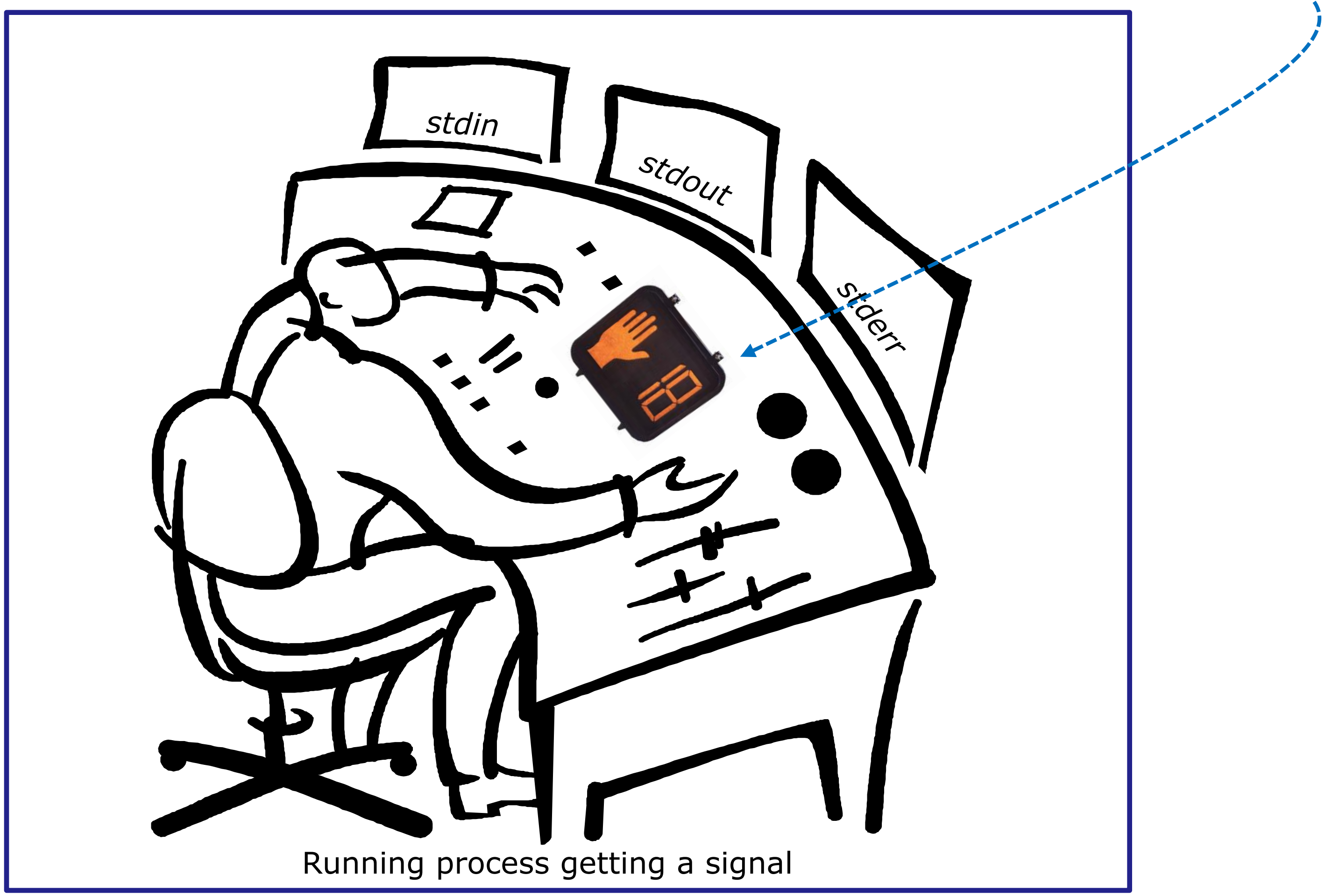

U. S. ARMY SIGNAL
Wig-Wagging

18


Right turn
Slow or stop
Left turn

# Signals

*Signals are asynchronous messages sent to processes*



Running process getting a signal

*Asynchronous means it can happen at any time*

# Signals

Signals are asynchronous messages sent to processes

They can result in one of three courses of action:

1. be ignored,
2. default action (die)
3. execute some predefined function.

Signals are sent:

- Using the kill command: **`$ kill -#`** ***`PID`***
  - Where # is the signal number and PID is the process id.
  - if no signal number is specified, SIGTERM is sent.
- Using special keystrokes (e.g. Ctrl-Z for SIGTSTP/20)
  - limited to just a few signals
  - sent to the process running in the foreground

*Use kill –l to see all signals*

# Signals

*Signals are asynchronous messages sent to processes*

Running process gets a signal

# Signals

| | | |
|---|---|---|
| SIGHUP | 1 | Hangup (POSIX) |
| SIGINT | 2 | Terminal interrupt (ANSI) ***Ctrl-C*** |
| SIGQUIT | 3 | Terminal quit (POSIX) ***Ctrl-\*** |
| SIGILL | 4 | Illegal instruction (ANSI) |
| SIGTRAP | 5 | Trace trap (POSIX) |
| SIGIOT | 6 | IOT Trap (4.2 BSD) |
| SIGBUS | 7 | BUS error (4.2 BSD) |
| SIGFPE | 8 | Floating point exception (ANSI) |
| SIGKILL | 9 | Kill (can't be caught or ignored) (POSIX) |
| SIGUSR1 | 10 | User defined signal 1 (POSIX) |
| SIGSEGV | 11 | Invalid memory segment access (ANSI) |
| SIGUSR2 | 12 | User defined signal 2 (POSIX) |
| SIGPIPE | 13 | Write on a pipe with no reader, Broken pipe (POSIX) |
| SIGALRM | 14 | Alarm clock (POSIX) |
| SIGTERM | 15 | Termination (ANSI) (default kill signal when not specified) |

*Use kill –l to see all signals*

# Signals

| | | |
|---|---|---|
| SIGSTKFLT | 16 | Stack fault |
| SIGCHLD | 17 | Child process has stopped or exited, changed (POSIX) |
| SIGCONT | 18 | Continue executing, if stopped (POSIX) |
| SIGSTOP | 19 | Stop executing(can't be caught or ignored) (POSIX) |
| SIGTSTP | 20 | Terminal stop signal (POSIX) ***Ctrl-Z or Ctrl-F*** |
| SIGTTIN | 21 | Background process trying to read, from TTY (POSIX) |
| SIGTTOU | 22 | Background process trying to write, to TTY (POSIX) |
| SIGURG | 23 | Urgent condition on socket (4.2 BSD) |
| SIGXCPU | 24 | CPU limit exceeded (4.2 BSD) |
| SIGXFSZ | 25 | File size limit exceeded (4.2 BSD) |
| SIGVTALRM | 26 | Virtual alarm clock (4.2 BSD) |
| SIGPROF | 27 | Profiling alarm clock (4.2 BSD) |
| SIGWINCH | 28 | Window size change (4.3 BSD, Sun) |
| SIGIO | 29 | I/O now possible (4.2 BSD) |
| SIGPWR | 30 | Power failure restart (System V) |

*Use kill –l to see all signals*

# Signals

## Use **kill –l** to see all of them

```
/home/cis90/rodduk $ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
/home/cis90/rodduk $
```

# Signals

## Special keystrokes

```
/home/cis90/rodduk $ stty -a
speed 38400 baud; rows 26; columns 78; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; swtch = <undef>; start = ^Q; stop = ^S; susp = ^F; rprnt = ^R;
werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
```

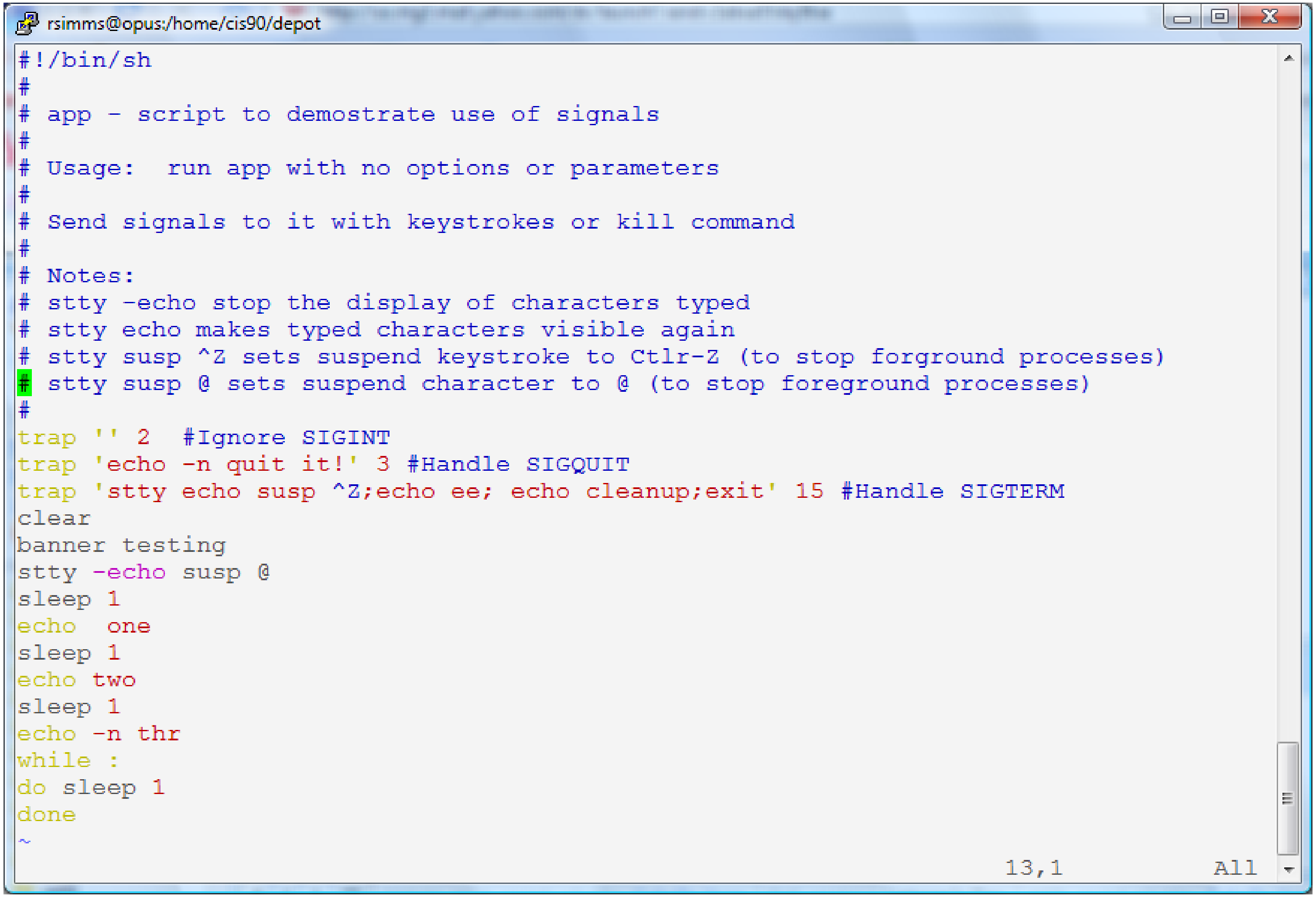
```
[rsimms@opus ~]$ stty -a
speed 38400 baud; rows 39; columns 84; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>;
swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V; flush = ^O; min = 1; time = 0;
```

*use Ctrl-C to send a SIGINT/2*

*or Ctrl-\ to send a SIGQUIT/3*

# Signals

## Jim's app script



```
#!/bin/sh
#
# app - script to demostrate use of signals
#
# Usage:  run app with no options or parameters
#
# Send signals to it with keystrokes or kill command
#
# Notes:
# stty -echo stop the display of characters typed
# stty echo makes typed characters visible again
# stty susp ^Z sets suspend keystroke to Ctlr-Z (to stop forground processes)
# stty susp @ sets suspend character to @ (to stop foreground processes)
#
trap '' 2  #Ignore SIGINT
trap 'echo -n quit it!' 3 #Handle SIGQUIT
trap 'stty echo susp ^Z;echo ee; echo cleanup;exit' 15 #Handle SIGTERM
clear
banner testing
stty -echo susp @
sleep 1
echo  one
sleep 1
echo two
sleep 1
echo -n thr
while :
do sleep 1
done
~
                                                                   13,1          All
```

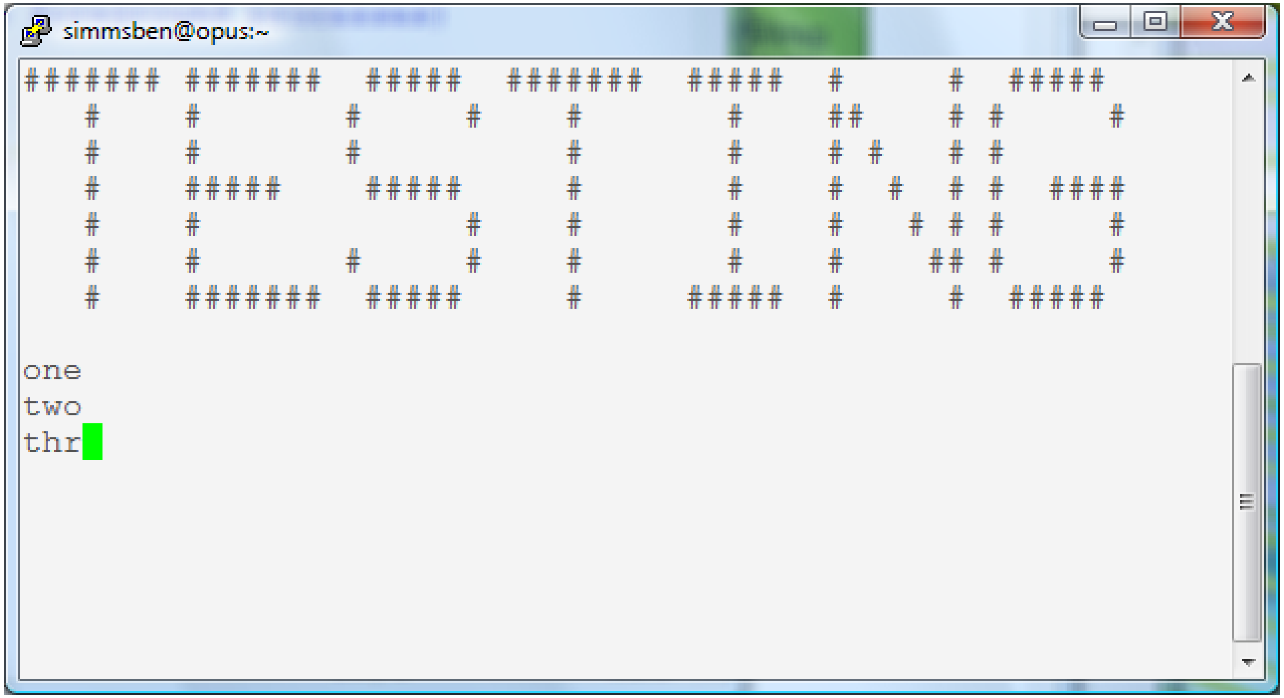
# Signals
## Class Exercise

- View with **cat bin/app**

- Look for the three trap handlers
  - Signal 2 (SIGINT)
  - Signal 3 (SIGQUIT)
  - Signal 15 (SIGTERM)

# Signals

## Benji runs app

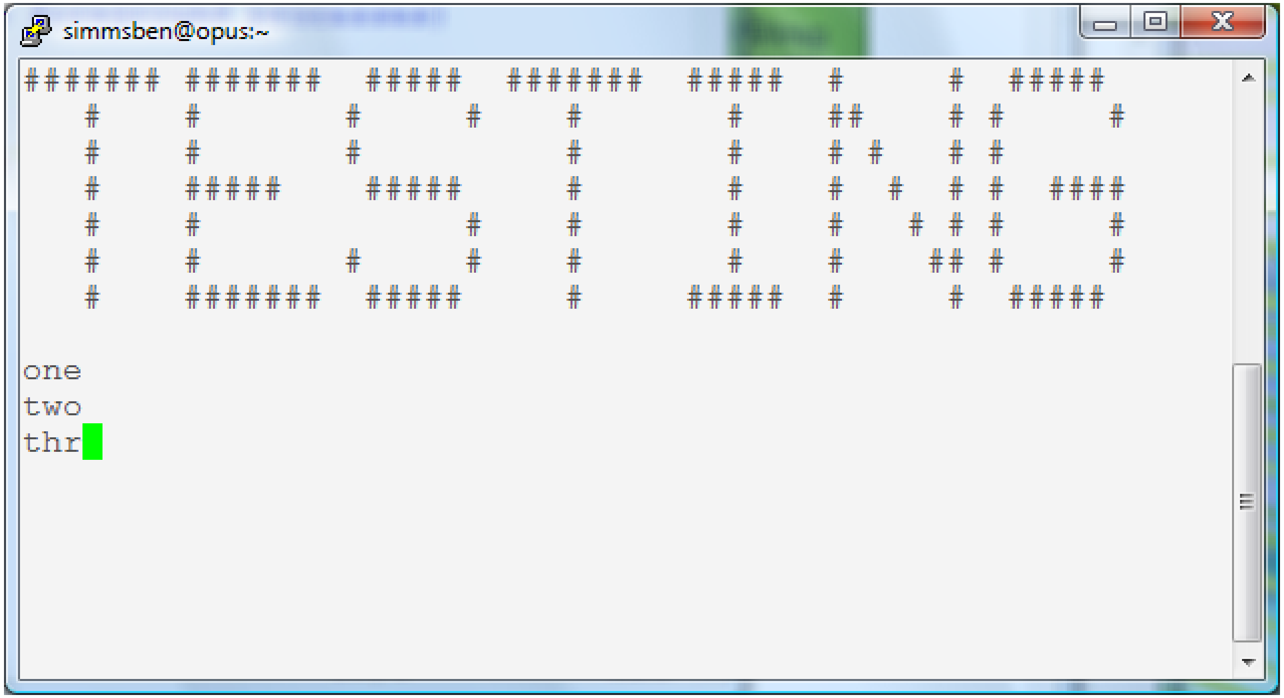


```
####### #######  #####  #######  #####  #     #  #####
   #    #       #     #    #       #    ##    # #     #
   #    #       #          #       #    # #   # #
   #    #####    #####     #       #    #  #  # #  ####
   #    #             #    #       #    #   # # #     #
   #    #       #     #    #       #    #    ## #     #
   #    #######  #####     #     #####  #     #  #####

one
two
thr
```

*Benji logs in and runs app … uh oh, its stuck !*

# Signals

## Benji runs app

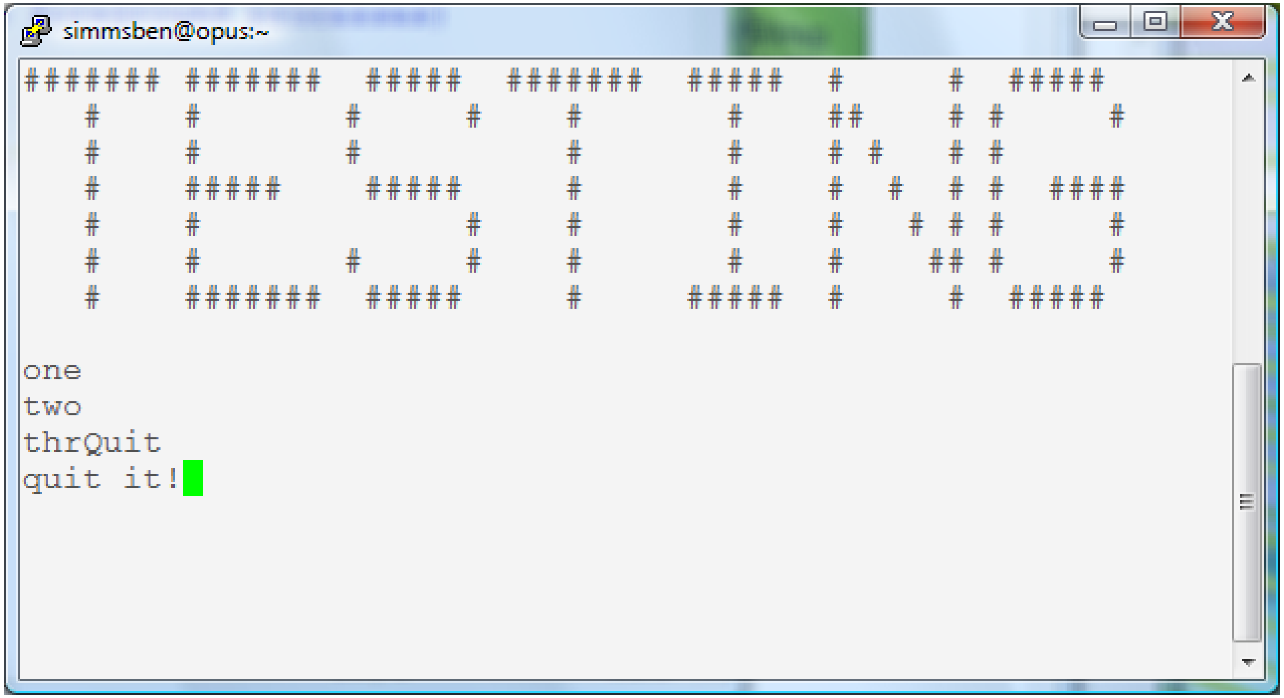


```
####### #######  #####  #######  #####  #     #  #####
   #    #       #     #    #       #    ##    # #     #
   #    #       #          #       #    # #   # #
   #    #####    #####     #       #    #  #  # #  ####
   #    #             #    #       #    #   # # #     #
   #    #       #     #    #       #    #    ## #     #
   #    #######  #####     #     #####  #     #  #####

one
two
thr
```

*Benji tries using the keyboard to send a SIGINT/2 using **Ctrl-C** but nothing happens (because app is ignoring SIGINT)*

# Signals

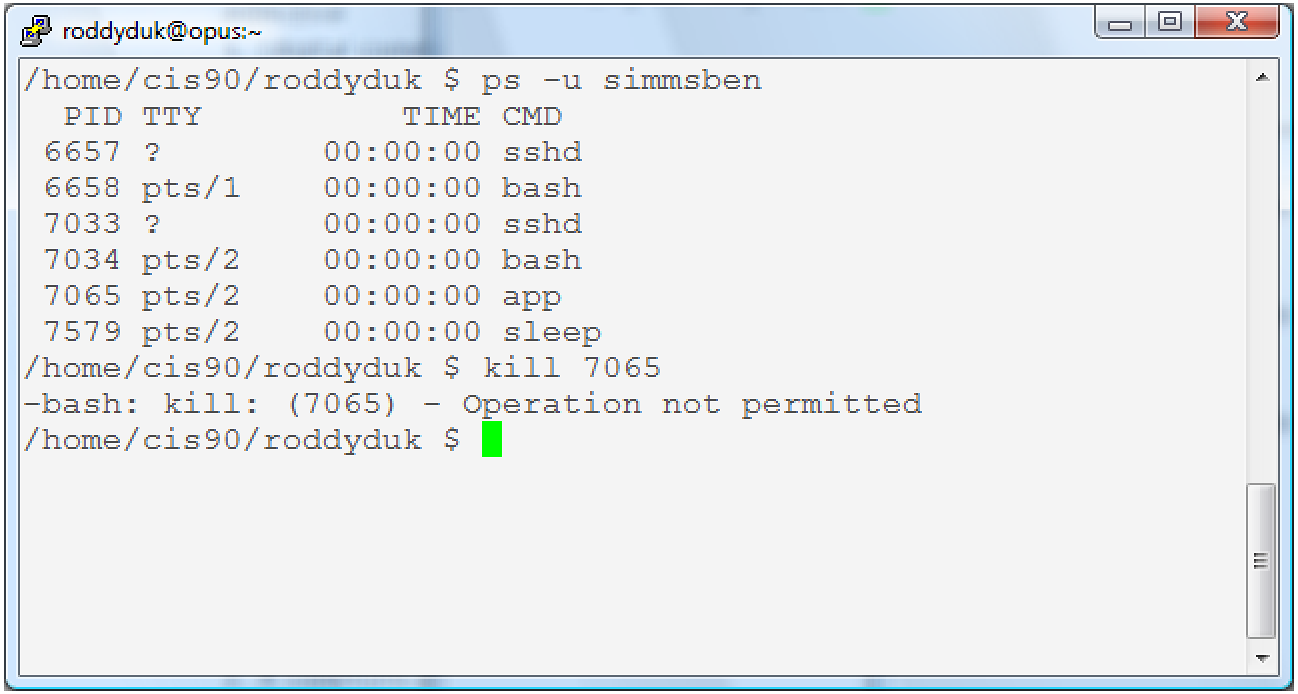
## Benji runs app



```
####### #######  #####  #######  #####  #     #  #####
   #    #       #     #    #       #     ##    # #     #
   #    #       #          #       #     # #   # #
   #    #####    #####     #       #     #  #  # #  ####
   #    #            #    #       #     #   # # #     #
   #    #       #     #    #       #     #    ## #     #
   #    #######  #####     #     #####  #     #  #####

one
two
thrQuit
quit it!
```

*Benji tries using the keyboard to send a SIGQUIT/3 using **Ctrl-\***
*but app reacts by saying "quit it"*

# Signals
## Benji runs app

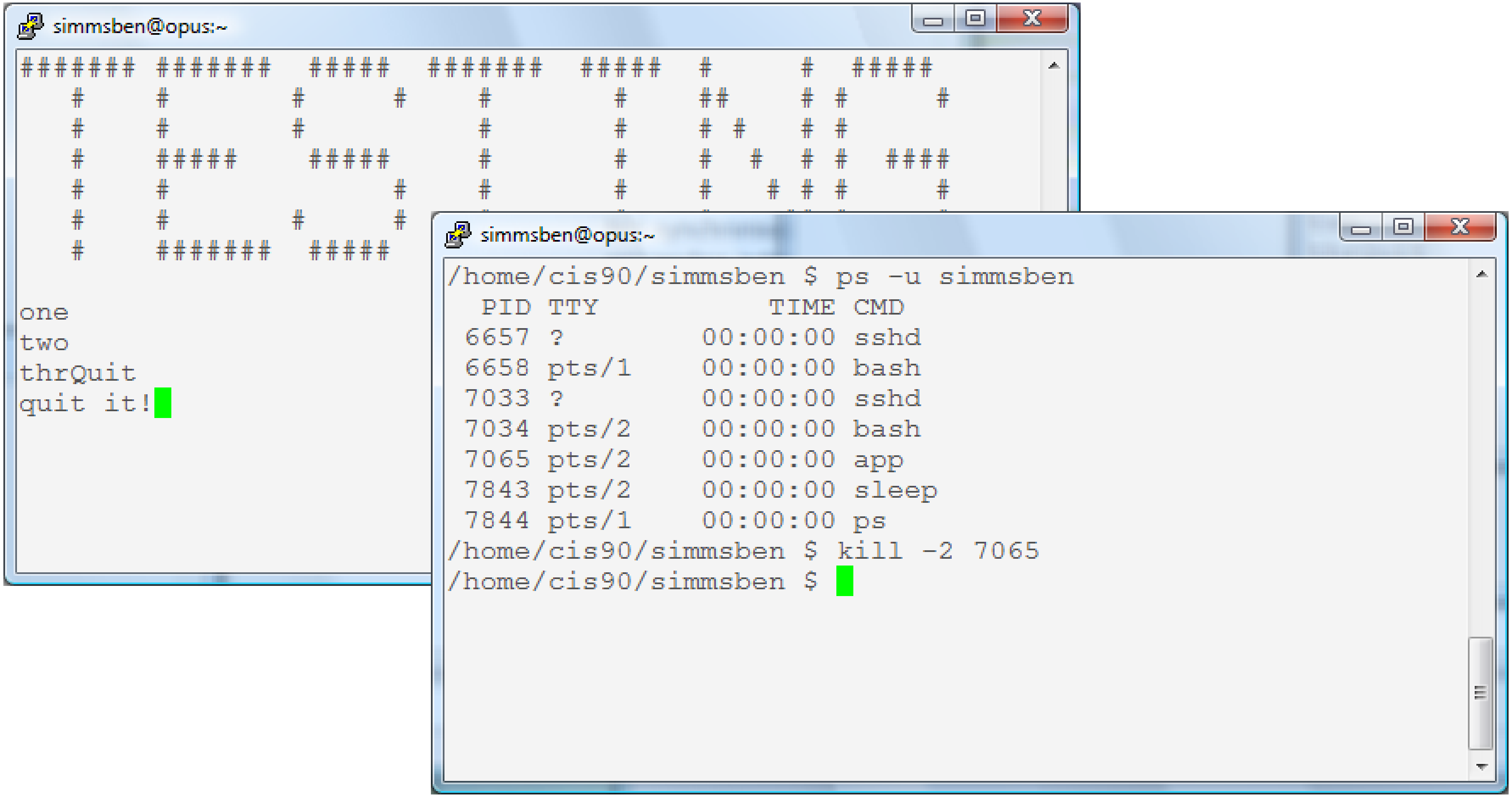


```
/home/cis90/roddyduk $ ps -u simmsben
  PID TTY          TIME CMD
 6657 ?        00:00:00 sshd
 6658 pts/1    00:00:00 bash
 7033 ?        00:00:00 sshd
 7034 pts/2    00:00:00 bash
 7065 pts/2    00:00:00 app
 7579 pts/2    00:00:00 sleep
/home/cis90/roddyduk $ kill 7065
-bash: kill: (7065) - Operation not permitted
/home/cis90/roddyduk $ 
```

*Benji asks his friend Duke to kill off his stalled app process.  Duke uses **ps** to look it up but does not have permission to kill it off*

# Signals

## Benji runs app



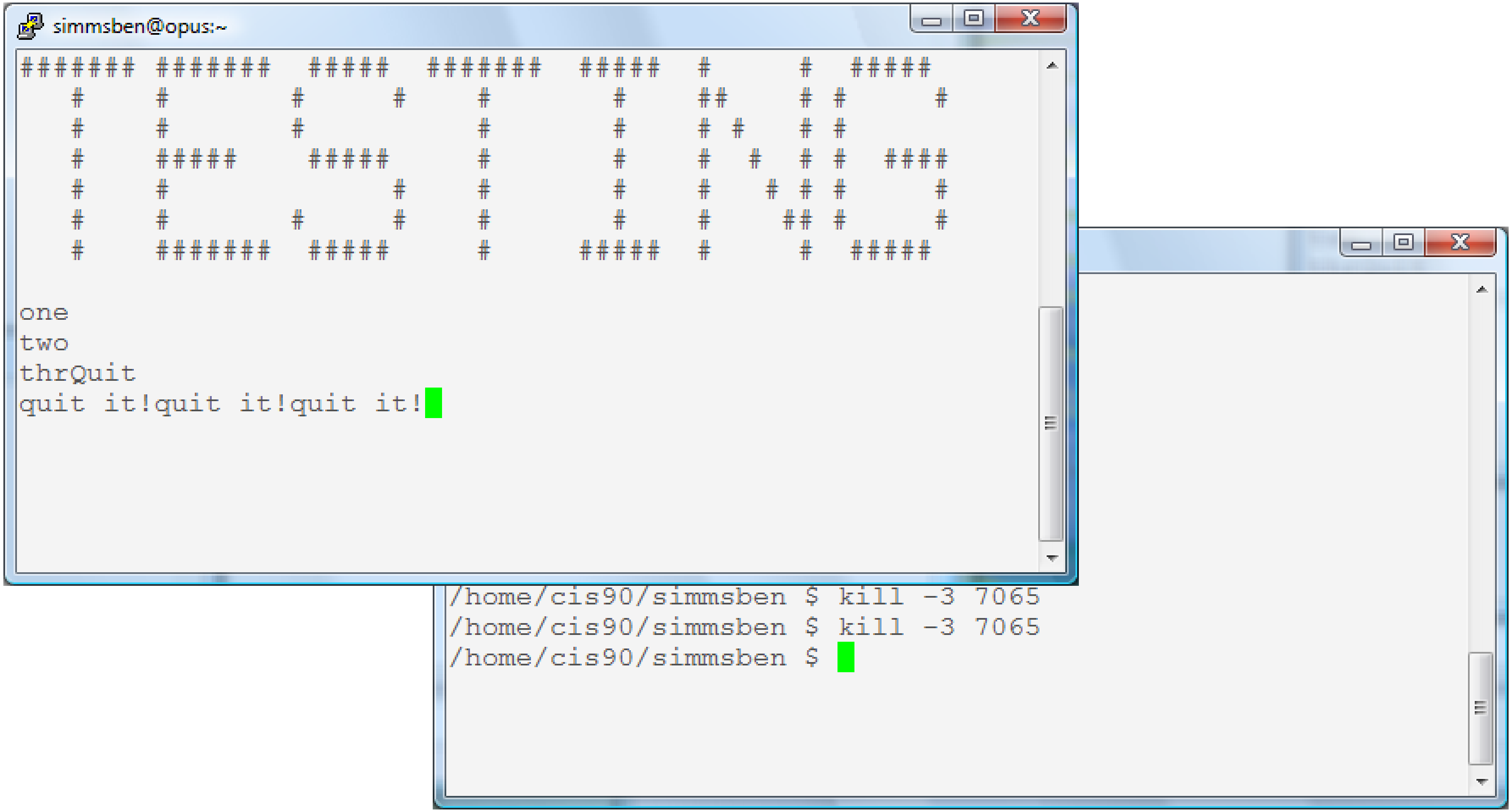
```
####### #######  #####  #######  #####  #     #  #####
   #    #       #     #    #       #    ##    # #     #
   #    #       #          #       #    # #   # #
   #    #####    #####     #       #    #  #  # #  ####
   #    #             #    #       #    #   # # #     #
   #    #       #     #
   #    #######  #####

one
two
thrQuit
quit it!
```

```
/home/cis90/simmsben $ ps -u simmsben
  PID TTY          TIME CMD
 6657 ?        00:00:00 sshd
 6658 pts/1    00:00:00 bash
 7033 ?        00:00:00 sshd
 7034 pts/2    00:00:00 bash
 7065 pts/2    00:00:00 app
 7843 pts/2    00:00:00 sleep
 7844 pts/1    00:00:00 ps
/home/cis90/simmsben $ kill -2 7065
/home/cis90/simmsben $
```

*Benji logs into another Putty session and sends a SIGINT/2 using the* ***kill*** *command …. but nothing happens*

# Signals

## Benji runs app



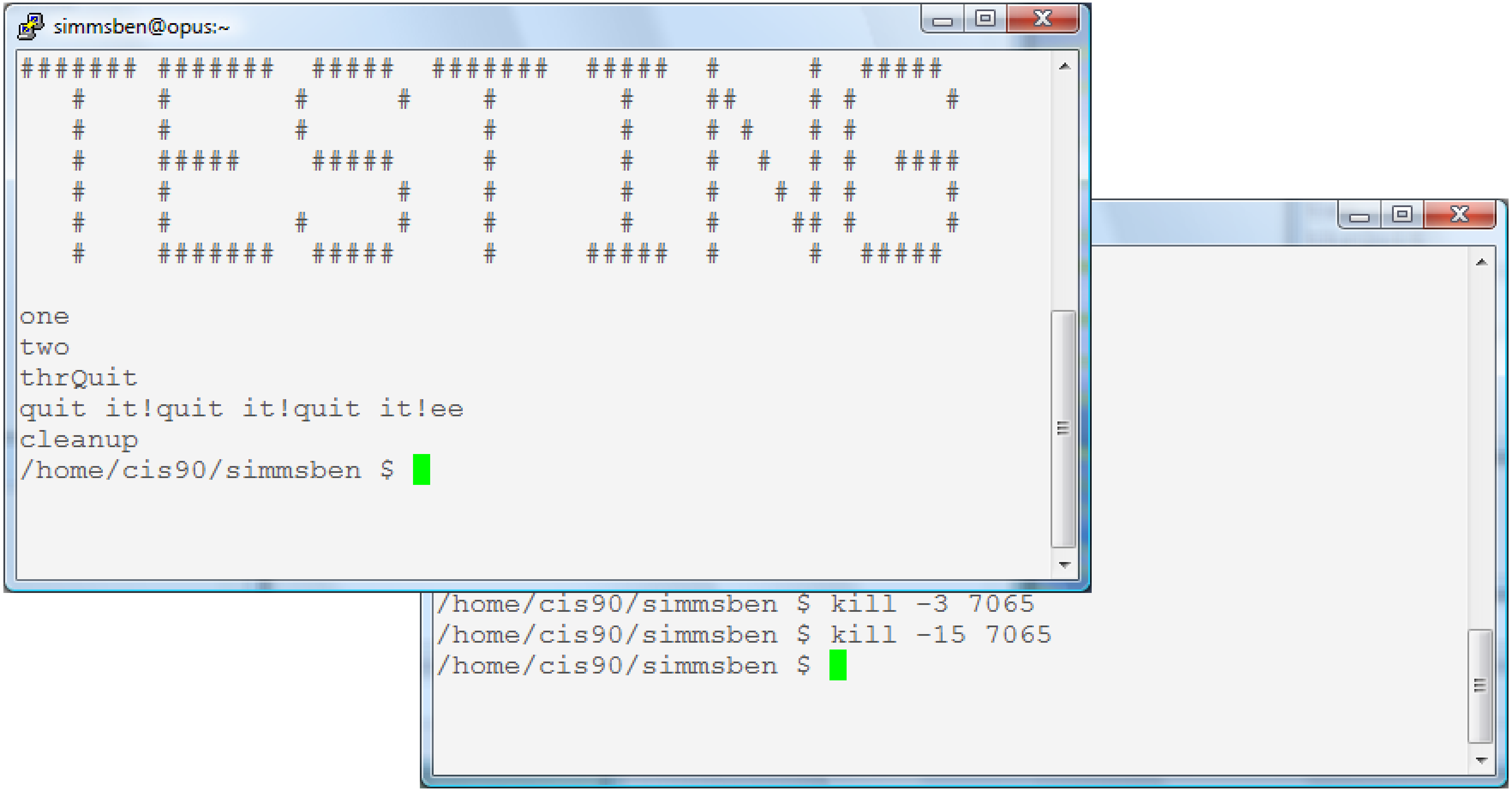
```
####### #######  #####  #######  #####  #     #  #####
   #    #       #     #    #       #    ##    # #     #
   #    #       #          #       #    # #   # #
   #    #####    #####     #       #    #  #  # #  ####
   #    #             #    #       #    #   # # #     #
   #    #       #     #    #       #    #    ## #     #
   #    #######  #####     #     #####  #     #  #####

one
two
thrQuit
quit it!quit it!quit it!
```

```
/home/cis90/simmsben $ kill -3 7065
/home/cis90/simmsben $ kill -3 7065
/home/cis90/simmsben $
```

*Benji ups the anty and sends two SIGQUIT/3's but the app process shrugs them off with "quit it!" messages*

# Signals

## Benji runs app



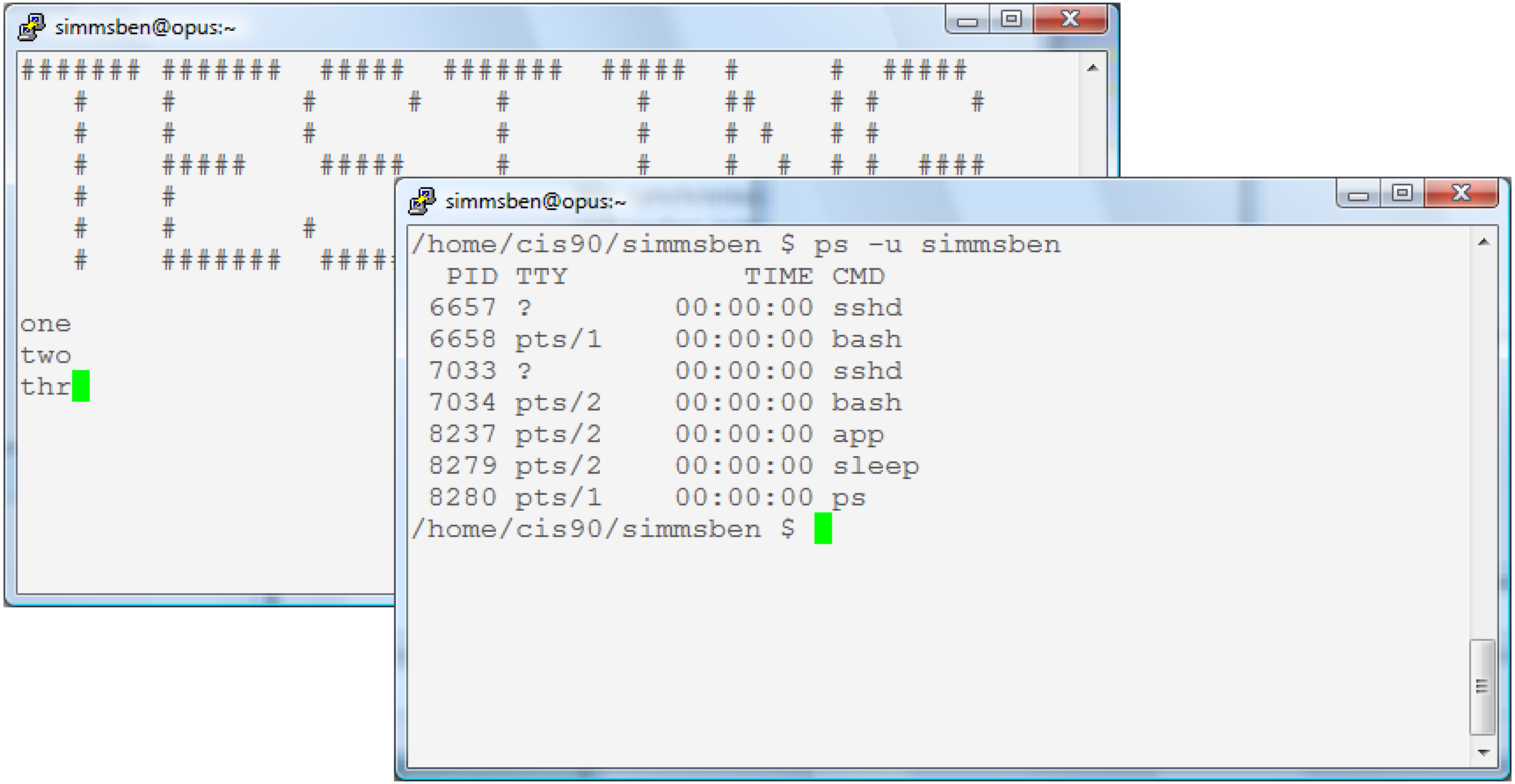
```
####### #######  #####  #######  #####  #     #  #####
   #    #       #     #    #       #    ##    # #     #
   #    #       #          #       #    # #   # #
   #    #####    #####     #       #    #  #  # #  ####
   #    #             #    #       #    #   # # #     #
   #    #       #     #    #       #    #    ## #     #
   #    #######  #####     #     #####  #     #  #####

one
two
thrQuit
quit it!quit it!quit it!ee
cleanup
/home/cis90/simmsben $
```

```
/home/cis90/simmsben $ kill -3 7065
/home/cis90/simmsben $ kill -15 7065
/home/cis90/simmsben $
```

*Benji decides to send a SIGTERM/15 this time and the app process finishes, cleans up and exits*

# Signals

## Benji runs app



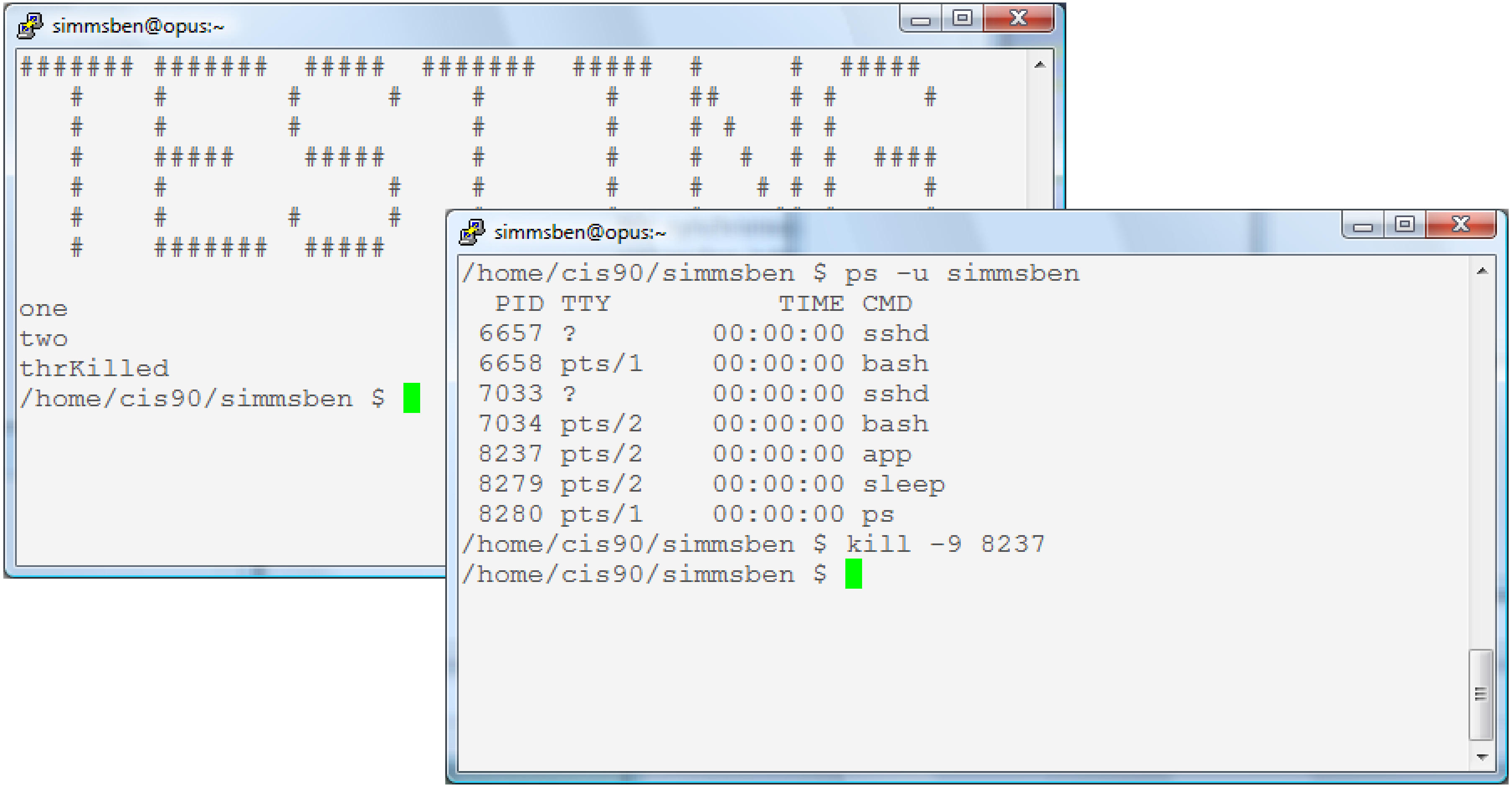
```
####### #######  #####  #######  #####  #     #  #####
   #    #       #     #    #       #    ##    # #     #
   #    #       #          #       #    # #   # #
   #    #####    #####     #       #    #  #  # #  ####
   #    #
   #    #       #
   #    #######  ####

one
two
thr
```

```
/home/cis90/simmsben $ ps -u simmsben
  PID TTY          TIME CMD
 6657 ?        00:00:00 sshd
 6658 pts/1    00:00:00 bash
 7033 ?        00:00:00 sshd
 7034 pts/2    00:00:00 bash
 8237 pts/2    00:00:00 app
 8279 pts/2    00:00:00 sleep
 8280 pts/1    00:00:00 ps
/home/cis90/simmsben $
```

*The same thing happens again another day. This time Benji does not care what happens with app …*

# Signals

## Benji runs app



```
####### #######  #####  #######  #####  #     #  #####
   #    #       #     #    #       #    ##    #  #     #
   #    #       #          #       #    # #   #  #
   #    #####    #####     #       #    #  #  #  #  ####
   #    #             #    #       #    #   # #  #     #
   #    #       #     #
   #    #######  #####

one
two
thrKilled
/home/cis90/simmsben $
```

```
/home/cis90/simmsben $ ps -u simmsben
  PID TTY          TIME CMD
 6657 ?        00:00:00 sshd
 6658 pts/1    00:00:00 bash
 7033 ?        00:00:00 sshd
 7034 pts/2    00:00:00 bash
 8237 pts/2    00:00:00 app
 8279 pts/2    00:00:00 sleep
 8280 pts/1    00:00:00 ps
/home/cis90/simmsben $ kill -9 8237
/home/cis90/simmsben $
```

*So he sends a SIGKILL/9 this time … and app never even sees it coming …. poof … app is gone*

# Signals
## Class Exercise

- Run app
- Try sending it a SIGINT from the keyboard (Ctrl-C)
- Try sending it a SIGQUIT from the keyboard (Ctrl-\)
- Login to another Putty session
  - Use the ps –u $LOGNAME to find the app PID
  - Send it a SIGINT (kill -2 PID)
  - Send it a SIGQUIT (kill -3 PID)
  - Now send either a SIGKILL (9) or SIGTERM (15)

# Load Balancing

# Load Balancing with **at** command

So that the multiprocessing CPU on a UNIX system does not get overloaded, some processes need to be run during low peak hours such as early in the morning or later in the day.

The **at** command reads from **stdin** for a list of commands to run, and begins running them at the time of day specified as the first argument

*example 1*

```
/home/cis90ol/simmsben $ at 10:30pm < script_file
```

*example 2*

```
/home/cis90ol/simmsben $ at 11:59pm
at> cat files.out bigshell > lab08
at> cp lab08 /home/rsimms/cis90/$LOGNAME
at> Ctrl-D
/home/cis90ol/simmsben $
```

*Note: the **Ctrl-D** must be entered as the first character on the last line.*

# **at** command scheduling examples

```
/home/cis90/rodduk $ cat job1
cp bin/myscript bin/myscript.bak
echo "Job 1 - finished, myscript has been backed up" | mail -s "Job 1" rodduk
```

*This job makes a backup of myscript and sends an email when finished*

# **at** command scheduling examples

*This job makes a backup of myscript and sends an email when finished*

```
/home/cis90/rodduk $ cat job1
cp bin/myscript bin/myscript.bak
echo "Job 1 - finished, myscript has been backed up" | mail -s "Job 1" rodduk

/home/cis90/rodduk $ at now + 5 minutes < job1
job 24 at 2008-11-12 12:14
/home/cis90/rodduk $ at now + 2 hours < job1
job 25 at 2008-11-12 14:09
/home/cis90/rodduk $ at teatime < job1
job 26 at 2008-11-12 16:00
/home/cis90/rodduk $ at now + 1 week < job1
job 27 at 2008-11-19 12:10
/home/cis90/rodduk $ at 3:00 12/12/2010 < job1
job 28 at 2008-12-12 03:00
```

*Many ways to specify a future time to run*

# **at** command scheduling examples

*This job makes a backup of myscript and sends an email when finished*

```
/home/cis90/rodduk $ cat job1
cp bin/myscript bin/myscript.bak
echo "Job 1 - finished, myscript has been backed up" | mail -s "Job 1" rodduk

/home/cis90/rodduk $ at now + 5 minutes < job1
job 24 at 2008-11-12 12:14
/home/cis90/rodduk $ at now + 2 hours < job1
job 25 at 2008-11-12 14:09
/home/cis90/rodduk $ at teatime < job1
job 26 at 2008-11-12 16:00
/home/cis90/rodduk $ at now + 1 week < job1
job 27 at 2008-11-19 12:10
/home/cis90/rodduk $ at 3:00 12/12/2010 < job1
job 28 at 2008-12-12 03:00

/home/cis90/rodduk $ atq
25      2008-11-12 14:09 a rodduk
28      2008-12-12 03:00 a rodduk
27      2008-11-19 12:10 a rodduk
26      2008-11-12 16:00 a rodduk
24      2008-11-12 12:14 a rodduk
/home/cis90/rodduk $
```

*Many ways to specify a future time to run*

*Use the **atq** command to show queued jobs*

# **at** command management

```
/home/cis90/rodduk $ jobs

/home/cis90/rodduk $ atq
25      2008-11-12 14:09 a rodduk
28      2008-12-12 03:00 a rodduk
27      2008-11-19 12:10 a rodduk
26      2008-11-12 16:00 a rodduk
24      2008-11-12 12:14 a rodduk
```

*The **jobs** command does not apply here. It lists processes running or suspended in the background.*

*The **atq** command lists jobs queued to run in the futures that were scheduled by at command*

```
/home/cis90/rodduk $ atrm 24
/home/cis90/rodduk $ atq
25      2008-11-12 14:09 a rodduk
28      2008-12-12 03:00 a rodduk
27      2008-11-19 12:10 a rodduk
26      2008-11-12 16:00 a rodduk
/home/cis90/rodduk $
```

*The **atrm** command is used to remove jobs from the queue*

# **at** command error handling

```
/home/cis90/simben $ at now + 1 minute
at> kitty letter
at> <EOT>
job 150 at 2011-04-20 10:47

/home/cis90/simben $ atq
150     2011-04-20 10:47 a simmsben
/home/cis90ol/simmsben $ atq

/home/cis90/simben $ mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/spool/mail/simben": 1 message 1 new
>N  1 simben@Opus.cabril  Wed Apr 20 10:47  16/709   "Output from your job "
& 1
Message 1:
From simben@Opus.cabrillo.edu  Wed Apr 20 10:47:01 2011
Date: Wed, 20 Apr 2011 10:47:01 -0700
From: Benji Simms <simben@Opus.cabrillo.edu>
Subject: Output from your job      150
To: simben@Opus.cabrillo.edu

/bin/bash: line 2: kitty: command not found

&
```

*Oops, specified a non-existent command to run in the future (**kitty** should have been **cat**)*

*Because, you may not be online when the command runs, any error messages are mailed to you.*

# Wrap up

New commands:

| | |
|---|---|
| Ctrl-Z or F | Suspends a foreground process |
| bg | Resumes suspended process |
| & | Runs command in the background |
| fg | Brings background job to foreground |
| jobs | show background jobs |
| kill | Send a signal to a process |
| at | Run job once in the future |
| atq | Show all *at* jobs queued to run |
| atrm | Remove *at* jobs from queue |
| sleep | Sleep for specified amount of time |
| stty | Terminal control |

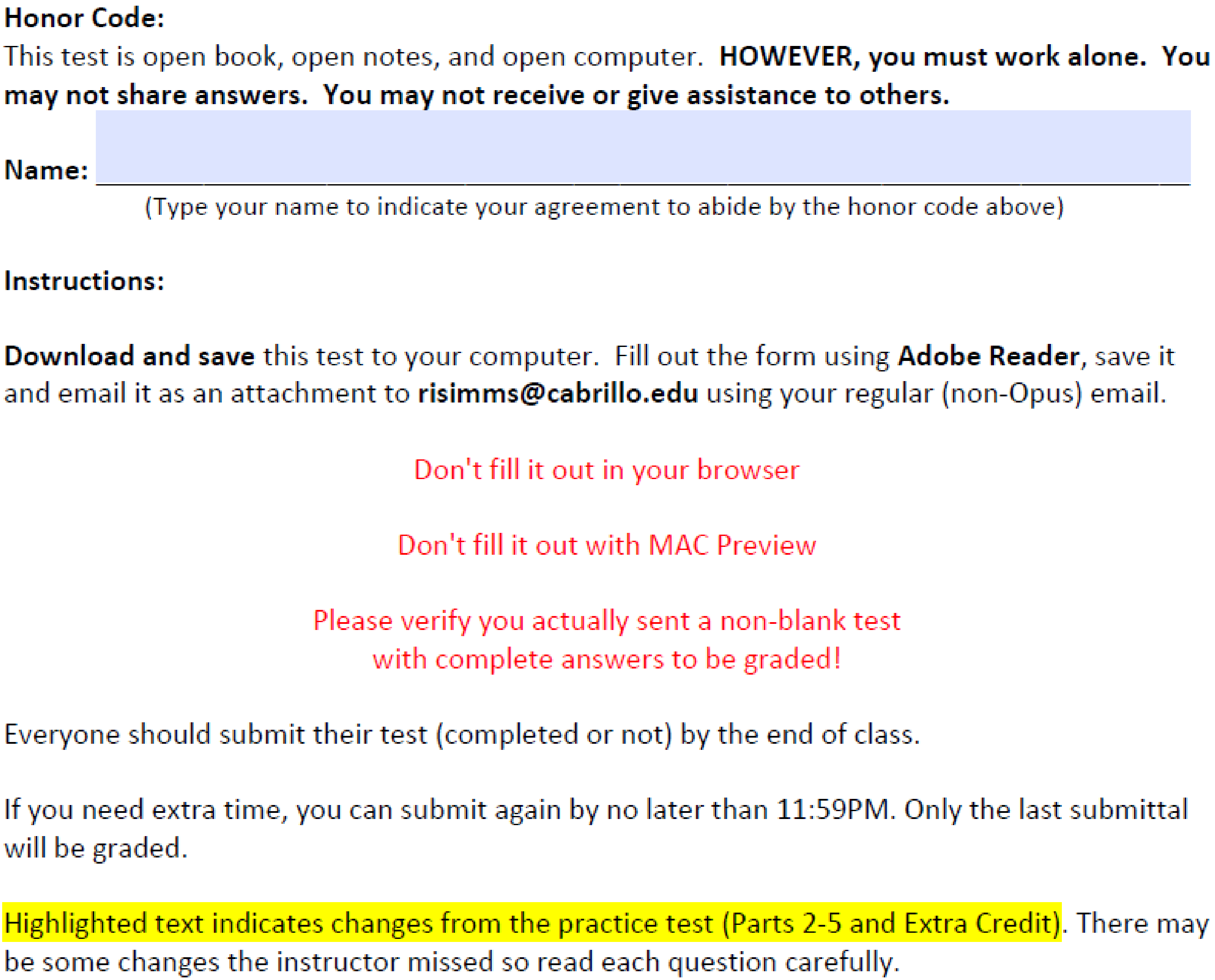
# Next Class

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

Lab 8 due

Quiz #8 questions for next class:

- What command shows the current running processes?
- Name four states a process can be in.
- What is the difference between the fork and exec system calls?



**Honor Code:**

This test is open book, open notes, and open computer. **HOWEVER, you must work alone. You may not share answers. You may not receive or give assistance to others.**

**Name:** ______________________________________________

(Type your name to indicate your agreement to abide by the honor code above)

**Instructions:**

**Download and save** this test to your computer. Fill out the form using **Adobe Reader**, save it and email it as an attachment to **risimms@cabrillo.edu** using your regular (non-Opus) email.

Don't fill it out in your browser

Don't fill it out with MAC Preview

Please verify you actually sent a non-blank test
with complete answers to be graded!

Everyone should submit their test (completed or not) by the end of class.

If you need extra time, you can submit again by no later than 11:59PM. Only the last submittal will be graded.

Highlighted text indicates changes from the practice test (Parts 2-5 and Extra Credit). There may be some changes the instructor missed so read each question carefully.

# The Test

- 10 minute break
- Instructor notes:
  - Power on Sun-Hwa-II or remove /etc/nologin
  - Q10-13 trouble-T2 (sun-hwa-ii)
  - Q13 set permissions on games (sun-hwa-ii)
  - Q14-16 hiderocks T2 (sun-hwa-ii)
  - Q18 randomwords file in depot
  - Q33 hide eggs for Benji (opus)
  - Change permissions on test to enable download
  - Shutdown Sun-Hwa

# Test 2

# Backup