



Lesson Module Status

- Slides – draft
 - Properties - done
 - Flash cards –
 - First minute quiz –
 - Web calendar summary –
 - Web book pages –
 - Commands – done
 - Lab tested and uploaded –
 - Supplies () - na
 - Class PC's – na
 - Supplies – chocolates
-
- Real test uploaded and permissions set – done
 - CCC Confer wall paper – done
-
- Materials uploaded – done
 - Backup headset charged – oops
 - Backup slides, CCC info, handouts on flash drive - done
-
- Check that room headset is charged – done



Dieskau



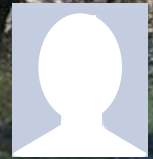
Jonathan



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



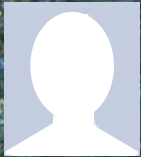
Ana



David



Obie



Dave



Cole



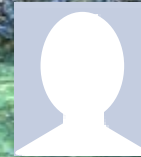
Corey



Nancy



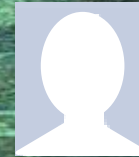
Ryan



Elia



Tasha



Darren



Scott



Devin



Everett



Juan



Raven



Rogan



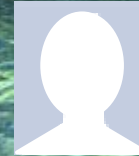
Mike



Mook



Melissa



Cameron



Jose



Jeff



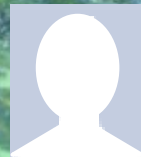
Matt



Kenneth



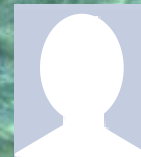
Ousmane



Ian



Solomon



Henry



Matthew



Mason



Chan

First Minute Quiz

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

No Quiz Today

But we do have a test!

email answers to: risimms@cabrillo.edu
(within the first few minutes of class)



- [] Has the phone bridge been added?
- [] Is recording on?
- [] Does the phone bridge have the mike?
- [] Share lesson slides, puttyx3, Chrome,
- [] Start 10 VMs on VLab
- [] Disable spelling on PowerPoint

UNIX Processes

Objectives	Agenda
<ul style="list-style-type: none">• Know the process life cycle• Interpret ps command output• Run or schedule jobs to run in the background• Send signals to processes• Configure process load balancing	<ul style="list-style-type: none">• Questions from last week• Housekeeping• Process definition• Process lifecycle• Process information• Job control• Signals• Load balancing• Wrap up• Test #2

?'S

Previous material and assignment

1. Questions on previous material?

- File management (Lesson 6)
- Permissions (Lesson 7)
- Input/output (Lesson 8)
- Labs
- Practice test

2. Questions regarding the test today?

- Test will start during the last hour of class.
- Should take about 45-60 minutes
- If you wish, you can keep working on it till midnight.
- You must do all the work on the test by yourself and not ask or give help to others regarding any of the test questions.

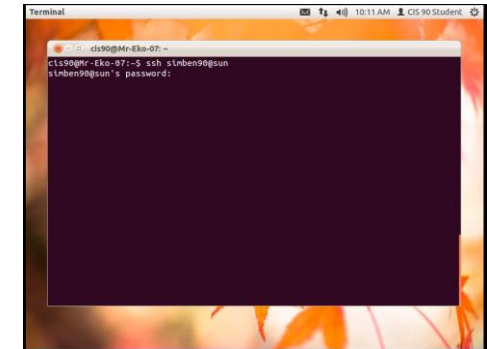
Getting to Sun

sun.cislab.net

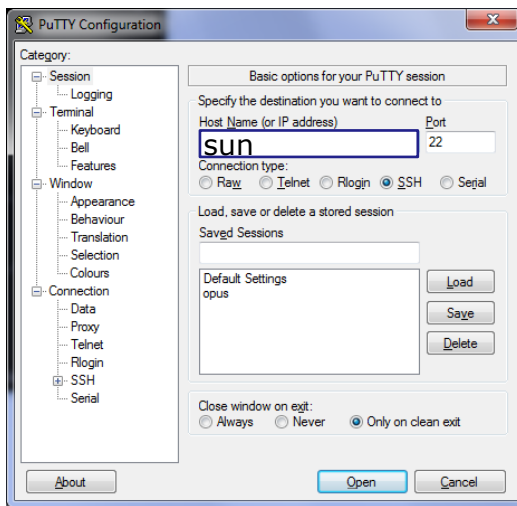
*Use your VLab
credentials to
log into Sun*



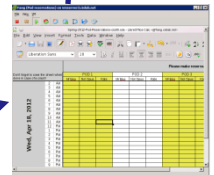
ssh username@sun



*Sun is on a private
network and not
available directly via
the Internet*



*From the classroom
or CIS Lab PCs, just
putty to sun*



*From off campus, use VLab to
reserve one of the CIS 90 VMs
and from the VM **ssh to sun***

umask

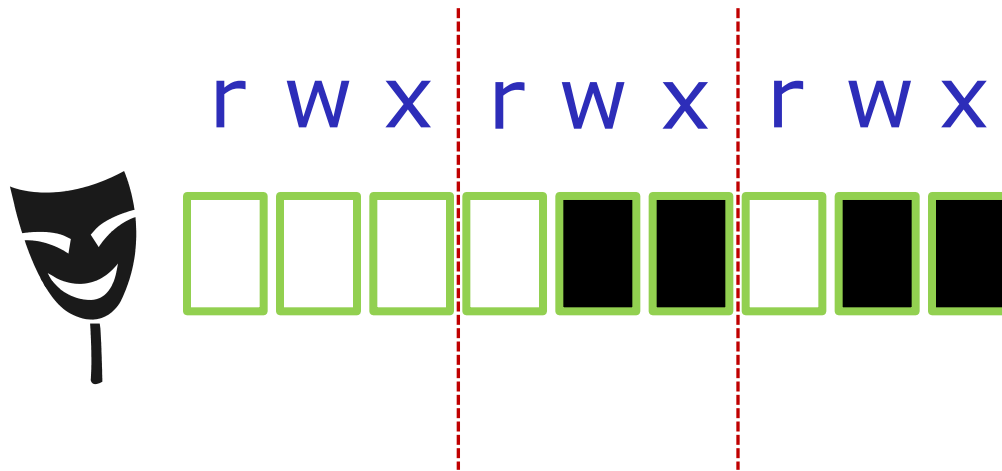
umask

- Use the **umask** command to specify the permissions you want stripped from future new files and directories

For example: **umask 022**

- Sets the umask value to 022 or --- -w- -w-
 - This will strip write permissions for group and others on any new files and directories created in the future
 - 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 file being copied

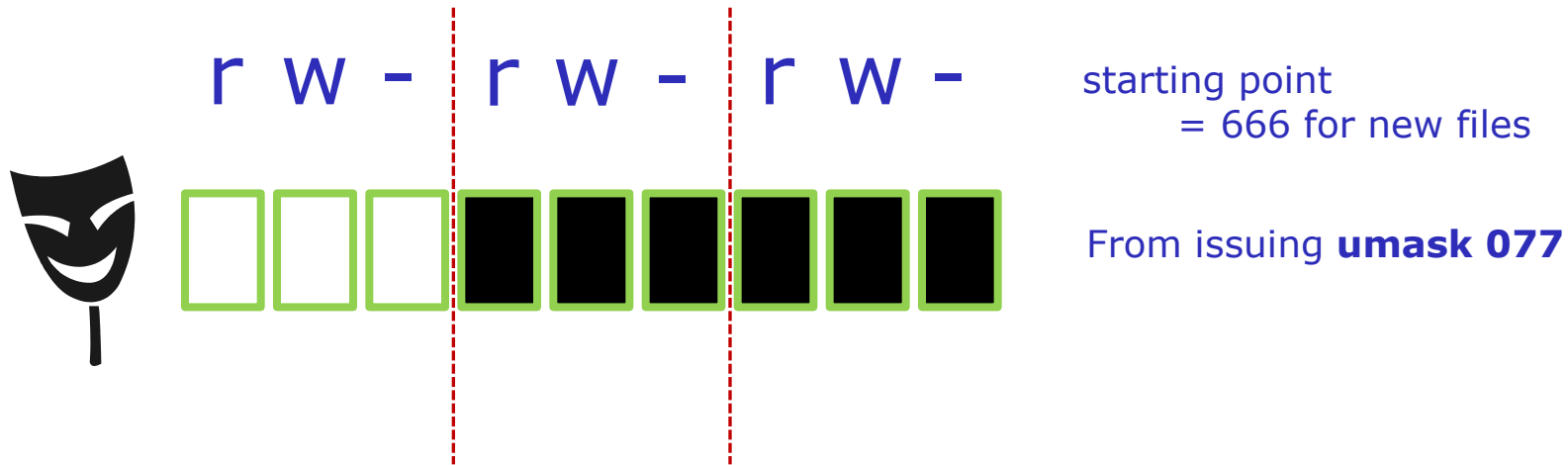
With a umask of 033 what permissions would a newly created directory have?



starting point
= 777 for new directories

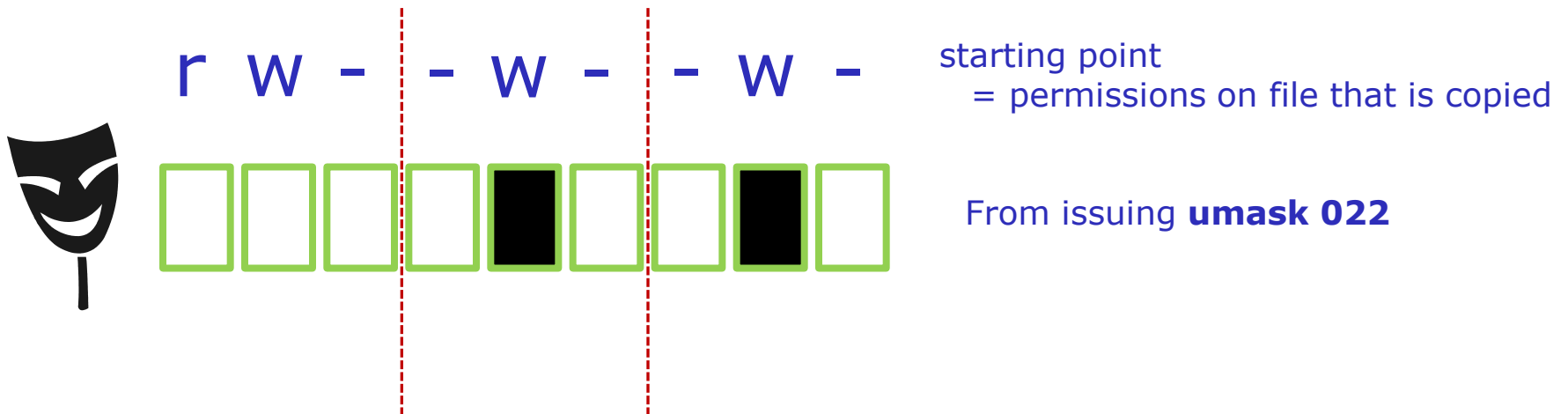
From issuing **umask 033**

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



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

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



FYI

shell debugging and {}

FYI **set -x**, **set +x**

```
/home/cis90/roddyduk $ set -x      Enable shell debugging
```

```
+ set -x
```

```
++ echo -ne '\033]0;roddyduk@opus:~'
```

```
/home/cis90/roddyduk $ type /bin/pi*
```

```
+ type /bin/ping /bin/ping6
```

```
/bin/ping is /bin/ping
```

```
/bin/ping6 is /bin/ping6
```

```
++ echo -ne '\033]0;roddyduk@opus:~'
```

*Shows how bash
expands pathnames*



```
/home/cis90/roddyduk $ 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;roddyduk@opus:~'
```

```
/home/cis90/roddyduk $ set +x      Disable shell debugging
```

```
+ set +x
```

```
/home/cis90/roddyduk $
```

shows what arguments are actually passed to the command when it's loaded

FYI **set -x, set +x**

```
/home/cis90/roddyduk $ set -x      Enable shell debugging
```

```
+ set -x
```

```
++ echo -ne '\033]0;roddyduk@opus:~'
```

```
/home/cis90/roddyduk $ find . -name '$LOGNAME'
```

```
+ find . -name '$LOGNAME'
```

```
find: ./Hidden: Permission denied
```

```
find: ./testdir: Permission denied
```

```
++ echo -ne '\033]0;roddyduk@opus:~'
```

*Shows how quoted
text strings get
handled for variables*

```
/home/cis90/roddyduk $ find . -name "$LOGNAME"
```

```
+ find . -name roddyduk
```

```
find: ./Hidden: Permission denied
```

```
./roddyduk
```

```
find: ./testdir: Permission denied
```

```
++ echo -ne '\033]0;roddyduk@opus:~'
```

```
/home/cis90/roddyduk $ set +x      Disable shell debugging
```

```
+ set +x
```

```
/home/cis90/roddyduk $
```

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

FYI **set -x**, **set +x**

```
/home/cis90/simben $ set -x Enable shell debugging
```

```
+ set -x
```

```
++ echo -ne '\033]0;simben90@opus:~'
```

```
/home/cis90/simben $ find . -name *.egg
```

```
+ find . -name 1991.egg
```

```
./1991.egg
```

```
++ echo -ne '\033]0;simben90@opus:~'
```

```
/home/cis90/simben $ find . -name "*.egg"
```

```
+ find . -name '*.egg'
```

```
./1991.egg
```

```
./basket/.1993.egg
```

```
< snipped >
```

```
./basket/.1969.egg
```

```
./basket/.1972.egg
```

```
++ echo -ne '\033]0;simben90@opus:~'
```

*Shows how quoted
text strings get
handled*

```
/home/cis90/simben $ set +x Disable shell debugging
```

```
+ set +x
```

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

Shows filename expansion metacharacters without quotes are expanded and those in quotes are not

FYI using {}

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

Managing your grade

Points gone by

- 7 quizzes - 21 points
 - 1 tests - 30 points
 - 2 forum periods - 40 points
 - 7 labs - 210 points
- } 301 points


Points yet to earn

- 3 quizzes - 9 points
 - 2 tests - 60 points
 - 2 forum periods - 40 points
 - 3 labs - 90 points
 - 1 final project - 60 points
- } 259 points
- Plus extra credit - up to 90 points

Managing your grade

adaldrida: 99% (300 of 301 points)
 alatar: 93% (281 of 301 points)
 amroth: 98% (297 of 301 points)
 arador: 90% (272 of 301 points)
 aragorn: 99% (298 of 301 points)
 arwen: 105% (319 of 301 points)
 carc: 60% (181 of 301 points)
 celebrian: 65% (198 of 301 points)
 cirdan: 104% (315 of 301 points)
 denethor: 30% (93 of 301 points)
 dori: 88% (267 of 301 points)
 dwalin: 80% (242 of 301 points)
 elrond: 106% (320 of 301 points)
 eomer: 85% (258 of 301 points)
 eowyn: 94% (283 of 301 points)
 frodo: 104% (316 of 301 points)
 gwaihir: 86% (259 of 301 points)
 haldir: 86% (260 of 301 points)
 ioreth: 27% (82 of 301 points)
 khamul: 95% (286 of 301 points)
 nazgul: 4% (14 of 301 points)
 nessa: 80% (242 of 301 points)
 orome: 101% (307 of 301 points)
 pippin: 65% (197 of 301 points)
 quickbeam: 101% (306 of 301 points)
 samwise: 50% (151 of 301 points)
 shadowfax: 105% (319 of 301 points)
 strider: 106% (320 of 301 points)
 theoden: 58% (177 of 301 points)
 treebeard: 100% (302 of 301 points)
 tulkas: 71% (215 of 301 points)
 ulmo: 103% (311 of 301 points)
 varda: 8% (27 of 301 points)

*As of April 15, 2012
 (using Jesse's
 checkgrades script)*



Rich's Cabrillo College CIS Classes
CIS 90 Grades

Home | Resources | Forums | CIS Lab | CTE

Login
 Homeworks
 Admin

CIS 90
 Previous Classes

60 days 18 term
 end!

Cabrillo College
 Web Advisor
 Math 101
 Comments and Files
 Accessing VLab
 R2P Dennis Ritchie

CIS 90 (Spring 2012) Grades
 Course Home | Calendar

Points can be earned from the following activities:

- 5% - Quizzes
- 16% - Tests
- 14% - Help forum participation
- 54% - Lab assignments
- 11% - Final project

How your grade is determined:
 A student can earn up to 560 total points during the activities listed above. The course grade is based on the number of points earned.

Percentage	Total Points	Letter Grade/Pass/Fail
90% or higher (304 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

For some flexibility, personal preferences or family emergencies there is an additional 50 points available of **extra credit** activities.

Choice of Grade or Pass/No Pass
 You indicate your grading choice on the Student Survey form passed out during the first class. You can verify your grading choice selection on the table below. Contact the instructor by email with any questions or to request a change in grading choice.

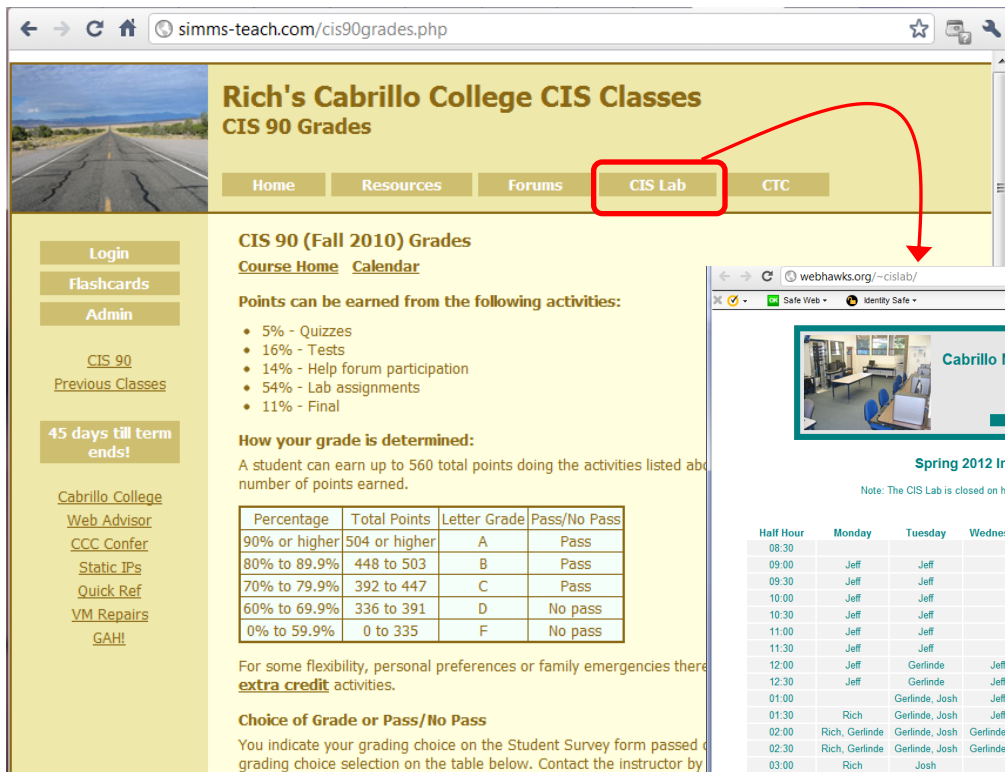
Recommendations
 The instructor may provide letters of recommendation upon request. When writing a recommendation the instructor will include both graded and non-graded areas of performance. Non-graded performance areas may include teamwork, helping others, quality, planning & organization skills, communication, documentation, motivation, and the desire to go above and beyond expectations. The forum is an excellent way to demonstrate teamwork and communication skills.

Current Progress

Code	Grading	Choice	Quizzes & Tests	Forum	Lab	Extra	Total	Grade
Max Points	3	3	3	3	3	3	30	30
adaldrida	3	1	1	2	3	3	28	28
alatar	3	1	1	2	3	3	28	28
amroth	3	1	1	2	3	3	28	28
arador	3	1	1	2	3	3	28	28
aragorn	3	1	1	2	3	3	28	28
arwen	3	1	1	2	3	3	28	28
carc	3	1	1	2	3	3	28	28
celebrian	3	1	1	2	3	3	28	28
cirdan	3	1	1	2	3	3	28	28
denethor	3	1	1	2	3	3	28	28
dori	3	1	1	2	3	3	28	28
dwalin	3	1	1	2	3	3	28	28
elrond	3	1	1	2	3	3	28	28
eomer	3	1	1	2	3	3	28	28
eowyn	3	1	1	2	3	3	28	28
frodo	3	1	1	2	3	3	28	28
gwaihir	3	1	1	2	3	3	28	28
haldir	3	1	1	2	3	3	28	28
ioreth	3	1	1	2	3	3	28	28
khamul	3	1	1	2	3	3	28	28
nessa	3	1	1	2	3	3	28	28
orome	3	1	1	2	3	3	28	28
pippin	3	1	1	2	3	3	28	28
quickbeam	3	1	1	2	3	3	28	28
samwise	3	1	1	2	3	3	28	28
shadowfax	3	1	1	2	3	3	28	28
strider	3	1	1	2	3	3	28	28
theoden	3	1	1	2	3	3	28	28
treebeard	3	1	1	2	3	3	28	28
tulkas	3	1	1	2	3	3	28	28
ulmo	3	1	1	2	3	3	28	28
varda	3	1	1	2	3	3	28	28

Math | Sitemap | WSC | WSC | Credits | Earth

Managing your grade Getting extra help for CIS 90



Rich's Cabrillo College CIS Classes
CIS 90 Grades

Home Resources Forums **CIS Lab** CTC

CIS 90 (Fall 2010) Grades
[Course Home](#) [Calendar](#)

Points can be earned from the following activities:

- 5% - Quizzes
- 16% - Tests
- 14% - Help forum participation
- 54% - Lab assignments
- 11% - Final

How your grade is determined:
A student can earn up to 560 total points doing the activities listed above based on the number of points earned.


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

For some flexibility, personal preferences or family emergencies there are **extra credit** activities.

Choice of Grade or Pass/No Pass
You indicate your grading choice on the Student Survey form passed out at the beginning of the semester. Contact the instructor by email for more information.

[Login](#)
[Flashcards](#)
[Admin](#)
[CIS 90](#)
[Previous Classes](#)
45 days till term ends!
[Cabrillo College](#)
[Web Advisor](#)
[CCC Confer](#)
[Static IPs](#)
[Quick Ref](#)
[VM Repairs](#)
[GAH!](#)

Come by the lab and get help from instructors and student assistants



Cabrillo Network & Systems Technology Lab
Aptos Campus

Home Resources **NETLAB** Location

Spring 2012 Instructor and Lab Assistant Hours
Note: The CIS Lab is closed on holidays and spring break (Feb 10, Feb 20, Apr 9-14, May 28)

Half Hour	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
08:30					closed	closed	closed
09:00	Jeff	Jeff					closed
09:30	Jeff	Jeff					closed
10:00	Jeff	Jeff					closed
10:30	Jeff	Jeff					closed
11:00	Jeff	Jeff					closed
11:30	Jeff	Jeff					closed
12:00	Jeff	Gerlinde	Jeff	Gerlinde	closed		
12:30	Jeff	Gerlinde	Jeff	Gerlinde	closed		
01:00		Gerlinde, Josh	Jeff	Gerlinde, Josh	closed		
01:30	Rich	Gerlinde, Josh	Jeff	Gerlinde, Josh	closed		
02:00	Rich, Gerlinde	Gerlinde, Josh	Gerlinde, Jeff	Gerlinde, Josh, Chelsea, Gary	closed		
02:30	Rich, Gerlinde	Gerlinde, Josh	Gerlinde, Jeff	Josh, Chelsea, Gary	closed		
03:00	Rich	Josh		Josh, Chelsea, Gary	closed		
03:30	Rich	Josh		Josh, Chelsea, Gary	closed		
04:00		Josh	Chelsea	Josh, Chelsea	closed	closed	closed
04:30		Josh	Chelsea	Josh, Chelsea	closed	closed	closed
05:00		Josh	Chelsea	Josh, Chelsea	closed	closed	closed
05:30			Chelsea	Josh, Chelsea	closed	closed	closed
06:00				Josh	closed	closed	closed
06:30					closed	closed	closed
07:00					closed	closed	closed
07:30					closed	closed	closed
08:00					closed	closed	closed
08:30					closed	closed	closed
09:00					closed	closed	closed

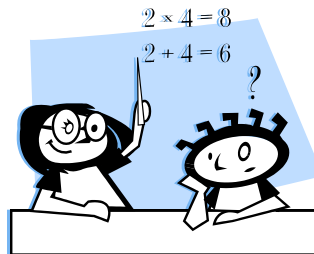
Gerlinde=Gerlinde Brady, Jeff=Jeff Watson, Jim=Jim Griffin, Rich=Rich Simms

W3C XHTML 1.0 W3C CSS

Managing your grade

Getting extra help for CIS 90

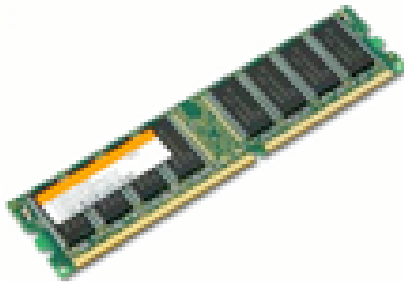
- Rich's Office Hours Wed 12:10-1:00 PM or TBA
- Ask questions on the Forum at:
<http://opus.cabrillo.edu/forum/>



Process Definition

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.*



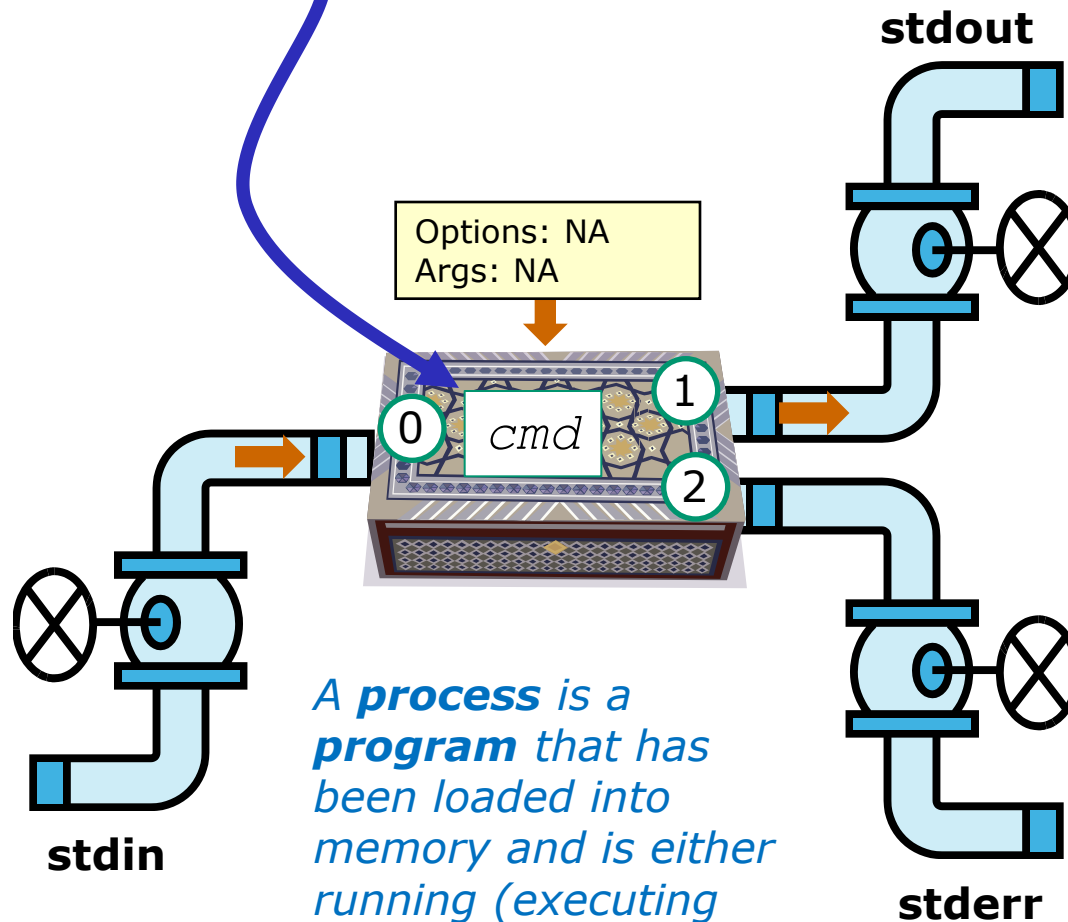
RAM (Random Access Memory) contains the instructions



The CPU executes the instructions in RAM

Program to process

```
/home/cis90/roddyduk $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

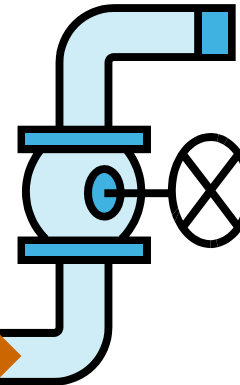
```
/home/cis90/roddyduk $ sort
duke
benji
star
homer
benji
duke
homer
star
/home/cis90/roddyduk $
```



Options: NA
Args: NA



stdout



/dev/pts/0

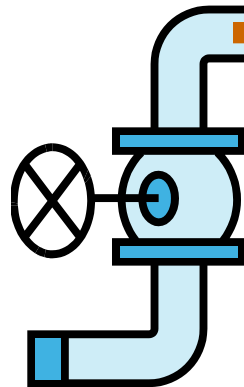


benji
duke
homer
star

/dev/pts/0

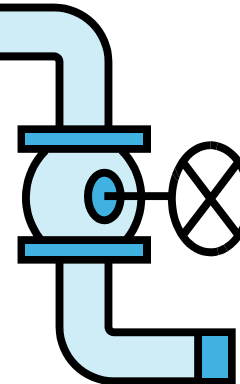


duke
benji
star
homer



stdin

*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.*



stderr

A simple example:

```
CODE
void function1() {
    int A = 10;
    A += 66;
}

compiles to...
function1:
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.

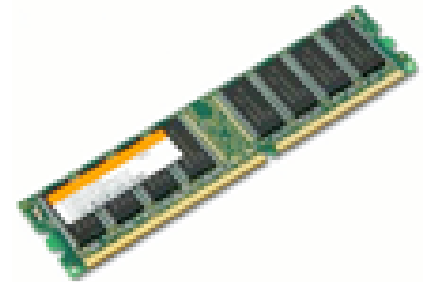
Example program to process: sort command

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

```
[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
```

```
00000000: 7f45 4c46 0101 0100 0000 0000 0000 0000  .ELF.....
00000100: 0200 0300 0100 0000 e093 0408 3400 0000  .....4...
00000200: 2cdb 0000 0000 0000 3400 2000 0800 2800  ,.....4. ...(.
00000300: 1f00 1e00 0600 0000 3400 0000 3480 0408  .....4...4...
00000400: 3480 0408 0001 0000 0001 0000 0500 0000  4.....
00000500: 0400 0000 0300 0000 3401 0000 3481 0408  .....4...4...
00000600: 3481 0408 1300 0000 1300 0000 0400 0000  4.....
00000700: 0100 0000 0100 0000 0000 0000 0080 0408  .....
00000800: 0080 0408 caca 0000 caca 0000 0500 0000  .....
00000900: 0010 0000 0100 0000 00d0 0000 0050 0508  .....P..
00000a00: 0050 0508 9404 0000 e80a 0000 0600 0000  .P.....
00000b00: 0010 0000 0200 0000 a0d1 0000 a051 0508  .....Q..
```

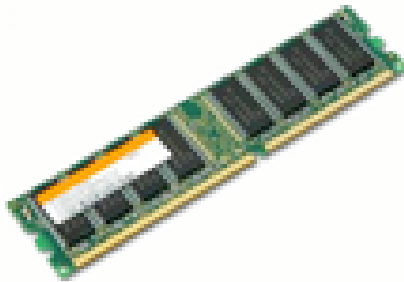


*A command like **sort** is a **program** when it is stored on the drive. It is a **process** when it is copied to memory by the kernel and either running or waiting to run by the CPU*



One more time ... this is important!

*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.*



RAM (Random Access Memory) contains the instructions

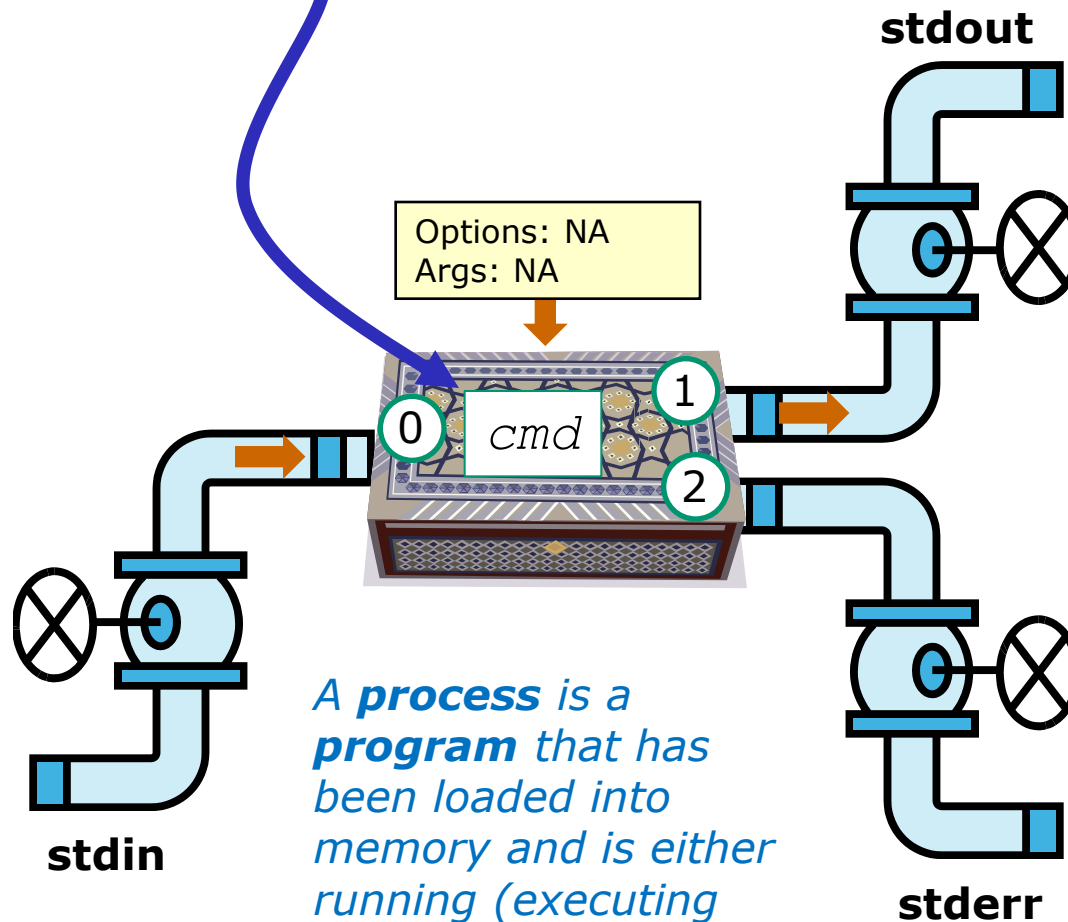


The CPU executes the instructions in RAM

Process Life Cycle

Program to process

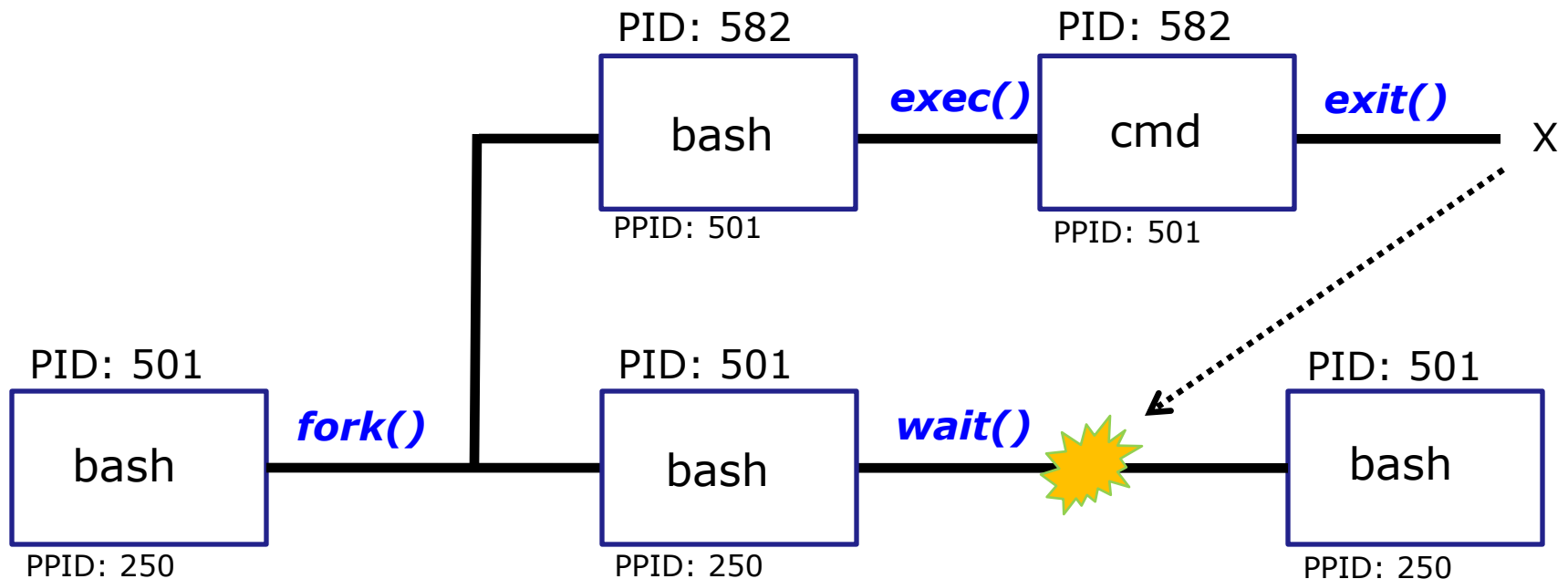
```
/home/cis90/roddyduk $cmd
```



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

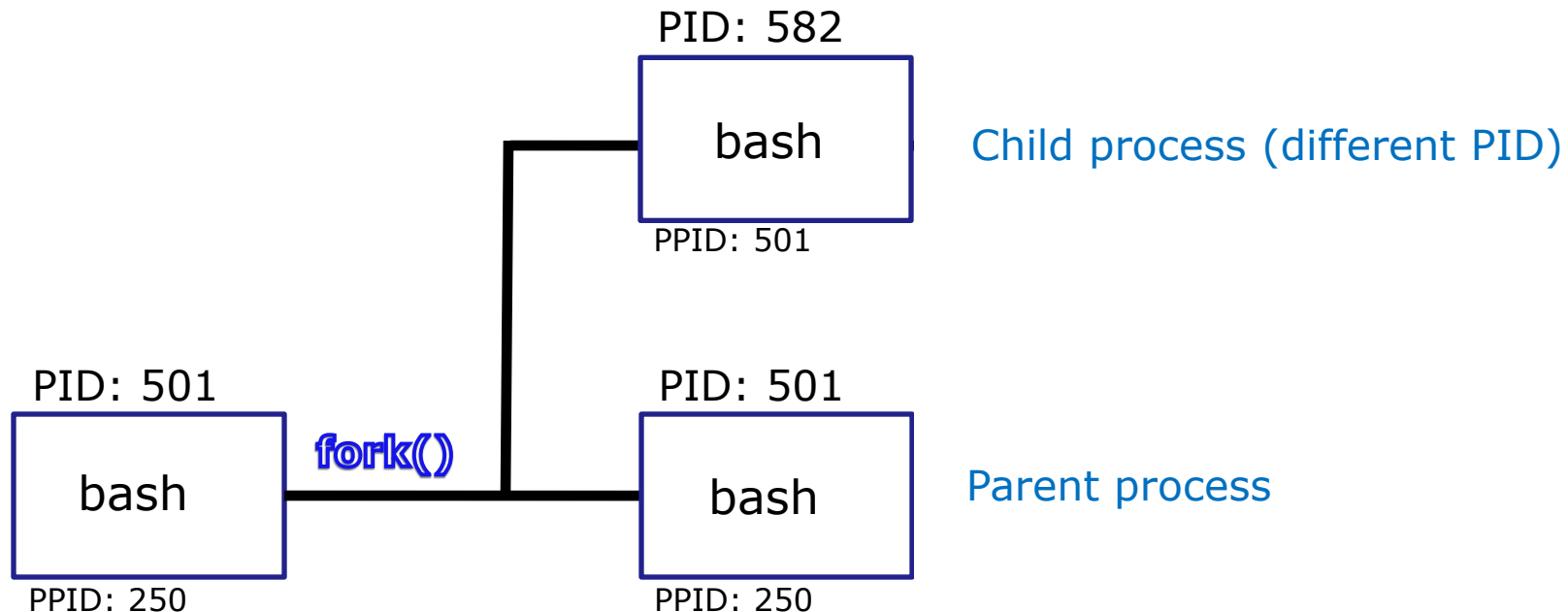
Process Lifecycle

Note: This diagram shows a generic command "cmd" being loaded and run by a user using the bash shell.



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

Process Lifecycle

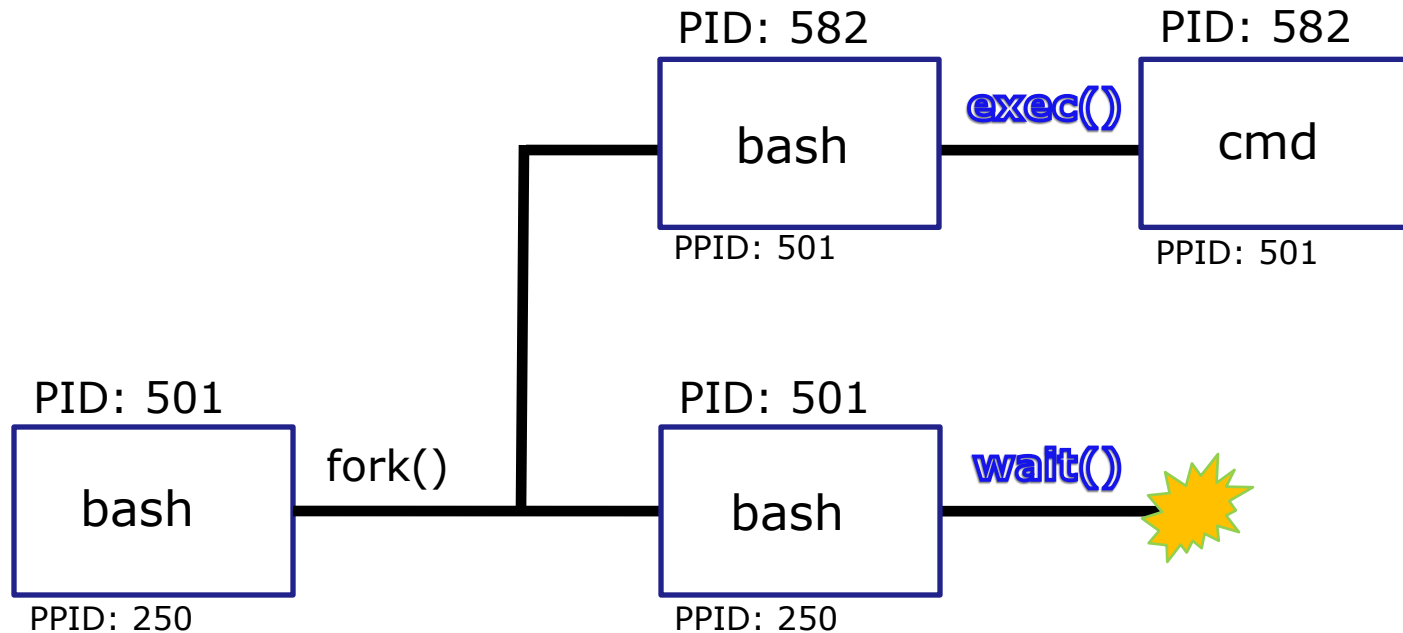


1) When a program is loaded into memory a new process must be created.

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.

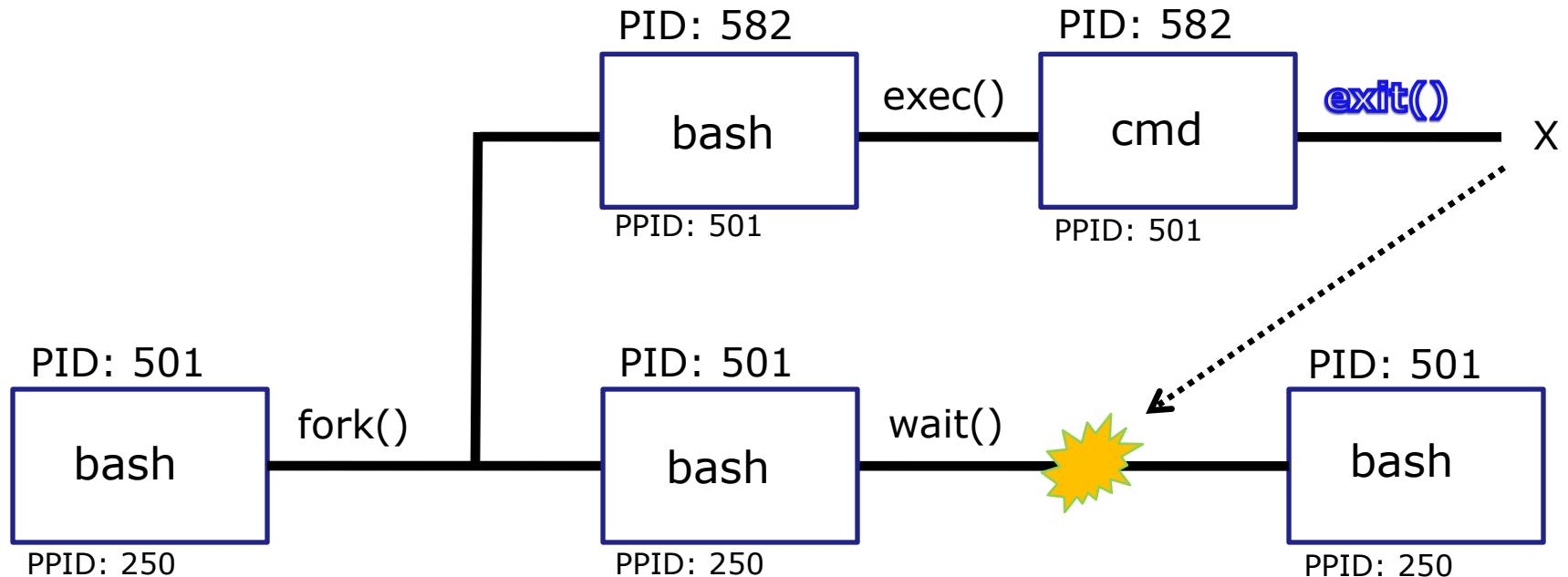
Process Lifecycle



2) 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.

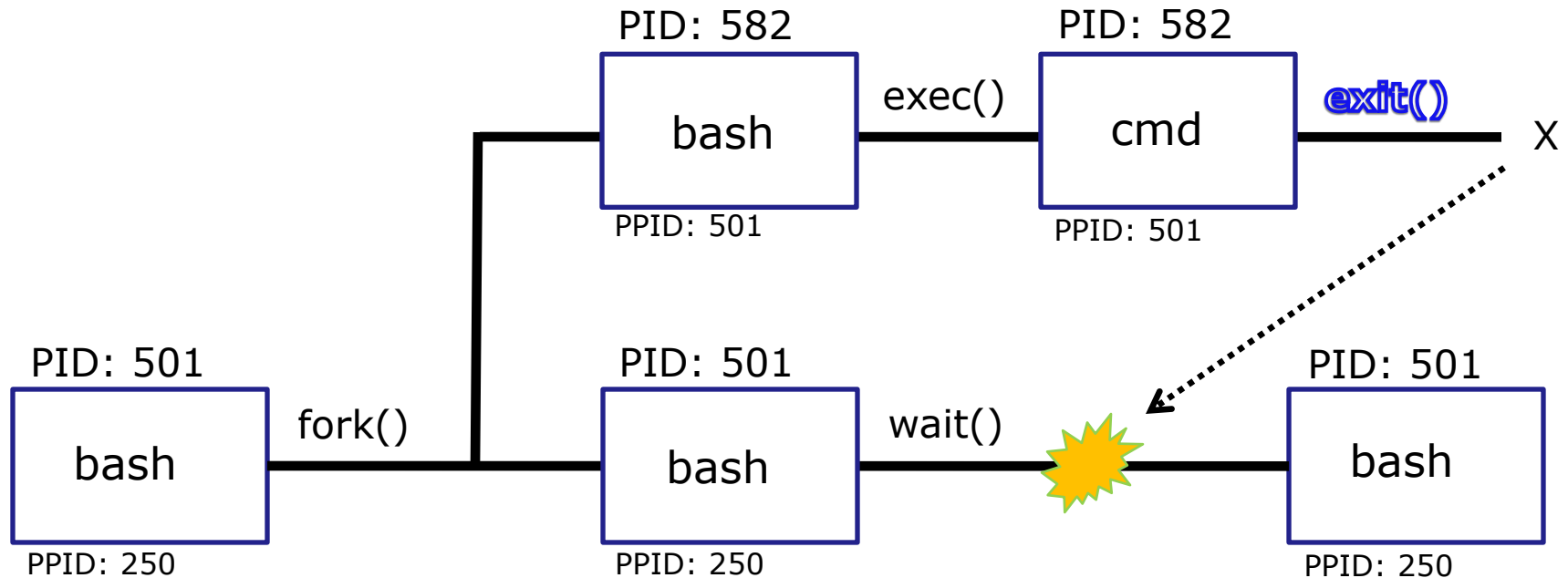
Process Lifecycle



3) 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 and 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



3) 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



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

Process Information

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

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

Process Information

```
[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 started my terminal session. ps is showing because it is running as this output is printed.

Process Information

```
[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 ~]$
```

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

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

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

Process Information

Use -l to show additional 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
```

Running or sleeping

 User ID

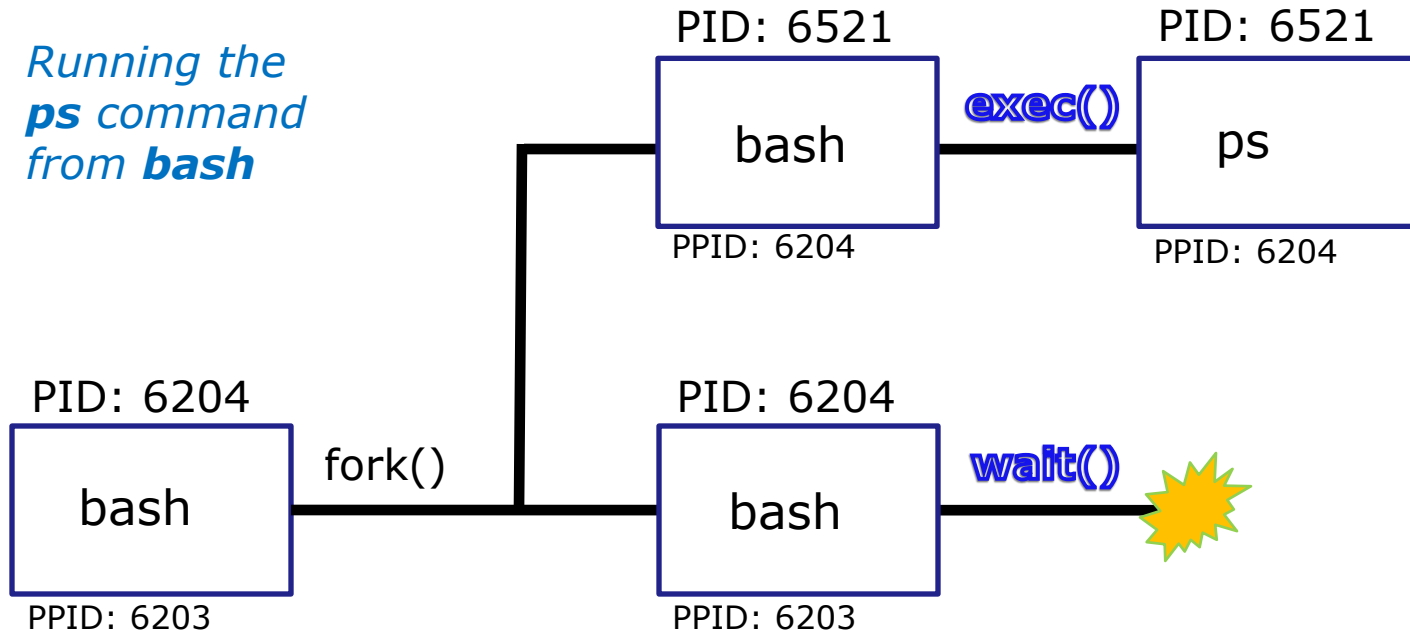
 Process ID

 Parent Process ID

 Size in 1K blocks

Process Lifecycle

Running the
ps command
from **bash**



```
[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

2) 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 **child** process (**ps**) runs (status=R)

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

Process Information

```
[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 every
process running*

Process Information

```

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 pcsd
root     2472       1    0 Sep10 ?          00:00:00 /usr/bin/hidd --server
root     2493       1    0 Sep10 ?          00:00:02 automount

```

Process Information

```

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

```

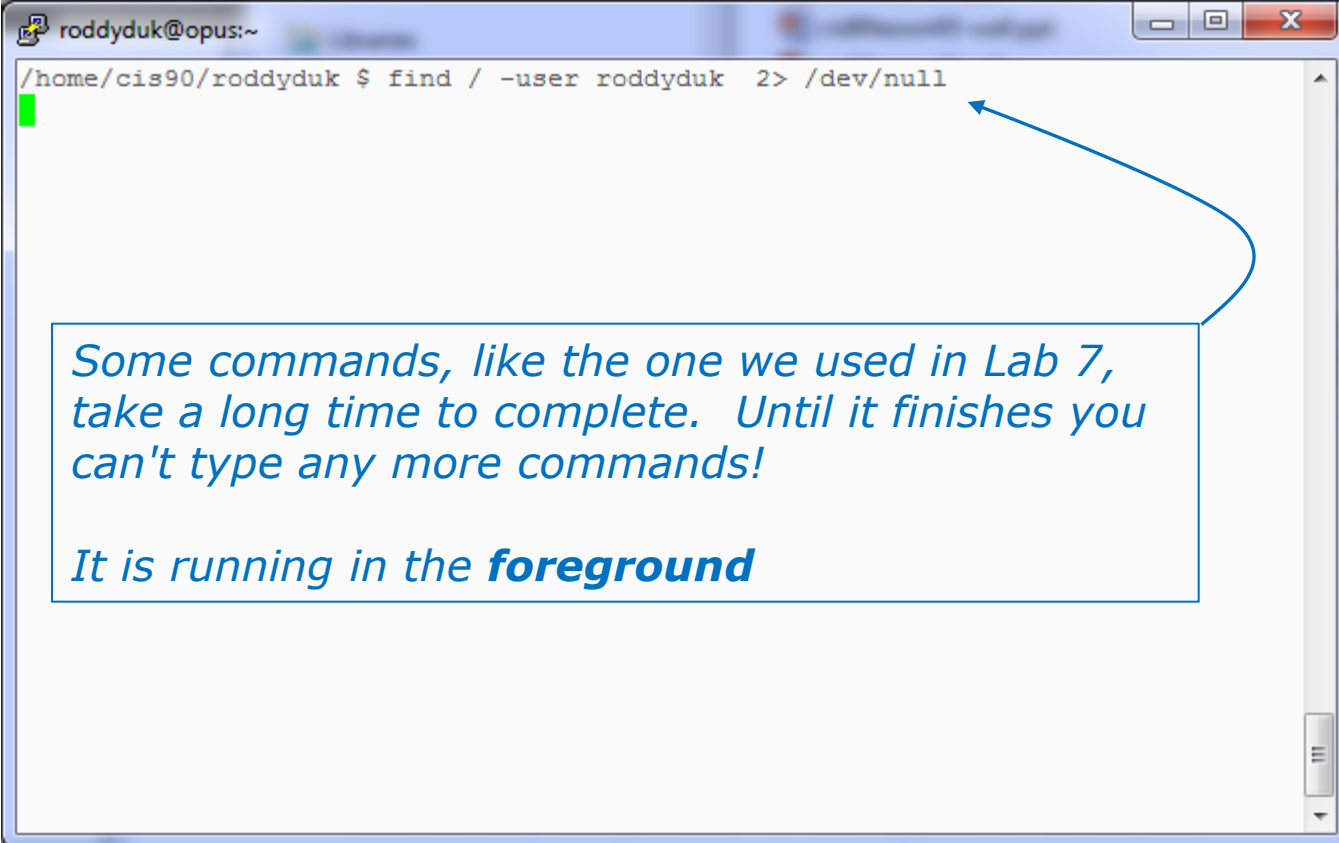
Process Information

```

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



A terminal window titled 'rododyduk@opus:~' shows the command `/home/cis90/rododyduk $ find / -user rododyduk 2> /dev/null` entered. A blue arrow points from a text box to the command line. The text box contains the following text:

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.

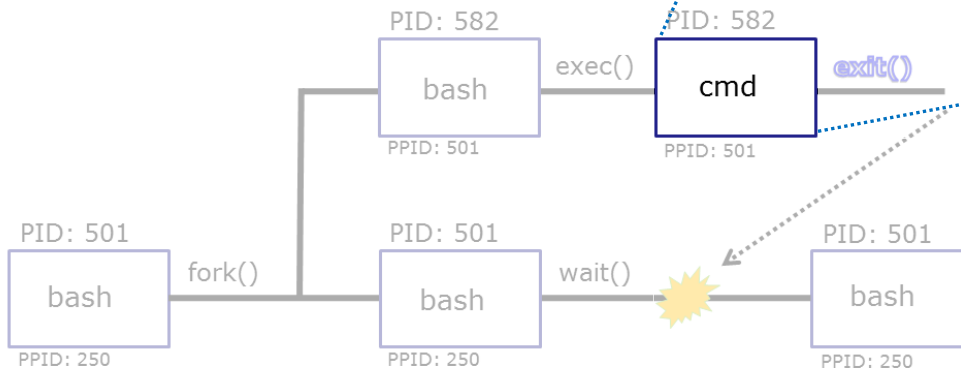
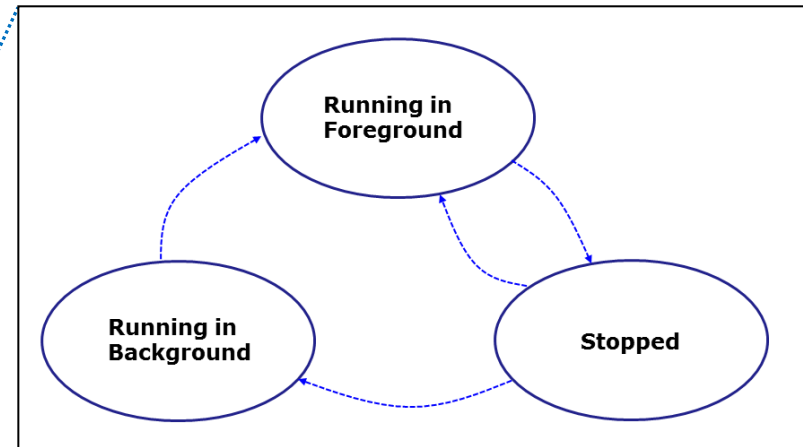
Background Processes

- 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

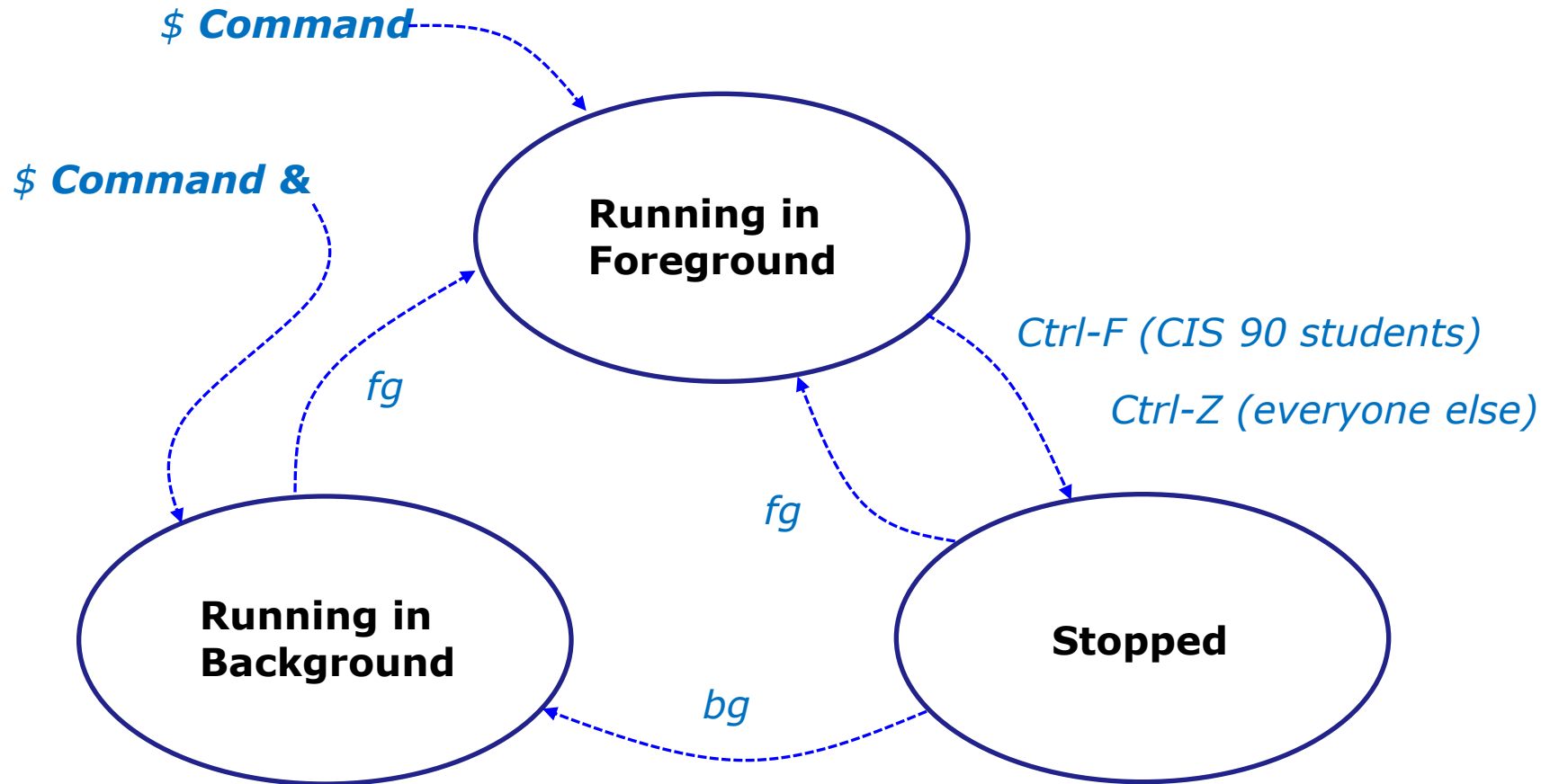
A feature of the bash shell

When a process is **running** (status=R) the user can **stop** it (status=T) 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

A feature of the bash shell

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

A feature of the bash shell

Ctrl-Z or Ctrl-F

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

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

CIS 90 accounts use Ctrl-F

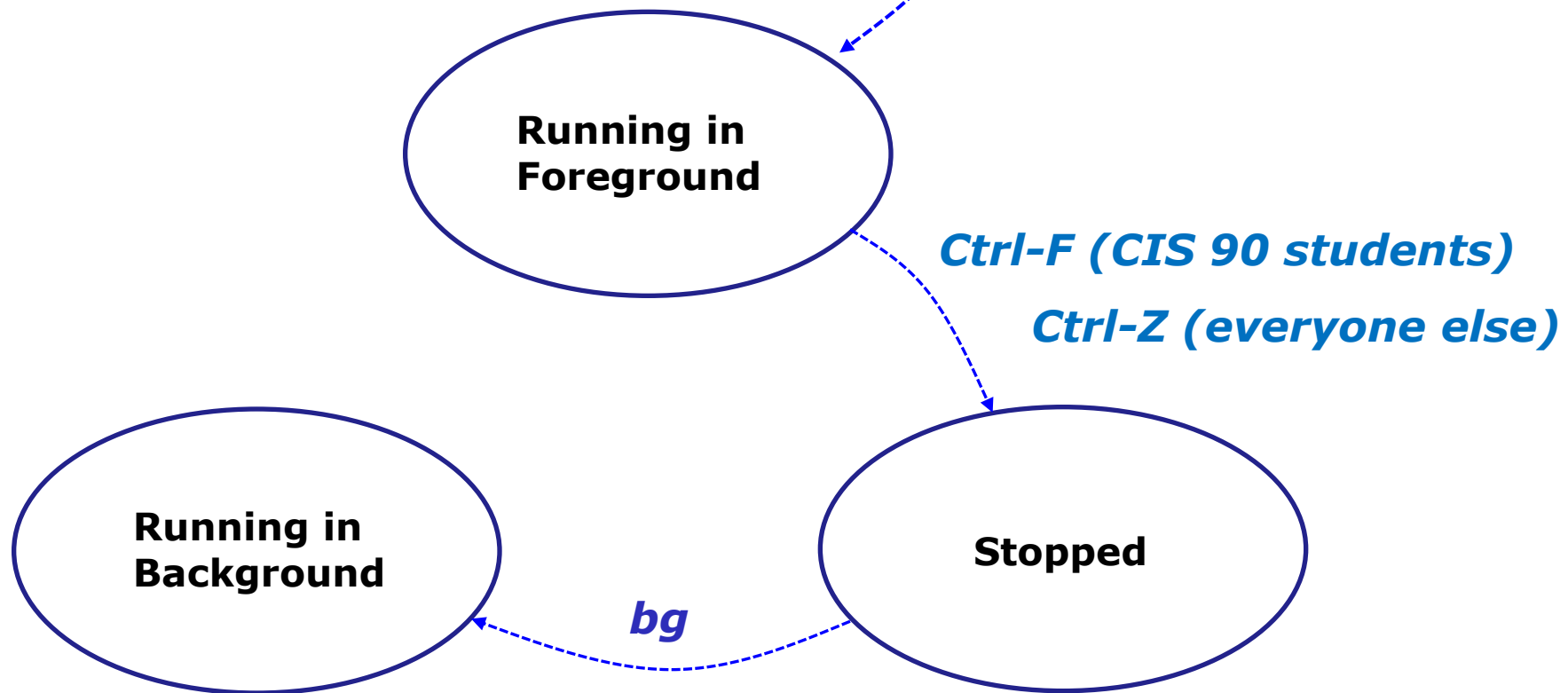
```
[rsimms@opus ~]$ stty -a  
speed 38400 baud; rows 39; columns 84; line = 0;  
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>;  
swch = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;  
lnext = ^V; flush = ^O; min = 1; time = 0;
```

Other Opus accounts use Ctrl-Z

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

Job Control Example 1

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



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

Job Control Example 1

```
[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 while find
is running

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

```
[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

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

Job Control Example 1

```
[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 ~]$
```

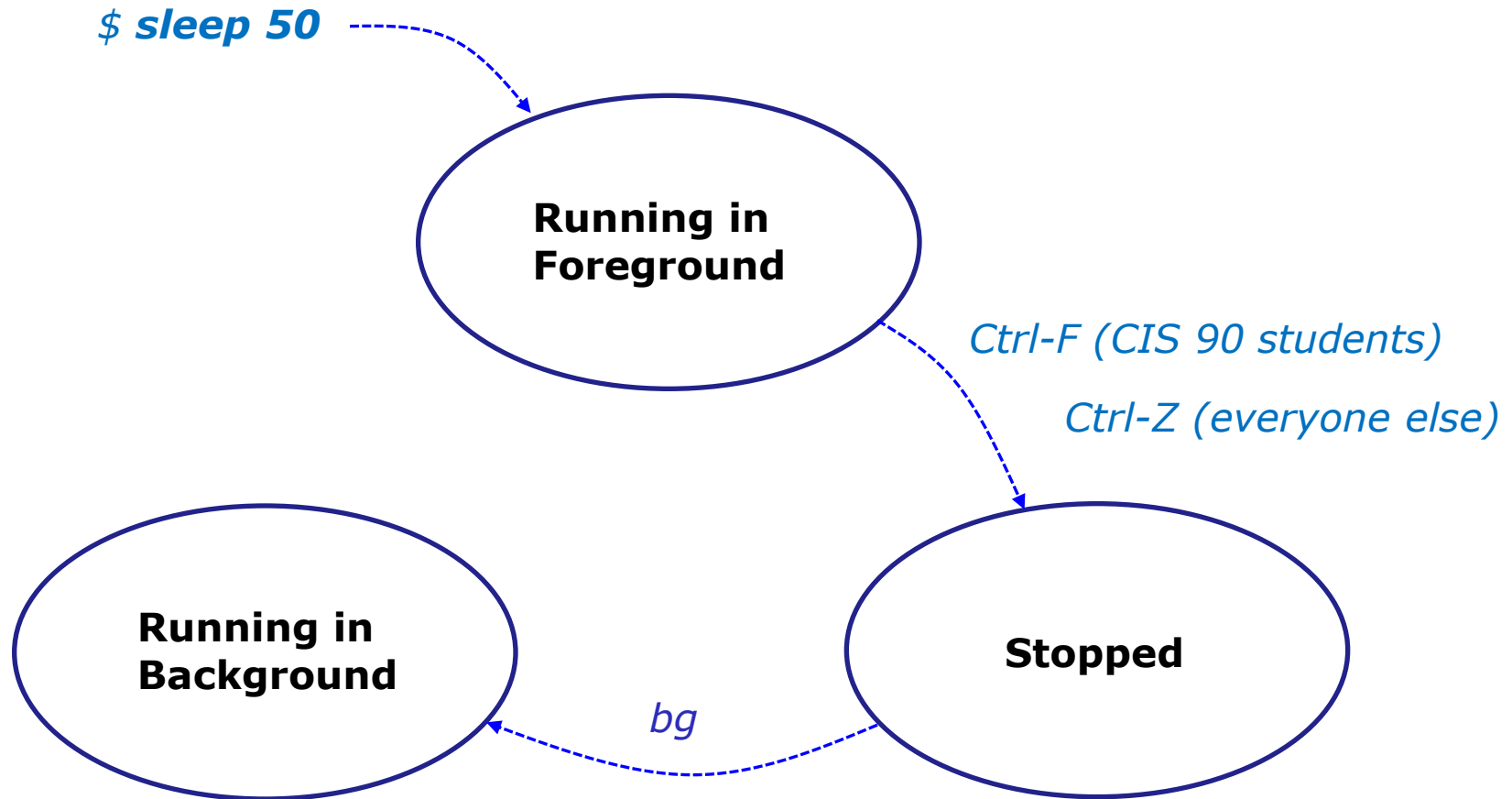
***bg** resumes the
find command in
the background*

```
[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
is running again
(status=R)*

Job Control Example 2



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

Job Control Example 2

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

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

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

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

jobs commands shows
sleep command is stopped

PID 25389
is stopped

```
[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

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


Job Control Example 2

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

*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

A feature of the bash shell

&

- Append to a command to run it in the background

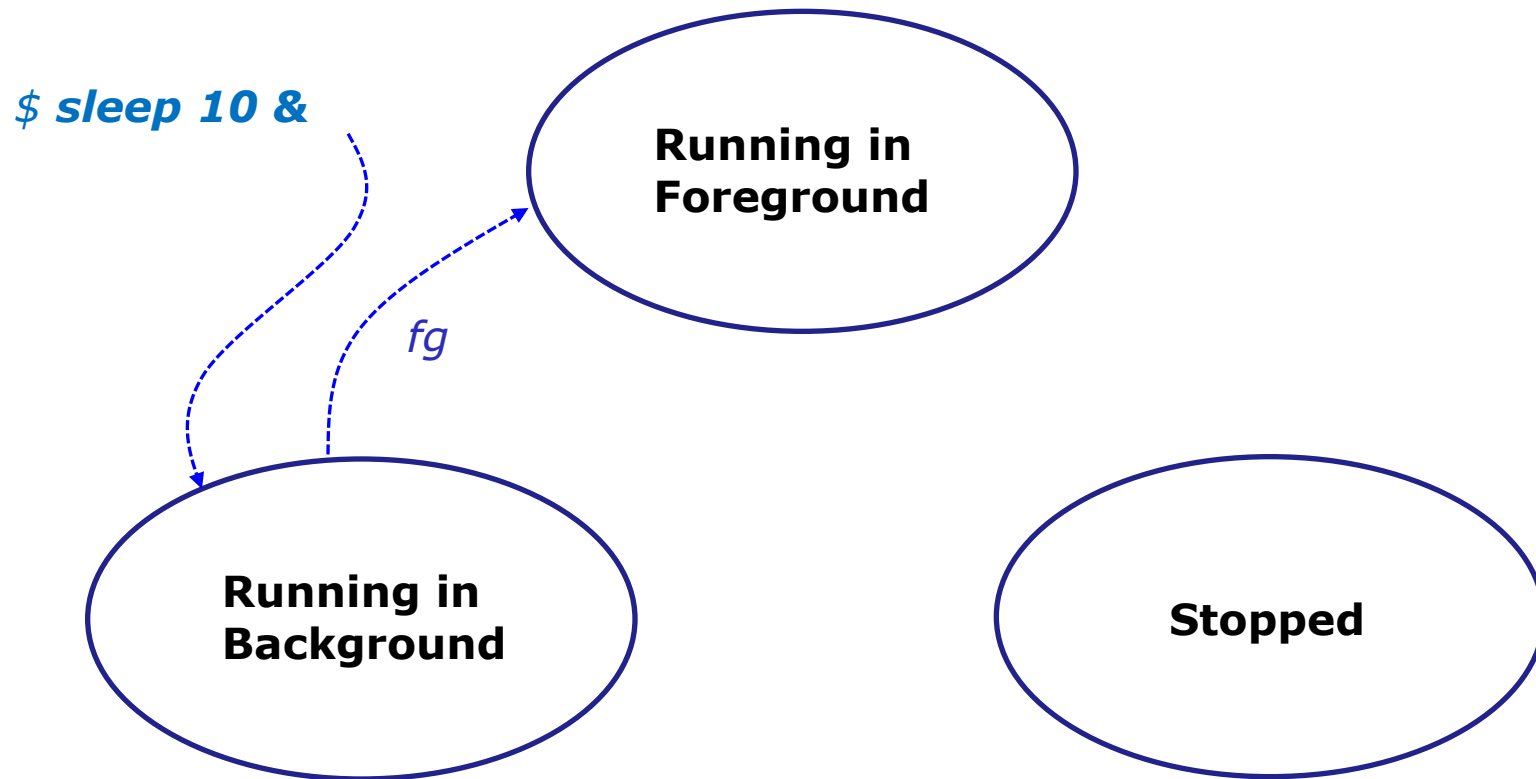
fg

- 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

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

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

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 ~]$
```

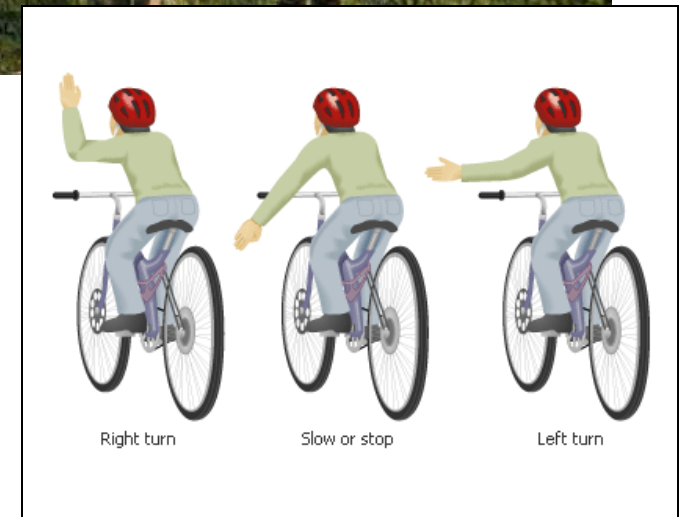
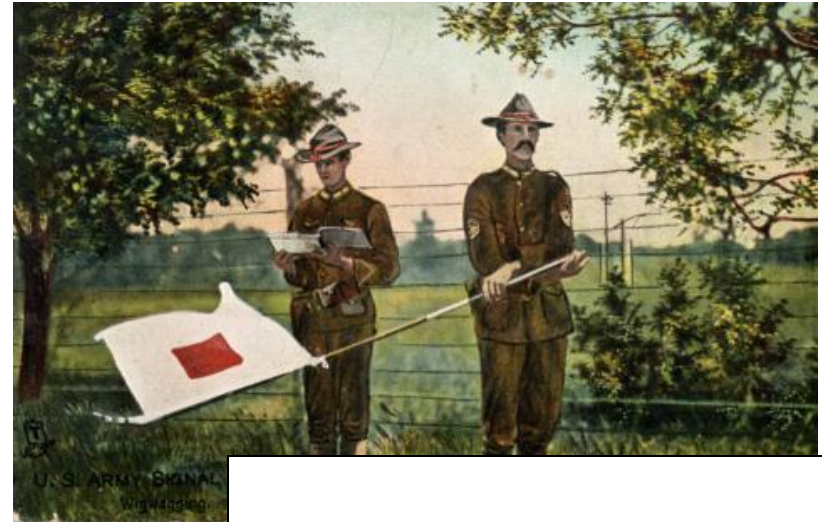
& is often used when running GUI tools like Firefox or Wireshark from the command line. This allows you to keep using the terminal for more commands while those applications run.

Signals

Signals

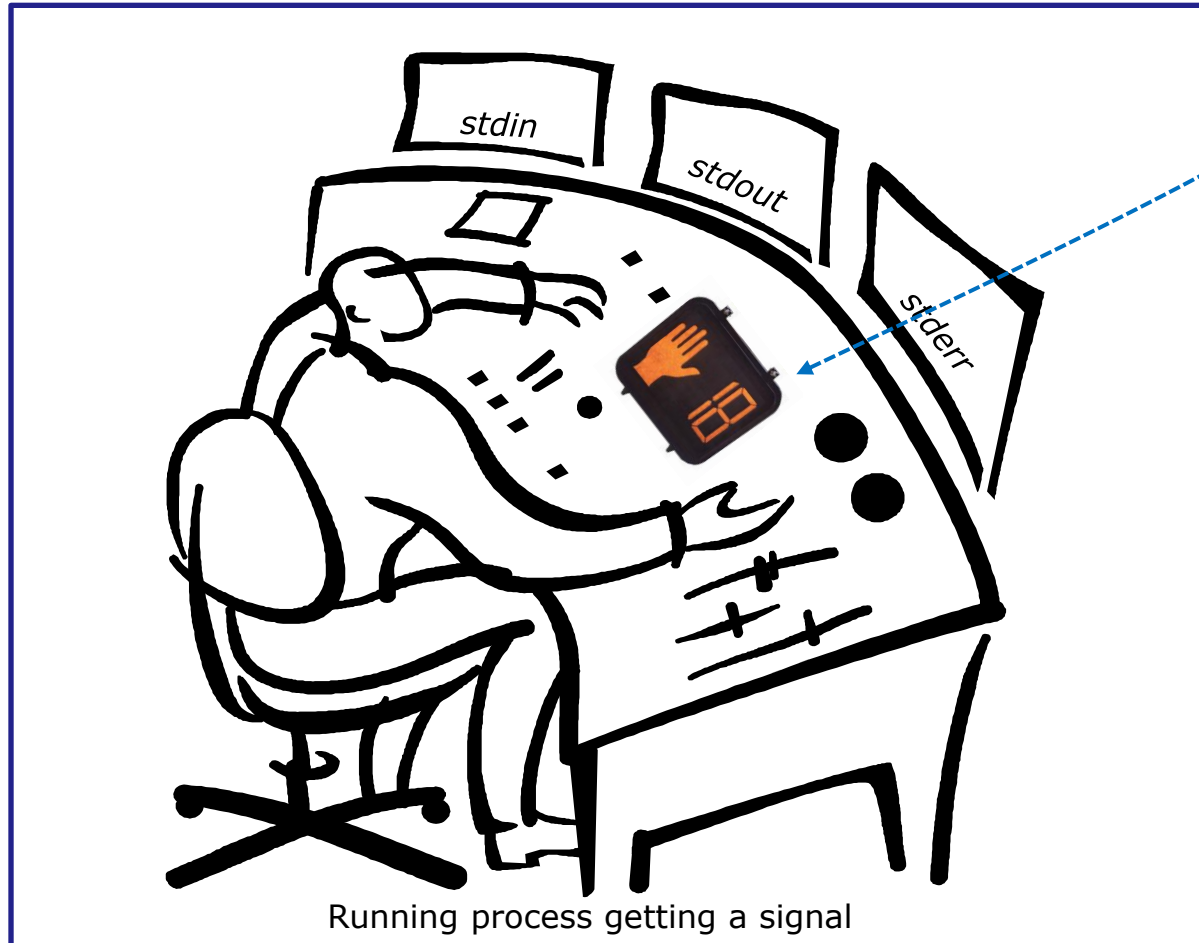
PLATE 4

COMMERCIAL CODE SIGNALS					
<p>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.</p>					
URGENT & IMPORTANT SIGNALS		COMPASS SIGNALS		3 FLAGS	
CODE FLAG OVER 1 FLAG OR 2 FLAG SIGNALS					
<p>CODE FLAG</p> <p>P</p> <p>"I Am about to Sail"</p>		<p>A</p> <p>C</p> <p>"Do Not"</p> <p>"abandon the Vessel"</p>		<p>A</p> <p>Q</p> <p>E</p> <p>N 1/2 E</p> <p>S 3/4 W</p>	
LATITUDE & LONGITUDE SIGNALS		CODE FLAG OVER 2 FLAGS			
<p>CODE FLAG</p> <p>A</p> <p>O</p> <p>12° Latitude</p>		<p>Q</p> <p>H</p> <p>X</p> <p>North Latitude</p>		<p>Q</p> <p>Y</p> <p>Z</p> <p>23° Longitude</p> <p>East Longitude</p>	
NUMERICAL TABLE		GENERAL VOCABULARY		GEOGRAPHICAL SIGNALS ALPHABETICAL ORDER.	
CODE FLAG UNDER 2 FLAGS		3 FLAG SIGNAL		4 FLAG SIGNAL	
<p>Y</p> <p>S</p> <p>CODE FLAG</p> <p>10,000</p>		<p>I</p> <p>X</p> <p>K</p> <p>Tons of Coal</p>		<p>A</p> <p>E</p> <p>Y</p> <p>Z</p> <p>Glasgow, Scotland.</p>	
ALPHABETICAL SPELLING TABLE		NAMES OF VESSELS FROM CODE LIST.			
<p>SPELLING SIGNAL</p> <p>J</p> <p>O</p> <p>H</p> <p>N</p> <p>John</p>		<p>C</p> <p>B</p> <p>D</p> <p>N</p> <p>Abb</p>		<p>C</p> <p>S</p> <p>F</p> <p>P</p> <p>off</p>	
				<p>H</p> <p>C</p> <p>L</p> <p>B</p> <p>Grunder of Glasgow</p> <p>1058 Tons No 32696</p>	



Signals

Signals are asynchronous messages sent to processes



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:

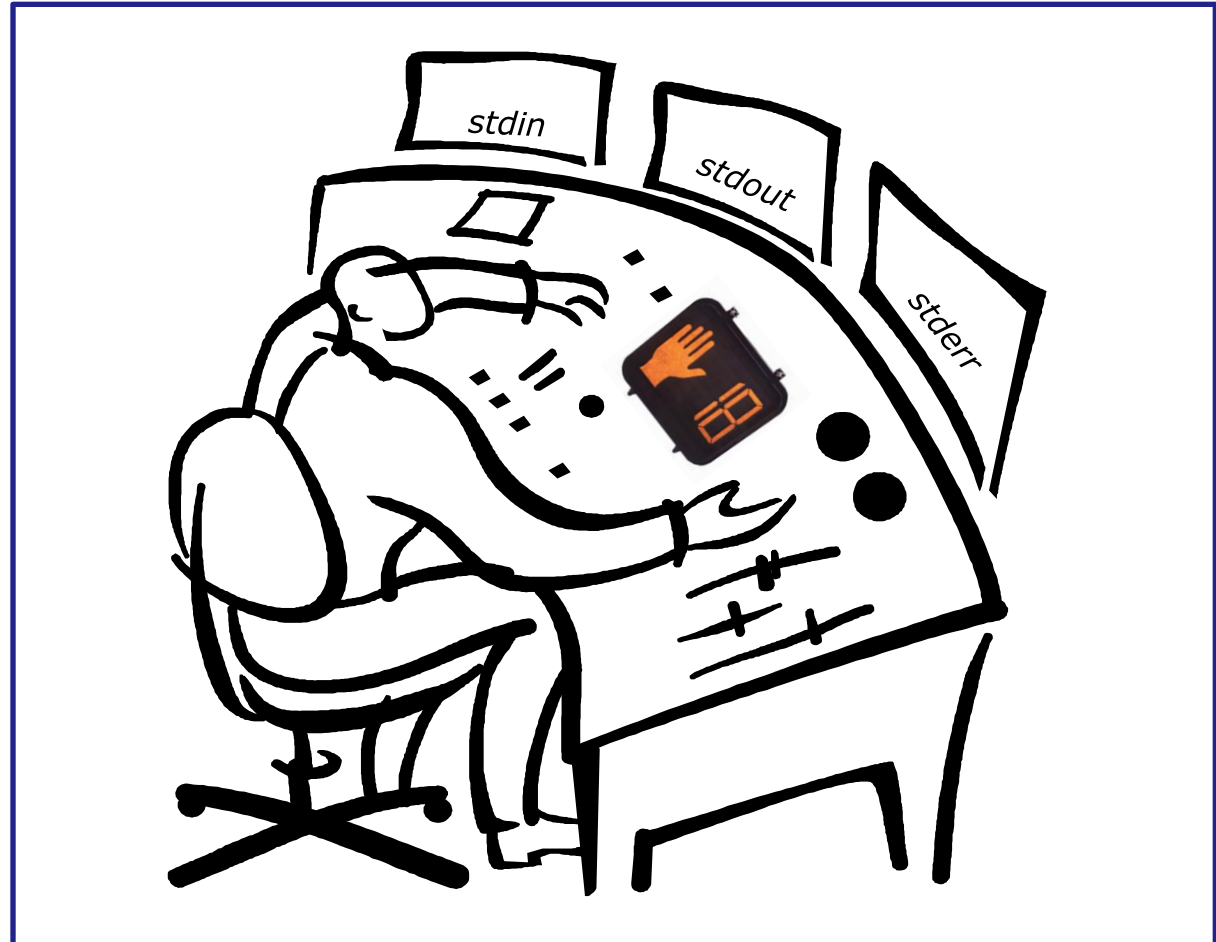
1. be ignored,
2. default action (die)
3. execute some predefined function.

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

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/roddyduk $ 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/roddyduk $
```

Signals

Special keystrokes

```
/home/cis90/roddyduk $ 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

```
rsimms@opus:/home/cis90/depot
#!/bin/sh
#
# app - script to demonstrate 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 Ctrl-Z (to stop foreground 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

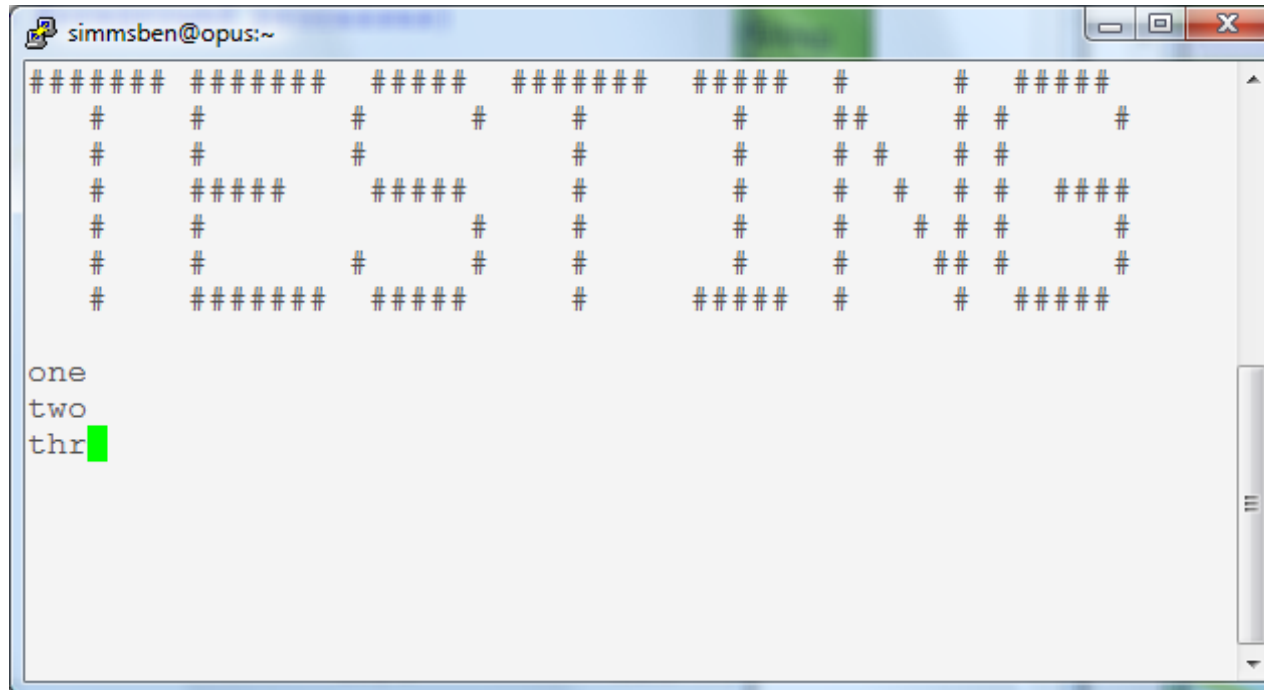
Signals

Class Exercise

- View with **cat ../depot/app**
- Look for the three trap handlers
 - Signal 2 (SIGINT)
 - Signal 3 (SIGQUIT)
 - Signal 15 (SIGTERM)

Signals

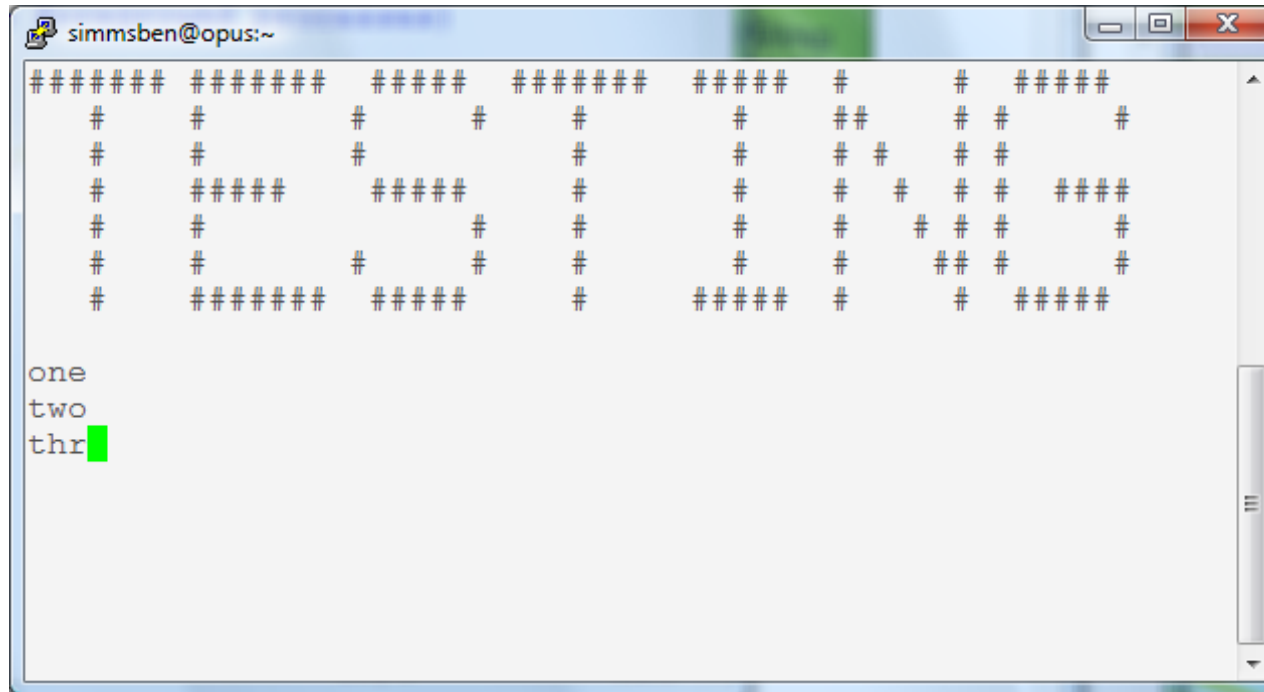
Benji runs app



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

Signals

Benji runs app



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



```

simmsben@opus:~
#####      #####      #####      #####      #####      #      #      #####
#          #          #          #          #          ##         # #          #
#          #          #          #          #          # #        # #         #
#          #####      #####      #          #          # #        # #         #####
#          #          #          #          #          #          # #        # #         #
#          #          #          #          #          #          # #        ## #         #
#          #####      #####      #          #####      #          #          #####
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

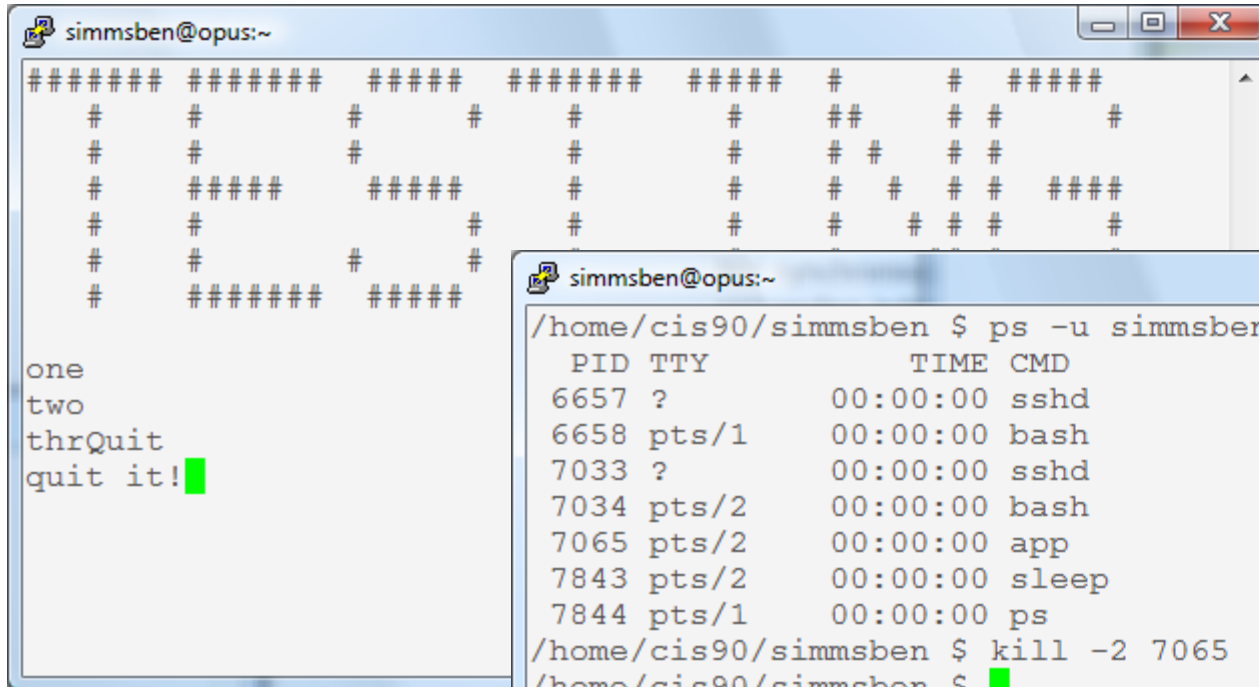


```
rododyduk@opus:~  
/home/cis90/rododyduk $ ps -u simmsben  
  PID TTY          TIME CMD  
 6657 ?            00:00:00 sshd  
 6658 pts/1        00:00:00 bash  
 7033 ?            00:00:00 sshd  
 7034 pts/2        00:00:00 bash  
 7065 pts/2        00:00:00 app  
 7579 pts/2        00:00:00 sleep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

*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



```

simmsben@opus:~
#####
#           #           #           #           #           #           #
#           #           #           #           #           #           #
#           #           #           #           #           #           #
#           #####       #####       #           #           #           #
#           #           #           #           #           #           #
#           #           #           #           #           #           #
#           #####       #####       #           #           #           #

one
two
thrQuit
quit it!

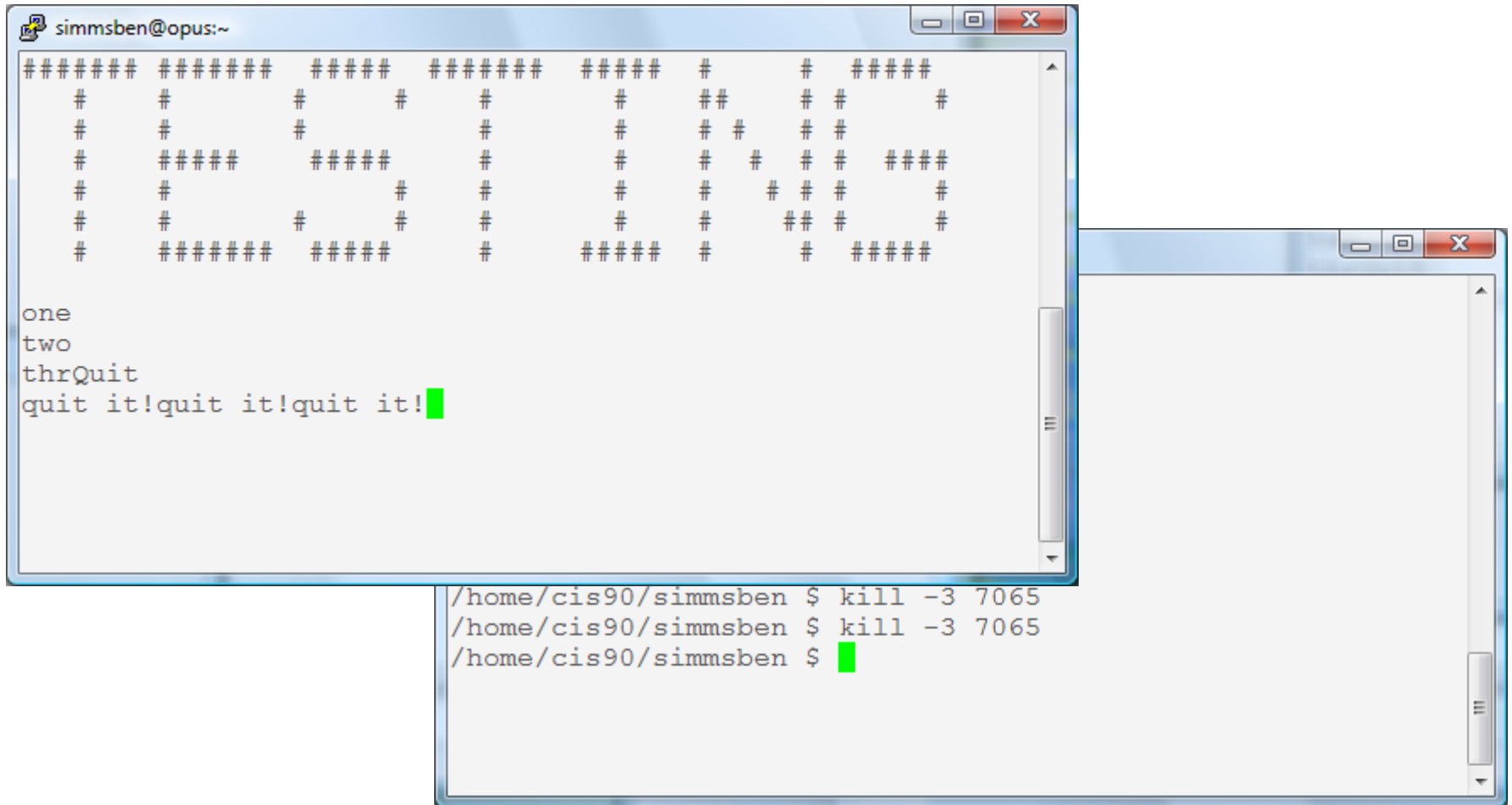
/home/cis90/simmsben $ ps -u simmsben
  PID TTY          TIME CMD
 6657 ?            00:00:00 sshd
 6658 pts/1        00:00:00 bash
 7033 ?            00:00:00 sshd
 7034 pts/2        00:00:00 bash
 7065 pts/2        00:00:00 app
 7843 pts/2        00:00:00 sleep
 7844 pts/1        00:00:00 ps
/home/cis90/simmsben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



```

simmsben@opus:~
#####  #####  #####  #####  #####  #  #  #####
#      #      #      #      #      #  ##  #  #      #
#      #      #      #      #      #  #  #  #  #      #
#      #####  #####  #      #      #  #  #  #  #  #####
#      #      #      #      #      #  #  #  #  #  #      #
#      #      #      #      #      #  #  #  #  #  #  #      #
#      #####  #####  #      #####  #      #  #####

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

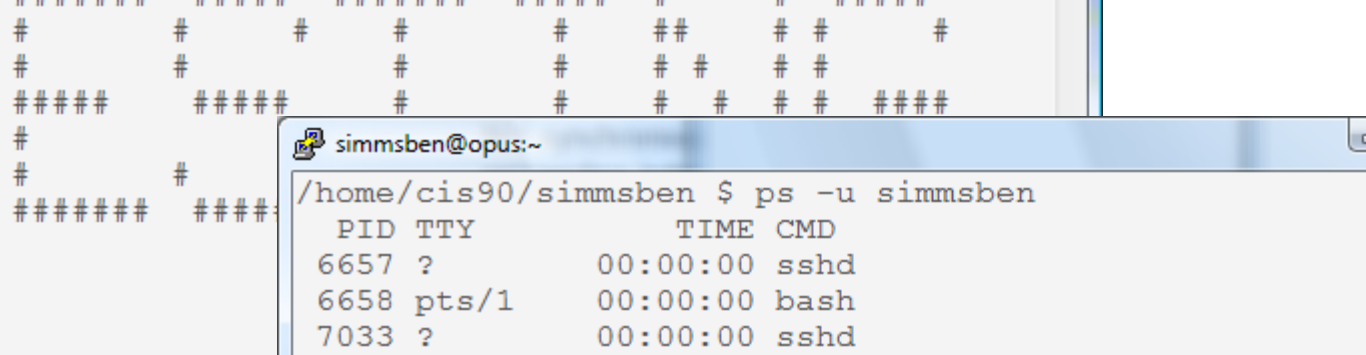
Benji runs app

[illegible]

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

Signals

Benji runs app



The image shows two terminal windows. The top window, titled 'simmsben@opus:~', displays a grid of '#' characters arranged in a pattern that resembles a stylized 'H' or a similar abstract shape. The bottom window, also titled 'simmsben@opus:~', shows the output of the command 'ps -u simmsben'. The output is a table with columns for PID, TTY, TIME, and CMD, listing the processes running for the user '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



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

Signals

Benji runs app

The image displays two overlapping terminal windows from a Linux environment. The background window, titled "simmsben@opus:~", shows a large ASCII art graphic composed of hash symbols (#) forming a rectangular shape with internal patterns. Below the graphic, the text "one", "two", and "thrKilled" are printed, followed by a shell prompt "/home/cis90/simmsben \$". The foreground window, also titled "simmsben@opus:~", shows the execution of the command "ps -u simmsben". This command outputs a table of processes running under the user 'simmsben'. The table has columns for PID, TTY, TIME, and CMD. It lists several instances of sshd, bash, app, sleep, and ps. Following the output, the command "kill -9 8237" is entered at the prompt, and the prompt returns without further output, indicating successful termination.

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

```
simmsben@opus:~/  
/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

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

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

```
/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.*

Load Balancing

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

```
/home/cis90/roddyduk $ cat job1
cp bin/myscript bin/myscript.bak
echo "Job 1 - finished, myscript has been backed up" | mail -s "Job 1" roddyduk
/home/cis90/roddyduk $ at now + 5 minutes < job1
job 24 at 2008-11-12 12:14
/home/cis90/roddyduk $ at now + 2 hours < job1
job 25 at 2008-11-12 14:09
/home/cis90/roddyduk $ at teatime < job1
job 26 at 2008-11-12 16:00
/home/cis90/roddyduk $ at now + 1 week < job1
job 27 at 2008-11-19 12:10
/home/cis90/roddyduk $ at 3:00 12/12/2010 < job1
job 28 at 2008-12-12 03:00
/home/cis90/roddyduk $ jobs
/home/cis90/roddyduk $ atq
25      2008-11-12 14:09 a roddyduk
28      2008-12-12 03:00 a roddyduk
27      2008-11-19 12:10 a roddyduk
26      2008-11-12 16:00 a roddyduk
24      2008-11-12 12:14 a roddyduk
/home/cis90/roddyduk $
```

*Several ways to specify
a future time to run*

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

Load Balancing

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

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

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

Load Balancing

```
/home/cis90ol/simmsben $ at now + 1 minute
```

```
at> kitty letter
```

```
at> <EOT>
```

```
job 150 at 2011-04-20 10:47
```

```
/home/cis90ol/simmsben $
```

Oops, specified a non-existent command to run in the future

```
/home/cis90ol/simmsben $ atq
```

```
150      2011-04-20 10:47 a simmsben
```

```
/home/cis90ol/simmsben $ atq
```

```
/home/cis90ol/simmsben $ mail
```

```
Mail version 8.1 6/6/93.  Type ? for help.
```

```
"/var/spool/mail/simmsben": 1 message 1 new
```

```
>N 1 simmsben@Opus.cabrill Wed Apr 20 10:47 16/709 "Output from your job "  
& 1
```

```
Message 1:
```

```
From simmsben@Opus.cabrillo.edu Wed Apr 20 10:47:01 2011
```

```
Date: Wed, 20 Apr 2011 10:47:01 -0700
```

```
From: Benji Simms <simmsben@Opus.cabrillo.edu>
```

```
Subject: Output from your job 150
```

```
To: simmsben@Opus.cabrillo.edu
```

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

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

Wrap up

New commands:

Ctrl-Z or F
bg

Suspends a foreground process
Resumes suspended process

&
fg

Runs command in the background
Brings background job to foreground

jobs

show background jobs

kill

Send a signal to a process

at
atq
atrm

Run job once in the future
Show all *at* jobs queued to run
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?

The Test



- 10 minute break
- Make more trouble on Sun
- Reserve VMs for remote test takers
- Add read permission on test2



Test 2

Backup

A message from ...





Greetings Professor Simms,

My name is Tim Hill. I'm chair of the MIS department in the College of Business at SJSU. I'm reaching out to computer information systems faculty and advisers in the local community colleges to raise awareness of our program and the important benefits community college transfer students should consider when exploring major options at SJSU. For students interested in combining computer technology and business, MIS represents an exceptional choice of concentration within the Bachelors of Science in Business Administration.

Management Information Systems (MIS) is the BSBA concentration at SJSU that currently garners *the highest starting salaries and yields the highest placement rate upon graduation*. And MIS students are the *only ones on the SJSU campus formally recruited Google!* But unfortunately, far too few transfer students are aware of MIS and its unique advantages. Please consider making them aware of the 4 attachments and the 6 critical bullet points below before they choose their concentration within the BSBA at SJSU.

If you would be so kind as to share this information as appropriate, it would be greatly appreciated. And please let me know if I can be of assistance. I would be happy to visit your campus to speak with you and/or your colleagues and/or students about the MIS program and job opportunities. Just call (408) 924-3512 or reply to this email and I'll get back to you promptly.



Recent developments to note:

The College of Business has just reduced the transfer GPA requirement from 3.4 down to 2.8, effective immediately.

SJSU is accepting applications for Fall 2011 through November 30.

(http://www.sjsu.edu/news/news_detail.jsp?id=3459)

SJSU has extended the Spring 2011 Admissions deadline to November 15 (http://www.sjsu.edu/news/news_detail.jsp?id=3468)

If you would be so kind as to share this information as appropriate, it would be greatly appreciated. And please let me know if I can be of assistance. I would be happy to visit your campus to speak with you and/or your colleagues and/or students about the MIS program and job opportunities. Just call (408) 924-3512 or reply to this email and I'll get back to you promptly.

Timothy R. Hill, Chair
Department of Management Information Systems
San Jose State University
One Washington Square
San Jose, CA 95192-0244
(408) 924-3512 (v)

http://www.cob.sjsu.edu/hill_t



College of Business Placement Rate at Graduation

(by concentration)

MIS Graduates lead in percentage of students with positions in career field upon graduation!

FALL 2009 Graduates***	
Accounting (28/82)	34%
AIS (2/6)	33%
Corporate Finance (1/10)	10%
Finance (13/45)	29%
HRM (5/19)	26%
International Business (2/13)	15%
Management (31/97)	32%
MIS (27/47)	57%
Marketing (14/74)	19%

*** 152 out of 434 participating graduates reported (35%)

SPRING 2010 Graduates**	
Accounting (28/101)	27%
AIS (3/12)	25%
Corporate Finance (3/16)	19%
Finance (16/66)	24%
HRM (4/19)	21%
International Business (8/35)	23%
Management (41/142)	29%
MIS (23/45)	51%
Marketing (25/108)	23%

** 170 out of 581 participating graduates reported (29.2%)

(Poll taken by SJSU Career Center)

* Data collected only from students who chose to participate in this survey at the College of Business Convocations.



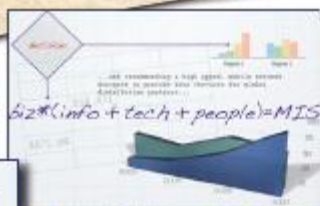
BS in Business Administration (avg new grad salary*)

Concentration:	<input checked="" type="checkbox"/> Management Information Systems	\$58,555
	<input type="checkbox"/> Accounting Information Systems	\$53,797
	<input type="checkbox"/> Finance	\$52,733
	<input type="checkbox"/> Management	\$52,174
	<input type="checkbox"/> International Business	\$51,887
	<input type="checkbox"/> Accounting	\$51,438
	<input type="checkbox"/> Corporate Financial Management	\$48,750
	<input type="checkbox"/> Marketing	\$46,597
	<input type="checkbox"/> Human Resources Management	\$44,365

MIS is managing business technology ⇒
getting people the info they need
to do their jobs



MIS students learn both sides



- ✓ interesting, dynamic, growing field – evolves with tech
- ✓ learn business + database, networking, web, security...
- ✓ award-winning faculty, academic & industry (IBM, Cisco, etc.)
- ✓ hands-on projects at local non-profits, eg. SJ City Parks Dept
- ✓ MIS grads work at HP, Cisco, Google, eBay...
- ✓ highest average salary of all SJSU Business concentrations*



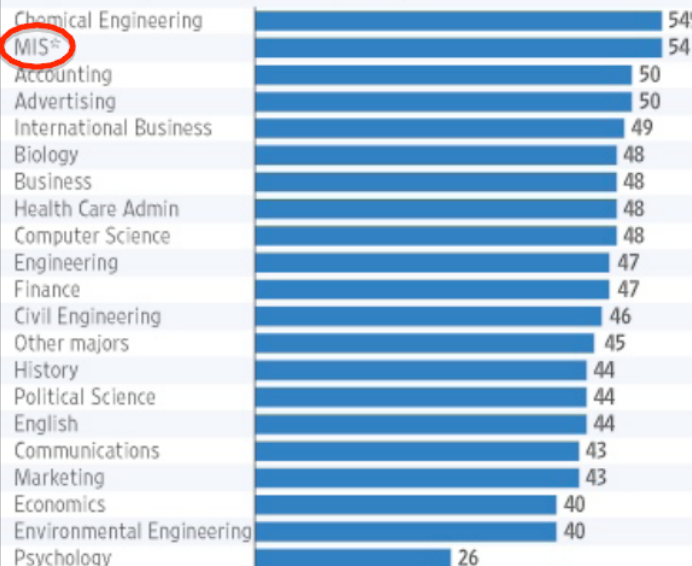
my the future is MIS

www.cob.sjsu.edu/mis

* according to 2009 SJSU Career Center survey

Satisfaction Not Guaranteed

Percentage of college graduates, sorted by major, who answered 'satisfied' or 'very satisfied' to the question: 'Overall, how satisfied are you with your current career path up to now?'



⁴Management Information Systems

Survey was conducted between April and June 2010, of people who graduated college between 1999 and 2010, with 10,800 respondents. Margin of error ranges between 2% and 7%, depending on major. Survey was limited to grads in a set of jobs deemed satisfying, well-paid and with growth potential. Source: PayScale.com

Google recruits SJSU business graduates

By Jaimie Collins
Spartan Daily
September 30, 2010



While visiting campus on Oct. 14, Google plans to recruit students from the College of Business for a two-year training program, said Google's global communications and public affairs representative.

"We are looking for people who are willing to tackle the big challenges and come up with innovative solutions — people who think outside the box," Jordan Newman said. "We definitely want people who aren't afraid to roll up their sleeves and get their hands dirty."

The Internal Technology Residency Program incorporates about 30 graduates and is designed to teach recruits how to support the technology and software systems used by Google employees, according to the program's website.

"We were very selective," Newman said. "At the end of the two years, there is always the possibility that (participants) will be converted into full-time employees."

Applications are only available to graduating seniors in the management information systems department, with interviews for those selected being held on Oct. 22, said department chair Timothy Hill.

"This is an exceptional program offered by the absolute world leader in technology, now and for the foreseeable future," Hill said. "It is really a golden opportunity for our graduates."

Junior accounting major Sarah Allen said she is glad Google will recruit from SJSU in the future.

"Having an opportunity like this will open tons of doors for grads," she said. "Being able to put Google on your resume when you've only been out of school for a few years — that's awesome."

The business department was honored last spring when four graduates were recruited for the program, Hill said.

According to an SJSU press release, the four students selected included Alex Khajehtoorian, Kobi Laredo, Marcos Ramirez and Ed Saucedo, all 2010 graduates from the management information systems department.

Of the students that were hired, Hill said two were members of the honors program and the entire group was highly distinguished among faculty.

"We are extremely proud," he said. "We think (their employment) says volumes about the kind of program we've built and the quality of

find command

Find all directories starting in my home directory that start with a capital B, S, Y or A.

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

Find all files starting in my home directory that contain town

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

grep command tips

*Use the "**^ *1**" to match all lines that start with zero or more blanks, a 1, followed by a blank.*

```
/home/cis90/roddyduk $ cat testfile
```

```
671 buster  
 99 scout  
125 benji  
 1 homer  
934 duke  
100 lucy  
 10 smokey  
322 sky
```

```
/home/cis90/roddyduk $ grep "^ *1 " testfile
```

```
1 homer
```

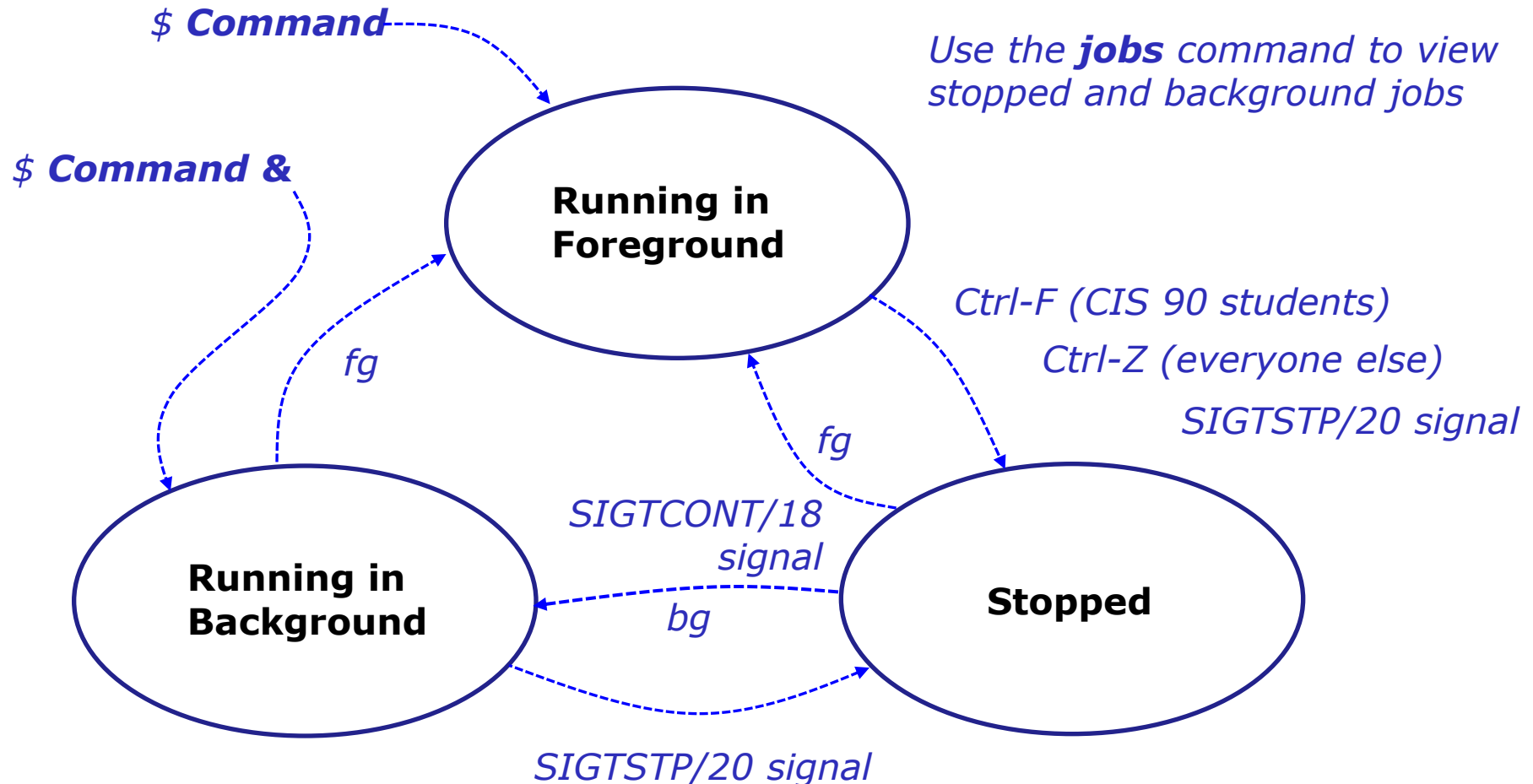
*Use the **B** option to list lines preceding the matched line*

```
/home/cis90/roddyduk $ grep -B 3 "^ *1 " testfile
```

```
671 buster  
 99 scout  
125 benji  
 1 homer
```

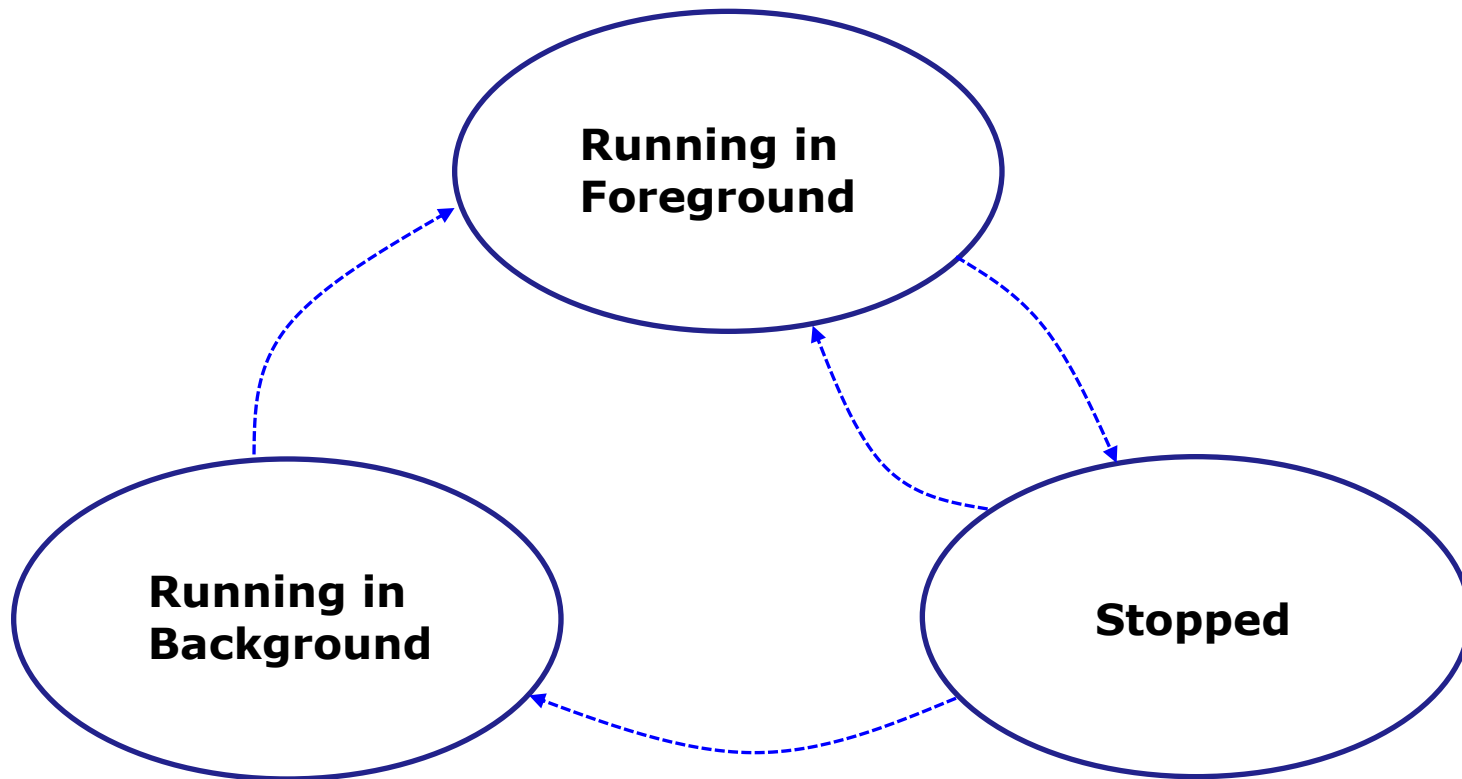
Job Control

A feature of the bash shell



Job Control

A feature of the bash shell



Process Information

Process ID number

```
[rsimms@opus ~]$ ps -a
  PID TTY          TIME CMD
 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
 6294 pts/6        00:00:00 ps
[rsimms@opus ~]$
```

-a option shows all my processes not associated with a terminal. This includes my other login session where I'm doing a man command on ps.

```
[rsimms@opus ~]$ ps x
  PID TTY          STAT       TIME COMMAND
 5368 ?            S           0:00 sshd: rsimms@pts/0
 5369 pts/0        Ss          0:00 -bash
 6173 pts/0        S+          0:00 man ps
 6176 pts/0        S+          0:00 sh -c (cd /usr/share/man && (echo ".ll 7.5i"; echo ".nr L
 6177 pts/0        S+          0:00 sh -c (cd /usr/share/man && (echo ".ll 7.5i"; echo ".nr L
 6182 pts/0        S+          0:00 /usr/bin/less -is
 6203 ?            S           0:00 sshd: rsimms@pts/6
 6204 pts/6        Ss          0:00 -bash
 6312 pts/6        R+          0:00 ps x
```

The x option shows full commands being run and states (most are asleep).

Sleeping (S) *Running (+ means running in the foreground)* (R+)

I'm using two Putty sessions, in one session I have the man page open for ps, the other I'm issuing ps commands