Practical Computing for Scientists

Armin Sobhani
CSCI 2000U
UOIT – Fall 2015

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

Checkpoint 4

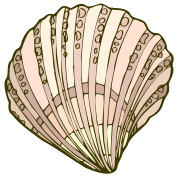
- Please answer the *Self-Assessment* :
  - Blackboard > Course Content > Week 2 (Sept. 21-25) > Wednesday Sept. 23 > Checkpoint 4

The Unix Shell

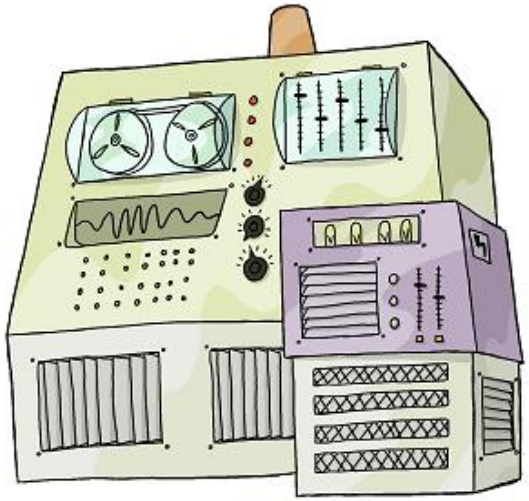
Permissions

Created by Greg Wilson

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

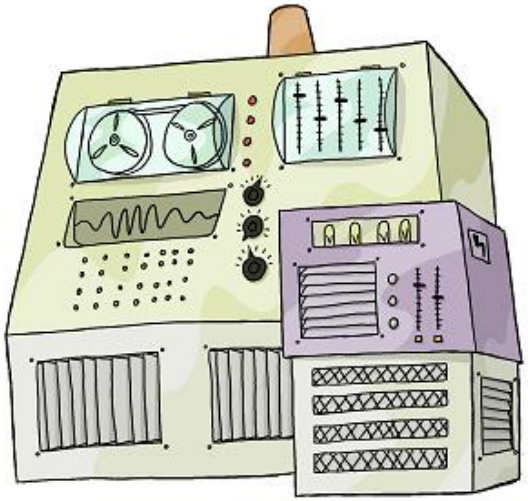


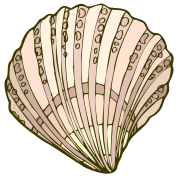
shell



shell

```
pwd, mkdir, cp, ...
```

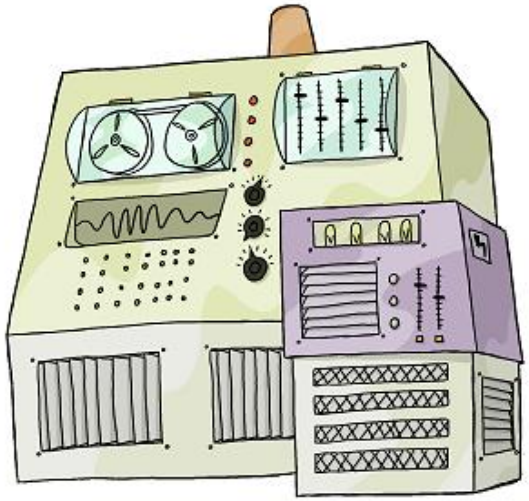


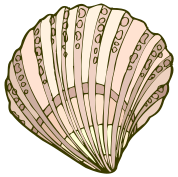


shell

pwd, mkdir, cp, ...

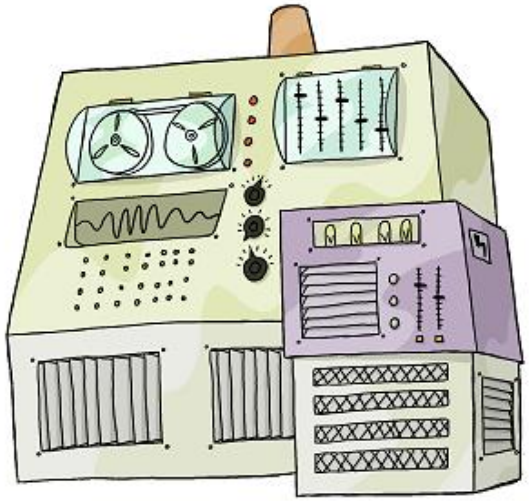
*

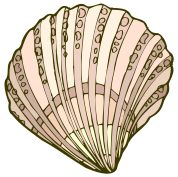




shell

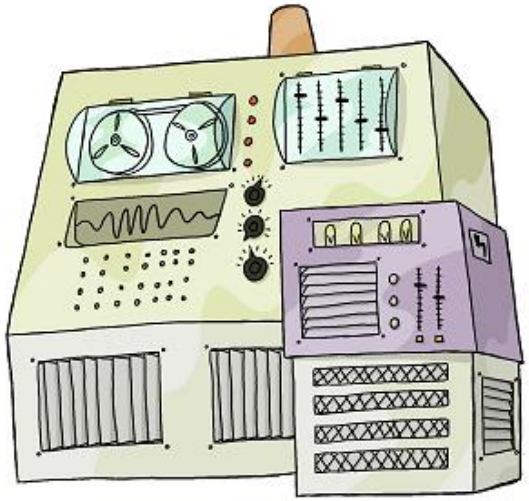
```
pwd, mkdir, cp, ...
*
>, |
```



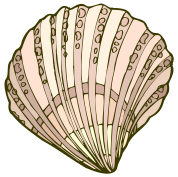


shell

```
pwd, mkdir, cp, ...
*
>, |
```

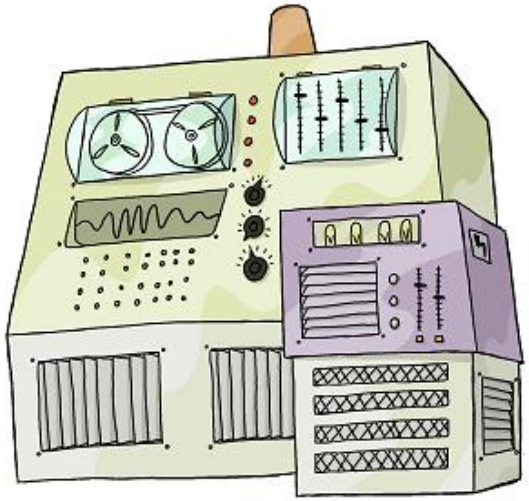


Who can see what?



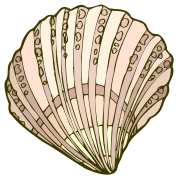
shell

```
pwd, mkdir, cp, ...
*
>, |
```



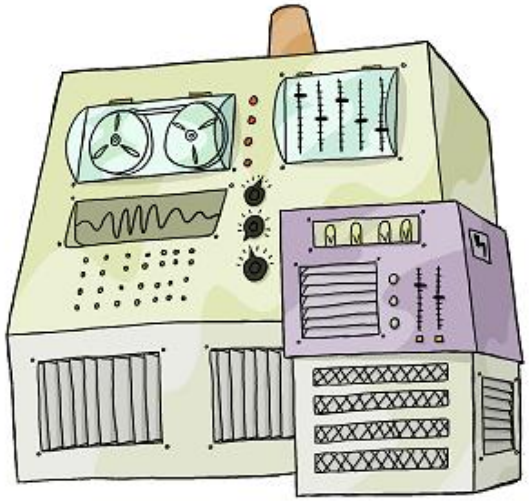
Who can see what?

change



shell

```
pwd, mkdir, cp, ...
*
>, |
```



Who can see what?

change

run

Simplified version of Unix permissions

Simplified version of Unix permissions

Windows uses similar concepts…

Simplified version of Unix permissions

Windows uses similar concepts…

…but there is no exact translation between the two

user

user

Has unique *user name* and *user ID*

user

Has unique *user name* and *user ID*

User name is text: "imhotep", "larry", “nelle", …

user

Has unique *user name* and *user ID*

User name is text: "imhotep", "larry", "vlad", ...

User ID is numeric (easier for computer to store)

user | group

user | group

Has unique *group name* and *group ID*

user | group

Has unique *group name* and *group ID*

User can belongs to zero or more groups

user | group

Has unique *group name* and *group ID*

User can belongs to zero or more groups

List is usually stored in /etc/group

user | group | all

user | group | all

Everyone else

user | group | all

Has user and group IDs



	user	group	all
read			



	user	group	all
read			
write			



	user	group	all
read			
write			
execute			



	user	group	all
read	✓	✓	✗
write	✓	✗	✗
execute	✗	✗	✗

File's owner can read and write it

	user	group	all
read	✓	✓	✗
write	✓	✗	✗
execute	✗	✗	✗

File's owner can read and write it

Others in group can read

	user	group	all
read	✓	✓	✗
write	✓	✗	✗
execute	✗	✗	✗

File's can read and write it

Others in group can read

That's all



	user	group	all
read	✓	✓	✗
write	✓	✗	✗
execute	✗	✗	✗

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$
```

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$
```

```
$ cd labs
$ ls
safety.txt     setup        waiver.txt
$ ls -F
safety.txt     setup*       waiver.txt
$
```

means "executable"

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

name

```
$ cd labs
$ ls
safety.txt     setup        waiver.txt
$ ls -F
safety.txt     setup*       waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

last modified

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

size (in bytes)

```
$ cd labs
$ ls
safety.txt     setup        waiver.txt
$ ls -F
safety.txt     setup*       waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

group owner

```
$ cd labs
$ ls
safety.txt     setup        waiver.txt
$ ls -F
safety.txt     setup*       waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

user owner

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

don't care (for now)

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

permissions

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

-rwxr-xr-x

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

-rwxr-xr-x

file type

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

-rwxr-xr-x

file type → '-' for regular

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

-rwxr-xr-x

file type

'-' for regular

'd' for directory

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

-rwxr-xr-x

user owner permissions

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

-rwxr-xr-x

group owner permissions

```
$ cd labs
$ ls
safety.txt     setup       waiver.txt
$ ls -F
safety.txt     setup*      waiver.txt
$ ls -l
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

-rwxr-xr-x

everyone else's permissions

```
$ ls -a -l
drwxr-xr-x 1 vlad bio     0  2010-08-14 09:55 .
drwxr-xr-x 1 vlad bio  8192  2010-08-27 23:11 ..
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

```
$ ls -a -l
drwxr-xr-x 1 vlad bio     0  2010-08-14 09:55 .
drwxr-xr-x 1 vlad bio  8192  2010-08-27 23:11 ..
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

```
$ ls -a -l
drwxr-xr-x 1 vlad bio     0  2010-08-14 09:55 .
drwxr-xr-x 1 vlad bio  8192  2010-08-27 23:11 ..
-rw-rw-r-- 1 vlad bio  1158  2010-07-11 08:22 safety.txt
-rwxr-xr-x 1 vlad bio 31988  2010-07-23 20:04 setup
-rw-rw-r-- 1 vlad bio  2312  2010-07-11 08:23 waiver.txt
$
```

What does "execute" mean for directories?

What does "execute" mean for directories?

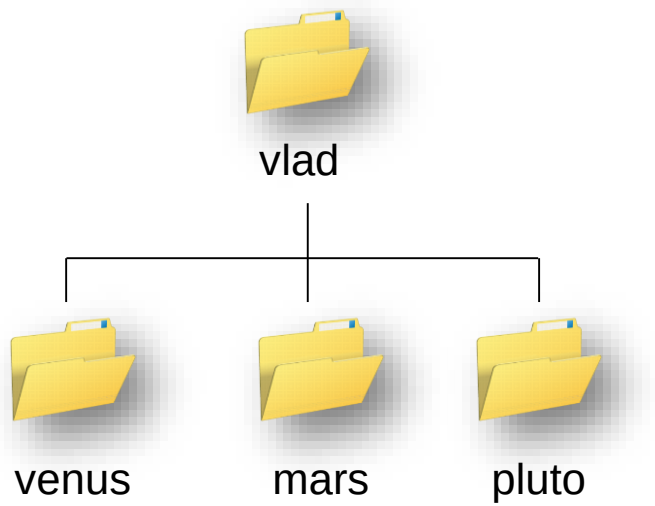
Gives the right to *traverse*

the directory

What does "execute" mean for directories?

Gives the right to *traverse*

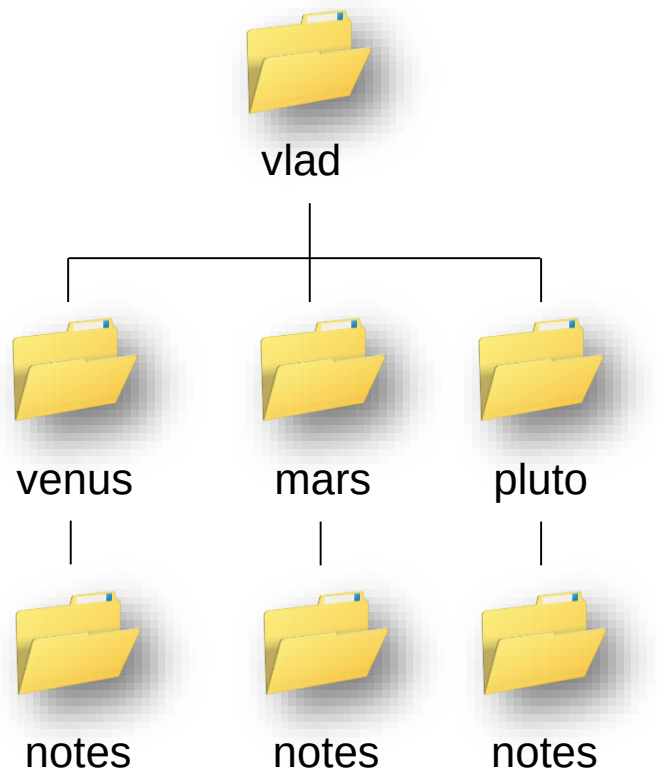
the directory



What does "execute" mean for directories?

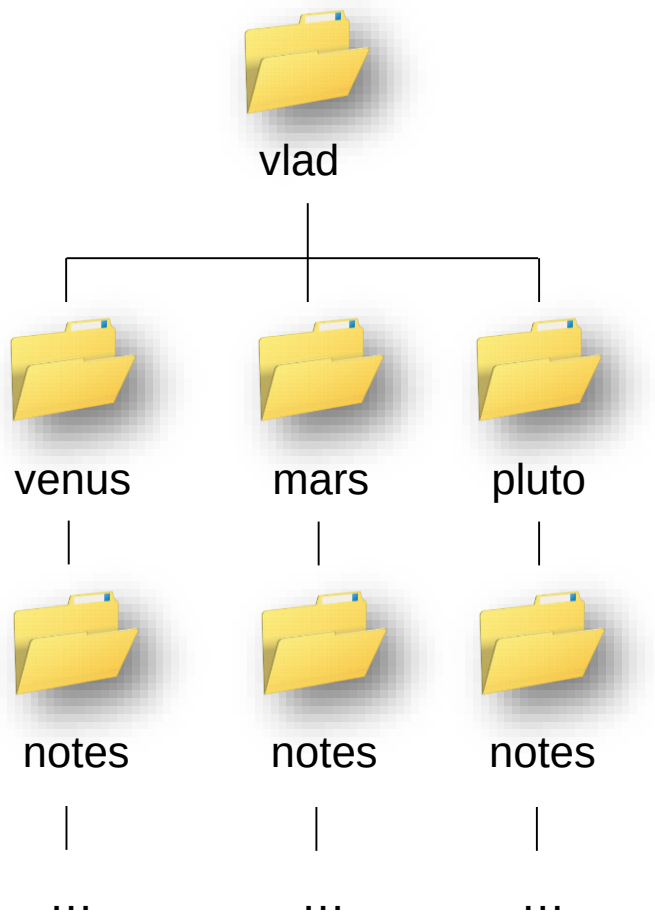
Gives the right to *traverse*

the directory



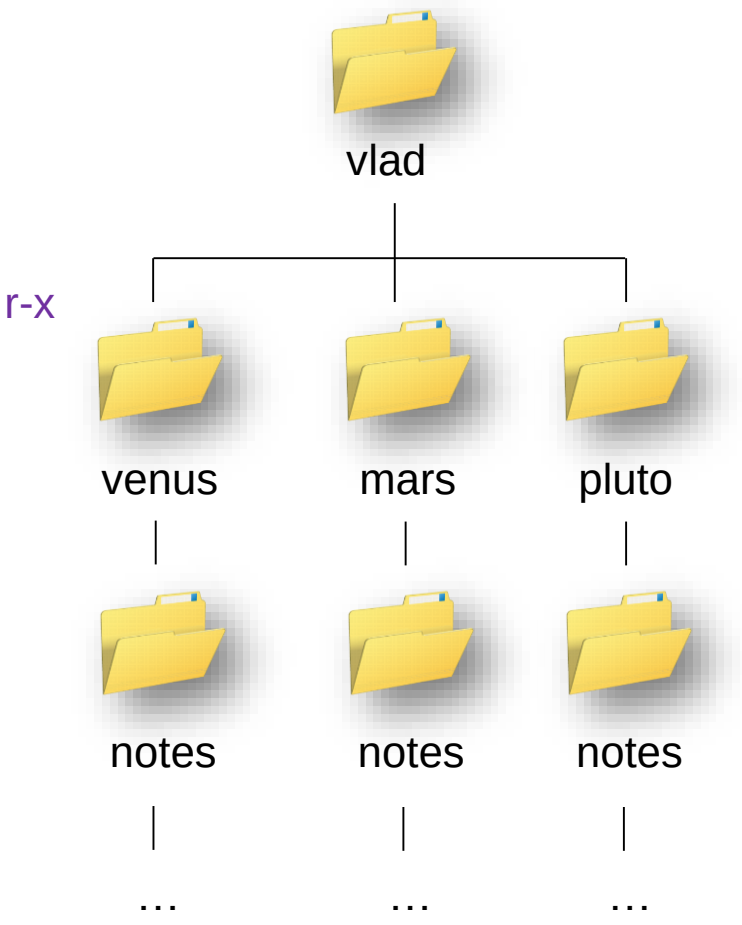
What does "execute" mean for directories?

Gives the right to *traverse* the directory



What does "execute" mean for directories?

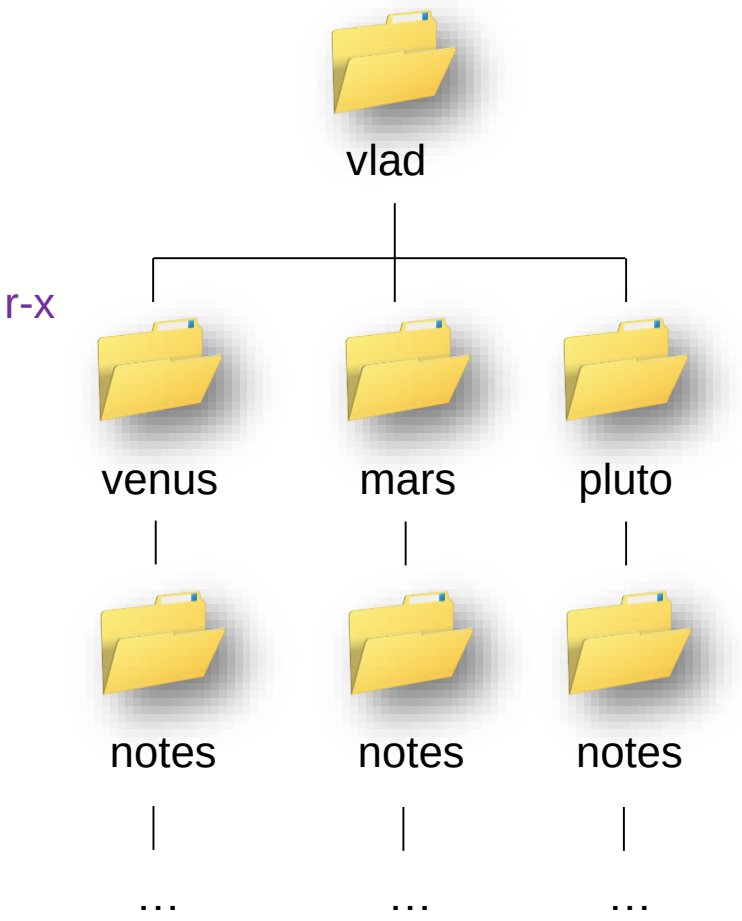
Gives the right to *traverse* the directory



What does "execute" mean for directories?

Gives the right to *traverse* the directory

```
$ ls venus venus/notes
```

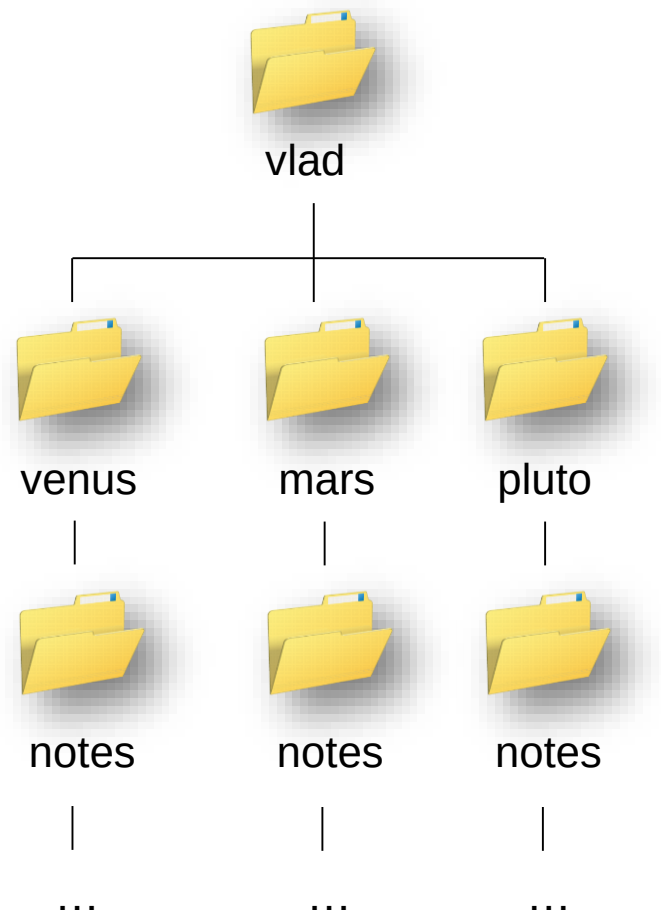


What does "execute" mean for directories?

Gives the right to *traverse* the directory

```
$ ls venus venus/notes ✓
```

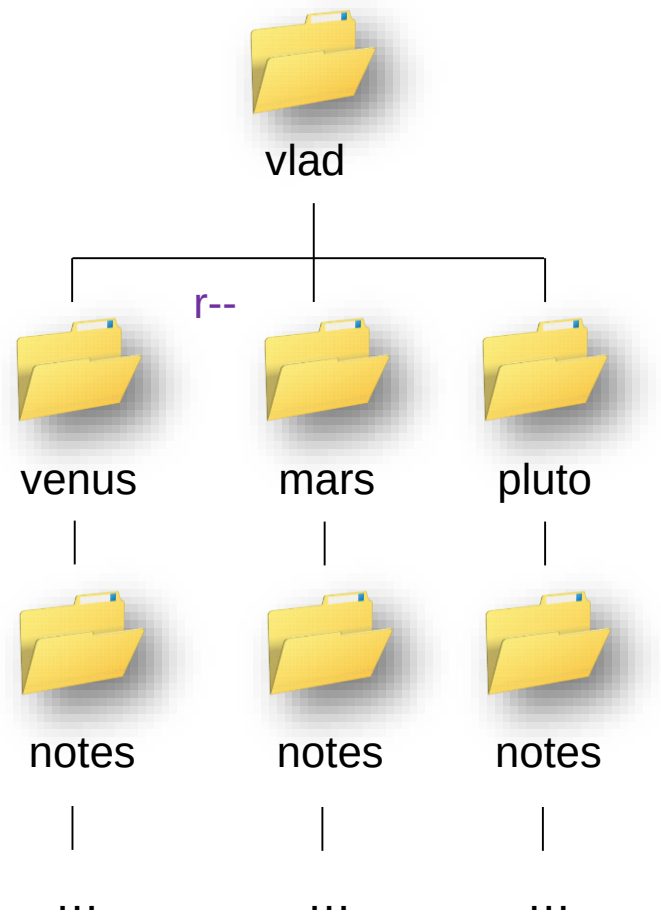
r-x



What does "execute" mean for directories?

Gives the right to *traverse* the directory

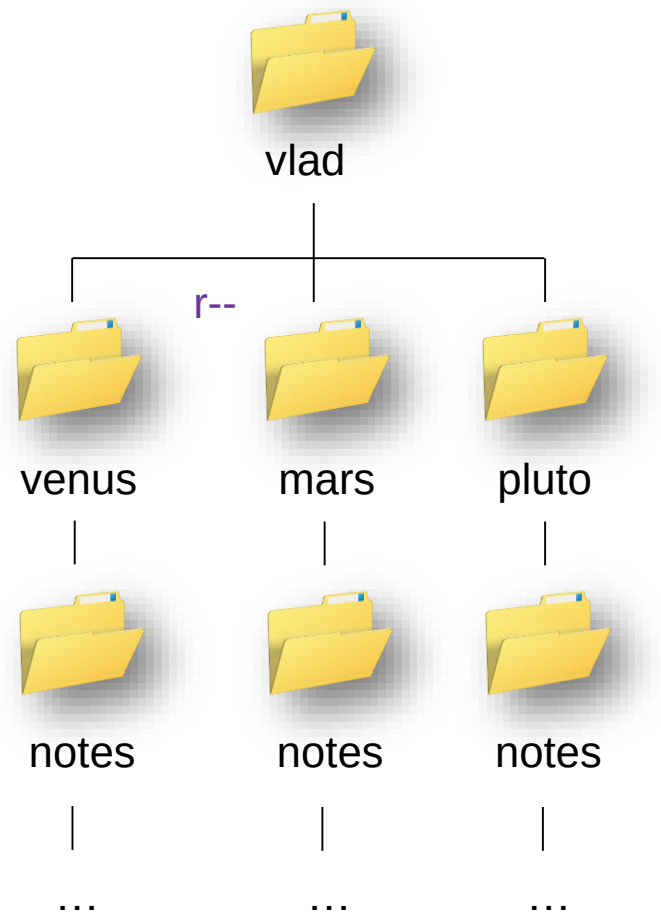
```
$ ls venus venus/notes ✓
```



What does "execute" mean for directories?

Gives the right to *traverse* the directory

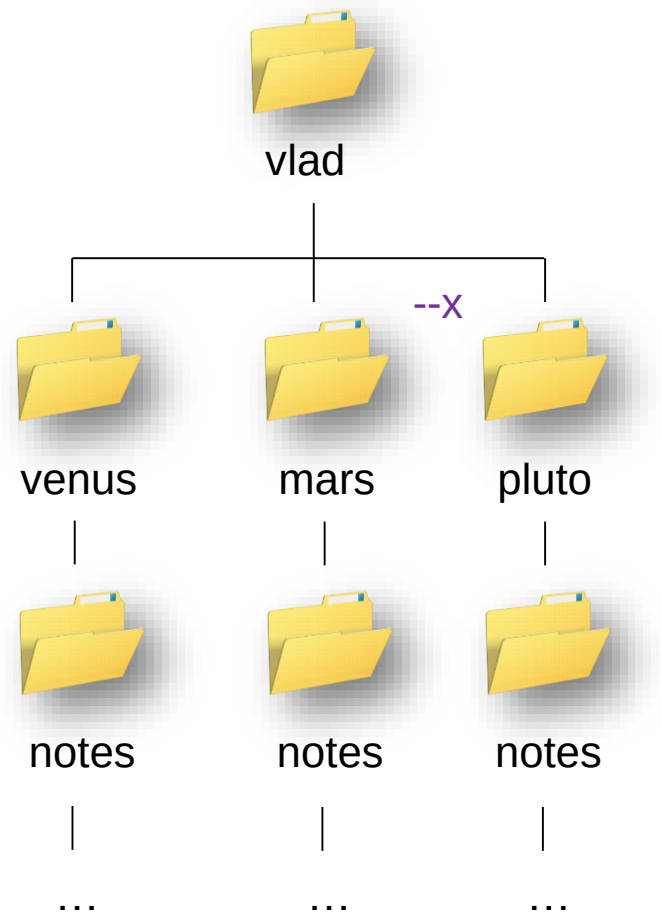
```
$ ls venus venus/notes ✓
$ ls mars mars/notes   ✓
```



What does "execute" mean for directories?

Gives the right to *traverse* the directory

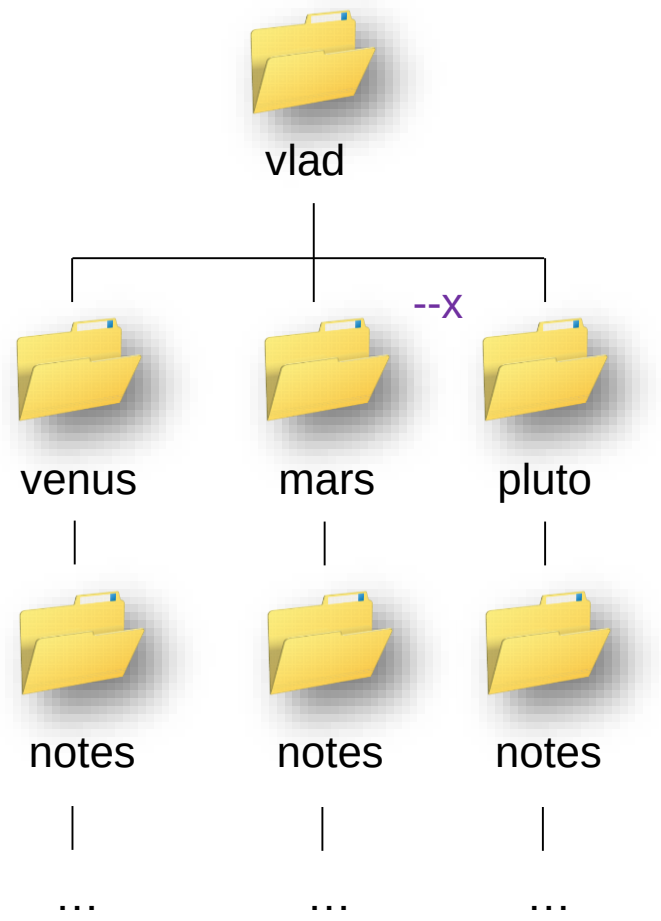
```
$ ls venus venus/notes ✓
$ ls mars mars/notes   ✓
```



What does "execute" mean for directories?

Gives the right to *traverse* the directory

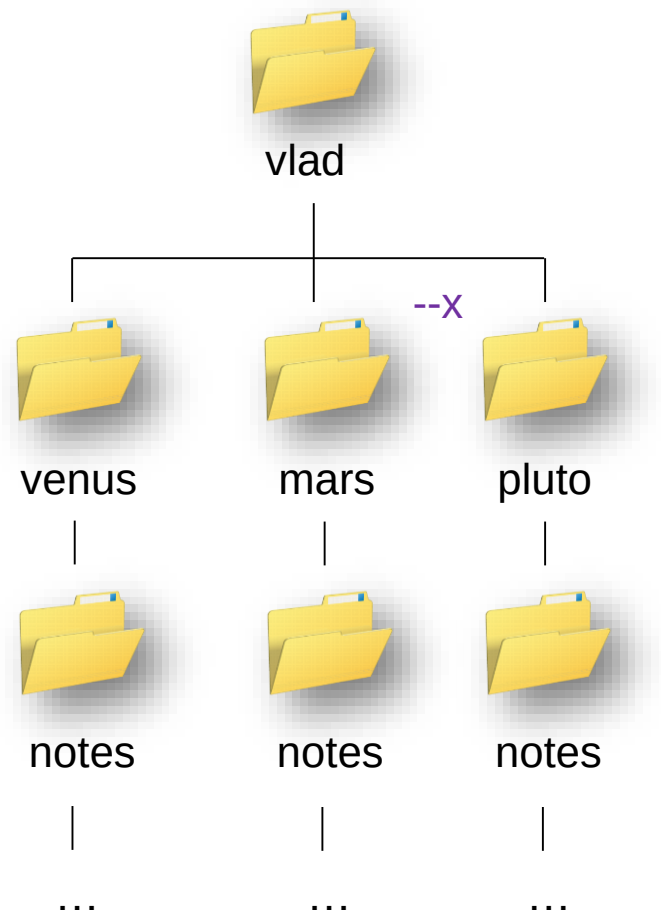
```
$ ls venus venus/notes ✓
$ ls mars mars/notes   ✓
$ ls pluto             ✗
```



What does "execute" mean for directories?

Gives the right to *traverse* the directory

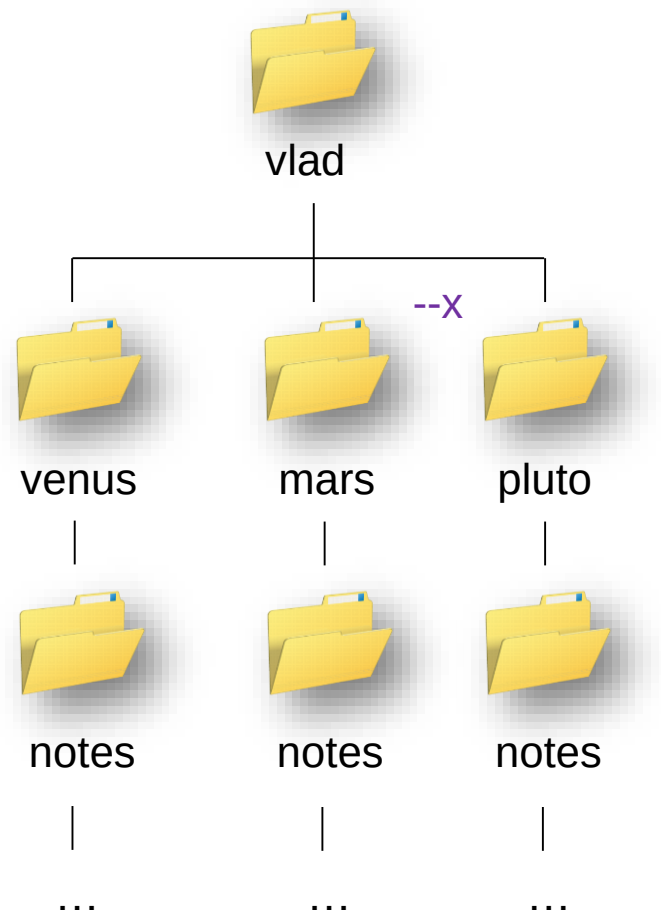
```
$ ls venus venus/notes ✓
$ ls mars mars/notes   ✓
$ ls pluto             ✗
$ ls pluto/notes
```



What does "execute" mean for directories?

Gives the right to *traverse* the directory

```
$ ls venus venus/notes ✓
$ ls mars mars/notes   ✓
$ ls pluto             ✗
$ ls pluto/notes       ✓
```



Change permission with `chmod` (change mode)

Change permission with `chmod` (change mode)

```
$ ls -l final.grd
-rwxrwxrwx 1 vlad bio  4215  2010-08-29 22:30 final.grd
```

Change permission with `chmod` (change mode)

```
$ ls -l final.grd
-rwxrwxrwx 1 vlad bio  4215  2010-08-29 22:30 final.grd
```

Everyone can read it

Change permission with `chmod` (change mode)

```
$ ls -l final.grd
-rwxrwxrwx 1 vlad bio  4215  2010-08-29 22:30 final.grd
```

Everyone can read it

Modify it

Change permission with `chmod` (change mode)

```
$ ls -l final.grd
-rwxrwxrwx 1 vlad bio  4215  2010-08-29 22:30 final.grd
```

Everyone can read it

Modify it

Try to run it (which probably doesn't make sense)

Change permission with `chmod` (change mode)

```
$ ls -l final.grd
-rwxrwxrwx 1 vlad bio  4215  2010-08-29 22:30 final.grd
$ chmod u=rw final.grd
$
```

Change permission with `chmod` (change mode)

```
$ ls -l final.grd
-rwxrwxrwx 1 vlad bio  4215  2010-08-29 22:30 final.grd
$ chmod u=rw final.grd
$
```

User (u) has read-write (rw)

Change permission with `chmod` (change mode)

```
$ ls -l final.grd
-rwxrwxrwx 1 vlad bio  4215  2010-08-29 22:30 final.grd
$ chmod u=rw final.grd
$ ls -l final.grd
-rw-rwxrwx 1 vlad bio  4215  2010-08-30 08:19 final.grd
$
```

Change permission with `chmod` (change mode)

```
$ ls -l final.grd
-rwxrwxrwx 1 vlad bio  4215  2010-08-29 22:30 final.grd
$ chmod u=rw final.grd
$ ls -l final.grd
-rw-rwxrwx 1 vlad bio  4215  2010-08-30 08:19 final.grd
$ chmod g=r final.grd; ls -l final.grd
-rw-r--rw- 1 vlad bio  4215  2010-08-30 08:19 final.grd
$
```

Change permission with `chmod` (change mode)

```
$ ls -l final.grd
-rwxrwxrwx 1 vlad bio  4215  2010-08-29 22:30 final.grd
$ chmod u=rw final.grd
$ ls -l final.grd
-rw-rwxrwx 1 vlad bio  4215  2010-08-30 08:19 final.grd
$ chmod g=r final.grd; ls -l final.grd
-rw-r--rw- 1 vlad bio  4215  2010-08-30 08:19 final.grd
$
```

Use ';' to put multiple commands on a single line

Change permission with `chmod` (change mode)

```
$ ls -l final.grd
-rwxrwxrwx 1 vlad bio  4215  2010-08-29 22:30 final.grd
$ chmod u=rw final.grd
$ ls -l final.grd
-rw-rwxrwx 1 vlad bio  4215  2010-08-30 08:19 final.grd
$ chmod g=r final.grd; ls -l final.grd
-rw-r--rw- 1 vlad bio  4215  2010-08-30 08:19 final.grd
$ chmod a= final.grd; ls -l final.grd
-rw-r----- 1 vlad bio  4215  2010-08-30 08:20 final.grd
```

Change permission with `chmod` (change mode)

```
$ ls -l final.grd
-rwxrwxrwx 1 vlad bio  4215  2010-08-29 22:30 final.grd
$ chmod u=rw final.grd
$ ls -l final.grd
-rw-rwxrwx 1 vlad bio  4215  2010-08-30 08:19 final.grd
$ chmod g=r final.grd; ls -l final.grd
-rw-r--rw- 1 vlad bio  4215  2010-08-30 08:19 final.grd
$ chmod a= final.grd; ls -l final.grd
-rw-r------ 1 vlad bio  4215  2010-08-30 08:20 final.grd
```

No permissions at all

Again, things are different on Windows

Again, things are different on Windows

Permissions defined by Access Control Lists (ACLs)

Again, things are different on Windows

Permissions defined by Access Control Lists (ACLs)

A list of (who, what) pairs

Again, things are different on Windows

Permissions defined by Access Control Lists (ACLs)

A list of (who, what) pairs

More flexible…

Again, things are different on Windows

Permissions defined by Access Control Lists (ACLs)

A list of (who, what) pairs

More flexible…

…but more complex to administer and understand

Again, things are different on Windows

Permissions defined by Access Control Lists (ACLs)

A list of (who, what) pairs

More flexible…

…but more complex to administer and understand

Some flavors of Unix provide ACLs, but hardly anyone uses them

Create your own commands

Create your own commands

```
$ cat > smallest
```

Create your own commands

```
$ cat > smallest
```

No input file specified, so read from keyboard

Create your own commands

```
$ cat > smallest
```

Send output to a file called `smallest`

Create your own commands

```
$ cat > smallest
wc -l *.pdb | sort | head -1
```

Create your own commands

```
$ cat > smallest
wc -l *.pdb | sort | head -1
^D
$
```

Create your own commands

```
$ cat > smallest
wc -l *.pdb | sort | head -1
^D
$
```

Ctrl-D means "end of input" in Unix

Create your own commands

```
$ cat > smallest
wc -l *.pdb | sort | head -1
^D
$
```

Ctrl-D means "end of input" in Unix

Ctrl-Z does the same thing in Windows

Create your own commands

```
$ cat > smallest
wc -l *.pdb | sort | head -1
^D
$ chmod u+x smallest
$
```

Create your own commands

```
$ cat > smallest
wc -l *.pdb | sort | head -1
^D
$ chmod u+x smallest
$
```

Give the user owner permission to run this file

Create your own commands

```
$ cat > smallest
wc -l *.pdb | sort | head -1
^D
$ chmod u+x smallest
$ ./smallest
```

Create your own commands

```
$ cat > smallest
wc -l *.pdb | sort | head -1
^D
$ chmod u+x smallest
$ ./smallest
```

Put ./ at the front to be sure of running the `smallest` that it's *this* directory

Create your own commands

```
$ cat > smallest
wc -l *.pdb | sort | head -1
^D
$ chmod u+x smallest
$ ./smallest
  9  methane.pdb
$
```

Create your own commands

```
$ cat > smallest
wc -l *.pdb | sort | head -1
^D
$ chmod u+x smallest
$ ./smallest
  9  methane.pdb
$
```

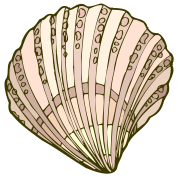
Try doing *that* with a desktop full of GUIs

The Unix Shell

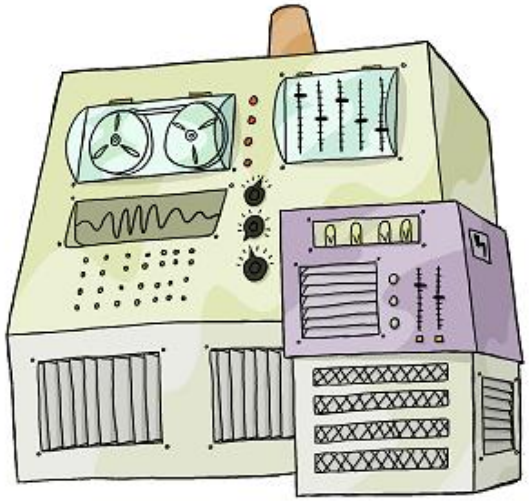
Finding Things

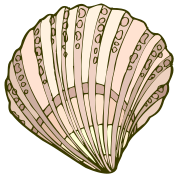
Created by Greg Wilson

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

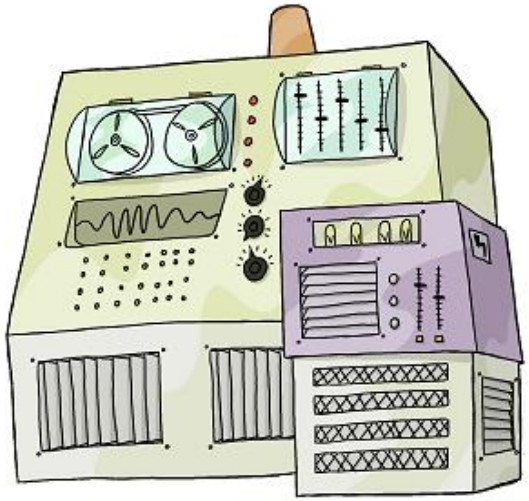


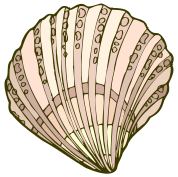
shell



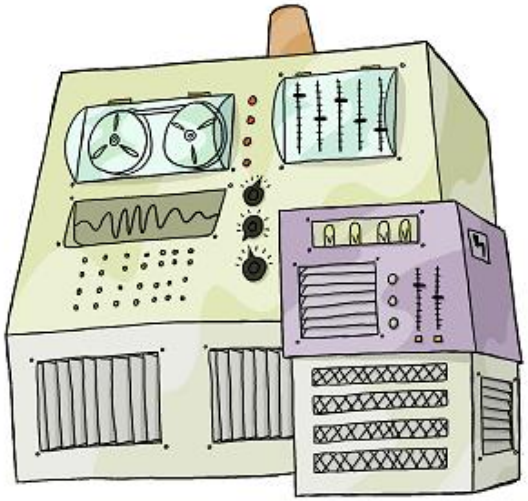


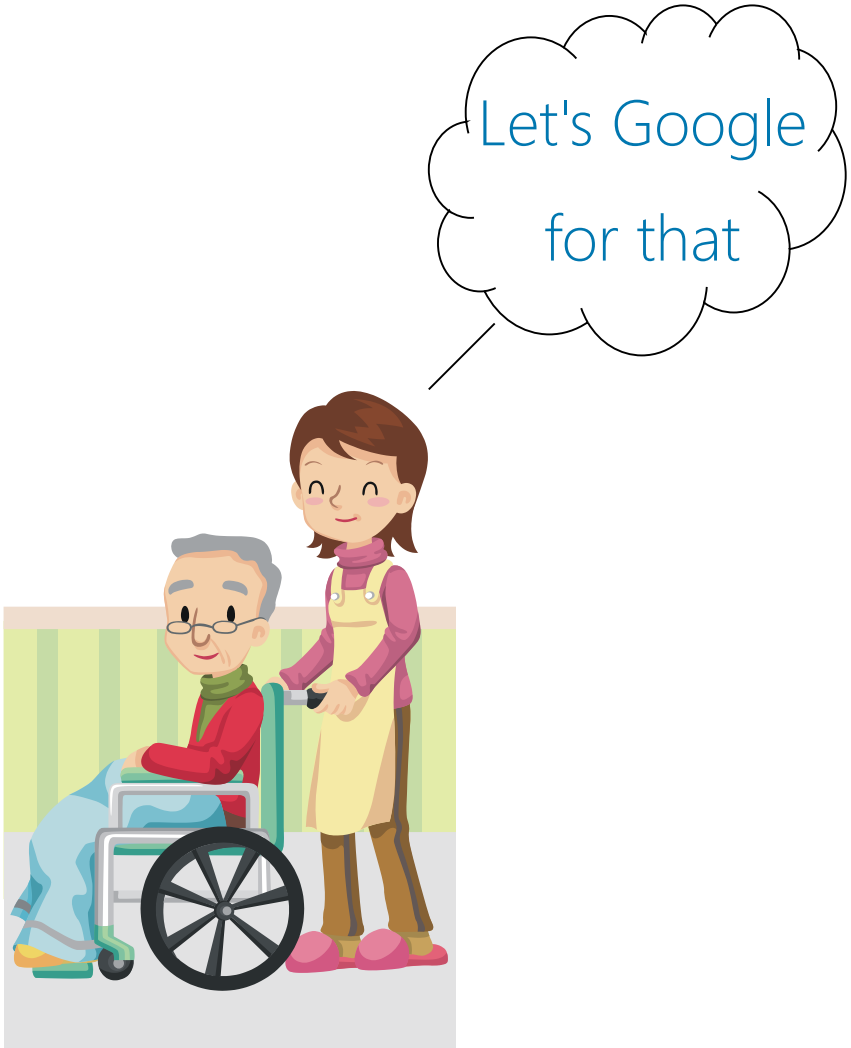
shell

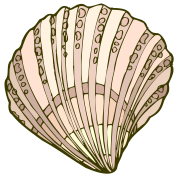




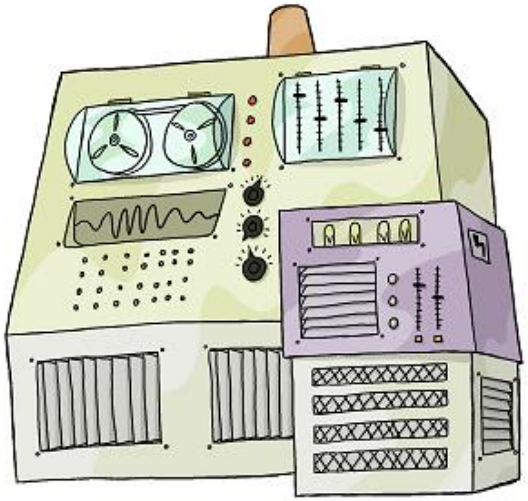
shell







shell



grep: global / regular expression / print

grep: global / regular expression / print

Finds and prints lines in files that match a pattern

grep: global / regular expression / print

Finds and prints lines in files that match a pattern

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

haiku.txt

grep: global / regular expression / print

Finds and prints lines in files that match a pattern

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

haiku.txt

```
$ grep not haiku.txt
```

grep: global / regular expression / print

Finds and prints lines in files that match a pattern

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

haiku.txt

```
$ grep not haiku.txt
```

Pattern

grep: global / regular expression / print

Finds and prints lines in files that match a pattern

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

haiku.txt

```
$ grep not haiku.txt
```

Pattern

Every letter matches itself

grep: global / regular expression / print

Finds and prints lines in files that match a pattern

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

haiku.txt

```
$ grep not haiku.txt
```

File(s)

grep: global / regular expression / print

Finds and prints lines in files that match a pattern

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

haiku.txt

```
$ grep not haiku.txt
Is not the true Tao, until
"My Thesis" not found
Today it is not working
$
```

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

```
$ grep day haiku.txt
Yesterday it worked
Today it is not working
$
```

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

```
$ grep day haiku.txt
Yesterday it worked
Today it is not working
$ grep -w day haiku.txt
$
```

Match whole words

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

```
$ grep day haiku.txt
Yesterday it worked
Today it is not working
$ grep -w day haiku.txt
$ grep -n it haiku.txt
```

Prefix matches with line numbers

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

```
$ grep day haiku.txt
Yesterday it worked
Today it is not working
$ grep -w day haiku.txt
$ grep -n it haiku.txt
5:With searching comes loss
9:Yesterday it worked
10:Today it is not working
$
```

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

```
$ grep day haiku.txt
Yesterday it worked
Today it is not working
$ grep -w day haiku.txt
$ grep -n it haiku.txt
5:With searching comes loss
9:Yesterday it worked
10:Today it is not working
$ grep -w -n it haiku.txt
```

Use multiple flags
to combine effects

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

```
$ grep day haiku.txt
Yesterday it worked
Today it is not working
$ grep -w day haiku.txt
$ grep -n it haiku.txt
5:With searching comes loss
9:Yesterday it worked
10:Today it is not working
$ grep -w -n it haiku.txt
9:Yesterday it worked
10:Today it is not working
$
```

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

```
$ grep -i -v the haiku.txt
You bring fresh toner.

With searching comes loss

Yesterday it worked
Today it is not working
Software is like that.
$
```

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

```
$ grep -i -v the haiku.txt
You bring fresh toner.

With searching comes loss

Yesterday it worked
Today it is not working
Software is like that.
$
```

-i case insensitive

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

```
$ grep -i -v the haiku.txt
You bring fresh toner.

With searching comes loss

Yesterday it worked
Today it is not working
Software is like that.
$
```

-i case insensitive

-v invert and print non-matches

Many more options

Many more options

Use `man grep` to get help

Many more options

Use `man` `grep` to get help

manual

Many more options

Use `man grep` to get help

Complex patterns use regular expressions

Many more options

Use `man grep` to get help

Complex patterns use regular expressions

(The 're' in `grep`)

Many more options

Use `man grep` to get help

Complex patterns use regular expressions

(The 're' in `grep`)

Ideas are covered in a separate lecture

Many more options

Use `man grep` to get help

Complex patterns use regular expressions

(The 're' in `grep`)

Ideas are covered in a separate lecture

`grep`'s regular expressions are slightly different

from those provided in most programming languages

Many more options

Use `man grep` to get help

Complex patterns use regular expressions

(The 're' in `grep`)

Ideas are covered in a separate lecture

`grep`'s regular expressions are slightly different

from those provided in most programming languages

But the ideas are the same

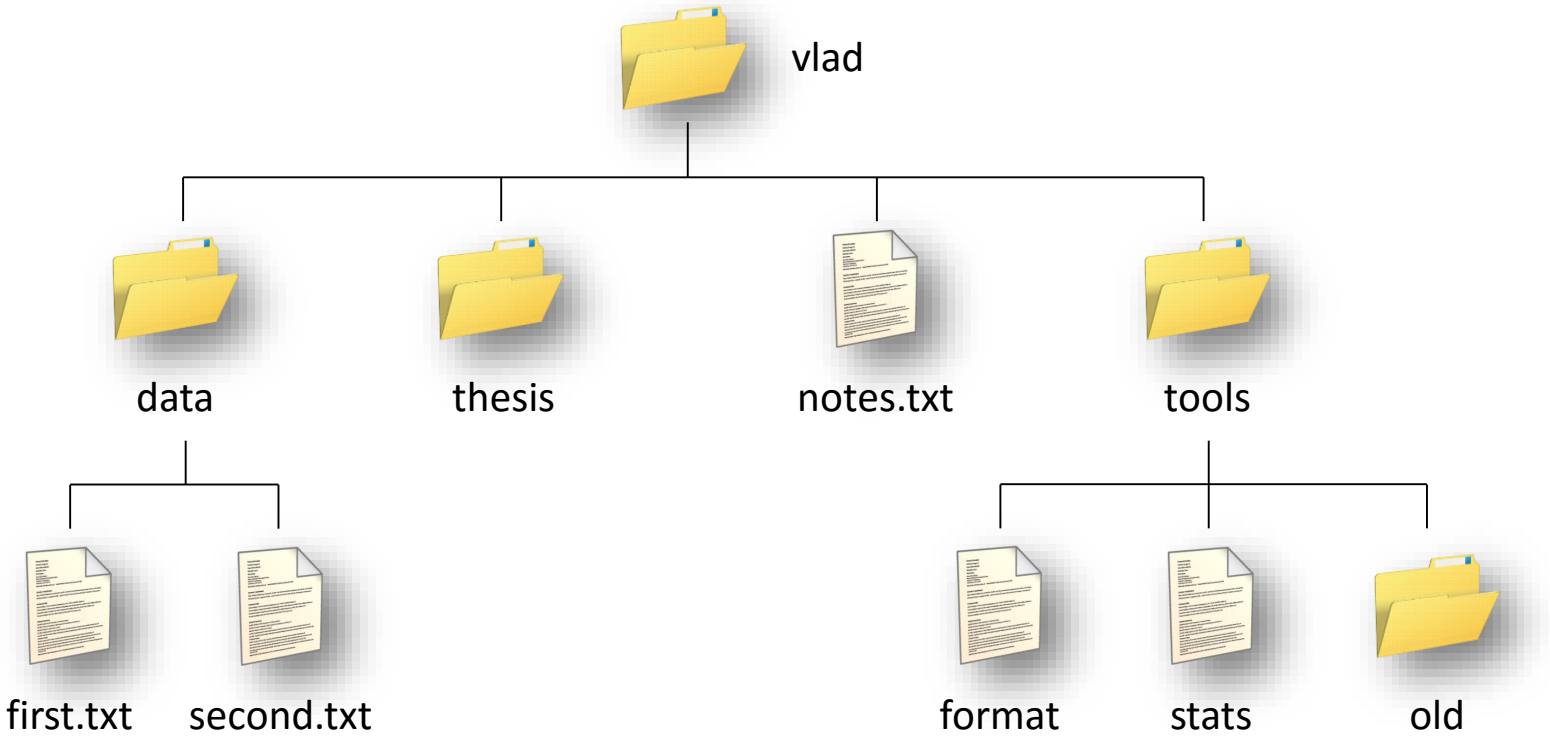
`find`: finds files (rather than lines in files)

`find`: finds files (rather than lines in files)

Again, too many options to cover here

`find`: finds files (rather than lines in files)

Again, too many options to cover here



`find`: finds files (rather than lines in files)

Again, too many options to cover here

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

Output of tree

`find`: finds files (rather than lines in files)

Again, too many options to cover here

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

Output of tree

Trailing / shows directories

`find`: finds files (rather than lines in files)

Again, too many options to cover here

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

Output of tree

Trailing / shows directories

Trailing * shows executables

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -type d
```

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -type d
```

Root directory of search

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -type d
```

Things of type 'd' (directory)

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -type d
./
./data
./thesis
./tools
./tools/old
$
```

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -type d
./
./data
./thesis
./tools
./tools/old
$ find . -type f
./data/first.txt
./data/second.txt
./notes.txt
./tools/format
./tools/stats
$
```

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -maxdepth 1 -type f
./notes.txt
$
```

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -maxdepth 1 -type f
./notes.txt
$ find . -mindepth 2 -type f
./data/first.txt
./data/second.txt
./tools/format
./tools/stats
$
```

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -maxdepth 1 -type f
./notes.txt
$ find . -mindepth 2 -type f
./data/first.txt
./data/second.txt
./tools/format
./tools/stats
$ find . -empty
./thesis
./tools/old
$
```

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -perm -u=x
./data
./thesis
./tools
./tools/format
./tools/old
./tools/stats
$
```

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -perm -u=x
./data
./thesis
./tools
./tools/format
./tools/old
./tools/stats
$ find . -perm -u=x -type f
./tools/format
./tools/stats
$
```

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -name *.txt
./notes.txt
$
```

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -name *.txt
./notes.txt
$
```

* expanded by shell *before* command runs

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -name notes.txt
./notes.txt
$
```

* expanded by shell *before* command runs

This is the actual command

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -name *.txt
./notes.txt
$ find . -name '*.txt'
```

Single quotes prevent shell from expanding wildcards

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -name *.txt
./notes.txt
$ find . -name '*.txt'
```

Single quotes prevent shell from expanding wildcards

So find gets the pattern

```
./
+-- data/
|   +-- first.txt
|   +-- second.txt
+-- notes.txt
+-- thesis/
+-- tools/
    +-- format*
    +-- old/
    +-- stats*
```

```
$ find . -name *.txt
./notes.txt
$ find . -name '*.txt'
./data/first.txt
./data/second.txt
./notes.txt
$
```

The command line's power lies in *combining* tools

The command line's power lies in *combining* tools

```
$ find . -name '*.txt'
./data/first.txt
./data/second.txt
./notes.txt
$
```

The command line's power lies in *combining* tools

```
$ find . -name '*.txt'
./data/first.txt
./data/second.txt
./notes.txt
$ wc -l `find . -name '*.txt'`
```

The command line's power lies in *combining* tools

```
$ find . -name '*.txt'
./data/first.txt
./data/second.txt
./notes.txt
$ wc -l `find . -name '*.txt'`
```

Back quotes

The command line's power lies in *combining* tools

```
$ find . -name '*.txt'
./data/first.txt
./data/second.txt
./notes.txt
$ wc -l `find . -name '*.txt'`
```

Back quotes

Replace what's inside with output from running that command

The command line's power lies in *combining* tools

```
$ find . -name '*.txt'
./data/first.txt
./data/second.txt
./notes.txt
$ wc -l `find . -name '*.txt'`
```

Back quotes

Replace what's inside with output from running that command

Like wildcards * and ?, but more flexible

The command line's power lies in *combining* tools

```
$ find . -name '*.txt'
./data/first.txt
./data/second.txt
./notes.txt
$ wc -l `find . -name '*.txt'`
./data/first.txt ./data/second.txt ./notes.txt
```

The command line's power lies in *combining* tools

```
$ find . -name '*.txt'
./data/first.txt
./data/second.txt
./notes.txt
$ wc -l `find . -name '*.txt'`
```

```
$ wc -l ./data/first.txt ./data/second.txt ./notes.txt
```

The command line's power lies in *combining* tools

```
$ find . -name '*.txt'
./data/first.txt
./data/second.txt
./notes.txt
$ wc -l `find . -name '*.txt'`
  70  ./data/first.txt
 420  ./data/second.txt
  30  ./notes.txt
 520  total
$
```

Use `find` and `grep` together

Use `find` and `grep` together

```
$ grep FE `find . -name '*.pdb'`
./human/heme.pdb:ATOM  25  FE  1  -0.924  0.535  -0.518
$
```

What if your data isn't text?

What if your data isn't text?

Images, databases, spreadsheets…

What if your data isn't text?

Images, databases, spreadsheets…

1. Teach standard tools about all these formats

What if your data isn't text?

Images, databases, spreadsheets…

1. Teach standard tools about all these formats

   Hasn't happened, and probably won't

What if your data isn't text?

Images, databases, spreadsheets…

1. Teach standard tools about all these formats

   Hasn't happened, and probably won't

2. Convert data to text (or extract text from data)

What if your data isn't text?

Images, databases, spreadsheets…

1. Teach standard tools about all these formats

   Hasn't happened, and probably won't

2. Convert data to text (or extract text from data)

   Simple things are easy

What if your data isn't text?

Images, databases, spreadsheets…

1. Teach standard tools about all these formats

   Hasn't happened, and probably won't

2. Convert data to text (or extract text from data)

   Simple things are easy

   Complex things are impossible

What if your data isn't text?

Images, databases, spreadsheets…

1. Teach standard tools about all these formats

   Hasn't happened, and probably won't

2. Convert data to text (or extract text from data)

   Simple things are easy

   Complex things are impossible

3. Use a programming language

What if your data isn't text?

Images, databases, spreadsheets…

1. Teach standard tools about all these formats

   Hasn't happened, and probably won't

2. Convert data to text (or extract text from data)

   Simple things are easy

   Complex things are impossible

3. Use a programming language

   Many have borrowed ideas from the shell

FAQTS – the Game

- Frequently Asked Questions with Tiny Sentences
- Both Q and A with least possible words
- The ideal word count for answers is *two*
- Our second round:

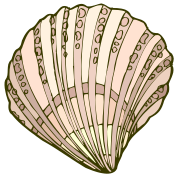
What is  model ?

The Unix Shell

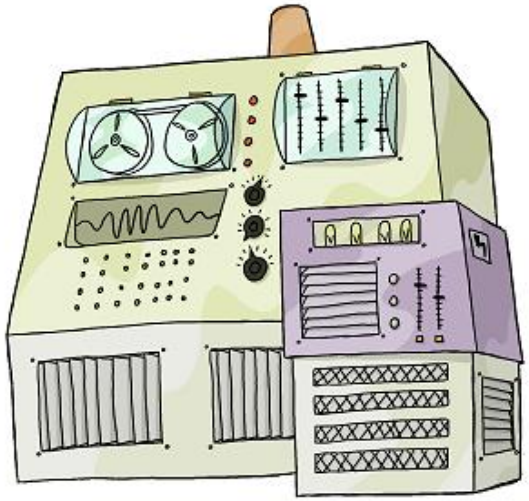
Job Control

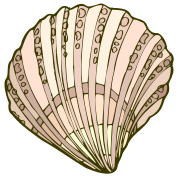
Created by Greg Wilson

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY



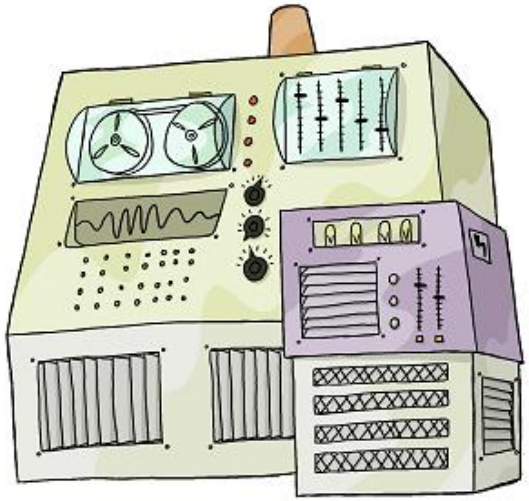
shell

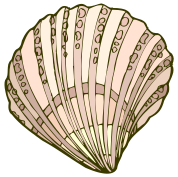




shell

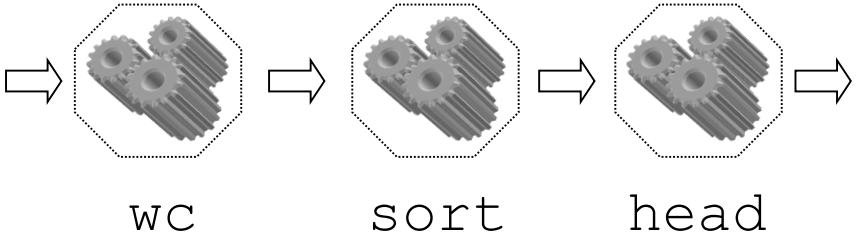
```
$ wc -l *.pdb | sort | head -1
```

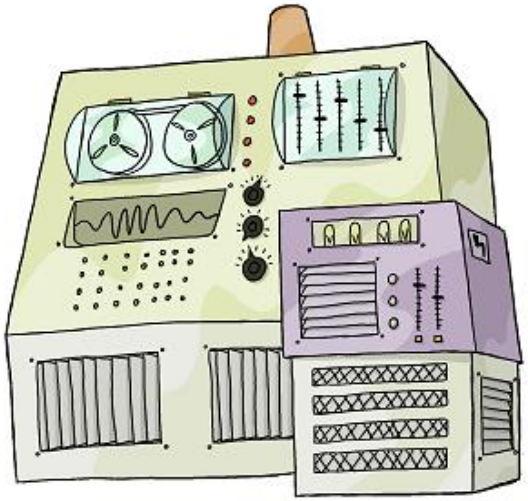


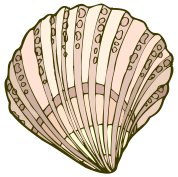


shell

```
$ wc -l *.pdb | sort | head -1
```

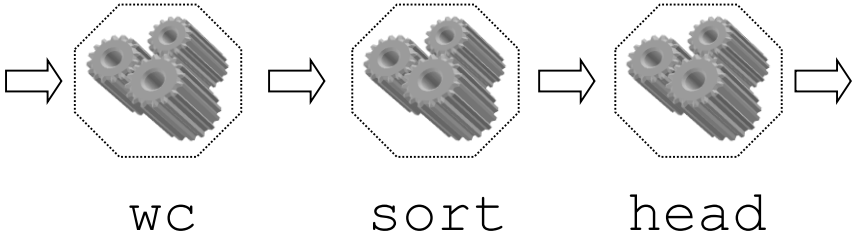


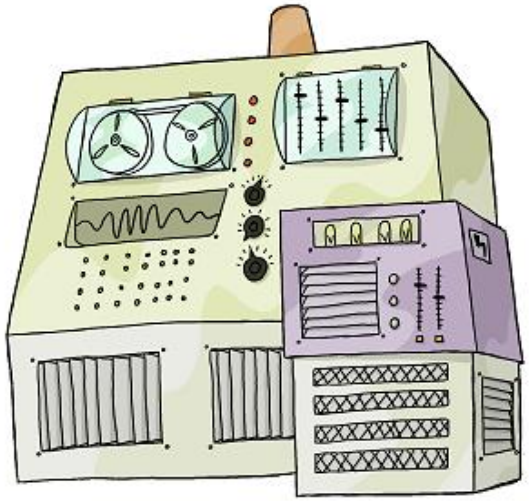




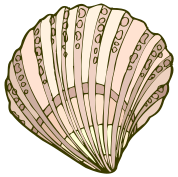
shell

```
$ wc -l *.pdb | sort | head -1
```



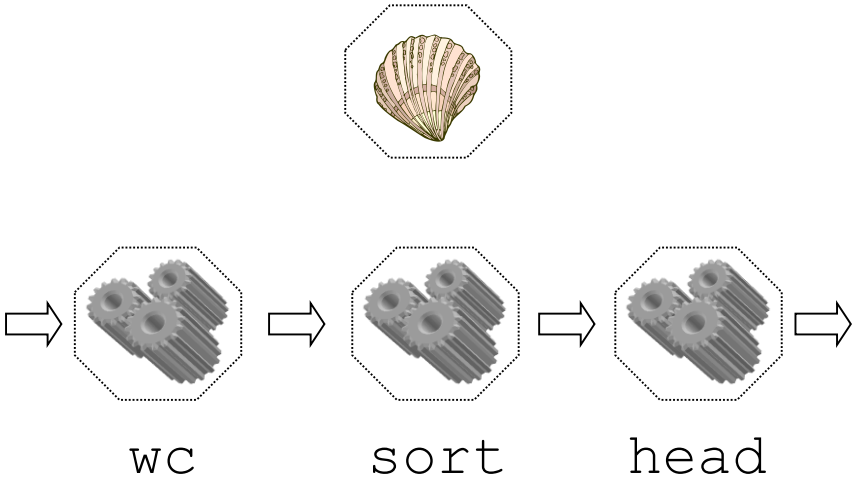


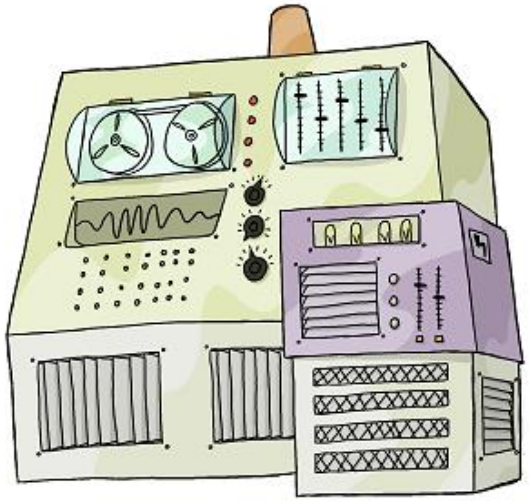
Control programs while they run



shell

```
$ wc -l *.pdb | sort | head -1
```





Control ~~programs~~ processes while they run

A *process* is a running program

A *process* is a running program

Some are yours

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

```
$ ps
PID     PPID    PGID    TTY    UID      STIME    COMMAND
2152       1    2152    con   1000   13:19:07    /usr/bin/bash
2276    2152    2276    con   1000   14:53:48    /usr/bin/ps
$
```

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

```
$ ps
PID     PPID    PGID    TTY    UID      STIME    COMMAND
2152       1    2152    con   1000   13:19:07    /usr/bin/bash
2276    2152    2276    con   1000   14:53:48    /usr/bin/ps
$
```

Process ID (unique at any moment)

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

```
$ ps
PID     PPID    PGID    TTY    UID      STIME    COMMAND
2152       1    2152    con   1000   13:19:07    /usr/bin/bash
2276    2152    2276    con   1000   14:53:48    /usr/bin/ps
$
```

Parent process ID

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

```
$ ps
PID     PPID    PGID    TTY    UID      STIME    COMMAND
2152       1    2152    con   1000   13:19:07    /usr/bin/bash
2276    2152    2276    con   1000   14:53:48    /usr/bin/ps
$
```

Parent process ID

What process created this one?

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

```
$ ps
PID     PPID    PGID    TTY    UID      STIME    COMMAND
2152       1    2152    con   1000   13:19:07    /usr/bin/bash
2276    2152    2276    con   1000   14:53:48    /usr/bin/ps
$
```

Process group ID

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

```
$ ps
PID     PPID    PGID    TTY    UID      STIME    COMMAND
2152       1    2152    con   1000   13:19:07    /usr/bin/bash
2276    2152    2276    con   1000   14:53:48    /usr/bin/ps
$
```

What terminal (TTY) is it running in?

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

```
$ ps
PID     PPID    PGID    TTY    UID      STIME    COMMAND
2152       1    2152    con   1000   13:19:07    /usr/bin/bash
2276    2152    2276    con   1000   14:53:48    /usr/bin/ps
$
```

What terminal (TTY) is it running in?

'?' indicates a system service (no TTY)

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

```
$ ps
PID     PPID    PGID    TTY    UID      STIME    COMMAND
2152       1    2152    con   1000   13:19:07    /usr/bin/bash
2276    2152    2276    con   1000   14:53:48    /usr/bin/ps
$
```

The user ID of the process's owner

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

```
$ ps
PID     PPID    PGID    TTY    UID      STIME    COMMAND
2152       1    2152    con   1000   13:19:07    /usr/bin/bash
2276    2152    2276    con   1000   14:53:48    /usr/bin/ps
$
```

The user ID of the process's owner

Controls what the process can read, write, execute, …

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

```
$ ps
PID     PPID    PGID    TTY    UID     STIME    COMMAND
2152       1    2152    con   1000   13:19:07   /usr/bin/bash
2276    2152    2276    con   1000   14:53:48   /usr/bin/ps
$
```

When the process was started

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

```
$ ps
PID     PPID    PGID    TTY    UID      STIME    COMMAND
2152       1    2152    con   1000   13:19:07    /usr/bin/bash
2276    2152    2276    con   1000   14:53:48    /usr/bin/ps
$
```

The program the process is executing

Can stop, pause, and resume running processes

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
```

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
```

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$
```

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$
```

Stop the running program

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$
```

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$
```

Run in the background

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$
```

Run in the *background*

Shell returns right away instead of waiting for the program to finish

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$
```

Can run other programs in the *foreground* while waiting for background process(es) to finish

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$ jobs
[1] ./analyze results01.dat results02.dat results03.dat
$
```

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$ jobs
[1] ./analyze results01.dat results02.dat results03.dat
$
```

Show background processes

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$ jobs
[1] ./analyze results01.dat results02.dat results03.dat
$ fg
```

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$ jobs
[1] ./analyze results01.dat results02.dat results03.dat
$ fg
```

Bring background job to foreground

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$ jobs
[1] ./analyze results01.dat results02.dat results03.dat
$ fg
```

Bring background job to foreground

Use `fg %1`, `fg %2`, etc. if there are several background jobs

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$ jobs
[1] ./analyze results01.dat results02.dat results03.dat
$ fg
...a few minutes pass...
$
```

← And finally it's done

Use `^Z` to pause a program that's already running

Use `^Z` to pause a program that's already running

`fg` to resume it in the foreground

Use `^Z` to pause a program that's already running

`fg` to resume it in the foreground

Or `bg` to resume it as a background job

Use `^Z` to pause a program that's already running

`fg` to resume it in the foreground

Or `bg` to resume it as a background job

```
$ ./analyze results01.dat
```

Use `^Z` to pause a program that's already running

`fg` to resume it in the foreground

Or `bg` to resume it as a background job

```
$ ./analyze results01.dat
^Z
[1]  Stopped    ./analyze results01.dat
$
```

Use `^Z` to pause a program that's already running

`fg` to resume it in the foreground

Or `bg` to resume it as a background job

```
$ ./analyze results01.dat
^Z
[1]  Stopped    ./analyze results01.dat
$ bg %1
$
```

Use `^Z` to pause a program that's already running

`fg` to resume it in the foreground

Or `bg` to resume it as a background job

```
$ ./analyze results01.dat
^Z
[1]  Stopped    ./analyze results01.dat
$ bg %1
$ jobs
[1] ./analyze results01.dat
$
```

Use `^Z` to pause a program that's already running

`fg` to resume it in the foreground

Or `bg` to resume it as a background job

```
$ ./analyze results01.dat
^Z
[1]  Stopped    ./analyze results01.dat
$ bg %1
$ jobs
[1] ./analyze results01.dat
$ kill %1
$
```

Job control mattered a lot when users only had one terminal window

Job control mattered a lot when users only had one terminal window

Less important now: just open another window

Job control mattered a lot when users only had one terminal window

Less important now: just open another window

Still useful when running programs remotely

QOTD

- The best way to predict the future is to invent it!

Alan Kay

(at a 1971 meeting of PARC)

American computer scientist