

# STATS 418 - WEEK 2

TOOLS IN DATA SCIENCE, UNIX AND THE CLOUD

EDUARDO ARIÑO DE LA RUBIA  
CHIEF DATA SCIENTIST, DOMINO DATA LAB  
[EARINO@GMAIL.COM](mailto:EARINO@GMAIL.COM)



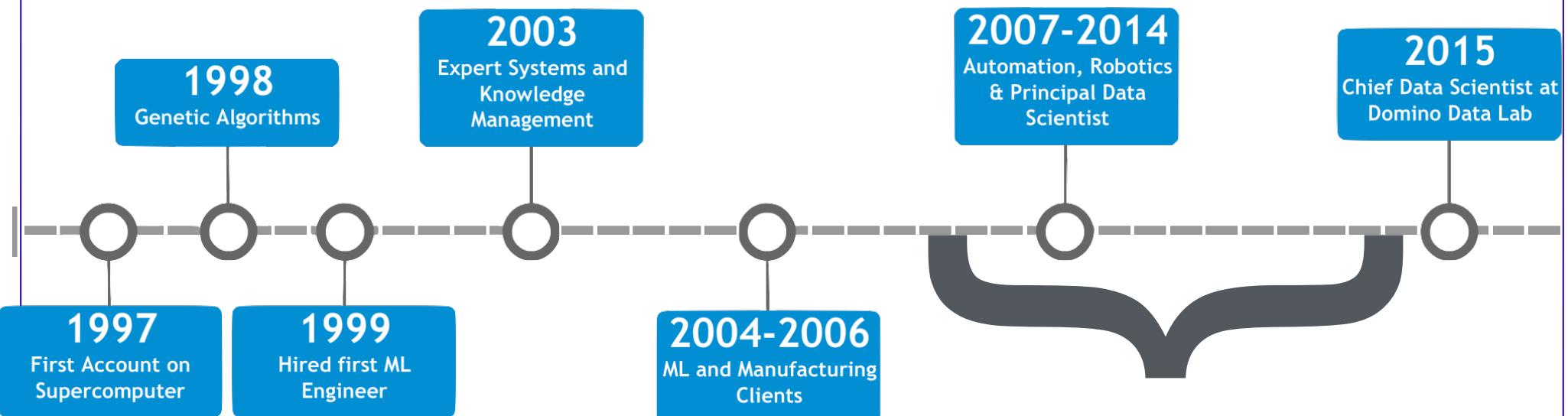
A BIT ABOUT ME



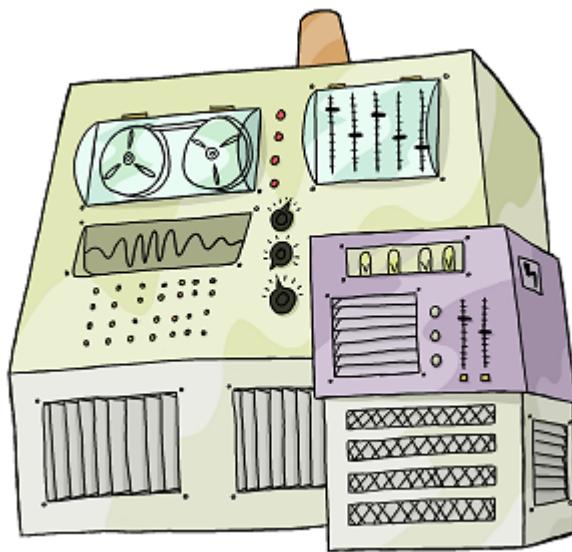
# DATA SCIENTIST

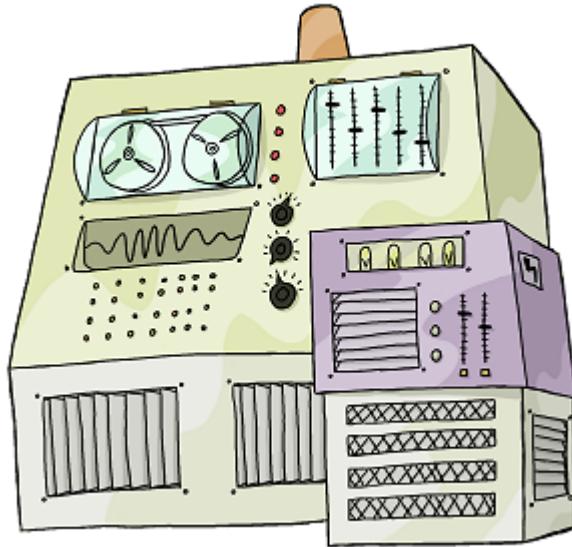
## PICTURE SLIDE





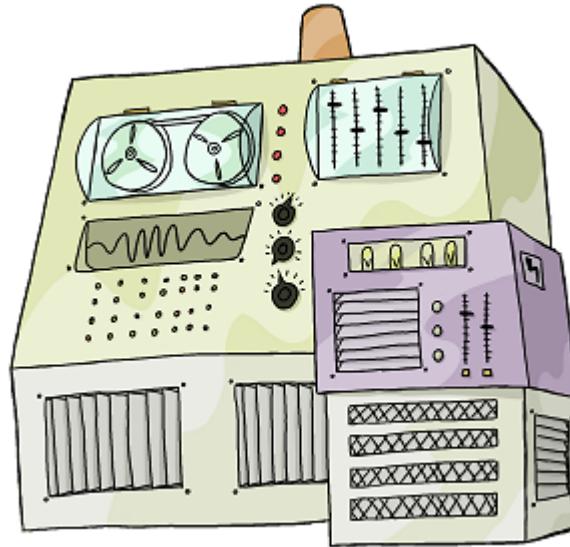
Robotics, Vision Systems  
Job Shop Scheduling,  
Optimization/Ops,  
Neural Networks, NLP





Run

Programs

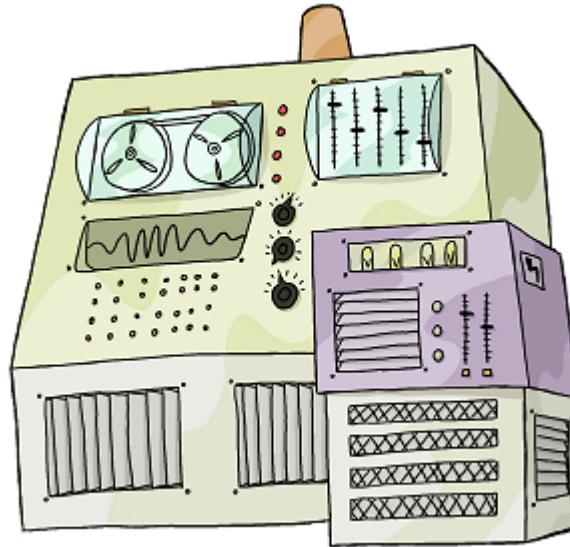


Run

Programs

Store

Data



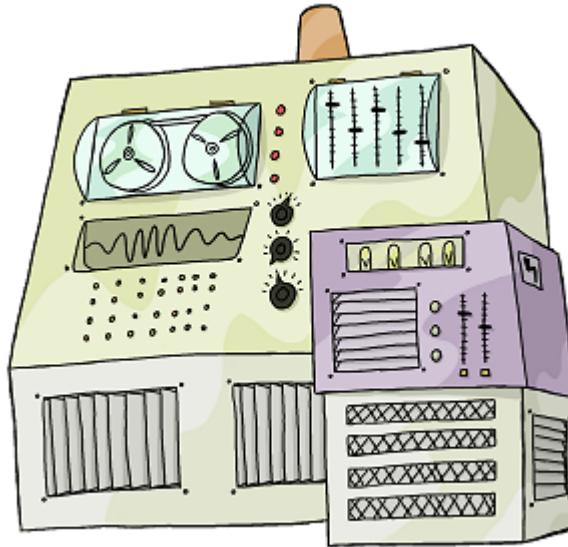
Run

Programs

Store

Data

Communicate  
with each other



Run

Programs

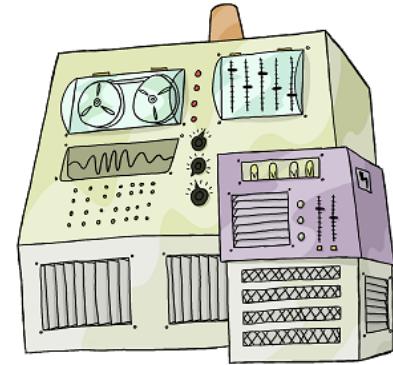
Communicate  
with each other

Store

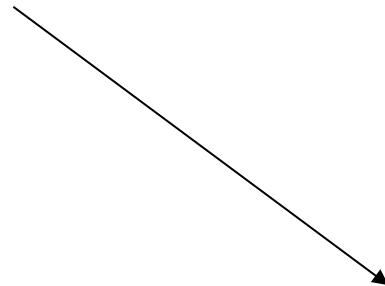
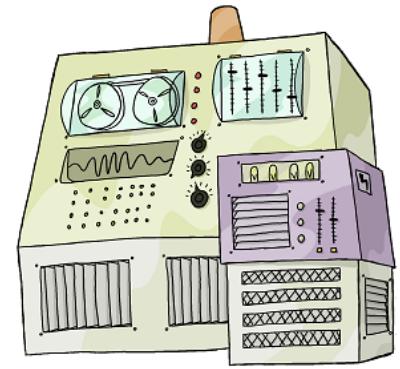
Data

Interact  
with us

Interact  
with us

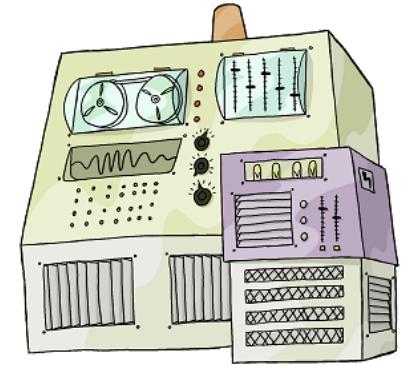


Interact  
with us



Telepathy

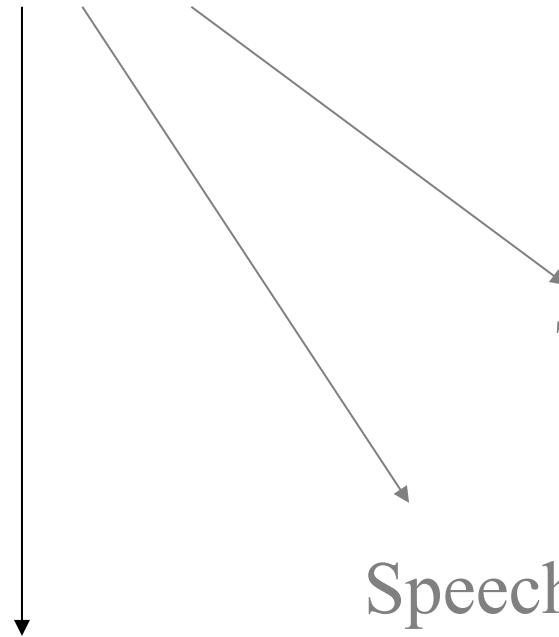
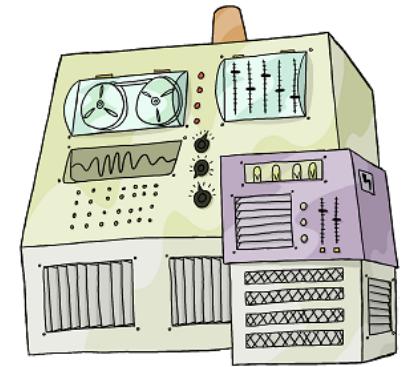
Interact  
with us



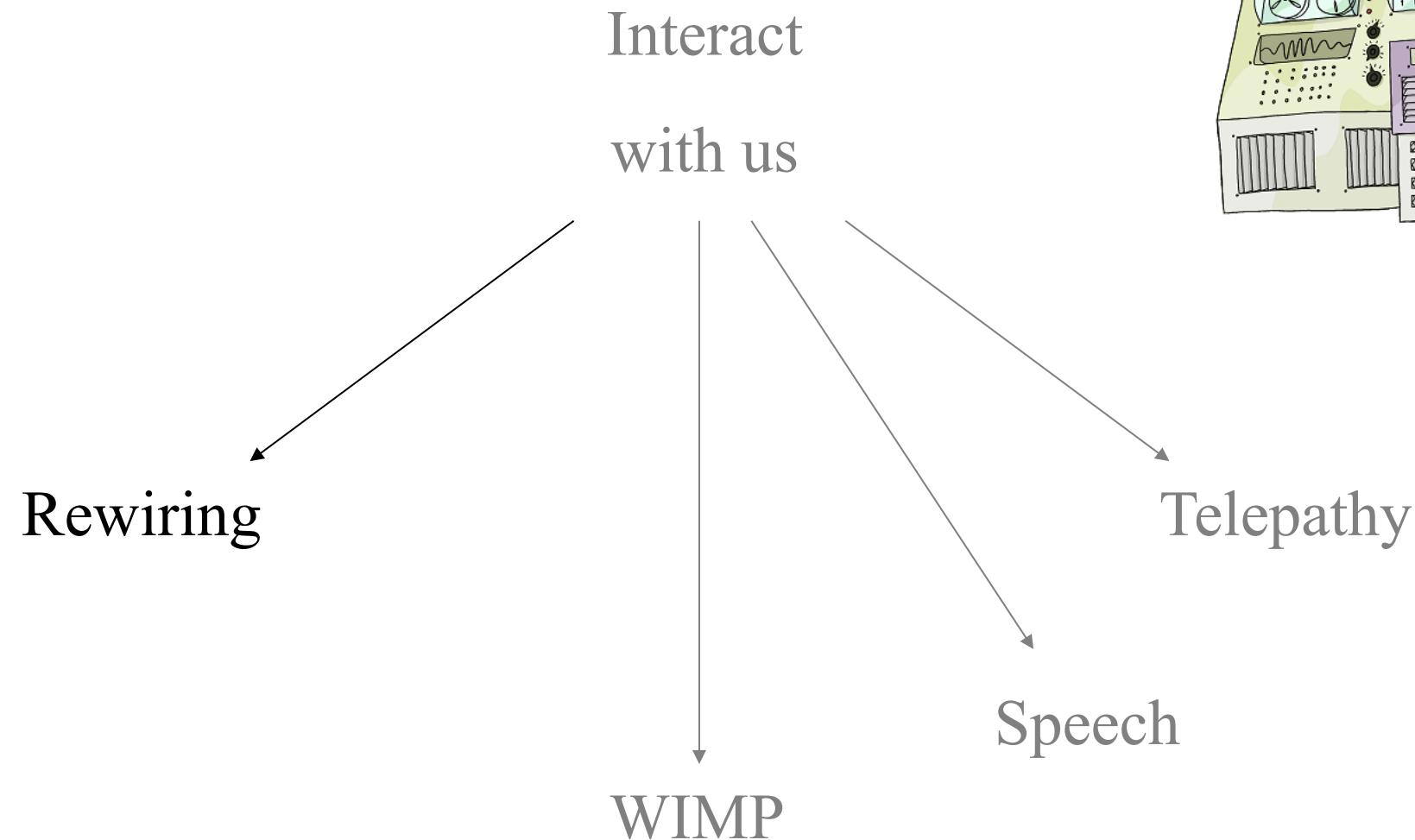
Telepathy

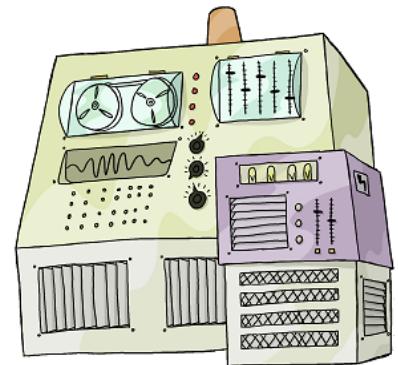
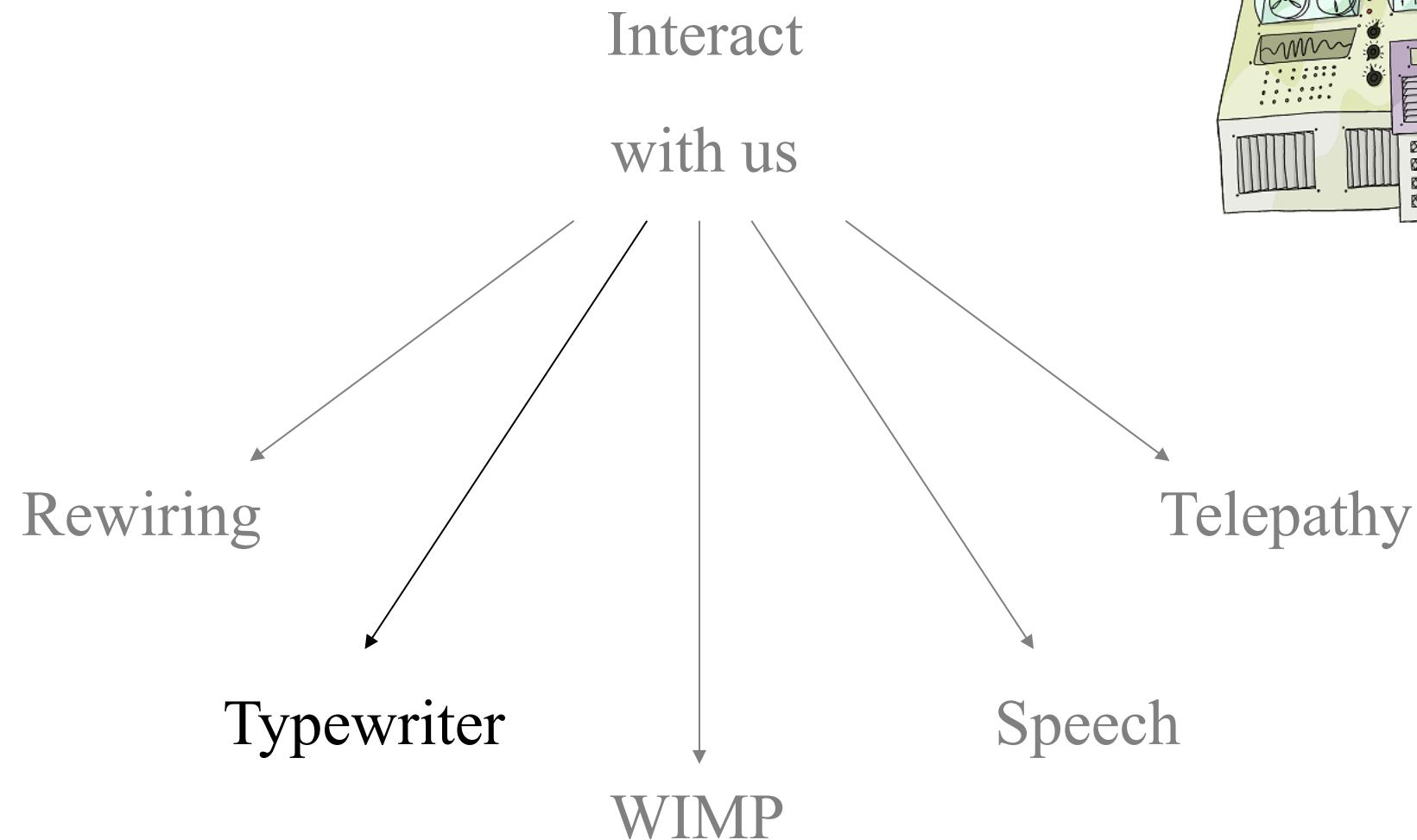
Speech

Interact  
with us

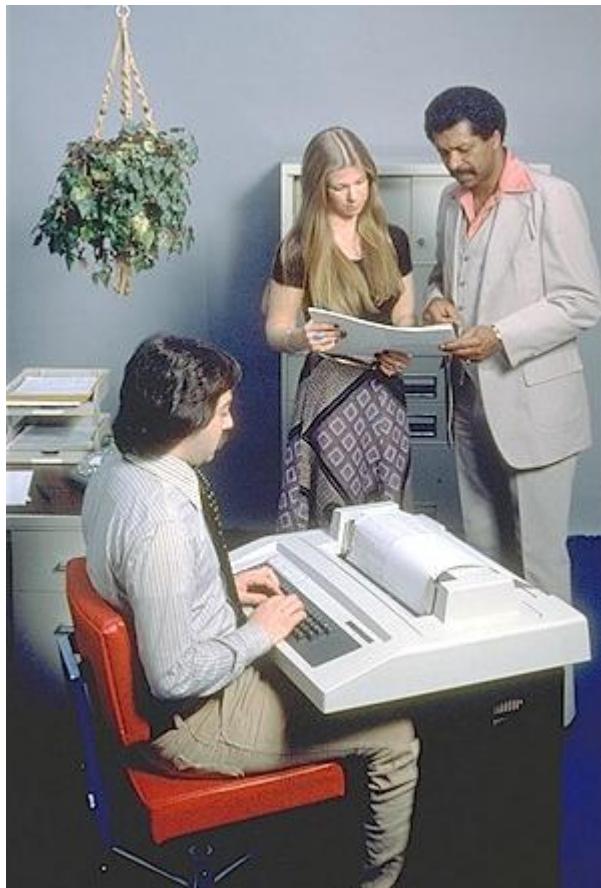


WIMP  
(windows, icons, mice, pointers)



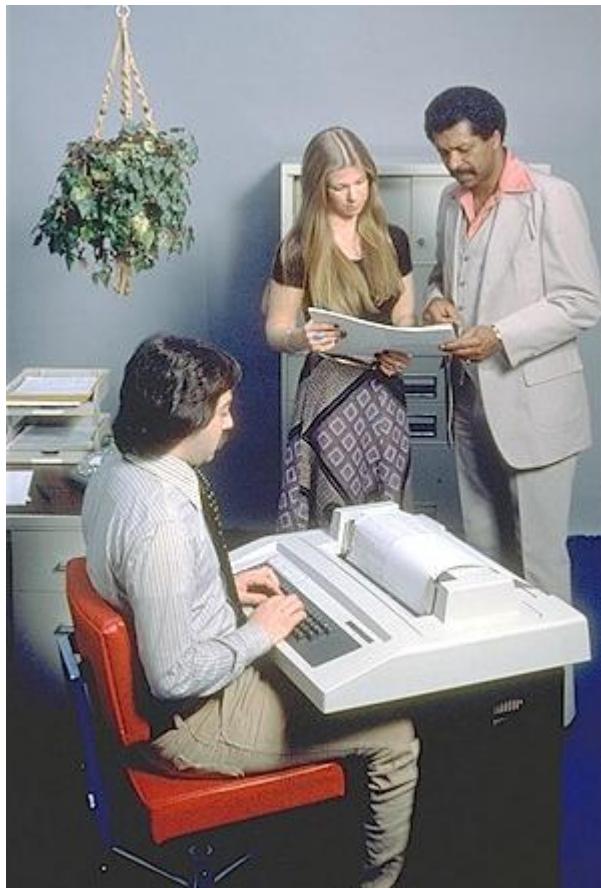


# Typewriter



~~Typewriter~~

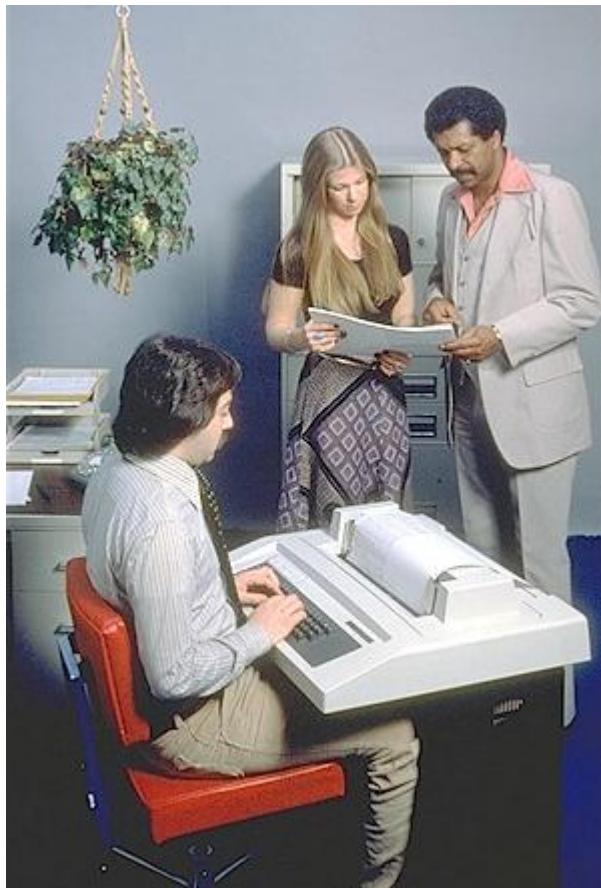
Line printer + keyboard



~~Typewriter~~

Line printer + keyboard

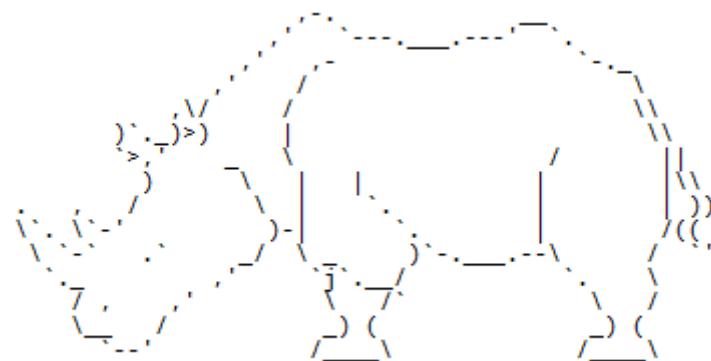
Text only

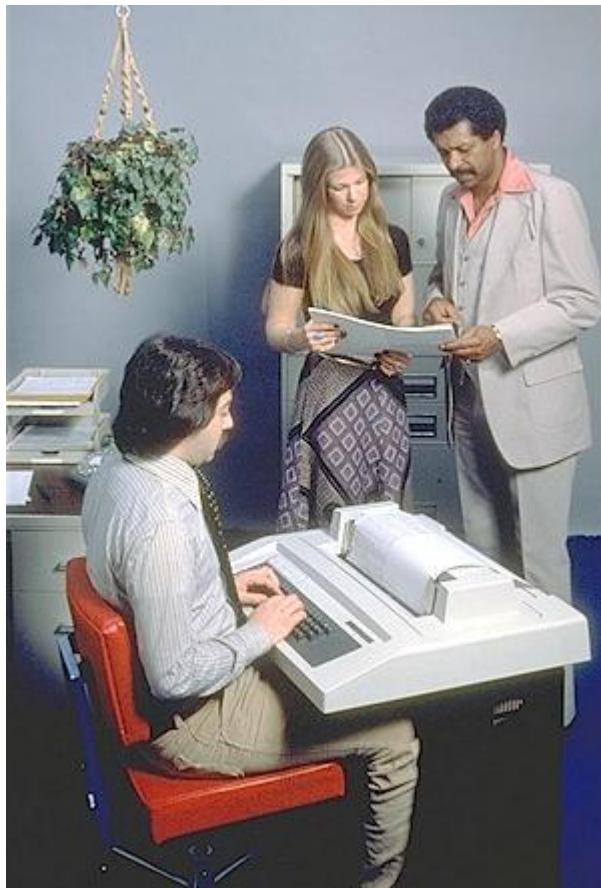


~~Typewriter~~

Line printer + keyboard

Text only

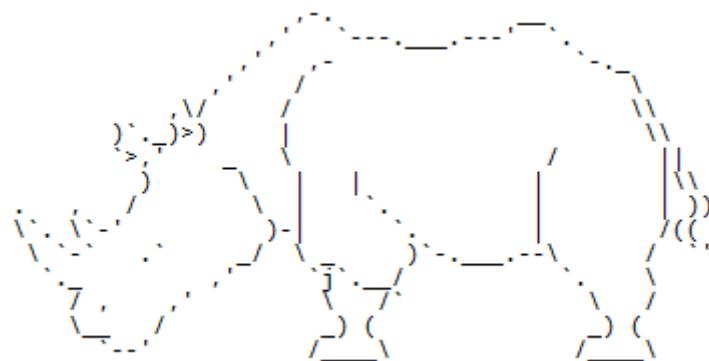




~~Typewriter~~

Line printer + keyboard

Text only



CLUI: command-line user interface

user logs in



user logs in

user types command



user logs in  
user types command  
computer executes command  
and prints output



user logs in

user types command

computer executes command  
and prints output

user types another command



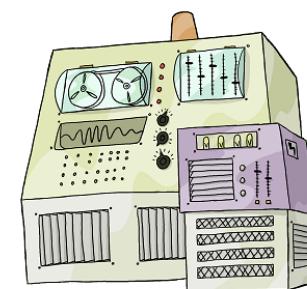
user logs in  
user types command  
computer executes command  
and prints output  
user types another command  
computer executes command  
and prints output



user logs in  
user types command  
computer executes command  
and prints output  
user types another command  
computer executes command  
and prints output  
:  
user logs off



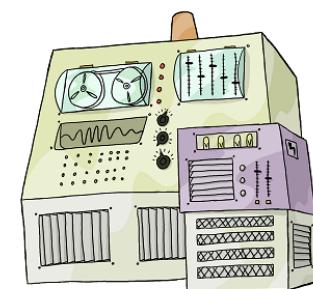
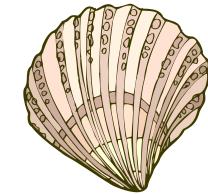
user logs in  
user types command  
computer executes command  
and prints output  
user types another command  
computer executes command  
and prints output  
:  
user logs off



user logs in  
user types command  
computer executes command  
and prints output  
user types another command  
computer executes command  
and prints output  
:  
user logs off

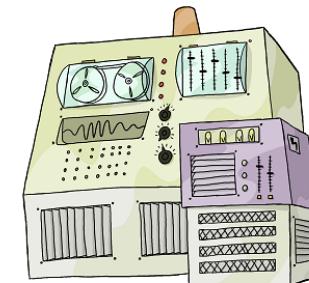
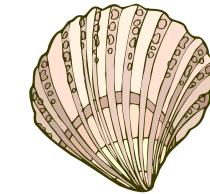


shell



user logs in  
user types command  
computer executes command  
and prints output  
user types another command  
computer executes command  
and prints output  
:  
user logs off

shell



A shell is just a program that runs other programs

A shell is just a program that runs other programs

Most popular is bash (the Bourne again shell)

A shell is just a program that runs other programs

Most popular is bash (the Bourne again shell)





A shell is just a program that runs other programs

Most popular is bash (the Bourne again shell)

Using it feels a lot more like programming

than using windows, a mouse, etc.



A shell is just a program that runs other programs

Most popular is bash (the Bourne again shell)

Using it feels a lot more like programming

than using windows, a mouse, etc.

Commands are terse and often cryptic



A shell is just a program that runs other programs

Most popular is bash (the Bourne again shell)

Using it feels a lot more like programming

than using windows, a mouse, etc.

Commands are terse and often cryptic

Use it because:

A shell is just a program that runs other programs

Most popular is bash (the Bourne again shell)



Using it feels a lot more like programming

than using windows, a mouse, etc.

Commands are terse and often cryptic

Use it because:

- many tools only have command-line interfaces



A shell is just a program that runs other programs

Most popular is bash (the Bourne again shell)

Using it feels a lot more like programming

than using windows, a mouse, etc.

Commands are terse and often cryptic

Use it because:

- many tools only have command-line interfaces
- allows you to combine tools in powerful new ways



# The Unix Shell

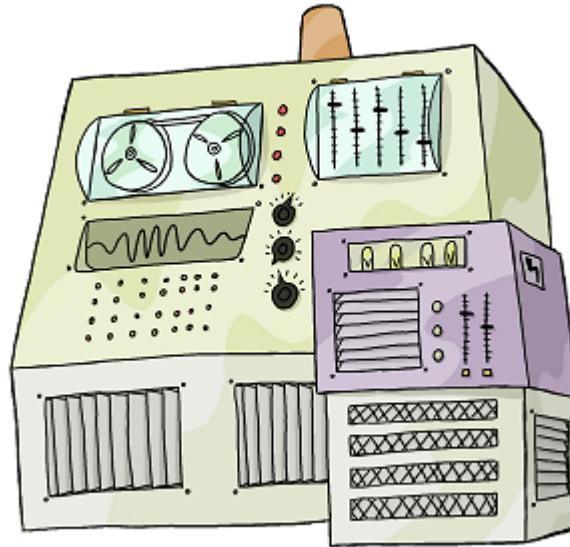
## Files and Directories



Copyright © Software Carpentry 2010

This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.



Run

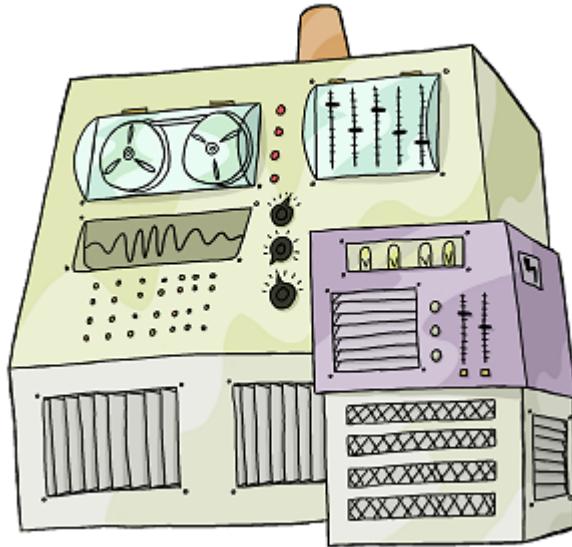
Programs

Store

Data

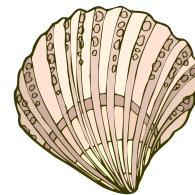
Communicate  
with each other

Interact  
with us



Run

Programs



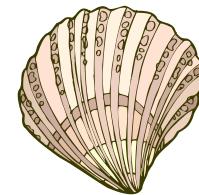
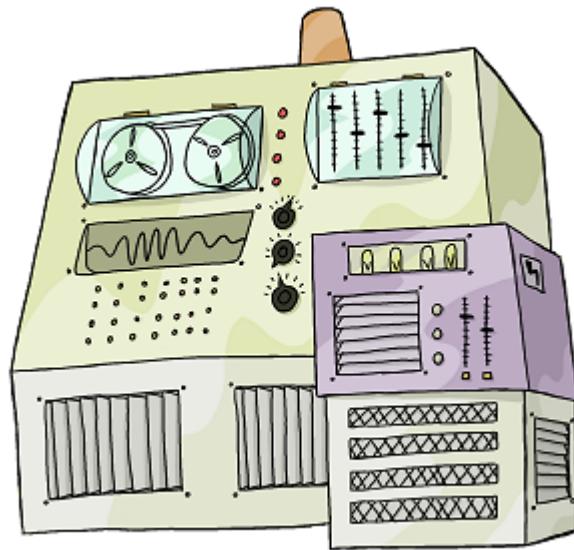
shell

Store

Data

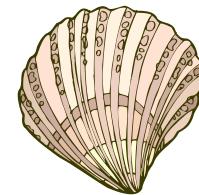
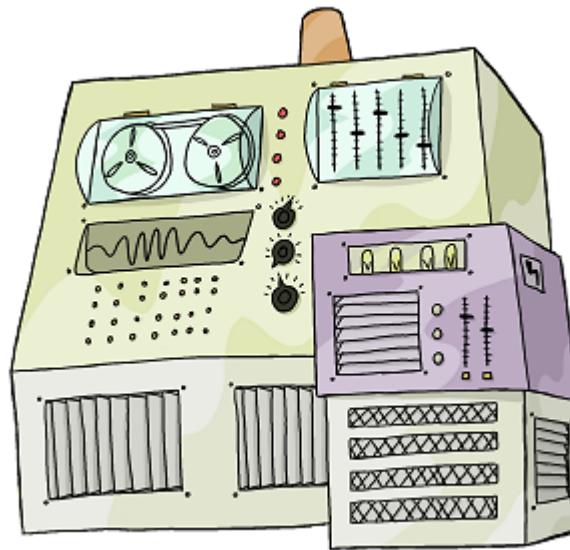
Communicate  
with each other

Interact  
with us



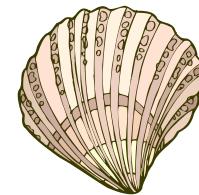
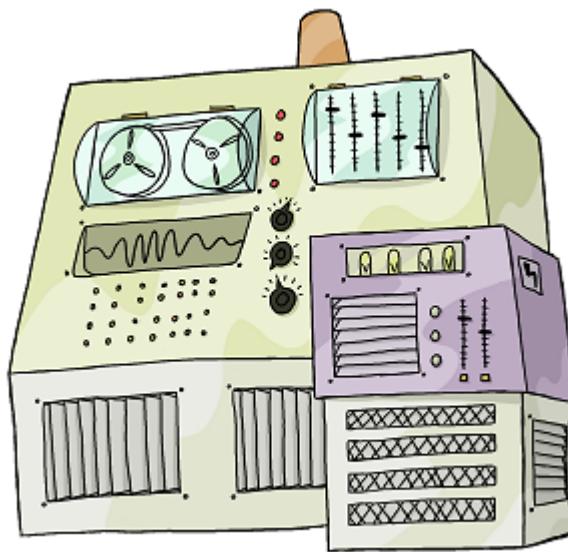
shell

Store  
Data



shell

Store  
Data  
file system



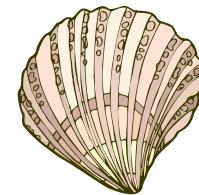
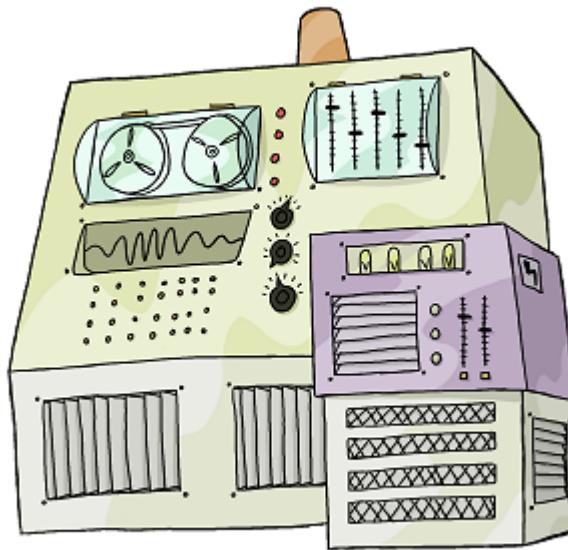
shell

Store

Data

file system

files



shell

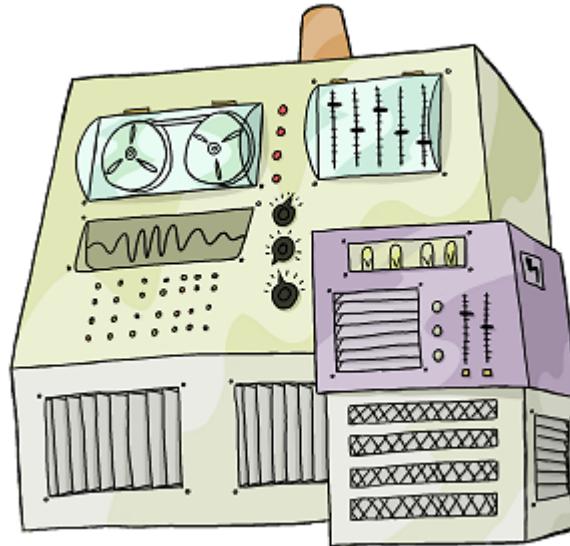
Store

Data

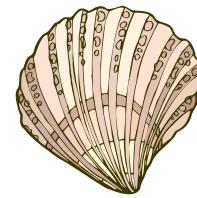
file system

files

directories



Use the shell  
to view and change  
the file system



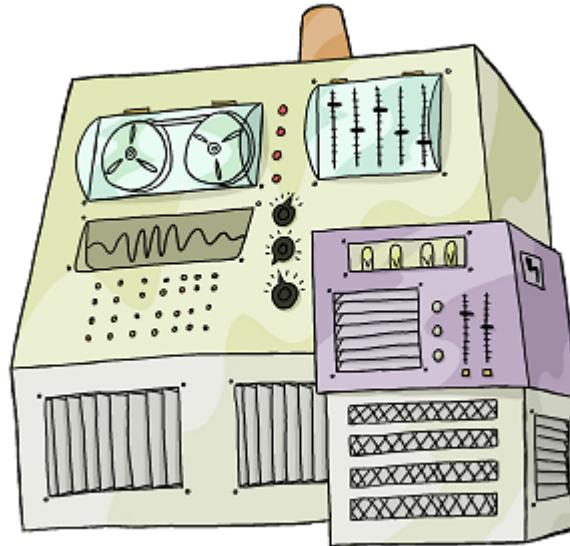
shell

Store  
Data

file system

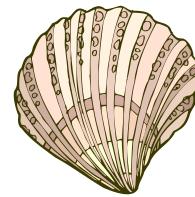
files

directories



Use the shell

*to run commands*  
to view what's in  
the file system



shell

Store

Data

file system  
files  
directories

**login:**

**login:**

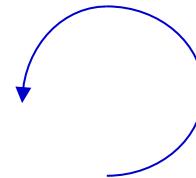


computer prompt in **bold**

**login:**



computer prompt in **bold**



explanatory text in blue

login: vlad



user input in green

login: vlad

password: \*\*\*\*\*



password

**login:** vlad

**password:** \*\*\*\*\*

\$  shell prompt

**login:** vlad

**password:** \*\*\*\*\*

\$

shell prompt

like Python's >>> and ...

**login:** vlad

**password:** \*\*\*\*\*

**\$ whoami** ← check user ID

**login:** vlad

**password:** \*\*\*\*\*

**\$ whoami**

check user ID

shell finds the whoami program

**login:** vlad

**password:** \*\*\*\*\*

**\$ whoami**

check user ID

shell finds the whoami program  
runs it

**login:** vlad

**password:** \*\*\*\*\*

**\$ whoami**

*vlad*

check user ID

shell finds the whoami program

runs it

prints its output

**login:** vlad

**password:** \*\*\*\*\*

**\$ whoami**

*vlad*

**\$**

check user ID

shell finds the whoami program  
runs it

prints its output

displays a new prompt

**login:** vlad

**password:** \*\*\*\*\*

**\$ whoami**

*vlad*

**\$ pwd**



what is the *working directory*

**login:** vlad

**password:** \*\*\*\*\*

**\$ whoami**

*vlad*

**\$ pwd**



what is the *working directory*  
the directory used when no other  
directory is explicitly specified

**login:** vlad

**password:** \*\*\*\*\*

**\$ whoami**

*vlad*

**\$ pwd**

*/users/vlad*

**\$**

```
login: vlad
password: *****
$ whoami
vlad
$ pwd
/users/vlad
$
```



root

```
login: vlad
password: *****
$ whoami
vlad
$ pwd
/users/vlad
$
```



root  
/

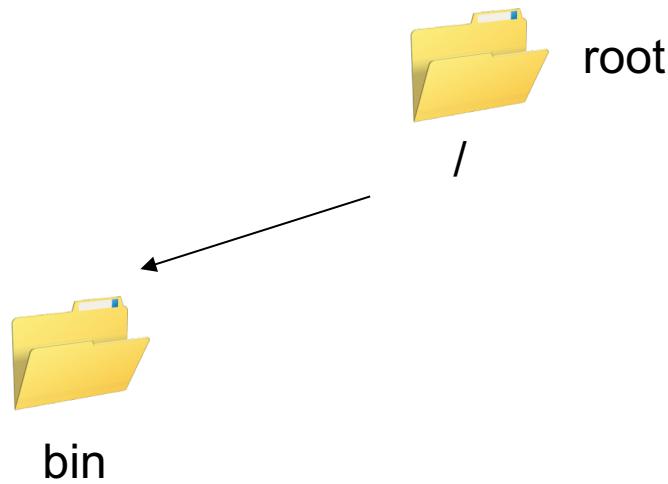
```
login: vlad
password: *****
$ whoami
vlad
$ pwd
/users/vlad
$
```



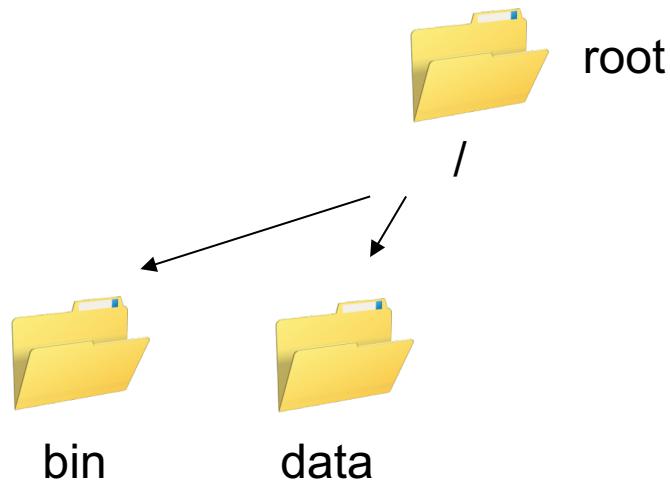
root

/

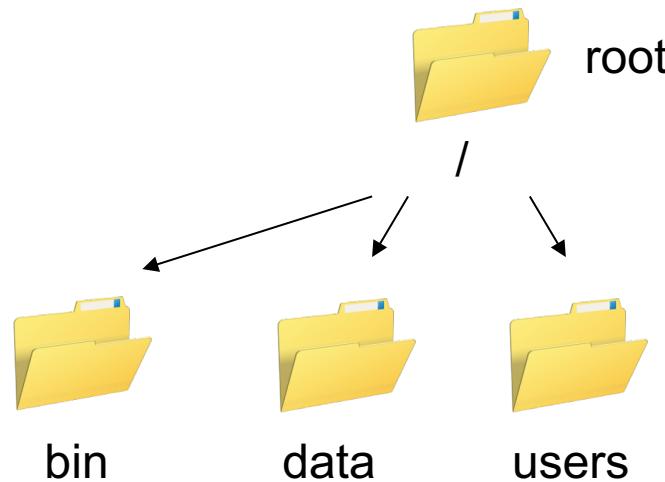
```
login: vlad
password: *****
$ whoami
vlad
$ pwd
/users/vlad
$
```



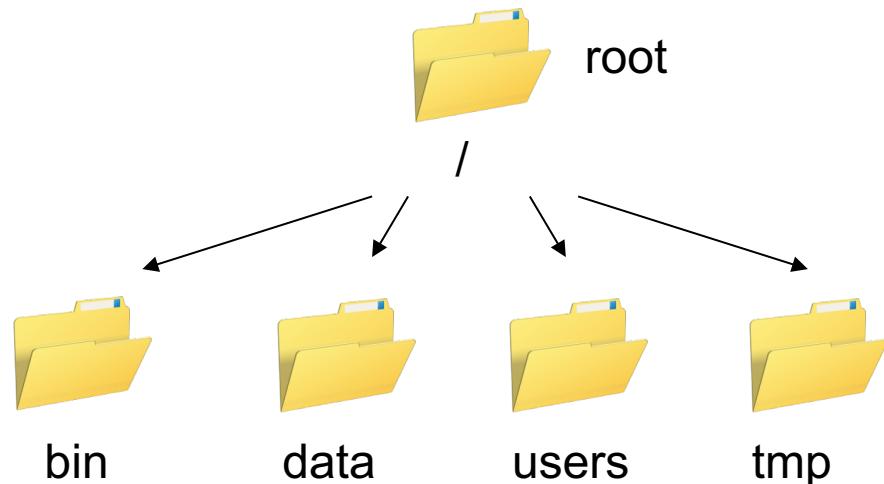
```
login: vlad
password: *****
$ whoami
vlad
$ pwd
/users/vlad
$
```



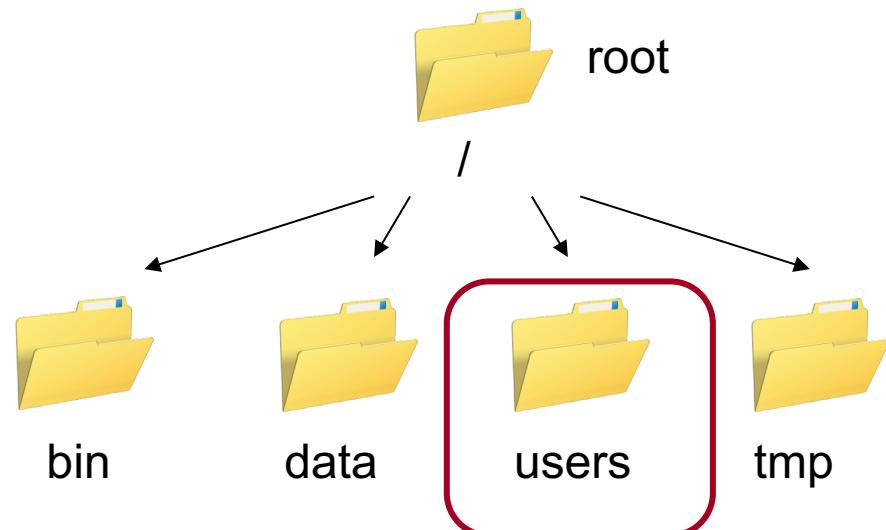
```
login: vlad
password: *****
$ whoami
vlad
$ pwd
/users/vlad
$
```



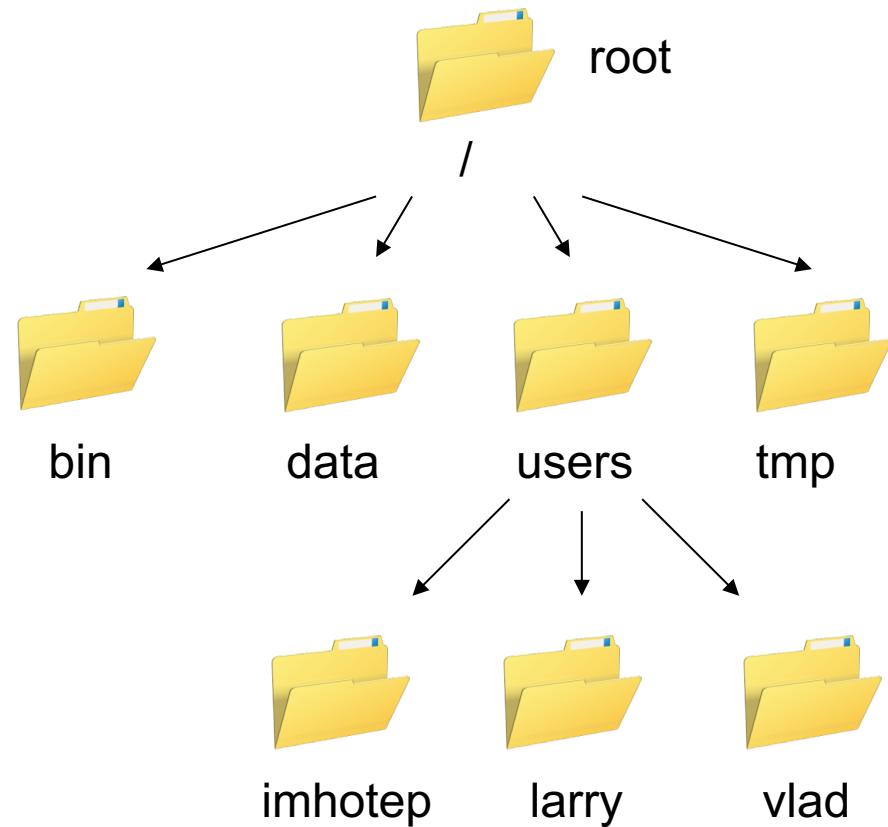
```
login: vlad
password: *****
$ whoami
vlad
$ pwd
/users/vlad
$
```



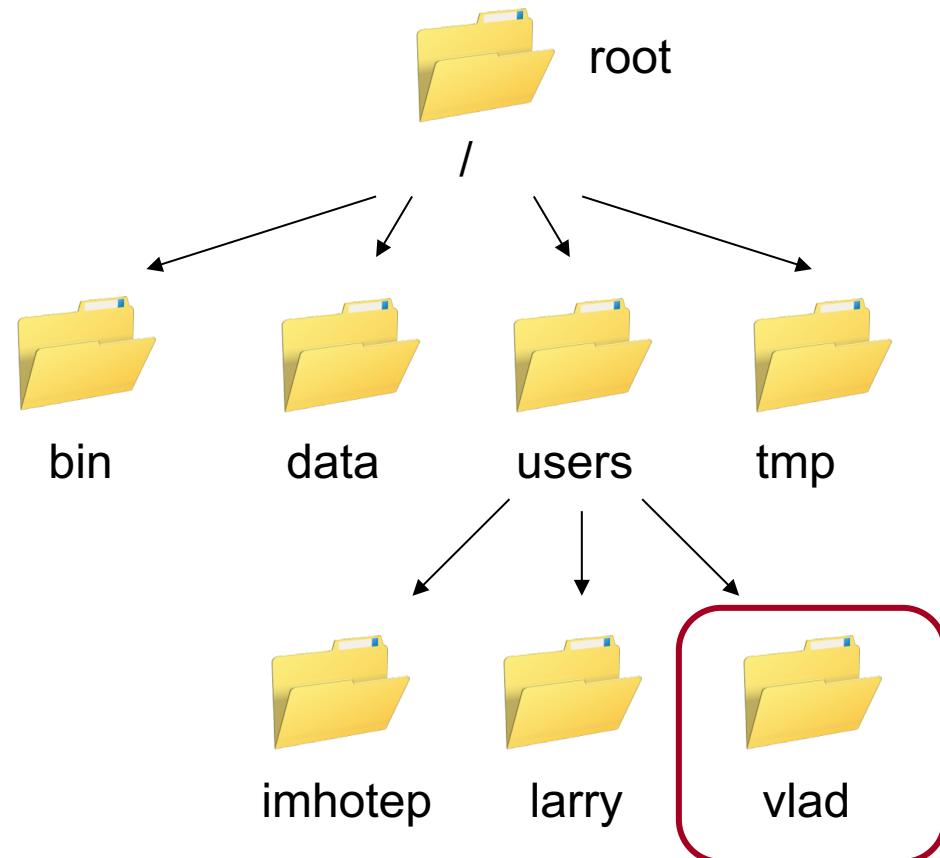
```
login: vlad
password: *****
$ whoami
vlad
$ pwd
/users/vlad
$
```



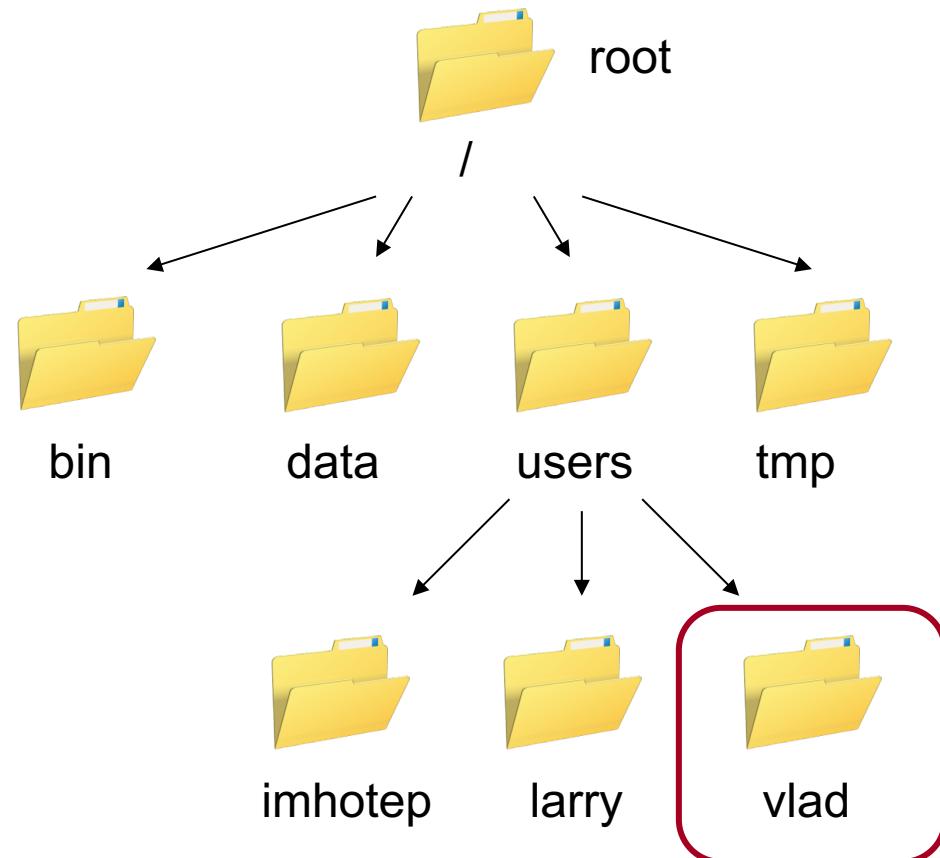
```
login: vlad
password: *****
$ whoami
vlad
$ pwd
/users/vlad
$
```



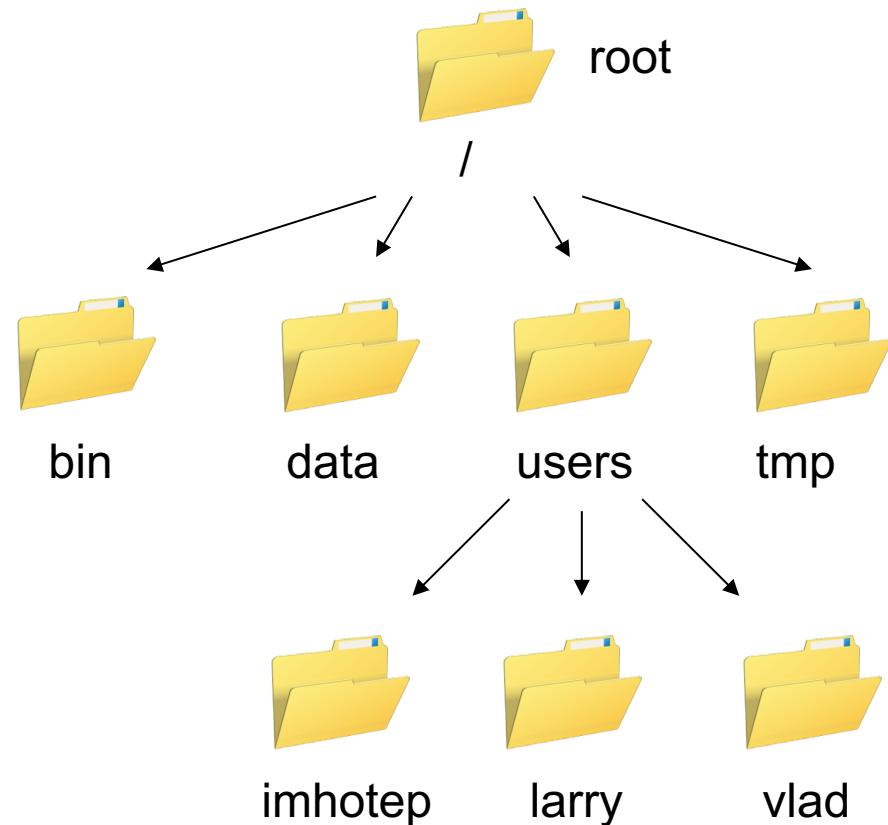
```
login: vlad
password: *****
$ whoami
vlad
$ pwd
/users/vlad
$
```



```
login: vlad
password: *****
$ whoami
vlad
$ pwd
/users/vlad
$
```



```
login: vlad
password: *****
$ whoami
vlad
$ pwd
/ /users/vlad
$
```



**login:** vlad

**password:** \*\*\*\*\*

**\$ whoami**

*vlad*

**\$ pwd**

*/users/vlad*

**\$ ls**



stands for "listing"

**login:** vlad

**password:** \*\*\*\*\*

**\$ whoami**

*vlad*

**\$ pwd**

*/users/vlad*

**\$ ls**



stands for "listing"

sadly more memorable than  
most command names

**login:** vlad

**password:** \*\*\*\*\*

**\$ whoami**

*vlad*

**\$ pwd**

*/users/vlad*

**\$ ls**

*bin data mail music*

*notes.txt papers pizza.cfg solar*

*solar.pdf swc*

**\$**

login: vlad

password: \*\*\*\*\*

\$ whoami

vlad

\$ pwd

/users/vlad

\$ ls -F

bin/ data/ mail/ music/

notes.txt papers/ pizza.cfg solar/

solar.pdf swc/

\$

an *argument* or *flag* modifying  
the command's behavior

login: vlad

password: \*\*\*\*\*

\$ whoami

*vlad*

\$ pwd

*/users/vlad*

\$ ls -F

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

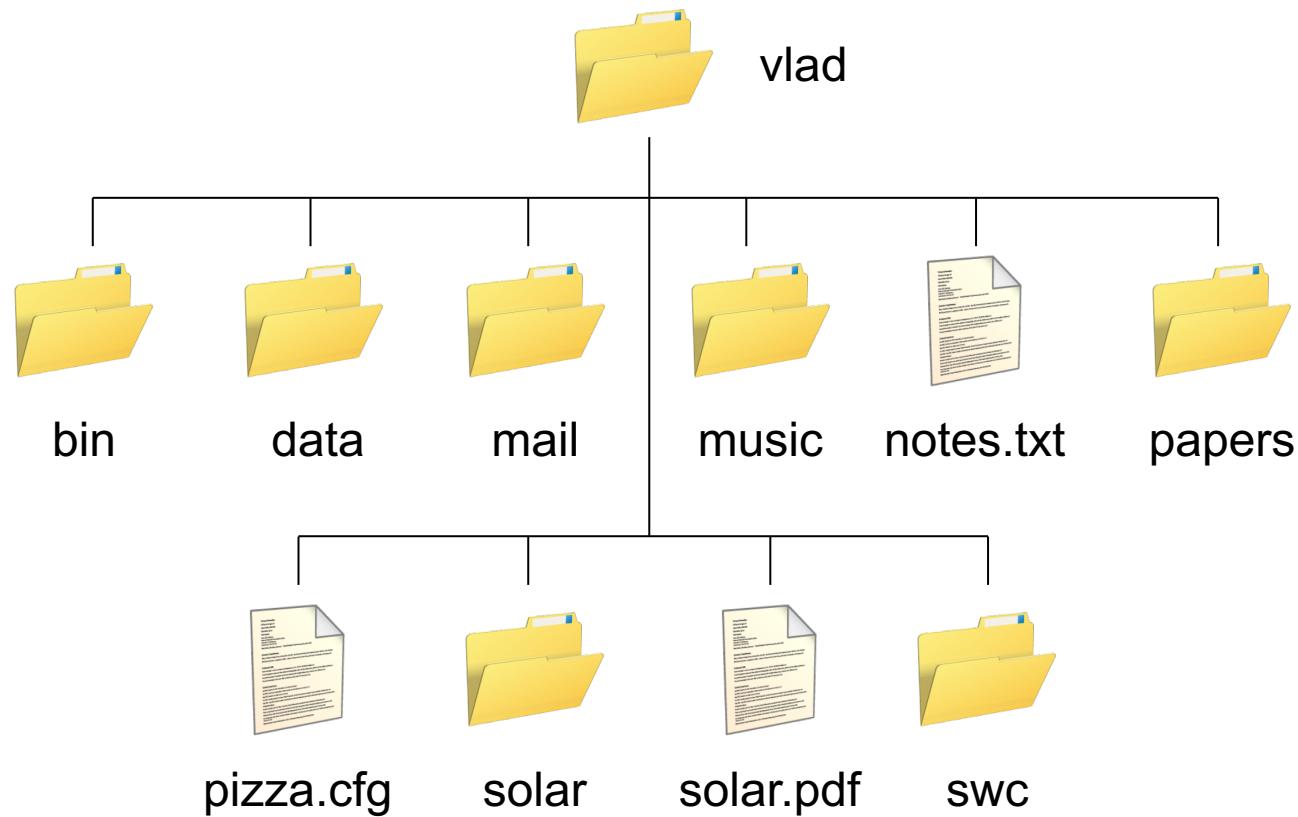
\$

adds a trailing '/' to  
directory names



```
$ ls -F
```

```
bin/      data/    mail/    music/  
notes.txt  papers/   pizza.cfg  solar/  
solar.pdf  swc/
```



`$ ls -F`

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

By convention, use *filename extension* to indicate file type

```
$ ls -F
```

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

By convention, use *filename extension* to indicate file type  
.txt for text, .pdf for PDF, .cfg for configuration file, etc.

`$ ls -F`

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

By convention, use *filename extension* to indicate file type  
.txt for text, .pdf for PDF, .cfg for configuration file, etc.

But this is only a convention, not a guarantee

```
$ ls -F data
```

```
$ ls -F data
```

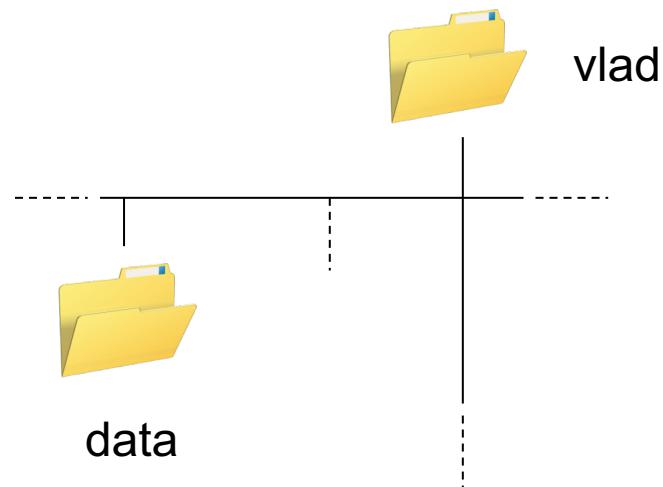
*amino\_acids.txt elements/ morse.txt*

*pdb/ planets.txt sunspot.txt*

```
$
```

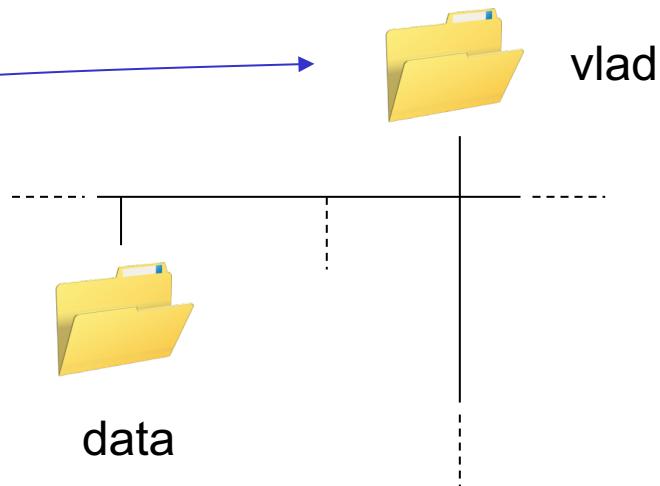
```
$ ls -F data
aminomino _acids.txt  elements/  morse.txt
pdb/                  planets.txt  sunspot.txt
$
```

*a relative path*



```
$ ls -F data
amino_acids.txt  elements/  morse.txt
pdb/            planets.txt  sunspot.txt
$
```

*a relative path  
relative to  
current working directory*



```
$ ls -F /data
```

*access.log backup/ hardware.cfg*

*network.cfg*

```
$
```

```
$ ls -F /data
```

*access.log* *backup/* *hardware.cfg*

*network.cfg*

\$

*an absolute path*

\$ ls -F /data

*access.log* ~~backup/~~ *hardware.cfg*  
*network.cfg*

\$

*an absolute path*

leading '/' means "from root"

```
$ ls -F /data
```

*access.log backup/ hardware.cfg*

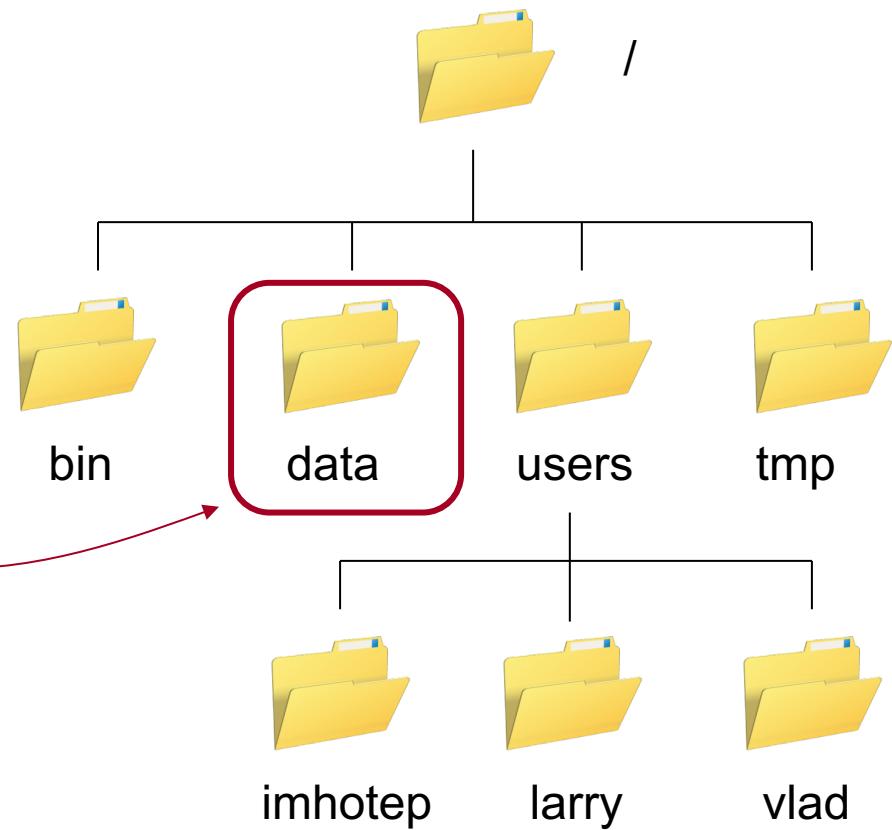
*network.cfg*

```
$
```

*an absolute path*

*leading '/' means "from root"*

*so it always refers to  
this directory*



\$ **pwd**

*/users/vlad*

\$

```
$ pwd
```

*/users/vlad*

```
$ ls
```

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

```
$
```

**\$** **pwd**

*/users/vlad*

**\$** **ls**

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

**\$** **cd data**

\$ **pwd**

*/users/vlad*

\$ **ls**

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

\$ **cd data** ← change directory

\$ **pwd**

*/users/vlad*

\$ **ls**

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

\$ **cd data**



change directory

actually doesn't change the directory

\$ **pwd**

*/users/vlad*

\$ **ls**

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

\$ **cd data**



change directory

actually doesn't change the directory

changes the shell's idea of

which directory we are in

```
$ pwd
```

*/users/vlad*

```
$ ls
```

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

```
$ cd data
```

```
$ pwd
```

*/users/vlad/data*

```
$
```

\$ **pwd**

*/users/vlad*

\$ **ls**

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

\$ **cd data**

\$ **pwd**

*/users/vlad/data*

\$ **ls**

*amino\_acids.txt elements/ morse.txt*

*pdb/ planets.txt sunspot.txt*

\$

\$ **pwd**

*/users/vlad*

\$ **ls**

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

\$ **cd data**

\$ **pwd**

*/users/vlad/data*

\$ **ls**

*amino\_acids.txt elements/ morse.txt*

*pdb/ planets.txt sunspot.txt*

\$

because we're now "in"  
this directory

\$ **pwd**

*/users/vlad/data*

\$

```
$ pwd
```

*/users/vlad/data*

```
$ cd ..
```

\$ **pwd**

*/users/vlad/data*

\$ **cd .** 

the directory above the current one

**\$** **pwd**

*/users/vlad/data*

**\$** **cd ..**



the directory above the current one  
its *parent directory*

```
$ pwd
```

```
/users/vlad/data
```

```
$ cd ..
```

```
$ pwd
```

```
/users/vlad
```

```
$
```

```
$ pwd
```

*/users/vlad/data*

```
$ cd ..
```

```
$ pwd
```

*/users/vlad*

```
$ ls
```

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

```
$
```

**\$** **pwd**

*/users/vlad/data*

**\$** **cd ..**

**\$** **pwd**

*/users/vlad*

**\$** **ls**

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

**\$** **ls -F -a**

*./ ../ bin/ data/*

*mail/ music/ notes.txt papers/*

*pizza.cfg solar/ solar.pdf swc/*

**\$ pwd**

*/users/vlad/data*

**\$ cd ..**

**\$ pwd**

*/users/vlad*

**\$ ls**

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

**\$ ls -F -a**



"show all"

*./ ../ bin/ data/*

*mail/ music/ notes.txt papers/*

*pizza.cfg solar/ solar.pdf swc/*

\$ **pwd**

*/users/vlad/data*

\$ **cd ..**

\$ **pwd**

*/users/vlad*

\$ **ls**

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

\$ **ls -F -a**

*./ ../ bin/ data/*

*mail/ music/ notes.txt papers/*

*pizza.cfg solar/ solar.pdf swc/*

parent directory

\$ **pwd**

*/users/vlad/data*

\$ **cd ..**

\$ **pwd**

*/users/vlad*

\$ **ls**

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

\$ **ls -F -a**

*./ ../ bin/ data/*

*mail/ music/ notes.txt papers/*

*pizza.cfg solar/ solar.pdf swc/*

parent directory

/users

**\$ pwd**

*/users/vlad/data*

**\$ cd ..**

**\$ pwd**

*/users/vlad*

**\$ ls**

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

**\$ ls -F -a**

**./** *../ bin/ data/*

*mail/ music/ notes.txt papers/*

*pizza.cfg solar/ solar.pdf swc/*

this directory  
itself

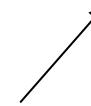
# Things are different on Windows

# Things are different on Windows

C:\Users\vlad

Things are different on Windows

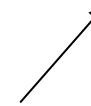
C:\Users\vlad



Drive letter

Things are different on Windows

C:\Users\vlad

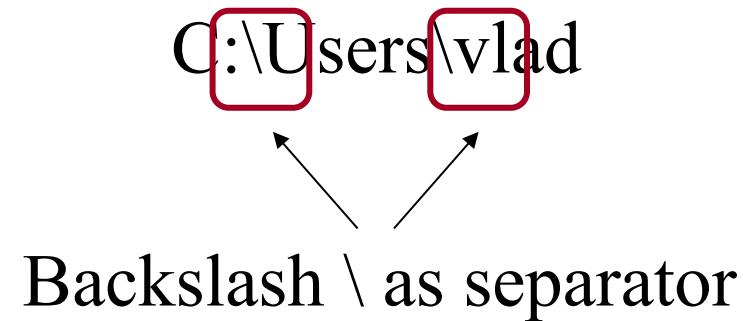


Drive letter

Each drive is a separate file system

Things are different on Windows

C:\Users\vlad



Backslash \ as separator

# Things are different on Windows

C:\Users\vlad

Backslash \ as separator

Unix uses \ to escape special characters

in names like my\ files.txt

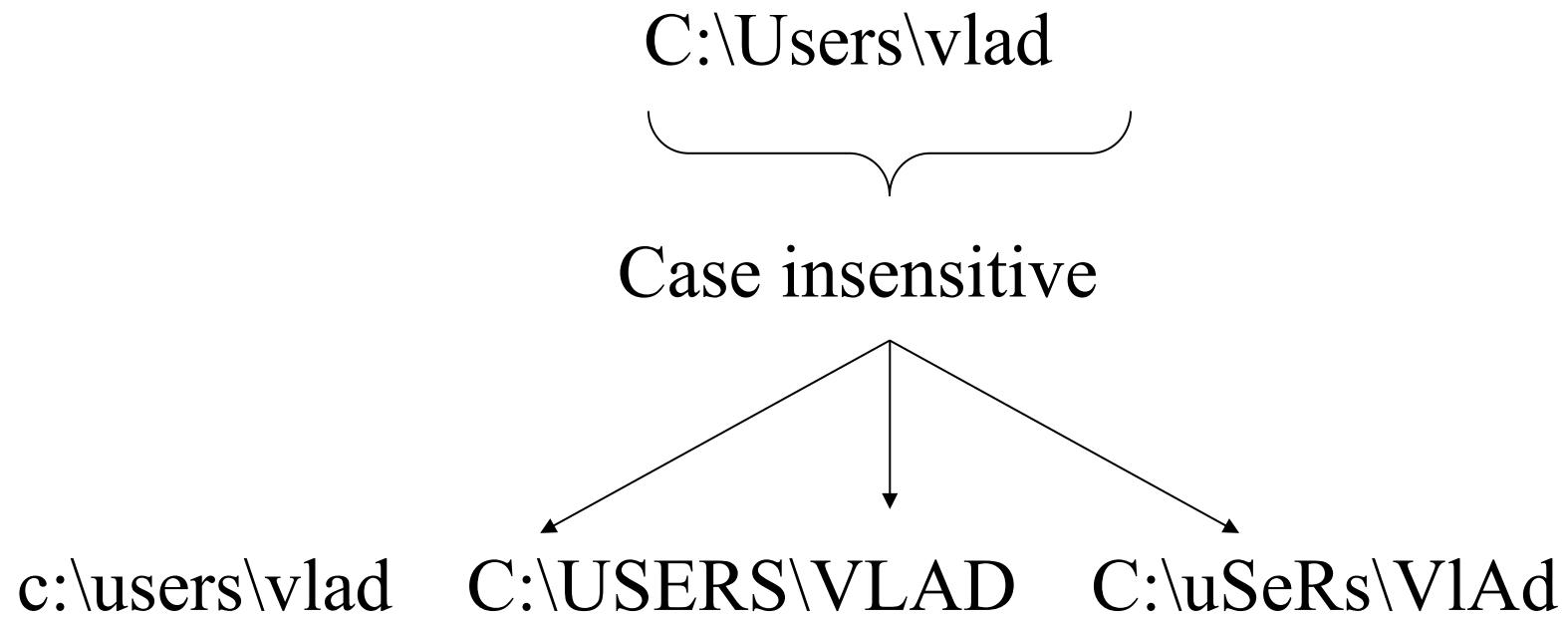
Things are different on Windows

C:\Users\vlad



Case insensitive

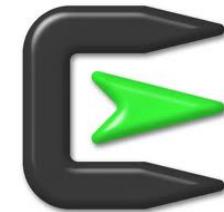
Things are different on Windows



Things are different on Windows

C:\Users\vlad

Cygwin: /cygdrive/c/Users/vlad

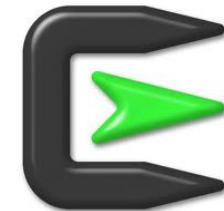


Map drive letters to "directories"

Things are different on Windows

C:\Users\vlad

Cygwin: /cygdrive/c/Users/vlad



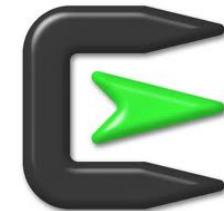
Map drive letters to "directories"

And use / instead of \

Things are different on Windows

C:\Users\vlad

Cygwin: /cygdrive/c/Users/vlad



Map drive letters to "directories"

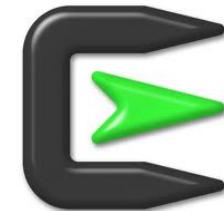
And use / instead of \

But still case insensitive

Things are different on Windows

C:\Users\vlad

Cygwin: /cygdrive/c/Users/vlad



Map drive letters to "directories"

And use / instead of \

But still case insensitive

Can't put backup.txt and Backup.txt in a directory

pwd

print working directory

cd

change working directory

ls

listing

.

current directory

..

parent directory



# The Unix Shell

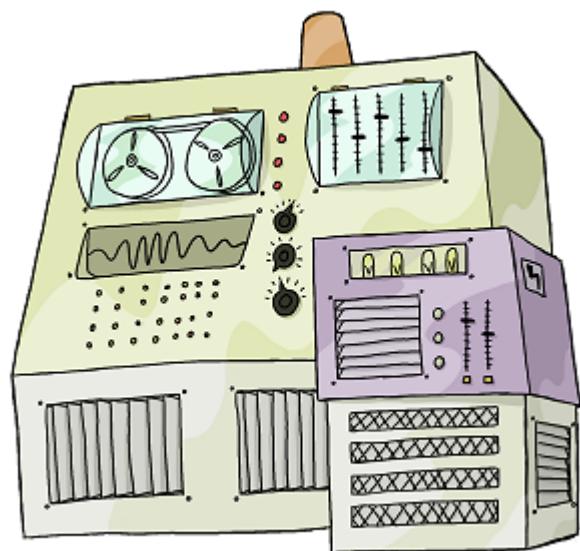
## Creating and Deleting

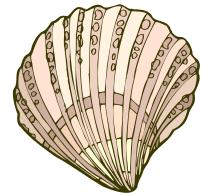


Copyright © Software Carpentry 2010

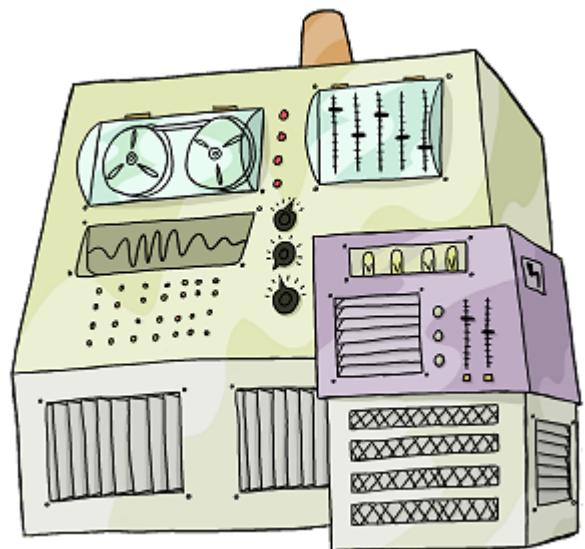
This work is licensed under the Creative Commons Attribution License

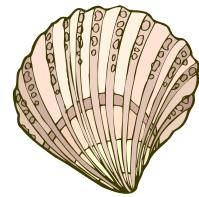
See <http://software-carpentry.org/license.html> for more information.





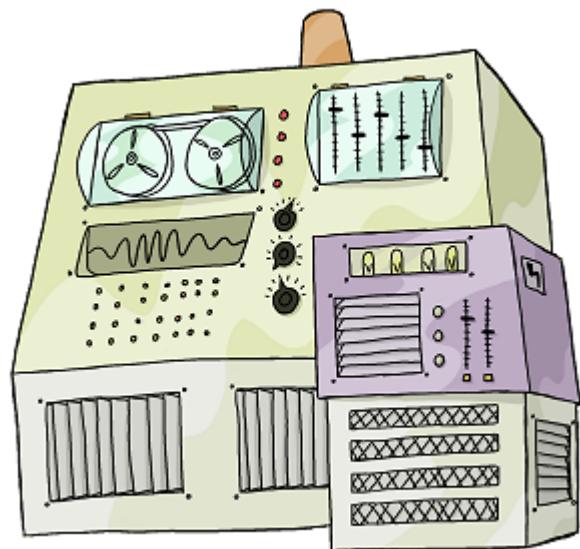
shell

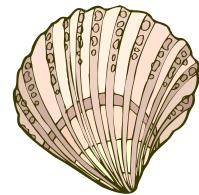




shell

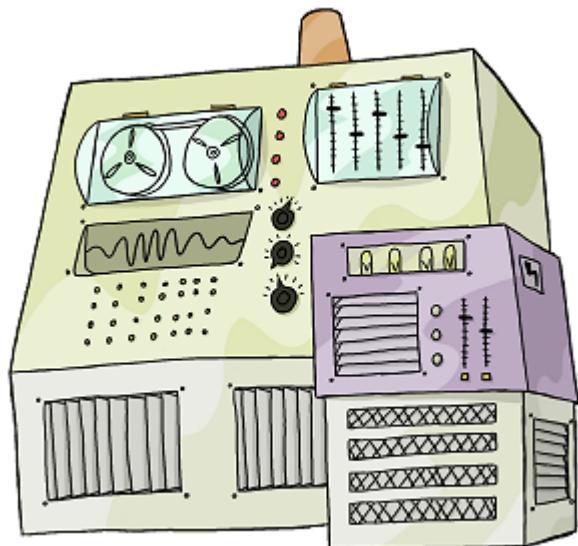
pwd	print working directory
cd	change working directory
ls	listing
.	current directory
.	parent directory





shell

pwd	print working directory
cd	change working directory
ls	listing
.	current directory
.	parent directory



*But how do we create things  
in the first place?*

\$ **pwd**

*/users/vlad*

\$

```
$ pwd
```

*/users/vlad*

```
$ ls -F
```

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

```
$
```

```
$ pwd
```

*/users/vlad*

```
$ ls -F
```

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

```
$ mkdir tmp
```

```
$ pwd
```

*/users/vlad*

```
$ ls -F
```

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

```
$ mkdir tmp ← make directory
```

```
$ pwd
```

*/users/vlad*

```
$ ls -F
```

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

```
$ mkdir tmp ← make directory
```

a relative path, so the new directory  
is made below the current one

```
$ pwd
```

*/users/vlad*

```
$ ls -F
```

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

```
$ mkdir tmp
```

```
$ ls -F
```

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/ tmp/*

```
$
```

\$ **pwd**

*/users/vlad*

\$ **ls -F**

*bin/ data/ mail/ music/*

*notes.txt papers/ pizza.cfg solar/*

*solar.pdf swc/*

\$ **mkdir tmp**

\$ **ls -F**

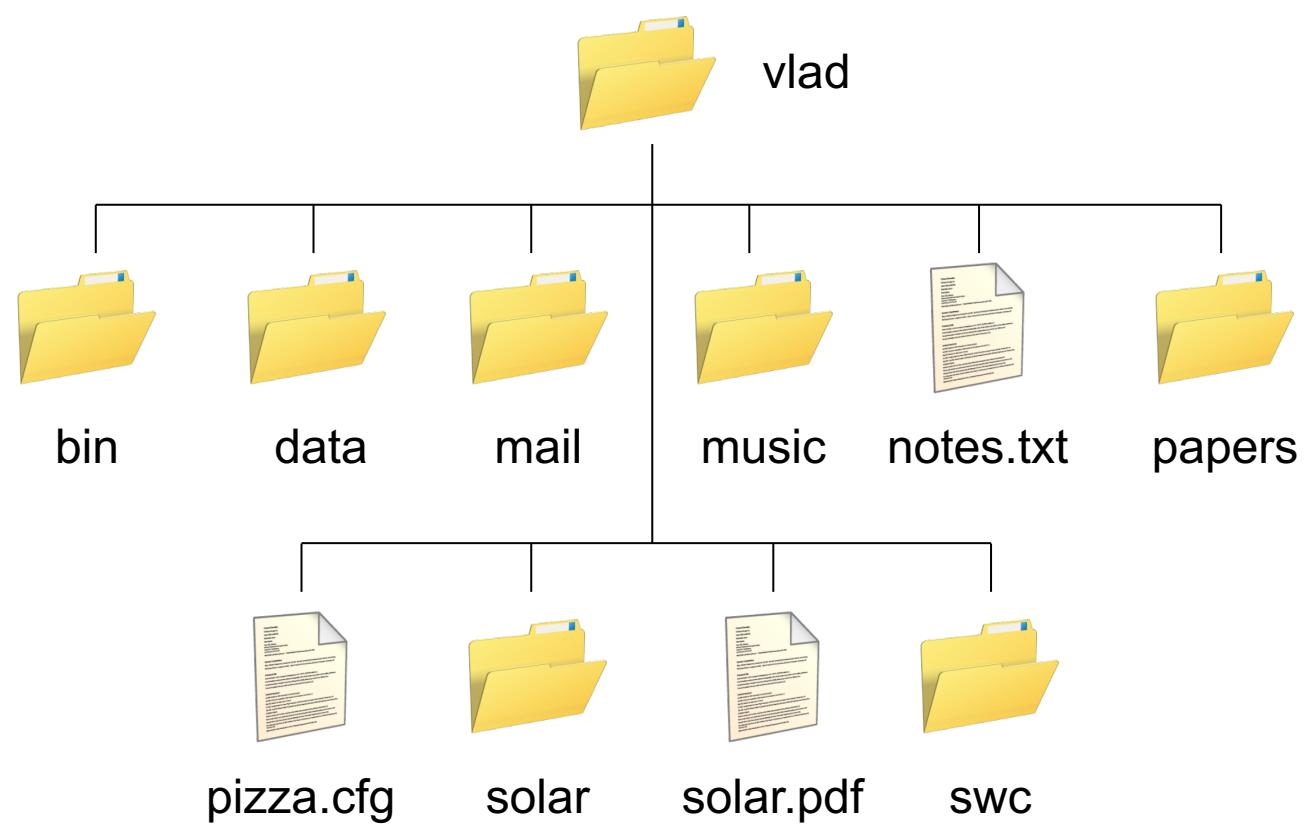
*bin/ data/ mail/ music/*

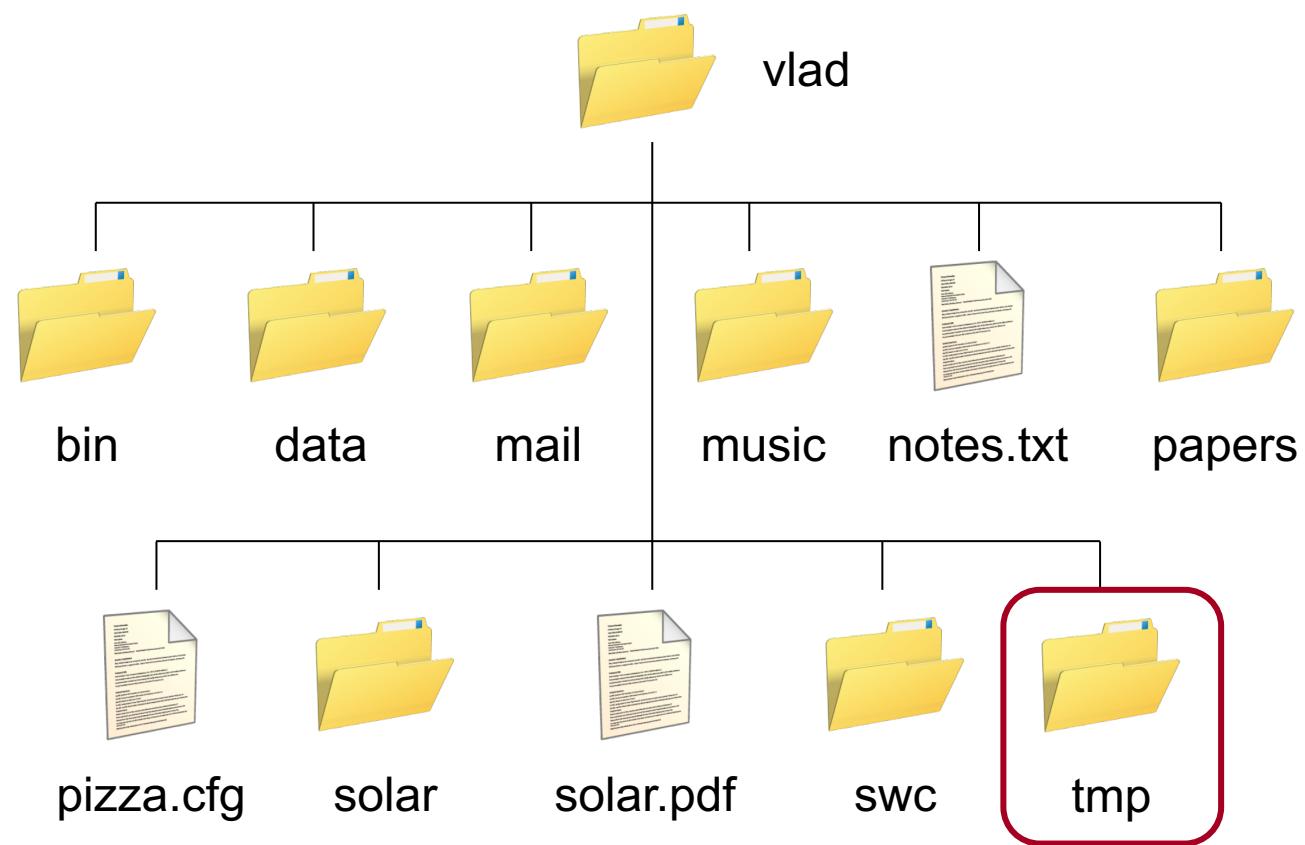
*notes.txt papers/ pizza.cfg solar/*

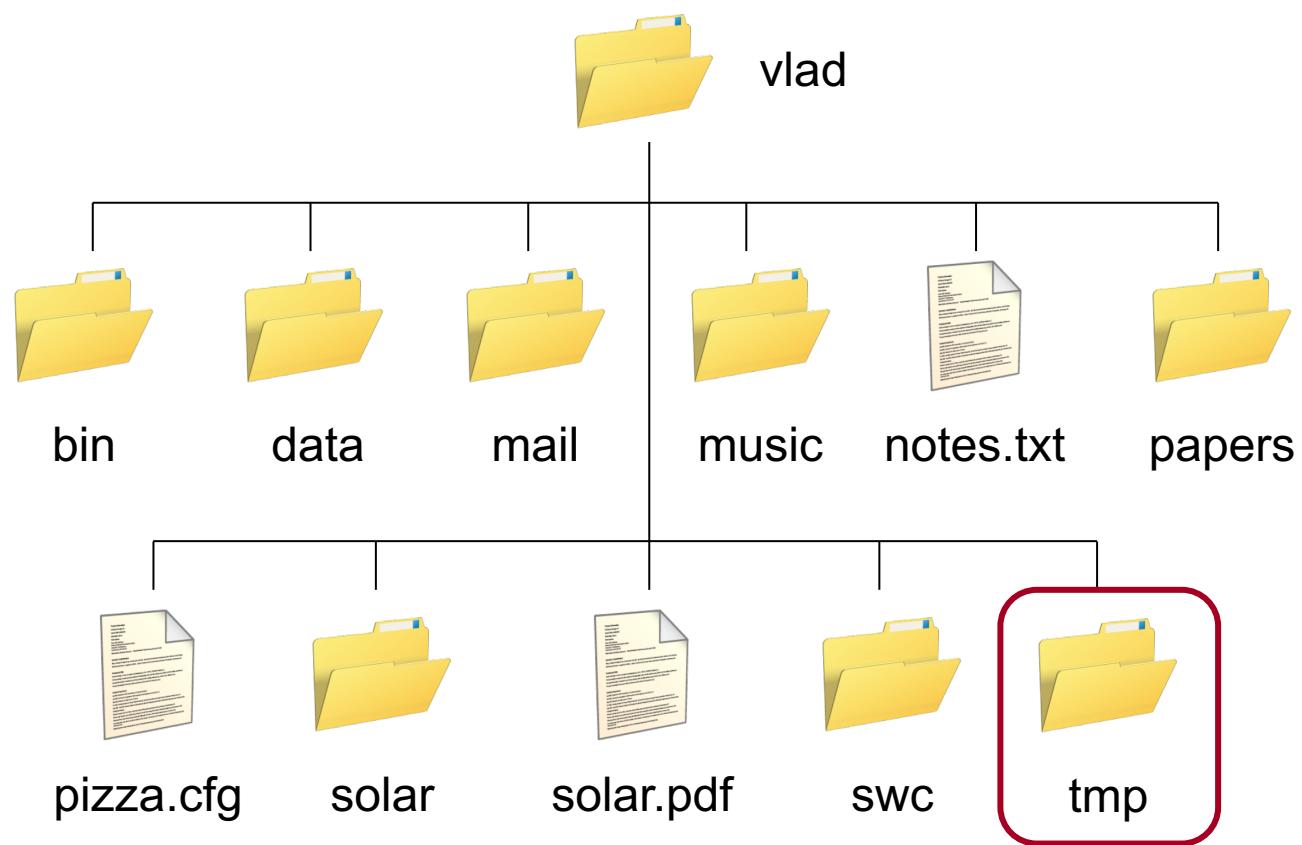
*solar.pdf swc/ tmp/*



\$







nothing below it yet

\$ **pwd**

*/users/vlad*

\$

```
$ pwd
```

```
/users/vlad
```

```
$ ls tmp
```

```
$
```

\$ **pwd**

*/users/vlad*

\$ **ls tmp**

\$  no output

```
$ pwd
```

```
/users/vlad
```

```
$ ls tmp
```

```
$ ls -a tmp
```

```
.
```

```
..
```

```
$
```

\$ **pwd**

*/users/vlad*

\$ **ls tmp**

\$ **ls -a tmp**

• ..



*/users/vlad/tmp*

\$ **pwd**

*/users/vlad*

\$ **ls tmp**

\$ **ls -a tmp**

..



*/users/vlad*

```
$ cd tmp  
$ nano junk
```

```
$ cd tmp  
$ nano junk
```

a text editor only a programmer could love

```
$ cd tmp  
$ nano junk
```

a text editor only a programmer could love  
really do mean "text"...

```
$ cd tmp  
$ nano junk
```

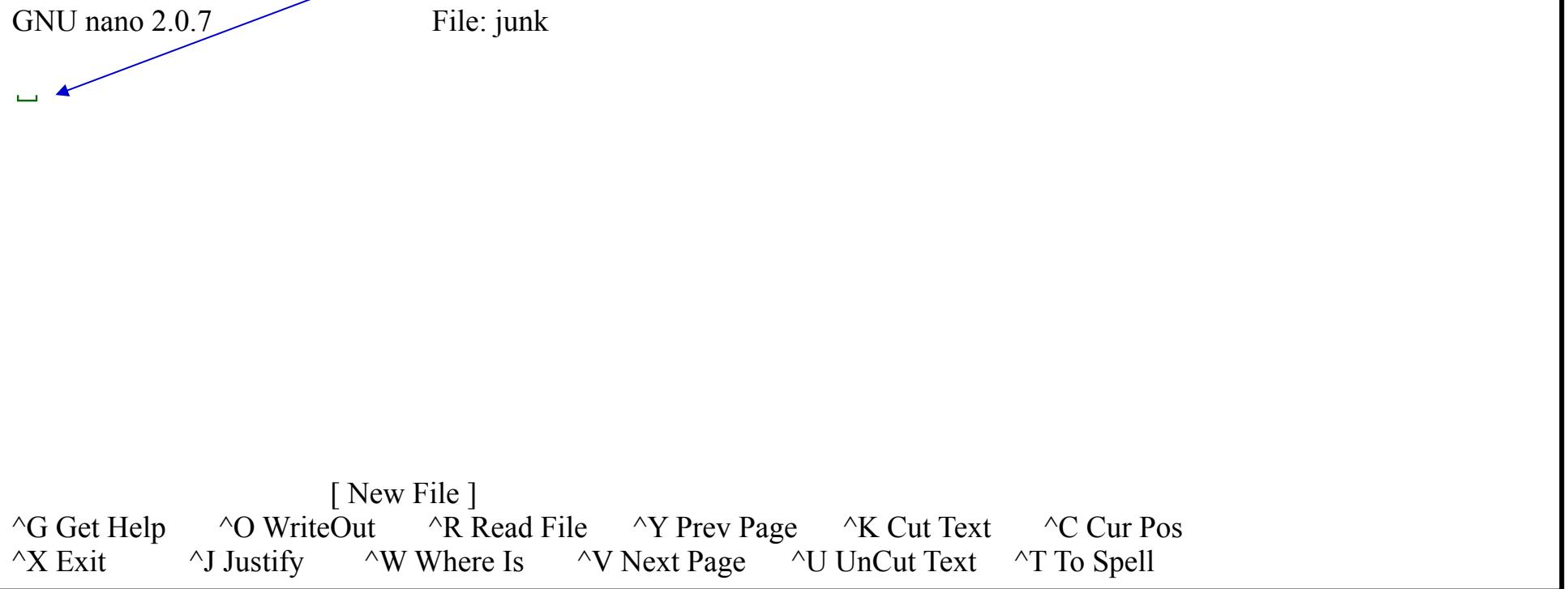
GNU nano 2.0.7

File: junk

[ New File ]  
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos  
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

```
$ cd tmp  
$ nano junk
```

That's your cursor



GNU nano 2.0.7

File: junk

[ New File ]

<sup>^G</sup> Get Help <sup>^O</sup> WriteOut <sup>^R</sup> Read File <sup>^Y</sup> Prev Page <sup>^K</sup> Cut Text <sup>^C</sup> Cur Pos  
<sup>^X</sup> Exit <sup>^J</sup> Justify <sup>^W</sup> Where Is <sup>^V</sup> Next Page <sup>^U</sup> UnCut Text <sup>^T</sup> To Spell

\$ cd tmp

\$ nano junk

GNU nano 2.0.7

File: junk

Make everything as simple as possible,  
but no simpler.  

[ New File ]  
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos  
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

```
$ cd tmp  
$ nano junk
```

GNU nano 2.0.7 File: junk

Make everything as simple as possible,  
but no simpler.  

[ New File ]  
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos  
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

^O means "Control + O" (to save changes)

```
$ cd tmp  
$ nano junk
```

GNU nano 2.0.7 File: junk

Make everything as simple as possible,  
but no simpler.  

[ New File ]  
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos  
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

^X to exit back to the shell

```
$ cd tmp  
$ nano junk  
$
```

nano doesn't leave any output  
on the screen after it exits

```
$ cd tmp  
$ nano junk  
$ ls  
junk ← but it has created the file  
$
```

```
$ cd tmp  
$ nano junk  
$ ls  
junk  
$ ls -s      ← use -s to show sizes  
  1 junk  
$
```

```
$ cd tmp  
$ nano junk  
$ ls  
junk  
$ ls -s ← use -s to show sizes  
  1 junk      reported in disk blocks  
$
```

```
$ cd tmp  
$ nano junk  
$ ls  
junk  
$ ls -s      ← use -s to show sizes  
  1 junk      reported in disk blocks  
$                      a less helpful default  
                      may have been possible...
```

```
$ cd tmp
$ nano junk
$ ls
junk
$ ls -s
  1 junk
$ ls -s -h
← use -h for human-friendly output
  512 junk
$
```

```
$ cd tmp  
$ nano junk  
$ ls  
junk  
$ ls -s  
 1 junk  
$ ls -s -h  
512 junk  
$
```



use -h for human-friendly output  
number of bytes

```
$ cd tmp  
$ nano junk  
$ ls  
junk
```

```
$ ls -s
```

```
1 junk
```

```
$ ls -s -h
```

```
512 junk
```

```
$
```

← use -h for human-friendly output  
number of bytes  
rounded up because computer stores  
things on disk using blocks of 512 bytes

```
$ cd tmp
$ nano junk
$ ls
junk
$ ls -s
  1 junk
$ ls -s -h
512 junk
$ rm junk
$
```

← remove (delete) file

```
$ cd tmp  
$ nano junk  
$ ls  
junk  
$ ls -s  
 1 junk  
$ ls -s -h  
512 junk  
$ rm junk  
$
```



remove (delete) file  
there is no (easy) un-delete!

```
$ cd tmp
$ nano junk
$ ls
junk
$ ls -s
  1 junk
$ ls -s -h
512 junk
$ rm junk
$ ls
← check that it's gone
```

\$ **pwd**

*/users/vlad/tmp*

\$ **nano junk**

\$ **ls**

*junk*

\$

\$ **pwd**

*/users/vlad/tmp*

\$ **nano junk**

\$ **ls**

*junk*

\$ **cd ..** ← change working directory to /users/vlad

\$

```
$ pwd
```

*/users/vlad/tmp*

```
$ nano junk
```

```
$ ls
```

*junk*

```
$ cd ..
```

```
$ rm tmp
```

rm only works on files

*rm: cannot remove 'tmp': Is a directory*

```
$
```

```
$ pwd  
/users/vlad/tmp  
$ nano junk  
$ ls  
junk  
$ cd ..  
$ rm tmp  
rm: cannot remove 'tmp': Is a directory  
$ rmdir tmp
```



use rmdir to remove directories

```
$ pwd  
/users/vlad/tmp
```

```
$ nano junk
```

```
$ ls
```

*junk*

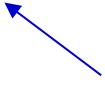
```
$ cd ..
```

```
$ rm tmp
```

*rm: cannot remove 'tmp': Is a directory*

```
$ rmdir tmp
```

*rmdir: failed to remove 'tmp': Directory not empty*

\$   
but it only works when the directory is empty

\$ **pwd**

*/users/vlad/tmp*

\$ **nano junk**

\$ **ls**

*junk*

\$ **cd ..**

\$ **rm tmp**

*rm: cannot remove 'tmp': Is a directory*

\$ **rmdir tmp**

*rmdir: failed to remove 'tmp': Directory not empty*

\$

but it only works when the directory is empty  
(safety feature)

```
$ pwd  
/users/vlad/tmp
```

```
$ nano junk  
$ ls
```

*junk*

```
$ cd ..  
$ rm tmp
```

*rm: cannot remove 'tmp': Is a directory*

```
$ rmdir tmp  
rmdir: failed to remove 'tmp': Directory not empty
```

```
$ rm tmp/junk
```

```
$
```

so get rid of the directory's contents...

```
$ pwd
```

*/users/vlad/tmp*

```
$ nano junk
```

```
$ ls
```

*junk*

```
$ cd ..
```

```
$ rm tmp
```

*rm: cannot remove 'tmp': Is a directory*

```
$ rmdir tmp
```

*rmdir: failed to remove 'tmp': Directory not empty*

```
$ rm tmp/junk
```

```
$ rmdir tmp
```



...then get rid of the directory

```
$
```

```
$ pwd
```

*/users/vlad/tmp*

```
$ mkdir tmp
```

```
$ nano tmp/junk
```

```
$ ls tmp
```

*junk*

```
$
```

```
$ pwd
```

*/users/vlad/tmp*

```
$ mkdir tmp
```

```
$ nano tmp/junk
```

```
$ ls tmp
```

*junk*

```
$ mv tmp/junk tmp/quotes.txt
```

```
$
```

```
$ pwd
```

*/users/vlad/tmp*

```
$ mkdir tmp
```

```
$ nano tmp/junk
```

```
$ ls tmp
```

*junk*

```
$ mv tmp/junk tmp/quotes.txt
```

```
$
```

move a file

```
$ pwd  
/users/vlad/tmp  
$ mkdir tmp  
$ nano tmp/junk  
$ ls tmp  
junk
```

```
$ mv tmp/junk tmp/quotes.txt
```

```
$
```

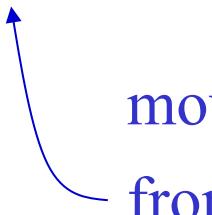
move a file (or directory)

```
$ pwd  
/users/vlad/tmp  
$ mkdir tmp  
$ nano tmp/junk  
$ ls tmp  
junk
```

```
$ mv tmp/junk tmp/quotes.txt
```

```
$
```

move a file (or directory)  
from here...



```
$ pwd  
/users/vlad/tmp  
$ mkdir tmp  
$ nano tmp/junk  
$ ls tmp  
junk
```

```
$ mv tmp/junk tmp/quotes.txt
```

\$

move a file (or directory)  
from here...  
...to here

```
$ pwd
```

*/users/vlad/tmp*

```
$ mkdir tmp
```

```
$ nano tmp/junk
```

```
$ ls tmp
```

*junk*

```
$ mv tmp/junk tmp/quotes.txt
```

```
$
```

move a file (or directory)

from here...

...to here

renames the file!

```
$ pwd
```

*/users/vlad/tmp*

```
$ mkdir tmp
```

```
$ nano tmp/junk
```

```
$ ls tmp
```

*junk*

```
$ mv tmp/junk tmp/quotes.txt
```

```
$ ls tmp
```

*quotes.txt*

```
$
```

```
$ pwd
```

*/users/vlad/tmp*

```
$ mkdir tmp
```

```
$ nano tmp/junk
```

```
$ ls tmp
```

*junk*

```
$ mv tmp/junk tmp/quotes.txt
```

```
$ ls tmp
```

*quotes.txt*

```
$ mv tmp/quotes.txt .
```

```
$
```

```
$ pwd
```

*/users/vlad/tmp*

```
$ mkdir tmp
```

```
$ nano tmp/junk
```

```
$ ls tmp
```

*junk*

```
$ mv tmp/junk tmp/quotes.txt
```

```
$ ls tmp
```

*quotes.txt*

```
$ mv tmp/quotes.txt .
```



current working directory

```
$
```

```
$ pwd
```

*/users/vlad/tmp*

```
$ mkdir tmp
```

```
$ nano tmp/junk
```

```
$ ls tmp
```

*junk*

```
$ mv tmp/junk tmp/quotes.txt
```

```
$ ls tmp
```

*quotes.txt*

```
$ mv tmp/quotes.txt .
```



move */users/vlad/tmp/quotes.txt*  
to */users/vlad/quotes.txt*

```
$
```

```
$ pwd
```

*/users/vlad/tmp*

```
$ mkdir tmp
```

```
$ nano tmp/junk
```

```
$ ls tmp
```

*junk*

```
$ mv tmp/junk tmp/quotes.txt
```

```
$ ls tmp
```

*quotes.txt*

```
$ mv tmp/quotes.txt .
```

```
$ ls tmp
```



nothing left in tmp

```
$
```

```
$ pwd
```

*/users/vlad/tmp*

```
$ mkdir tmp
```

```
$ nano tmp/junk
```

```
$ ls tmp
```

*junk*

```
$ mv tmp/junk tmp/quotes.txt
```

```
$ ls tmp
```

*quotes.txt*

```
$ mv tmp/quotes.txt .
```

```
$ ls tmp
```

```
$ ls quotes.txt
```

*quotes.txt*

← quotes.txt now in this directory

\$ **pwd**

*/users/vlad/tmp*

\$ **mkdir tmp**

\$ **nano tmp/junk**

\$ **ls tmp**

*junk*

\$ **mv tmp/junk tmp/quotes.txt**

\$ **ls tmp**

*quotes.txt*

\$ **mv tmp/quotes.txt .**

\$ **ls tmp**

\$ **ls quotes.txt**

*quotes.txt*



ls with a file or directory argument  
lists that file or directory

```
$ cp quotes.txt tmp/quotations.txt
```

```
$
```

copy a file

```
$ cp quotes.txt tmp/quotations.txt
```

```
$ ls quotes.txt tmp/quotations.txt
```

*quotes.txt tmp/quotations.txt*

```
$
```

```
$ cp quotes.txt tmp/quotations.txt
```

```
$ ls quotes.txt tmp/quotations.txt
```

*quotes.txt tmp/quotations.txt*

```
$ rm quotes.txt
```

```
$
```

```
$ cp quotes.txt tmp/quotations.txt
```

```
$ ls quotes.txt tmp/quotations.txt
```

*quotes.txt tmp/quotations.txt*

```
$ rm quotes.txt
```

```
$ ls quotes.txt tmp/quotations.txt
```

*ls: cannot access quotes.txt: No such file or directory*

*tmp/quotations.txt*

```
$
```

```
$ cp quotes.txt tmp/quotations.txt
```

```
$ ls quotes.txt tmp/quotations.txt
```

*quotes.txt tmp/quotations.txt*

```
$ rm quotes.txt
```

```
$ ls quotes.txt tmp/quotations.txt
```

*ls: cannot access quotes.txt: No such file or directory*

*tmp/quotations.txt*

```
$ cp tmp/quotations.txt .
```

```
$ ls quotations.txt
```

*quotations.txt*

```
$
```

```
$ cp quotes.txt tmp/quotations.txt
```

```
$ ls quotes.txt tmp/quotations.txt
```

*quotes.txt tmp/quotations.txt*

```
$ rm quotes.txt
```

```
$ ls quotes.txt tmp/quotations.txt
```

*ls: cannot access quotes.txt: No such file or directory*

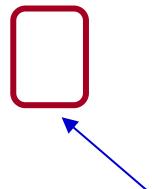
*tmp/quotations.txt*

```
$ cp tmp/quotations.txt .
```

```
$ ls quotations.txt
```

*quotations.txt*

```
$
```



this is a directory, so the copy has the same name as the original file

pwd	print working directory
cd	change working directory
ls	listing
.	current directory
..	parent directory
mkdir	make a directory
nano	text editor
rm	remove (delete) a file
rmdir	remove (delete) a directory
mv	move (rename) a file or directory
cp	copy a file



# The Unix Shell

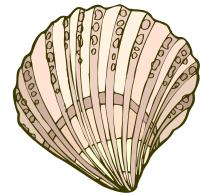
## Pipes and Filters



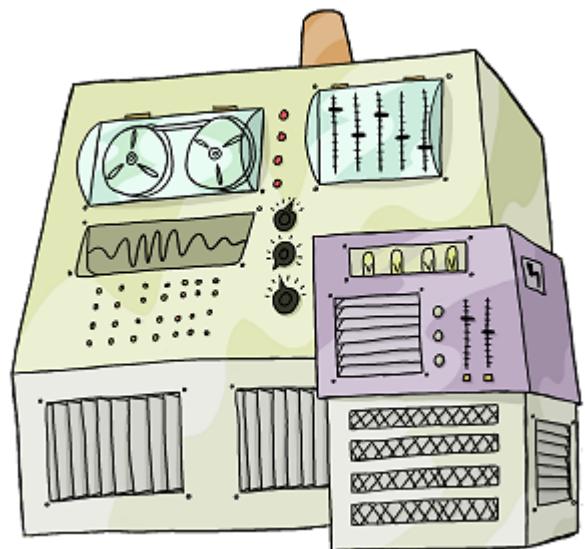
Copyright © Software Carpentry 2010

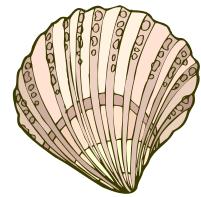
This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.

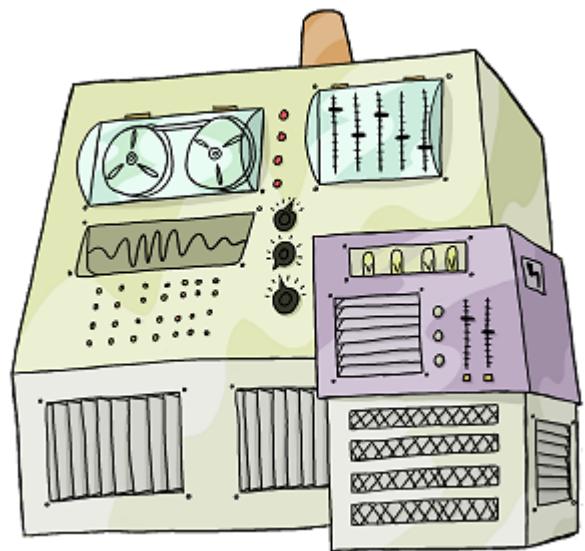


shell

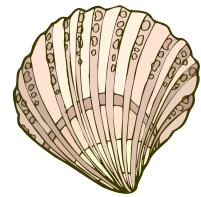




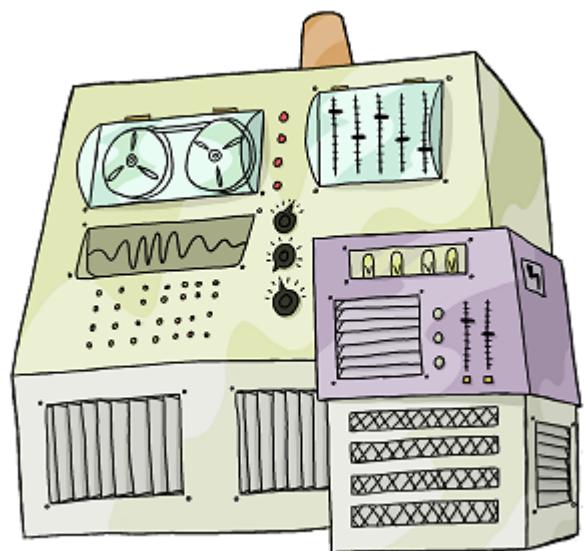
shell



pwd	mkdir
cd	nano
ls	rm
.	rmdir
..	mv
	cp



shell



pwd	mkdir
cd	nano
ls	rm
.	rmdir
..	mv
	cp

*More powerful  
when combined*

**\$ ls molecules**

*cubane.pdb ethane.pdb methane.pdb*

*octane.pdb pentane.pdb propane.pdb*

**\$**

**\$ ls molecules**

*cubane.pdb ethane.pdb methane.pdb*

*octane.pdb pentane.pdb propane.pdb*

**\$ cd molecules**

**\$**

**\$ ls molecules**

*cubane.pdb ethane.pdb methane.pdb*

*octane.pdb pentane.pdb propane.pdb*

**\$ cd molecules**

**\$ wc \*.pdb**

\$ ls molecules

*cubane.pdb ethane.pdb methane.pdb*

*octane.pdb pentane.pdb propane.pdb*

\$ cd molecules

\$ wc \*.pdb



\* is a *wild card*

`$ ls molecules`

*cubane.pdb ethane.pdb methane.pdb*

*octane.pdb pentane.pdb propane.pdb*

`$ cd molecules`

`$ wc *.pdb`



*\* is a wild card*

*matches zero or more characters*

`$ ls molecules`

*cubane.pdb ethane.pdb methane.pdb*

*octane.pdb pentane.pdb propane.pdb*

`$ cd molecules`

`$ wc *.pdb`



*\* is a wild card*

matches zero or more characters

so `*.pdb` matches all filenames

ending in `.pdb`

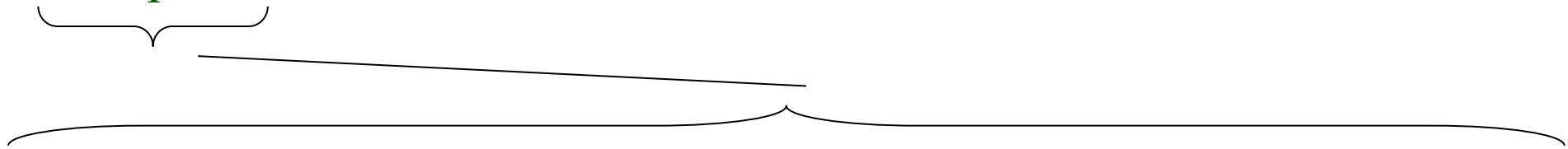
**\$ ls molecules**

*cubane.pdb ethane.pdb methane.pdb*

*octane.pdb pentane.pdb propane.pdb*

**\$ cd molecules**

**\$ wc \*.pdb**



Hand-drawn curly braces are drawn over the command line. The first brace is positioned under the command 'wc' and extends to the right, covering the file names 'cubane.pdb', 'ethane.pdb', 'methane.pdb', 'octane.pdb', 'pentane.pdb', and 'propane.pdb'. The second brace is positioned under the file names 'cubane.pdb', 'ethane.pdb', 'methane.pdb', 'octane.pdb', 'pentane.pdb', and 'propane.pdb'.

*wc cubane.pdb ethane.pdb methane.pdb octane.pdb pentane.pdb propane.pdb*

\$ ls molecules

*cubane.pdb ethane.pdb methane.pdb*

*octane.pdb pentane.pdb propane.pdb*

\$ cd molecules

\$ wc \*.pdb

← word count

**\$ ls molecules**

*cubane.pdb ethane.pdb methane.pdb*

*octane.pdb pentane.pdb propane.pdb*

**\$ cd molecules**

**\$ wc \*.pdb**



word count

counts lines, words, and  
characters in files

**\$ ls molecules**

*cubane.pdb ethane.pdb methane.pdb*

*octane.pdb pentane.pdb propane.pdb*

**\$ cd molecules**

**\$ wc \*.pdb**

*20 156 1158 cubane.pdb*

*12 84 622 ethane.pdb*

*9 57 422 methane.pdb*

*30 246 1828 octane.pdb*

*21 165 1226 pentane.pdb*

*15 111 825 propane.pdb*

*107 819 6081 total*

**\$**

```
$ wc -l *.pdb
```

report only lines

20 *cubane.pdb*

12 *ethane.pdb*

9 *methane.pdb*

30 *octane.pdb*

21 *pentane.pdb*

15 *propane.pdb*

107 *total*

\$

```
$ wc -l *.pdb
```

20 *cubane.pdb*

12 *ethane.pdb*

9 *methane.pdb*

30 *octane.pdb*

21 *pentane.pdb*

15 *propane.pdb*

107 *total*

```
$
```

report only lines

use -w for words or  
-c for characters

20 *cubane.pdb*  
12 *ethane.pdb*  
9 *methane.pdb*  
30 *octane.pdb*  
21 *pentane.pdb*  
15 *propane.pdb*  
107 *total*

Which file is shortest?

20 *cubane.pdb*  
12 *ethane.pdb*  
9 *methane.pdb*  
30 *octane.pdb*  
21 *pentane.pdb*  
15 *propane.pdb*  
107 *total*

Which file is shortest?

Easy to see when there are six...

20 *cubane.pdb*  
12 *ethane.pdb*  
9 *methane.pdb*  
30 *octane.pdb*  
21 *pentane.pdb*  
15 *propane.pdb*  
107 *total*

Which file is shortest?

Easy to see when there are six...

...but what if there were 6000?

```
$ wc -l *.pdb > lengths
```

```
$
```

```
$ wc -l *.pdb > lengths
```

```
$
```

*redirect output to a file*

```
$ wc -l *.pdb > lengths
```

```
$
```

*redirect output to a file  
create file if it doesn't exist*

```
$ wc -l *.pdb > lengths
```

```
$
```

*redirect output to a file  
create file if it doesn't exist  
overwrite it if it does*

```
$ wc -l *.pdb > lengths
```

```
$
```

no screen output

```
$ wc -l *.pdb > lengths
```

```
$ ls lengths
```

*lengths*

```
$
```

```
$ wc -l *.pdb > lengths
```

```
$ ls lengths
```

*lengths*

```
$ cat lengths
```

*20 cubane.pdb*

*12 ethane.pdb*

*9 methane.pdb*

*30 octane.pdb*

*21 pentane.pdb*

*15 propane.pdb*

*107 total*

```
$
```

```
$ wc -l *.pdb > lengths
```

```
$ ls lengths
```

*lengths*

```
$ cat lengths
```

20 *cubane.pdb*

12 *ethane.pdb*

9 *methane.pdb*

30 *octane.pdb*

21 *pentane.pdb*

15 *propane.pdb*

107 *total*

```
$
```



*concatenate files*

```
$ wc -l *.pdb > lengths
```

```
$ ls lengths
```

*lengths*

```
$ cat lengths
```

20 *cubane.pdb*

12 *ethane.pdb*

9 *methane.pdb*

30 *octane.pdb*

21 *pentane.pdb*

15 *propane.pdb*

107 *total*

```
$
```



*concatenate* files

in this case, only one

so file contents printed to screen

\$ sort lengths

9 *methane.pdb*

12 *ethane.pdb*

15 *propane.pdb*

20 *cubane.pdb*

21 *pentane.pdb*

30 *octane.pdb*

107 *total*

\$

\$ sort lengths > sorted-lengths

\$

```
$ sort lengths > sorted-lengths
```

```
$ head -1 sorted-lengths
```

*9 methane.pdb*

```
$
```

```
$ sort lengths > sorted-lengths
```

```
$ head -1 sorted-lengths
```

9 *methane.pdb*

```
$
```

get the first line of the file

```
$ sort lengths > sorted-lengths
```

```
$ head -1 sorted-lengths
```

9 *methane.pdb*

```
$
```

get the first line of the file  
this must be the PDB file  
with the fewest lines,  
since sorted-lengths holds  
files and line counts in  
order from least to greatest

```
$ sort lengths > sorted-lengths
```

```
$ head -1 sorted-lengths
```

9 *methane.pdb*

```
$
```

not particularly obvious

get the first line of the file  
this must be the PDB file  
with the fewest lines,  
since sorted-lengths holds  
files and line counts in  
order from least to greatest

```
$ sort lengths | head -1
```

*9 methane.pdb*

```
$
```

```
$ sort lengths | head -1
```

*9 methane.pdb*

```
$
```

*a pipe*

```
$ sort lengths | head -1
```

9 methane.pdb

```
$
```

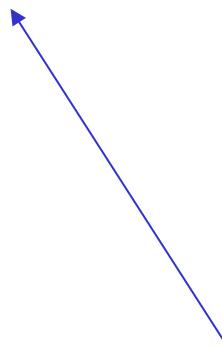
*a pipe*  
use output of left side

```
$ sort lengths | head -1
```

9 methane.pdb

\$

*a pipe*  
use output of left side  
as input to right side



```
$ sort lengths | head -1
```

*9 methane.pdb*

```
$
```

*a pipe*

*use output of left side*

*as input to right side*

*without creating temporary file*

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

*9 methane.pdb*

```
$
```

don't need to create lengths file

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

*9 methane.pdb*

```
$
```

This simple idea is why Unix has been so successful

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

*9 methane.pdb*

```
$
```

This simple idea is why Unix has been so successful

Create simple tools that:

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

*9 methane.pdb*

```
$
```

This simple idea is why Unix has been so successful

Create simple tools that:

- do one job well

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

*9 methane.pdb*

```
$
```

This simple idea is why Unix has been so successful

Create simple tools that:

- do one job well
- work well with each other

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

*9 methane.pdb*

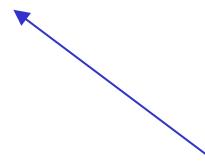
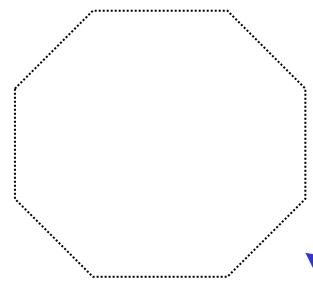
```
$
```

This simple idea is why Unix has been so successful

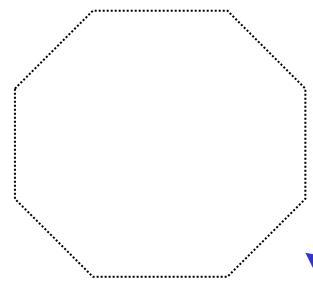
Create simple tools that:

- do one job well
- work well with each other

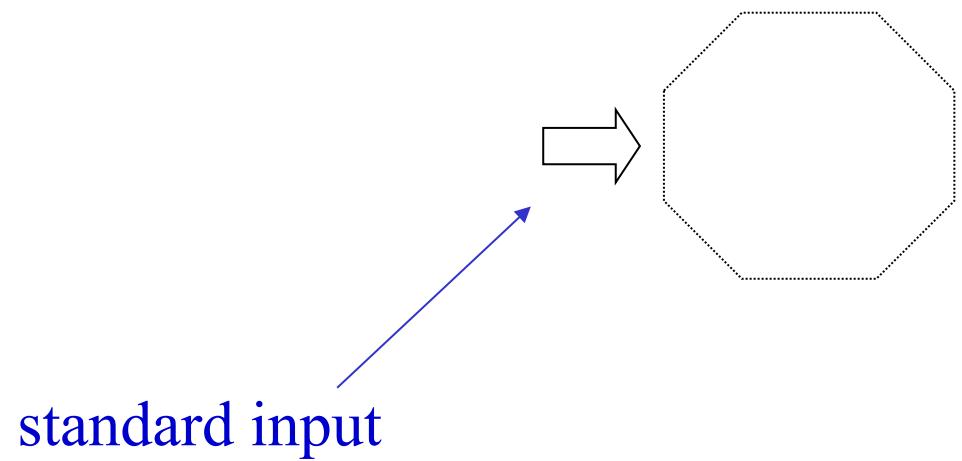
10 tools can be combined in 100 ways

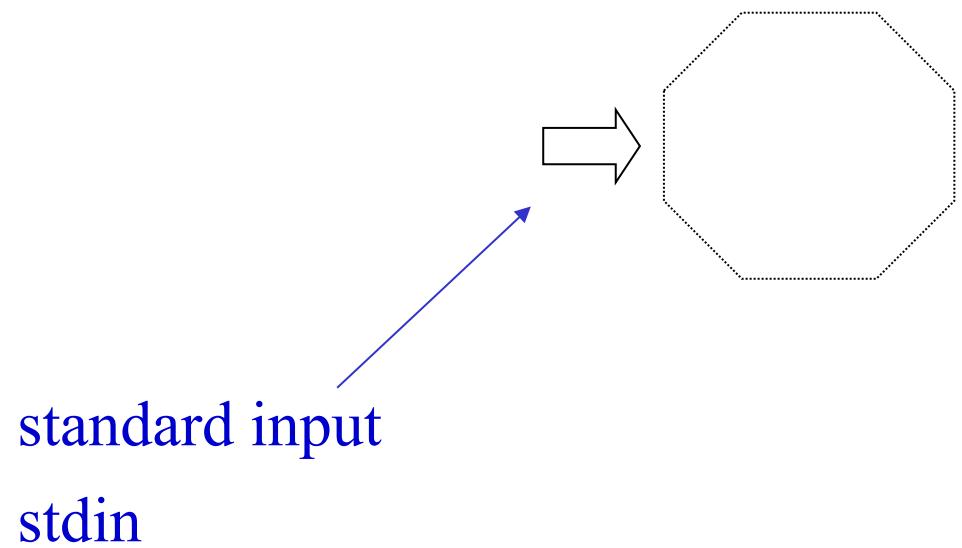


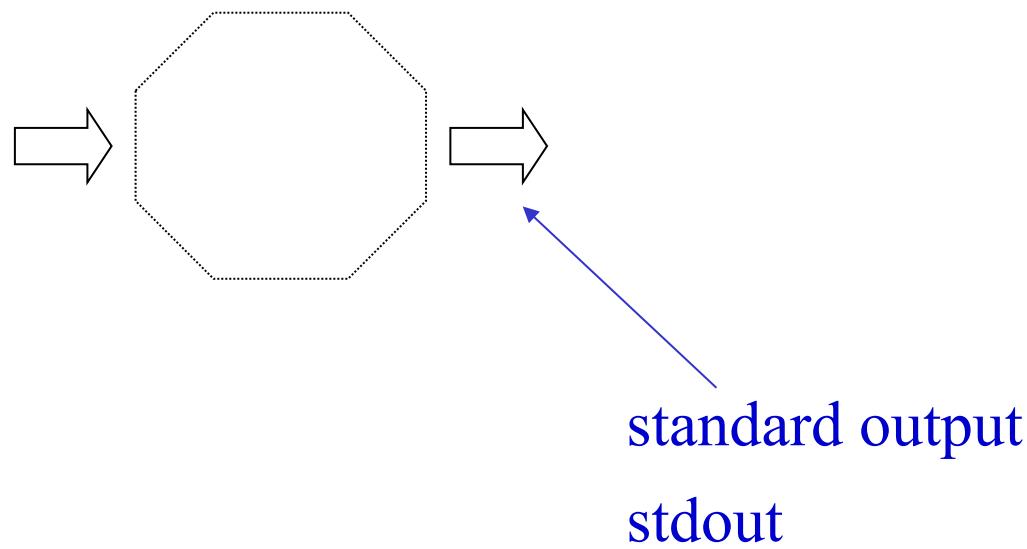
running program

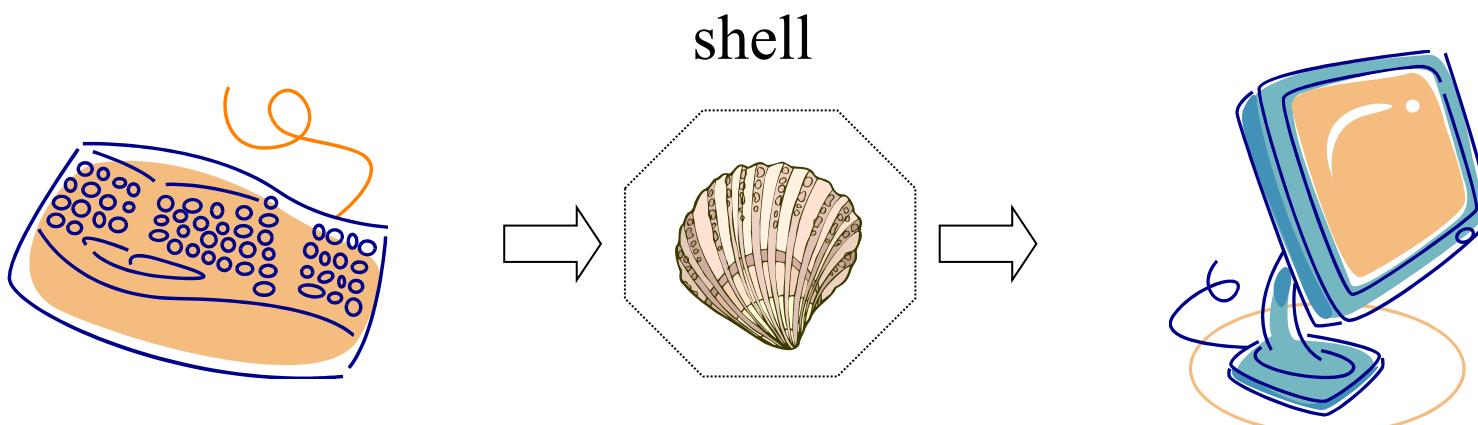


running program  
*process*

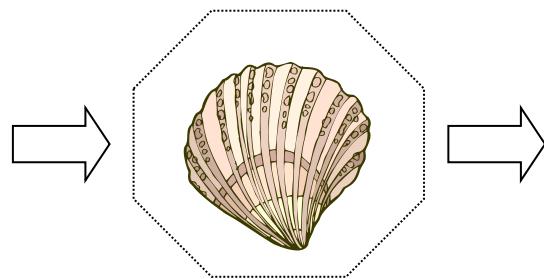




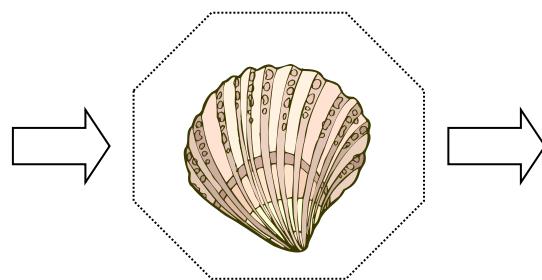




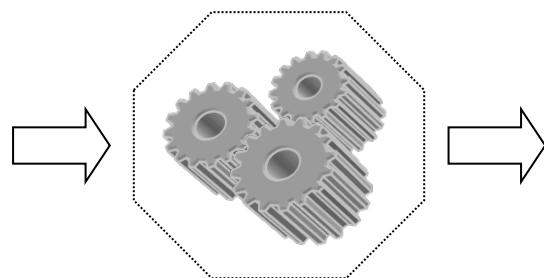
shell



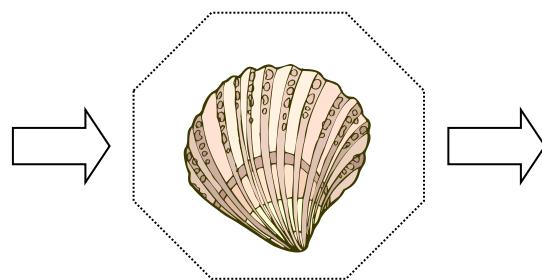
```
$ wc -l *.pdb > lengths
```



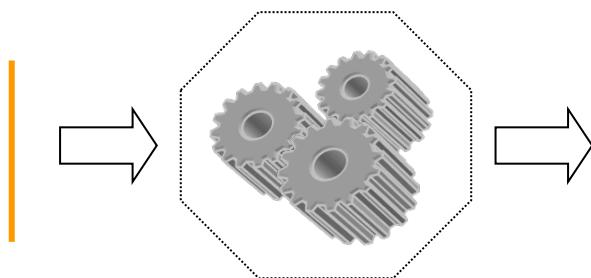
WC



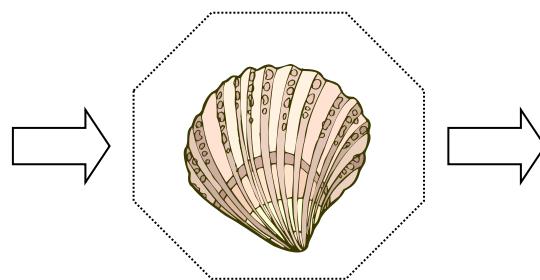
```
$ wc -l *.pdb > lengths
```



WC



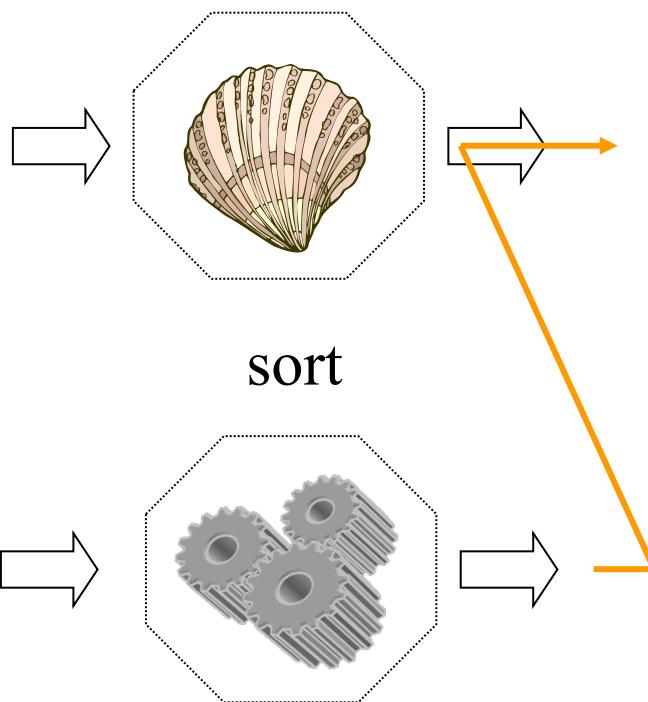
```
$ wc -l *.pdb > lengths
```



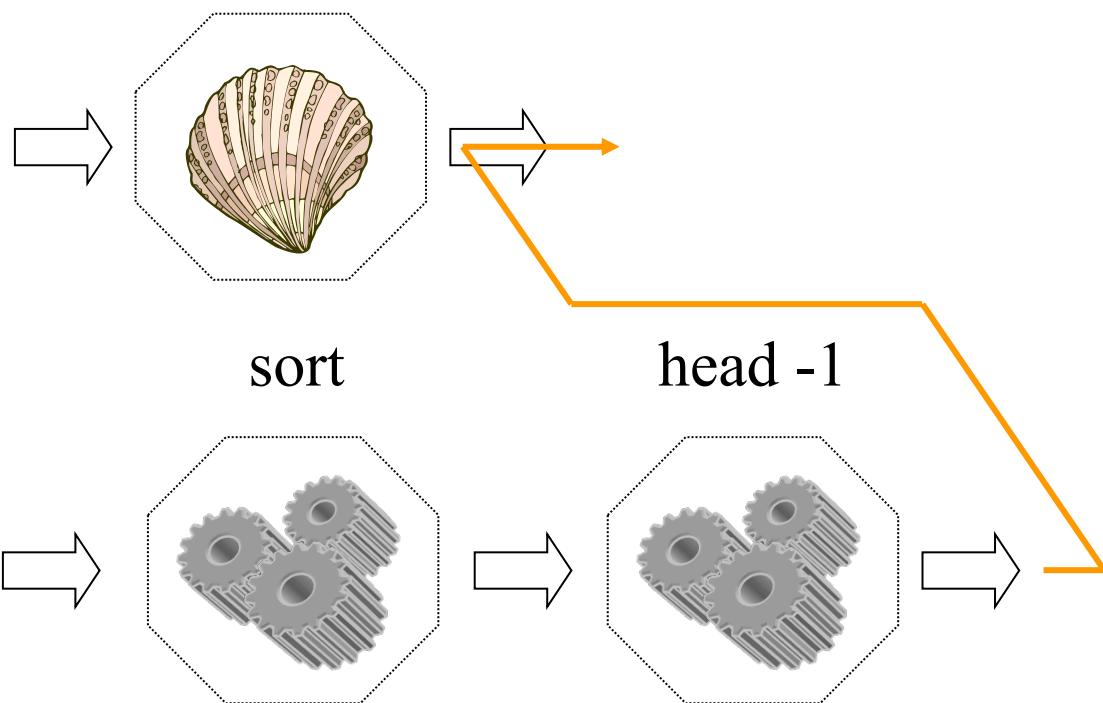
WC



```
$ wc -l *.pdb > lengths
```



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



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

This programming model called *pipes and filters*

This programming model called *pipes and filters*

A *filter* transforms a stream of input into a stream of output

This programming model called *pipes and filters*

A *filter* transforms a stream of input into a stream of output

A *pipe* connects two filters

This programming model called *pipes and filters*

A *filter* transforms a stream of input into a stream of output

A *pipe* connects two filters

Any program that reads lines of text from standard input, and writes lines of text to standard output, can work with every other

This programming model called *pipes and filters*

A *filter* transforms a stream of input into a stream of output

A *pipe* connects two filters

Any program that reads lines of text from standard input, and writes lines of text to standard output, can work with every other

You can (and should) write such programs

pwd	mkdir
cd	nano
ls	rm
.	rmdir
..	mv
	cp

pwd	mkdir	wc
cd	nano	sort
ls	rm	head
.	rmdir	
..	mv	
	cp	

pwd	mkdir	wc
cd	nano	sort
ls	rm	head
.	rmdir	<i>tail</i>
..	mv	<i>split</i>
	cp	<i>cut</i>
		<i>uniq</i>

pwd	mkdir	wc	*
cd	nano	sort	>
ls	rm	head	
.	rmdir	<i>tail</i>	
..	mv	<i>split</i>	
	cp	<i>cut</i>	
		<i>uniq</i>	

pwd	mkdir	wc	*
cd	nano	sort	>
ls	rm	head	
.	rmdir	<i>tail</i>	<
..	mv	<i>split</i>	?
	cp	<i>cut</i>	
		<i>uniq</i>	



# The Unix Shell

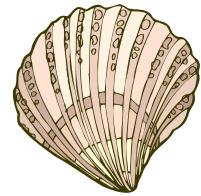
## Permissions



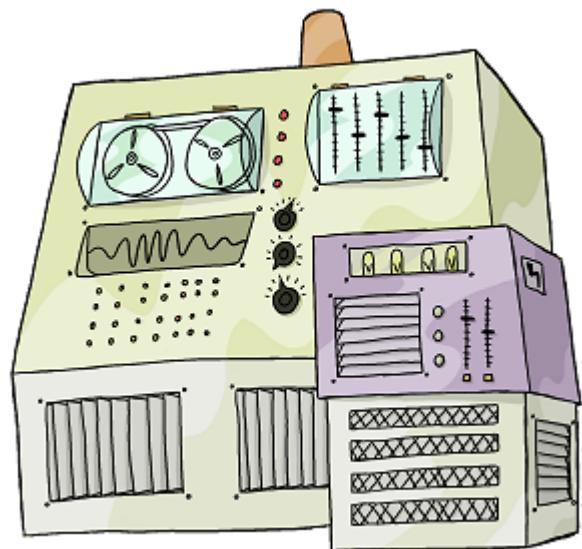
Copyright © Software Carpentry 2010

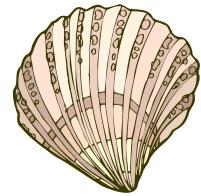
This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.



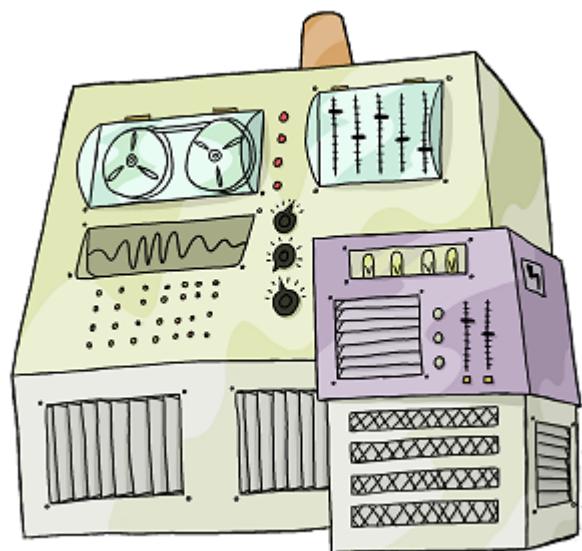
shell

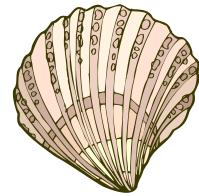




shell

pwd, mkdir, cp, ...

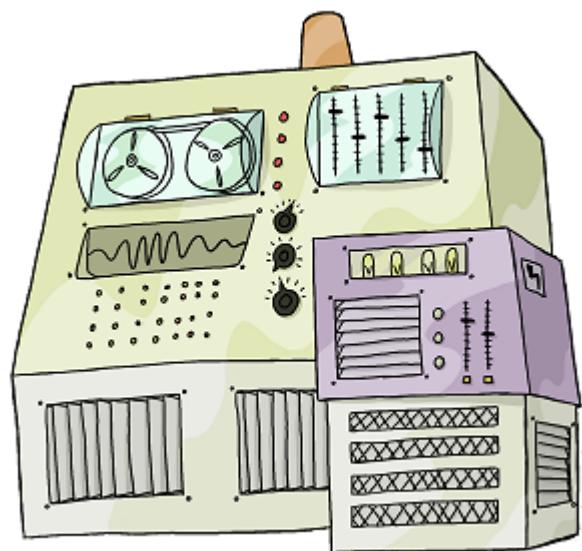


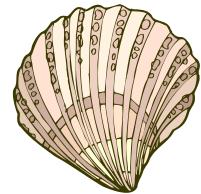


shell

pwd, mkdir, cp, ...

\*



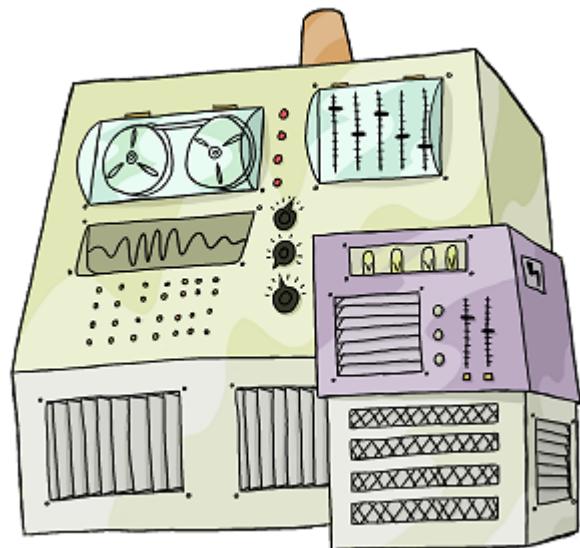


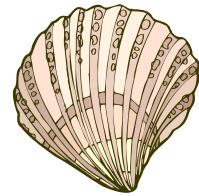
shell

pwd, mkdir, cp, ...

\*

>, |



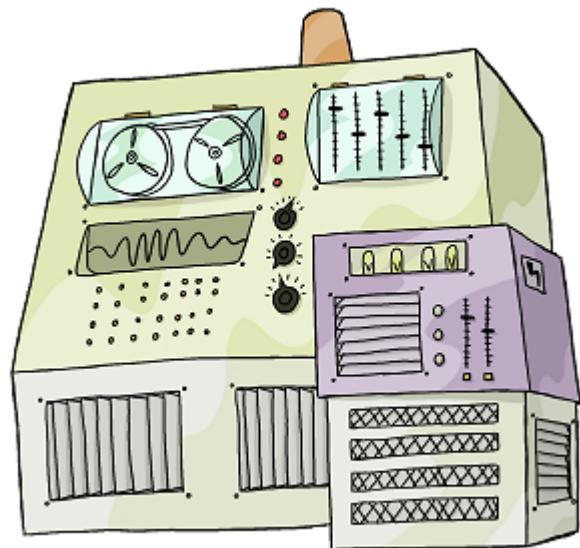


shell

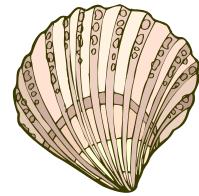
pwd, mkdir, cp, ...

\*

>, |



Who can see what?

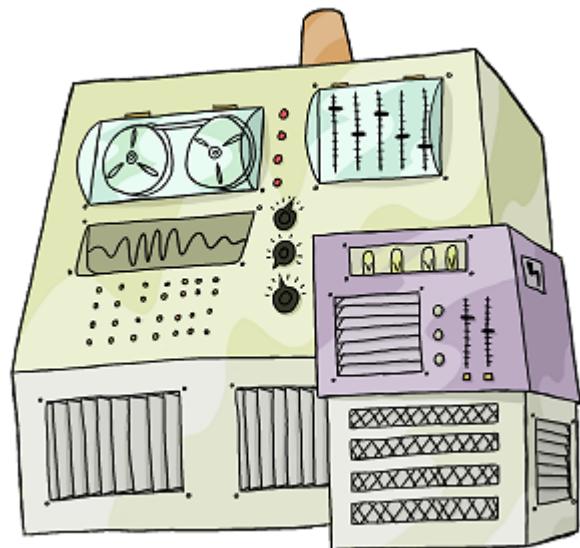


shell

pwd, mkdir, cp, ...

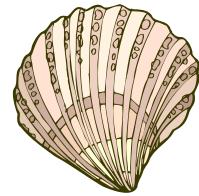
\*

>, |



Who can see what?

change

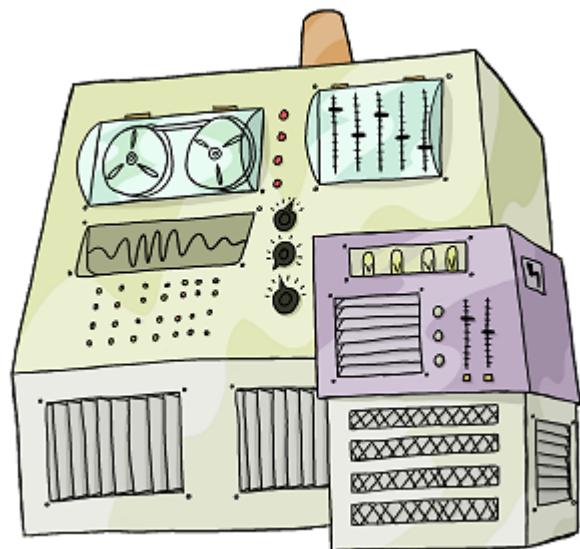


shell

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", "vlad", ...



user

Has unique *user name* and *user ID*

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

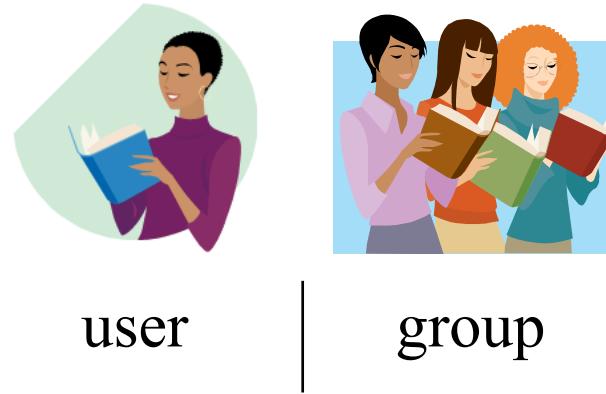
User ID is numeric (easier for computer to store)



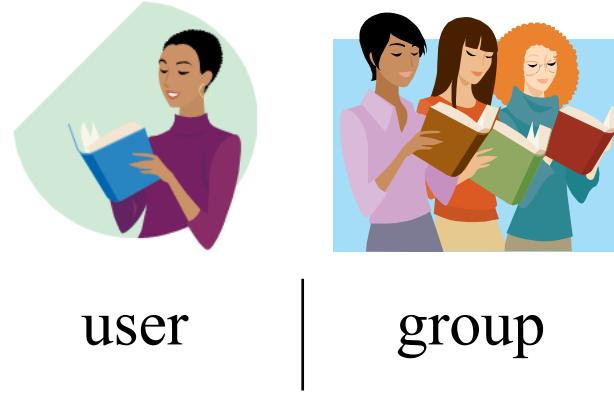
user



group

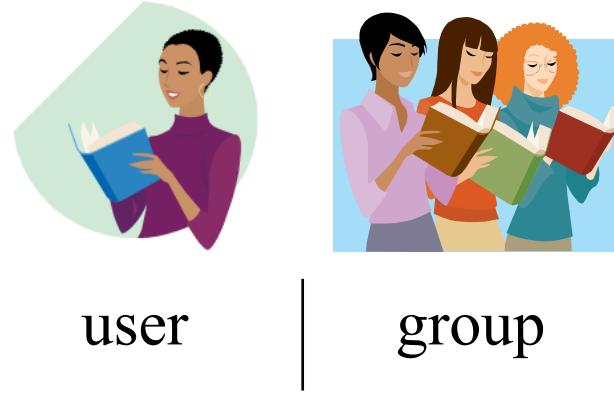


Has unique *group name* and *group ID*



Has unique *group name* and *group ID*

User can belongs to zero or more groups



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



read



user



group



all



	user	group	all
read			
write			





	user	group	all
read			
write			
execute			

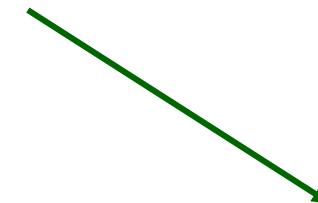




	user	group	all
read	✓	✓	✗
write	✓	✗	✗
execute	✗	✗	✗



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
```

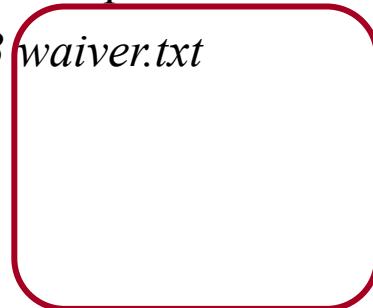
```
-rw-rw-r-- 1 vlad bio 1158 2010-07-11 08:22 safety.txt
```

```
-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
```

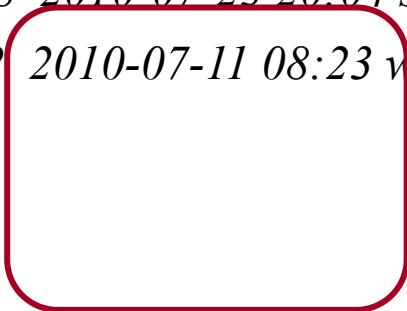
```
$
```

```
$ 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  
$
```



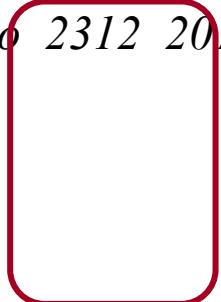
name

```
$ 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  
$
```



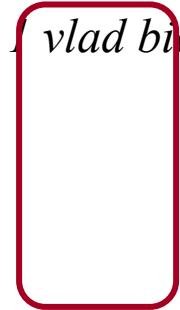
size (in bytes)

```
$ 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-rr-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?

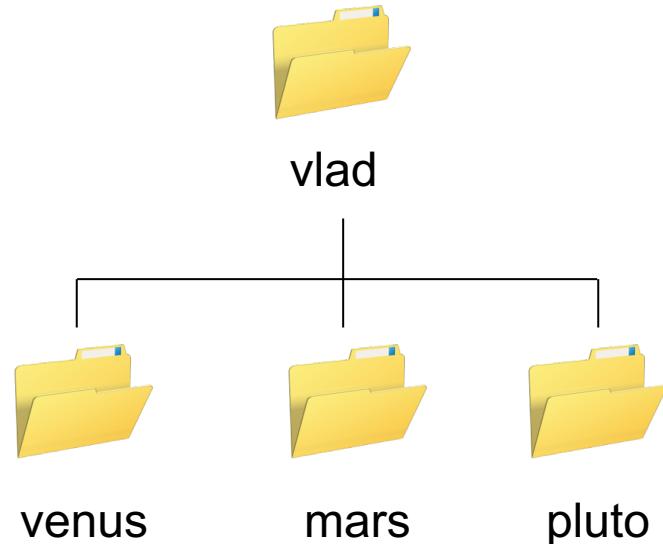
What does "execute" mean for directories?

Gives the right to *traverse*

the directory

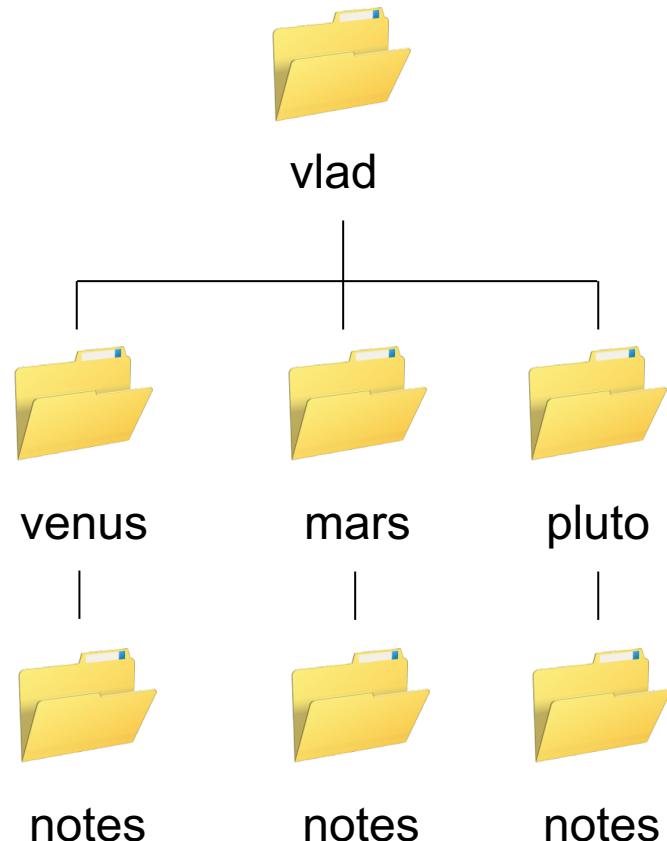
What does "execute" mean for directories?

Gives the right to *traverse*  
the directory



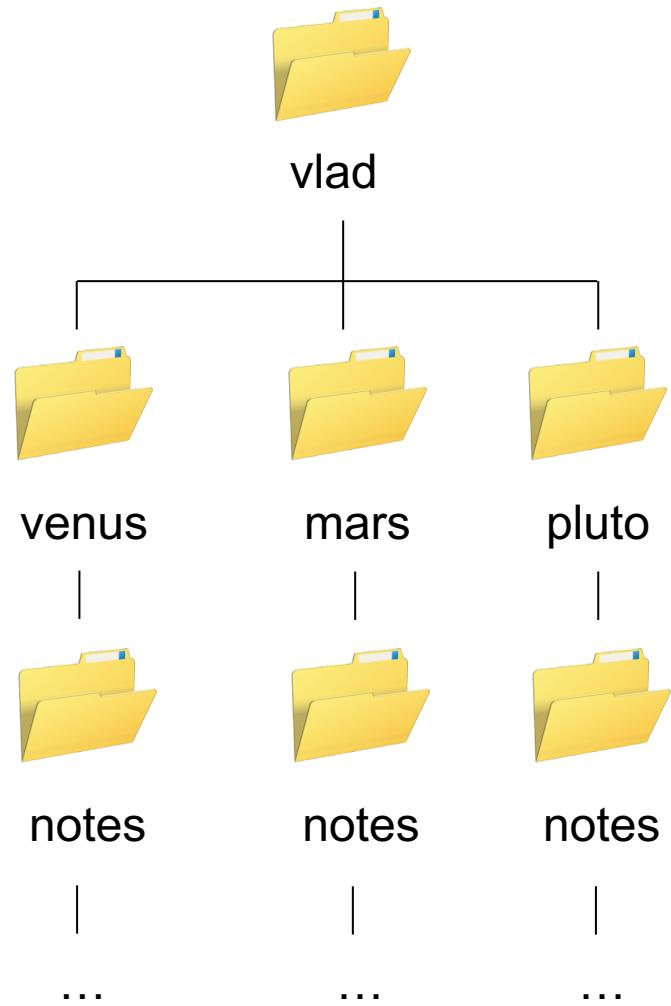
What does "execute" mean for directories?

Gives the right to *traverse*  
the directory



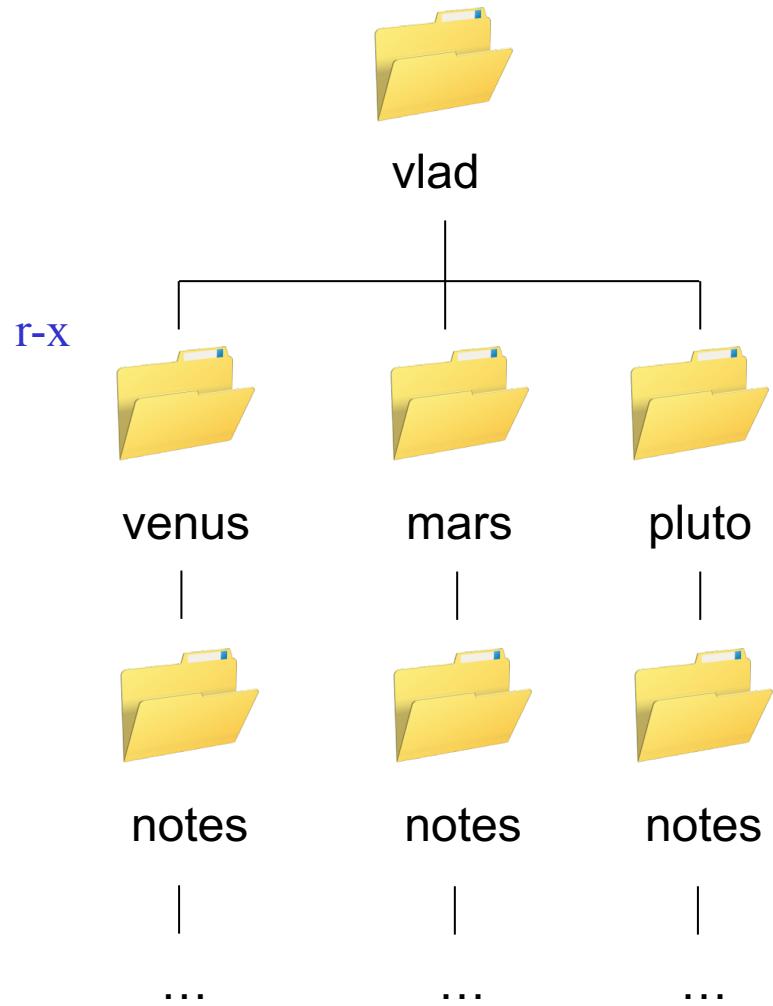
What does "execute" mean for directories?

Gives the right to *traverse*  
the directory



What does "execute" mean for directories?

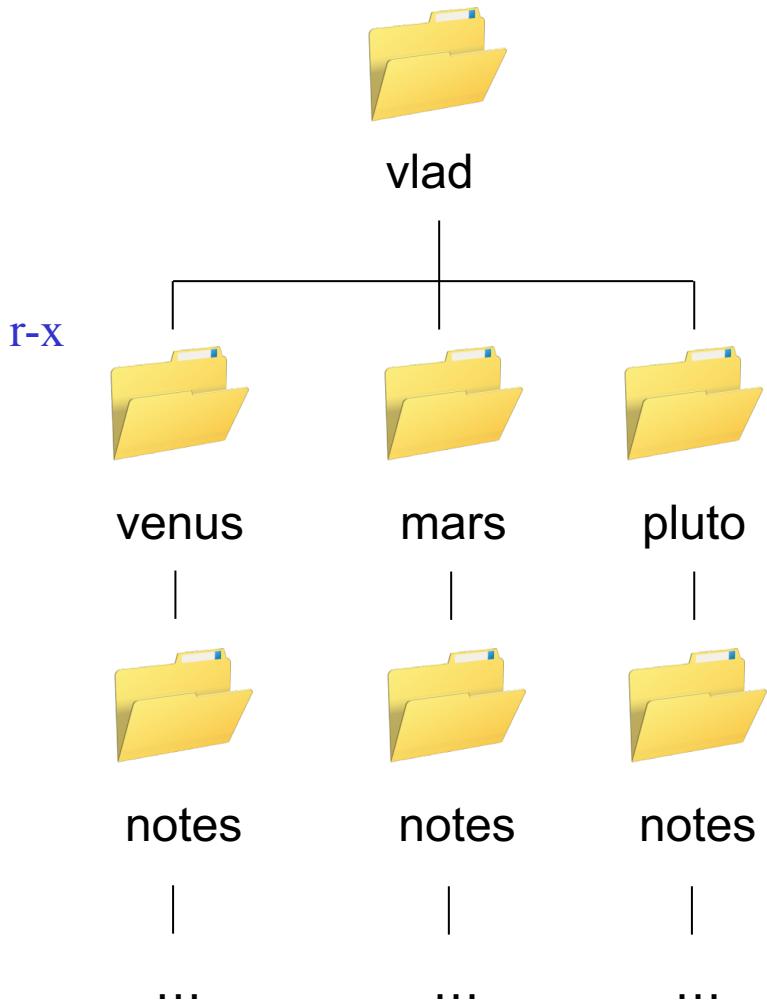
Gives the right to *traverse*  
the directory



What does "execute" mean for directories?

Gives the right to *traverse*  
the directory

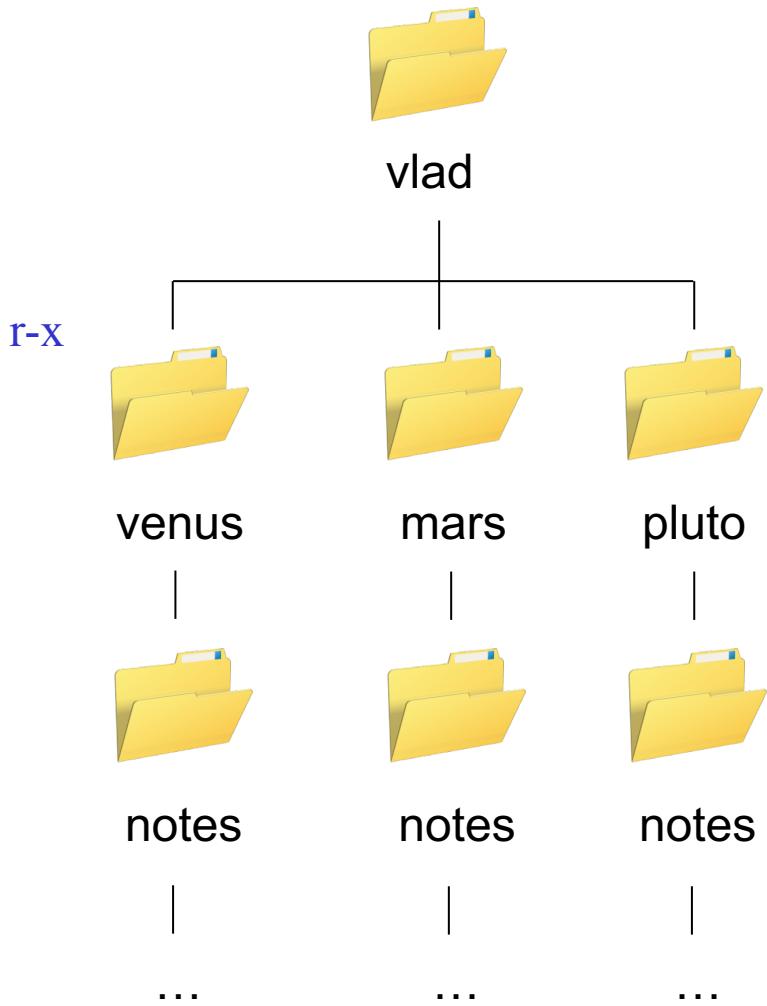
\$ ls venus venus/notes



What does "execute" mean for directories?

Gives the right to *traverse*  
the directory

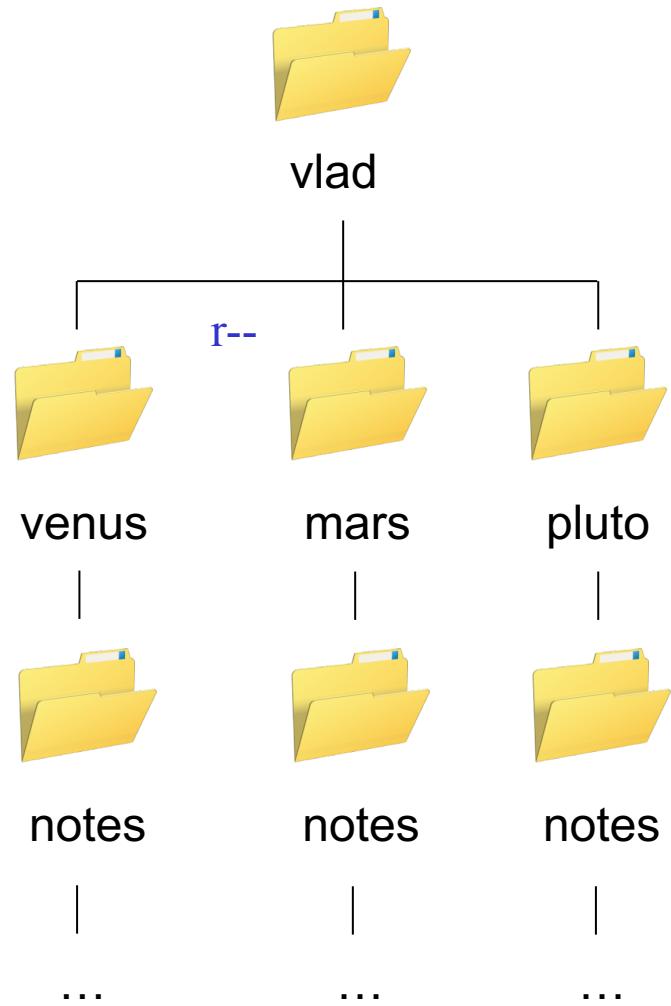
\$ ls venus venus/notes



What does "execute" mean for directories?

Gives the right to *traverse*  
the directory

\$ ls venus venus/notes

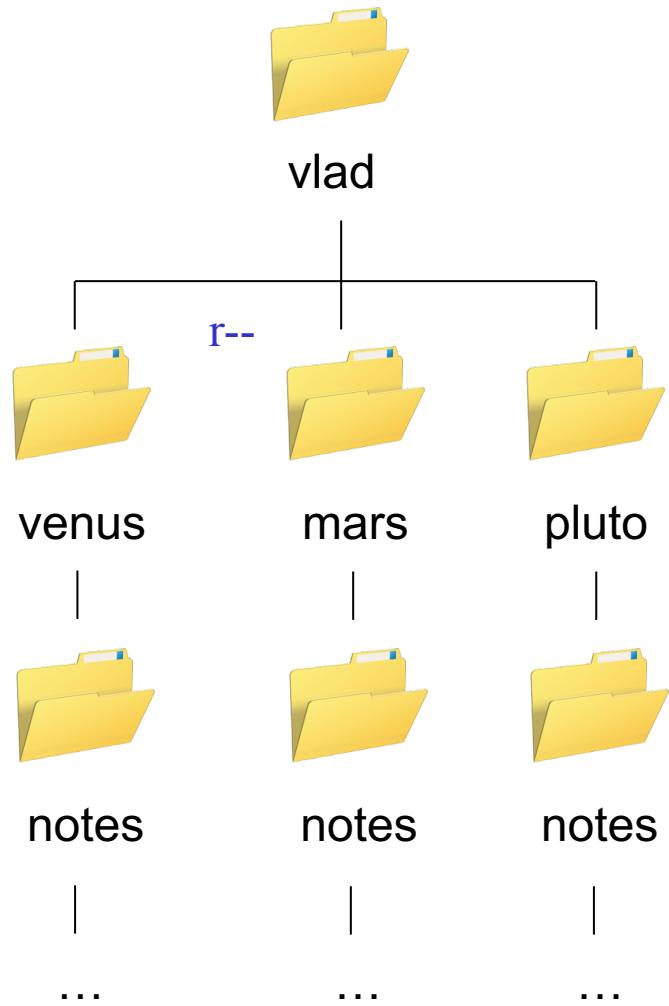


What does "execute" mean for directories?

Gives the right to *traverse*  
the directory

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

✓  
✓

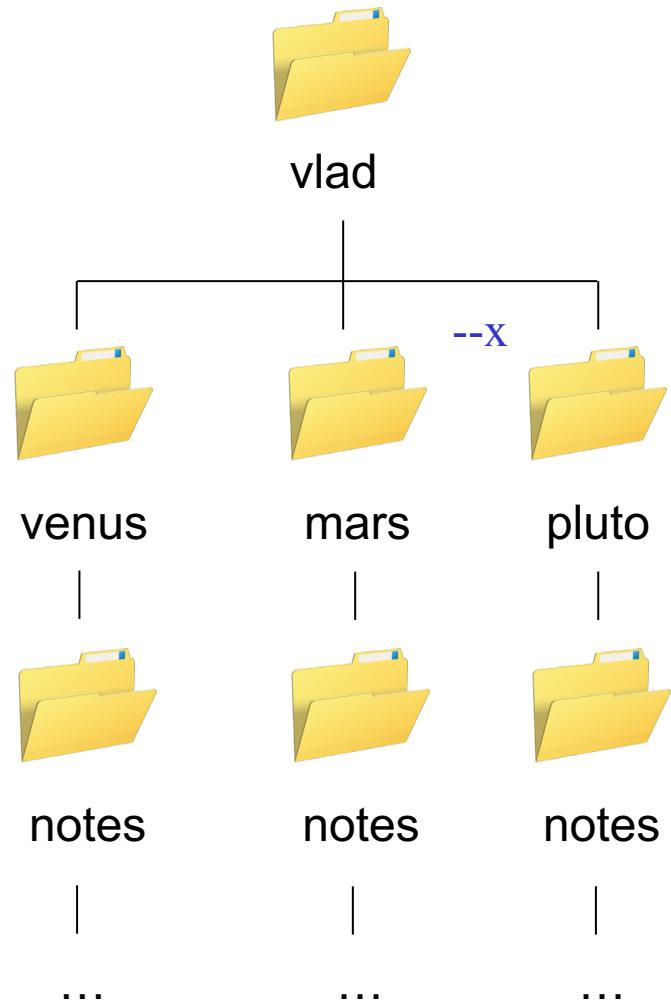


What does "execute" mean for directories?

Gives the right to *traverse*  
the directory

\$ ls venus venus/notes  
\$ ls mars mars/notes

✓  
✓

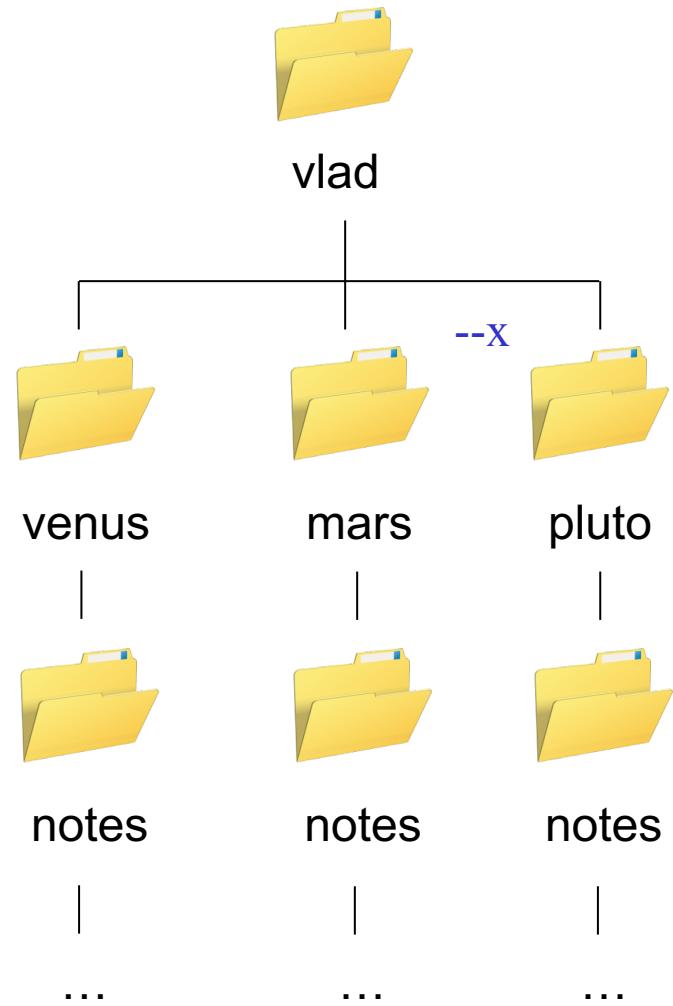


What does "execute" mean for directories?

Gives the right to *traverse*  
the directory

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

✓  
✓  
✗



What does "execute" mean for directories?

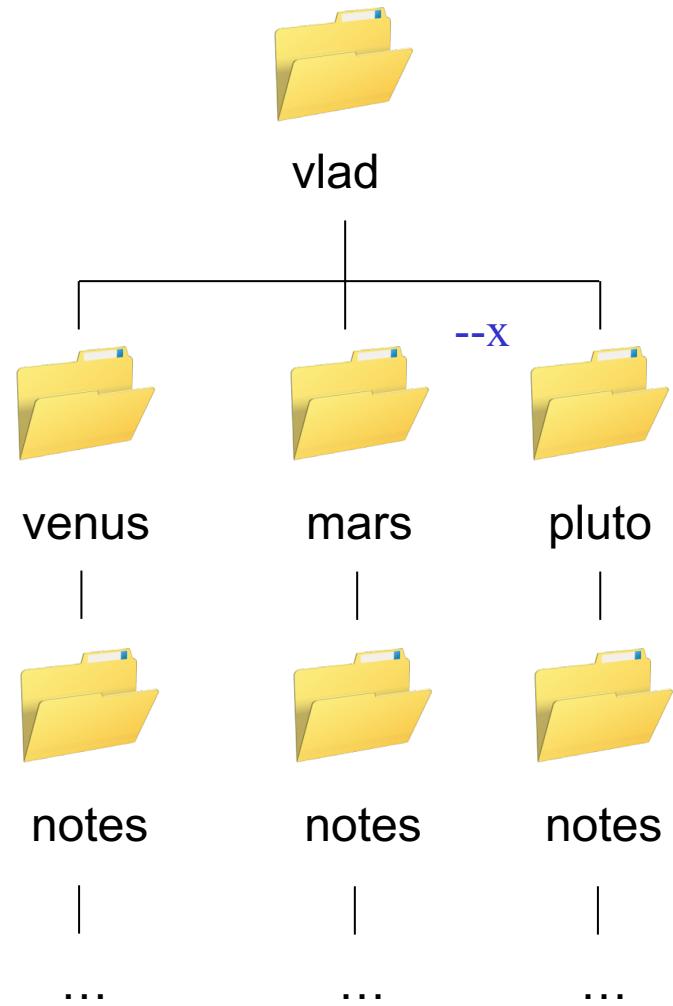
Gives the right to *traverse*  
the directory

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

✓

✓

✗

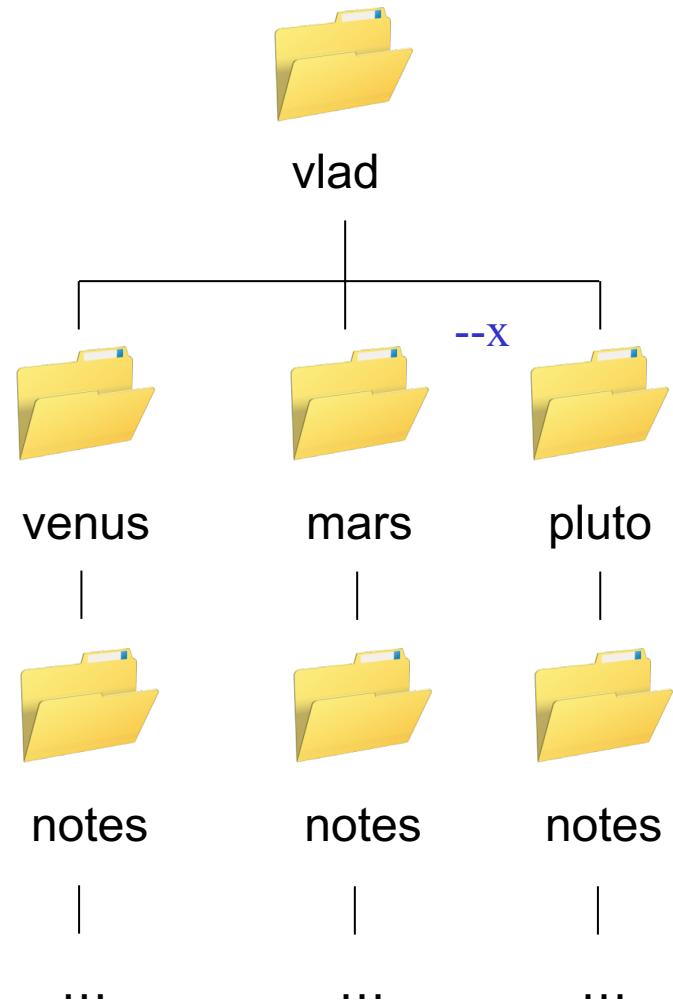


What does "execute" mean for directories?

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--r-- 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-rw-rwx 1 vlad bio 4215 2010-08-30 08:19 final.grd
```

```
$ chmod g=r final.grd; ls -l final.grd
```

```
-rw-r--r-- 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--r-- 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--r-- 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
```

<sup>^D</sup>

\$



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

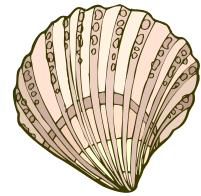
## Finding Things



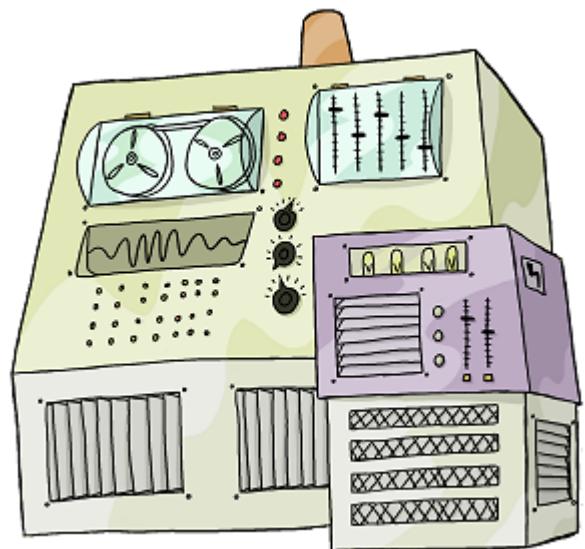
Copyright © Software Carpentry 2010

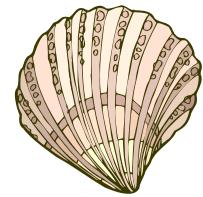
This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.

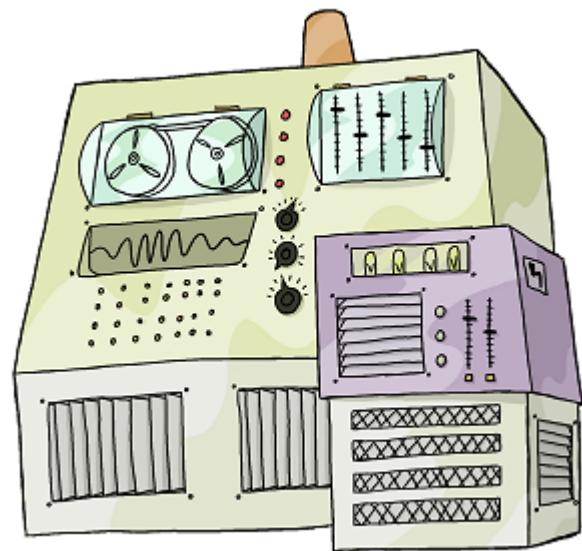


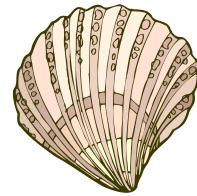
shell



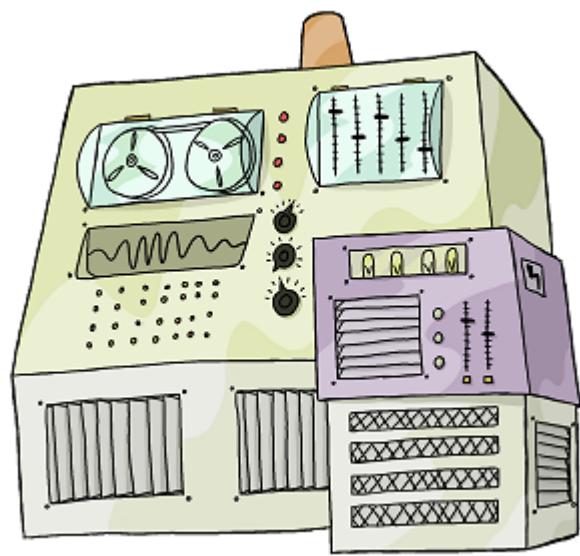


shell

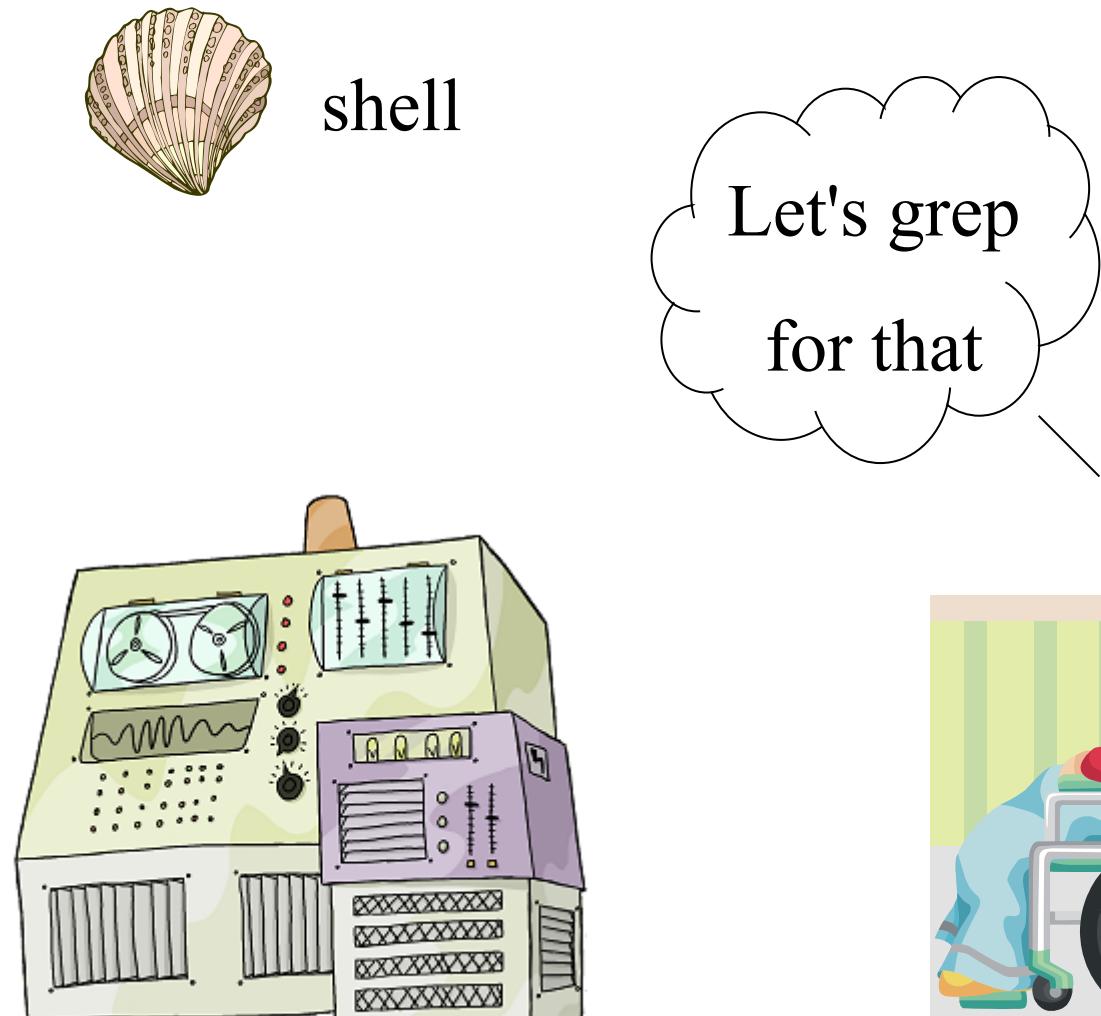




shell



Let's Google  
for that



# 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.
```

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

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.

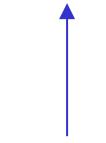
**\$ grep day haiku.txt**

*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.

**-i** case insensitive

`$ 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.

**-i** case insensitive

**-v** invert and print

non-matches

**\$ 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.*

**\$**

# 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

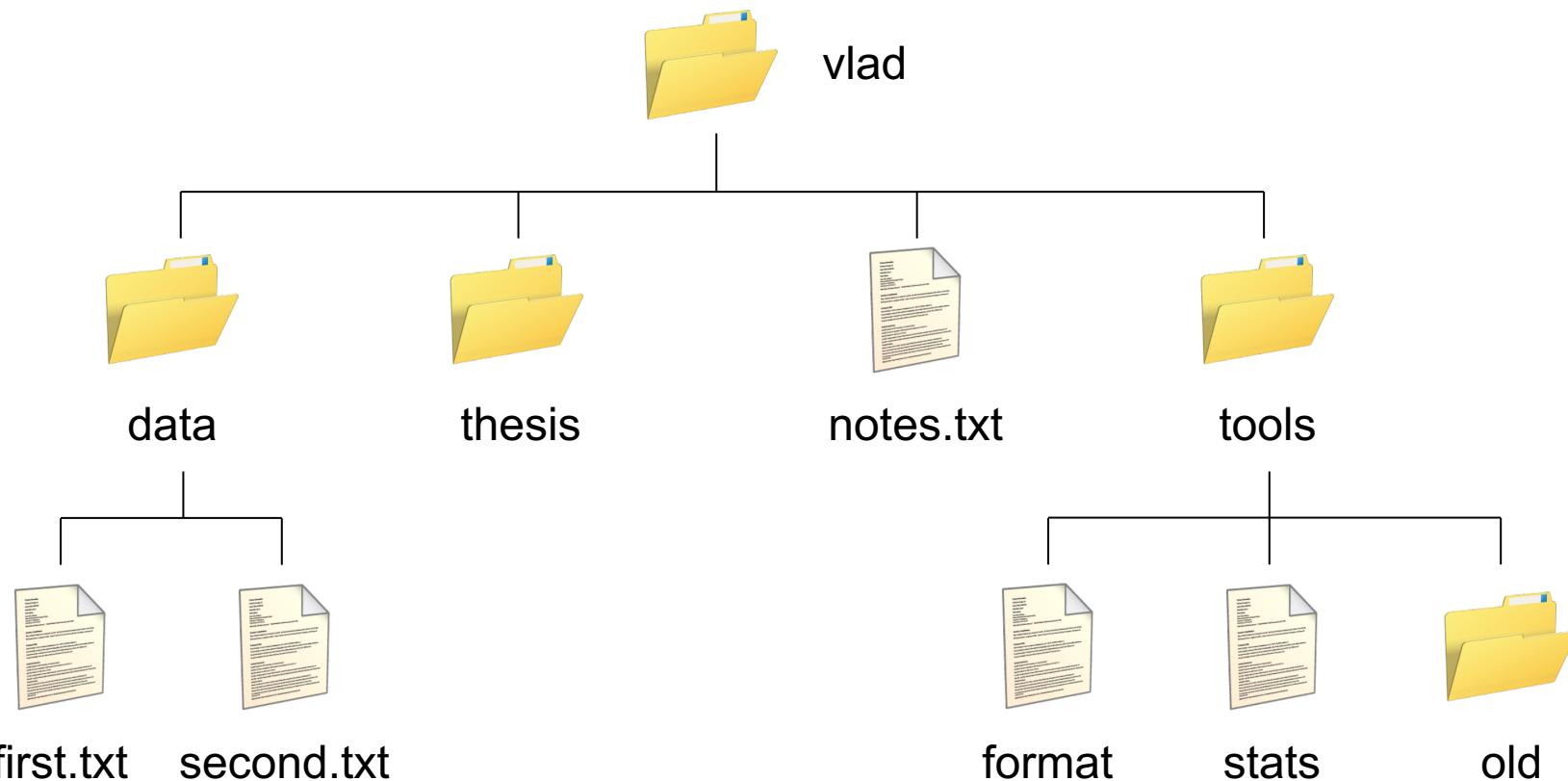
find: finds files (rather than lines in files)

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

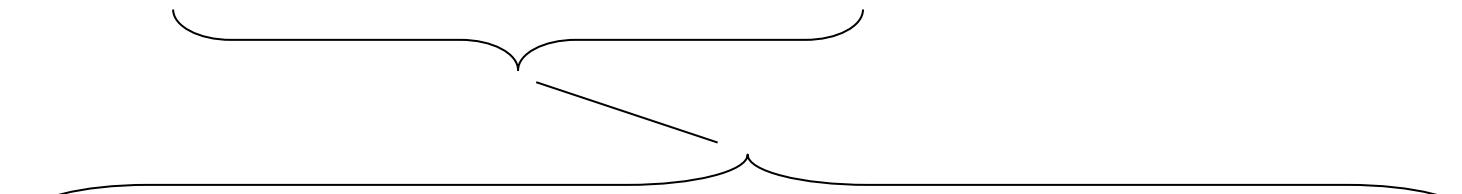
Replace what's inside with output from  
running that command

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'
```

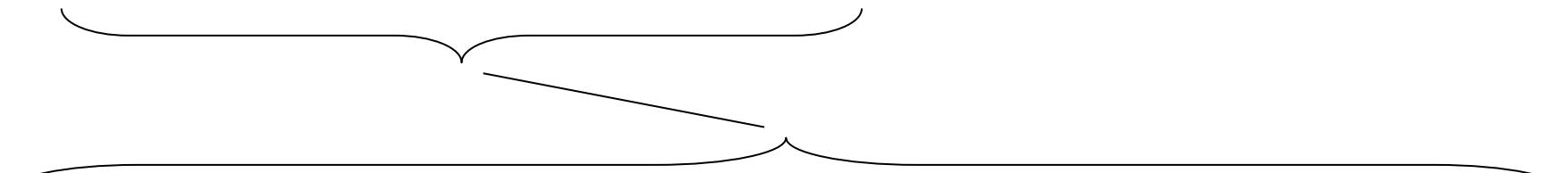


```
./data/first.txt ./data/second.txt ./notes.txt
```

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



# The Unix Shell

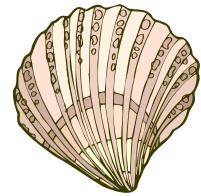
## Job Control



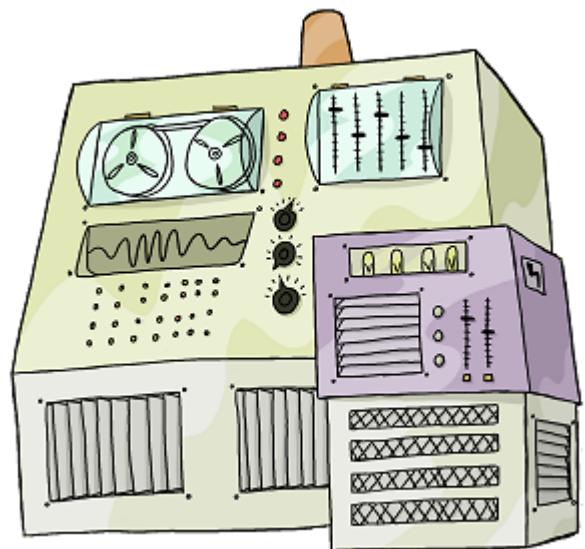
Copyright © Software Carpentry 2010

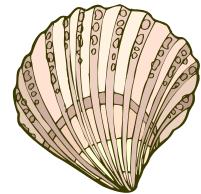
This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.



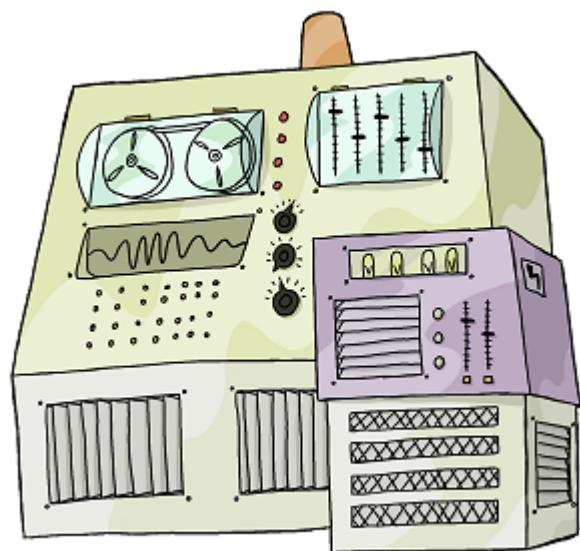
shell

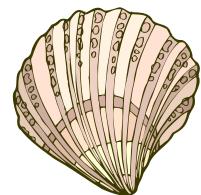




shell

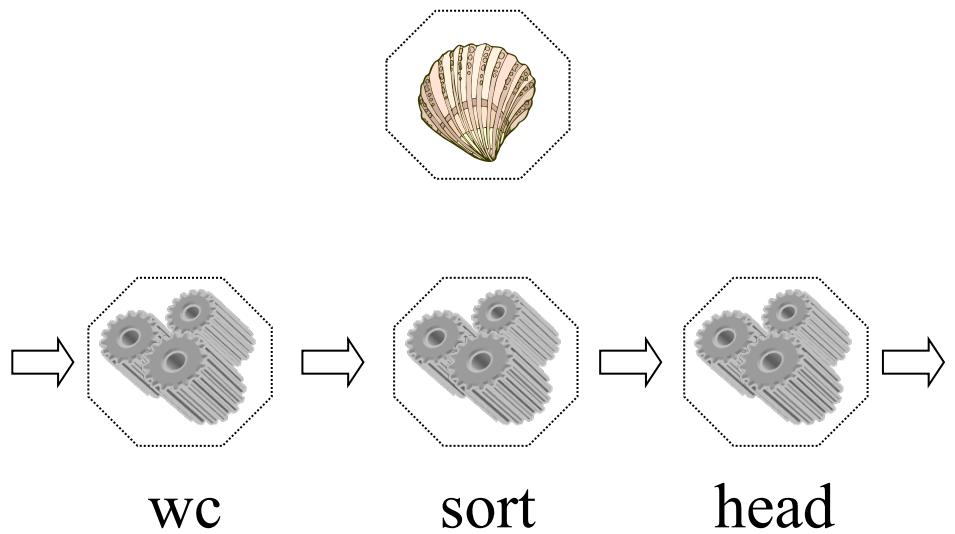
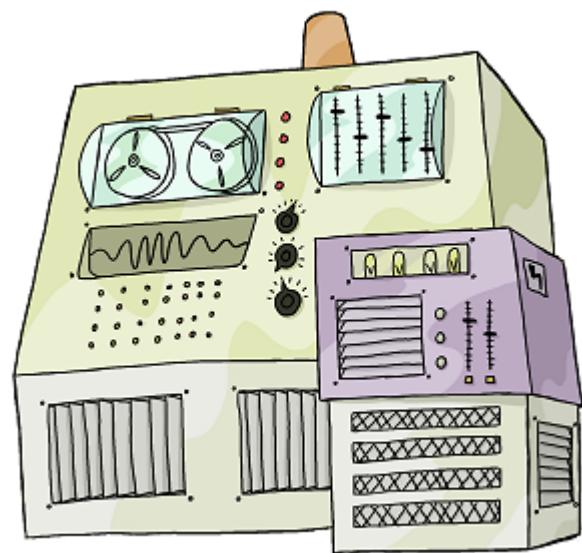
```
$ wc -l *.pdb | sort | head -1
```

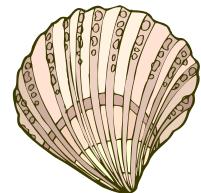




shell

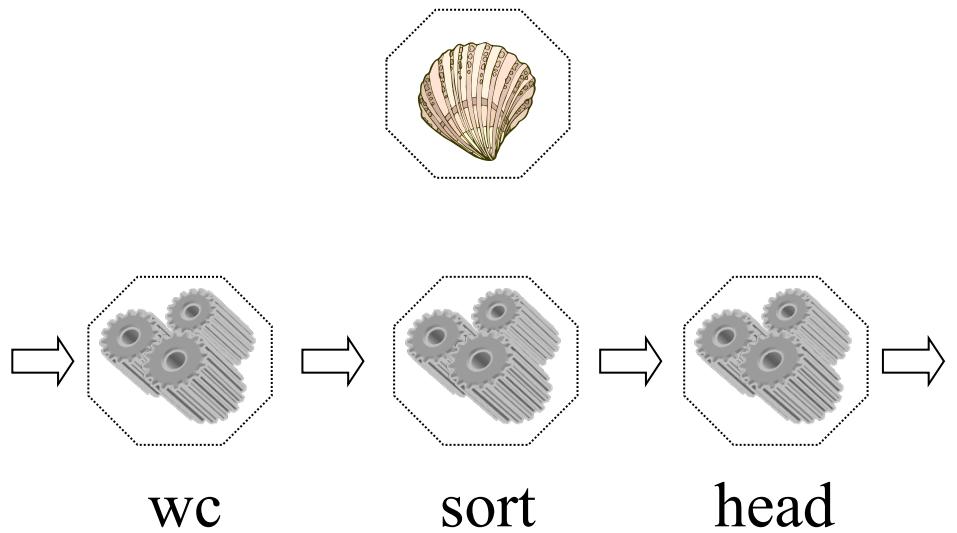
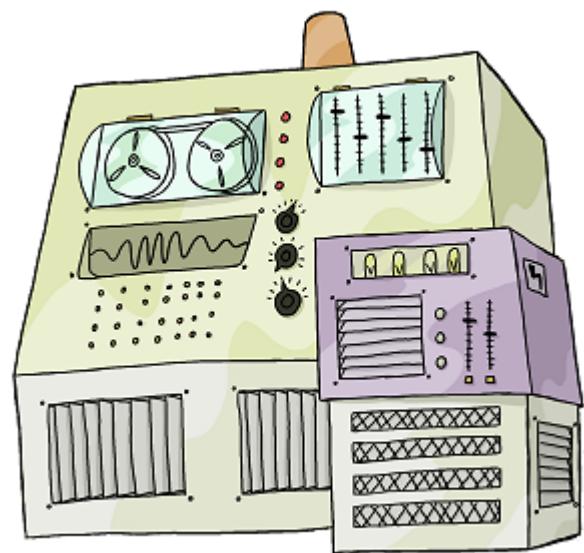
```
$ wc -l *.pdb | sort | head -1
```



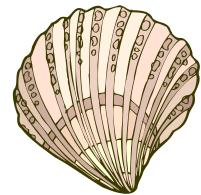


shell

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

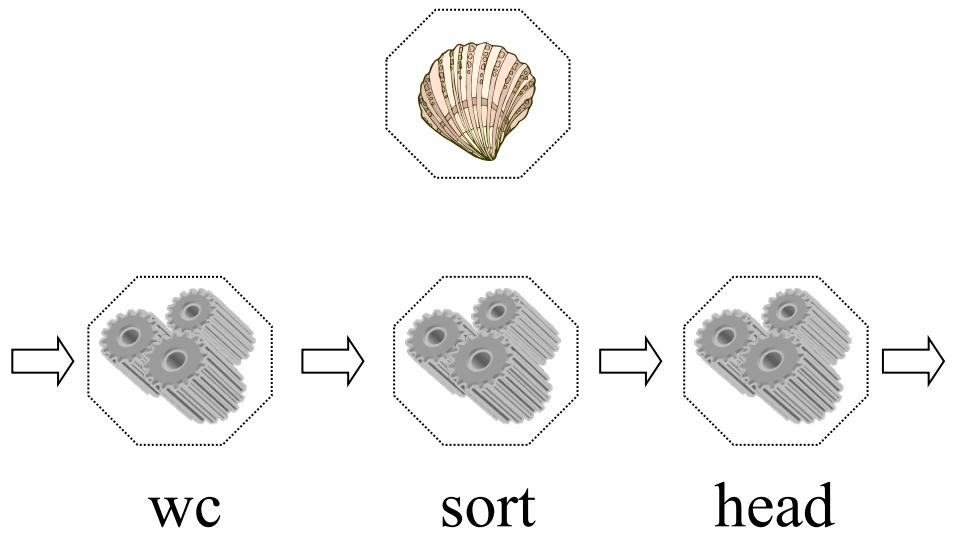
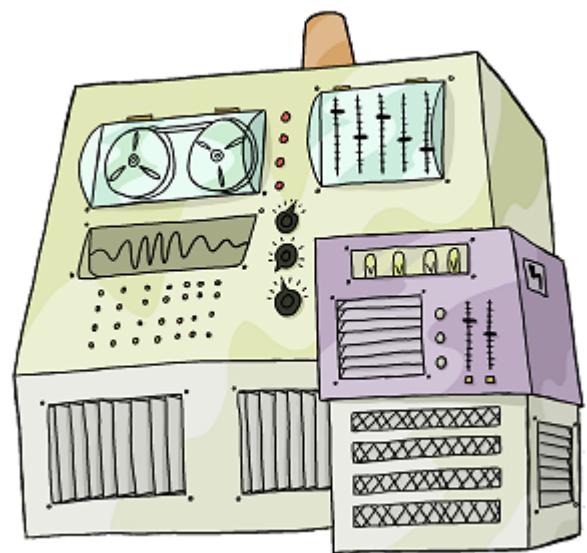


*Control programs while they run*



shell

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



*processes*

*Control programs 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
```

<i>PID</i>	<i>PPID</i>	<i>PGID</i>	<i>TTY</i>	<i>UID</i>	<i>STIME</i>	<i>COMMAND</i>
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
```

<i>PID</i>	<i>PPID</i>	<i>PGID</i>	<i>TTY</i>	<i>UID</i>	<i>STIME</i>	<i>COMMAND</i>
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
```

<i>PID</i>	<i>PPID</i>	<i>PGID</i>	<i>TTY</i>	<i>UID</i>	<i>STIME</i>	<i>COMMAND</i>
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
```

<i>PID</i>	<i>PPID</i>	<i>PGID</i>	<i>TTY</i>	<i>UID</i>	<i>STIME</i>	<i>COMMAND</i>
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

<i>PID</i>	<i>PPID</i>	<i>PGID</i>	<i>TTY</i>	<i>UID</i>	<i>STIME</i>	<i>COMMAND</i>
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
```

<i>PID</i>	<i>PPID</i>	<i>PGID</i>	<i>TTY</i>	<i>UID</i>	<i>STIME</i>	<i>COMMAND</i>
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

<i>PID</i>	<i>PPID</i>	<i>PGID</i>	<i>TTY</i>	<i>UID</i>	<i>STIME</i>	<i>COMMAND</i>
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

<i>PID</i>	<i>PPID</i>	<i>PGID</i>	<i>TTY</i>	<i>UID</i>	<i>STIME</i>	<i>COMMAND</i>
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

<i>PID</i>	<i>PPID</i>	<i>PGID</i>	<i>TTY</i>	<i>UID</i>	<i>STIME</i>	<i>COMMAND</i>
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

<i>PID</i>	<i>PPID</i>	<i>PGID</i>	<i>TTY</i>	<i>UID</i>	<i>STIME</i>	<i>COMMAND</i>
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

<i>PID</i>	<i>PPID</i>	<i>PGID</i>	<i>TTY</i>	<i>UID</i>	<i>STIME</i>	<i>COMMAND</i>
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

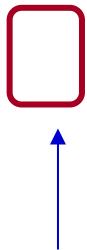
```
$ ./analyze results*.dat
```

*...a few minutes pass...*

```
^C
```

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

```
$
```



Run in the background

Can stop, pause, and resume running processes

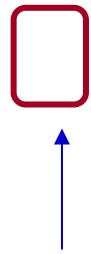
`$ ./analyze results*.dat`

*...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
```



Show background processes

```
[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
```

```
$ 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



# The Unix Shell

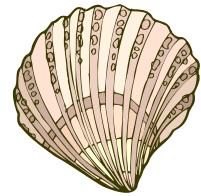
## Variables



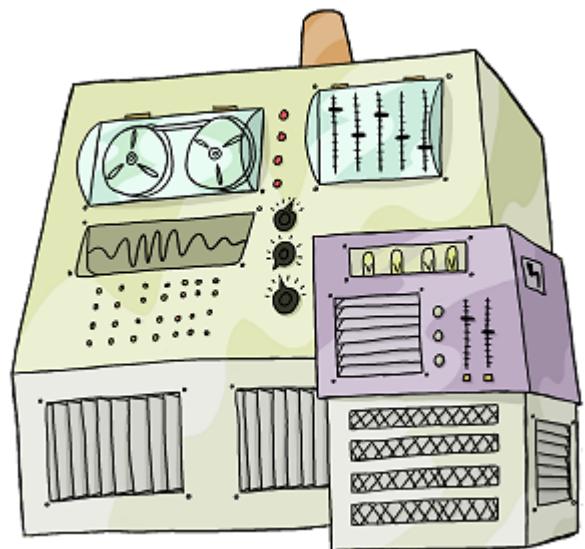
Copyright © Software Carpentry 2010

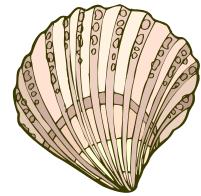
This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.



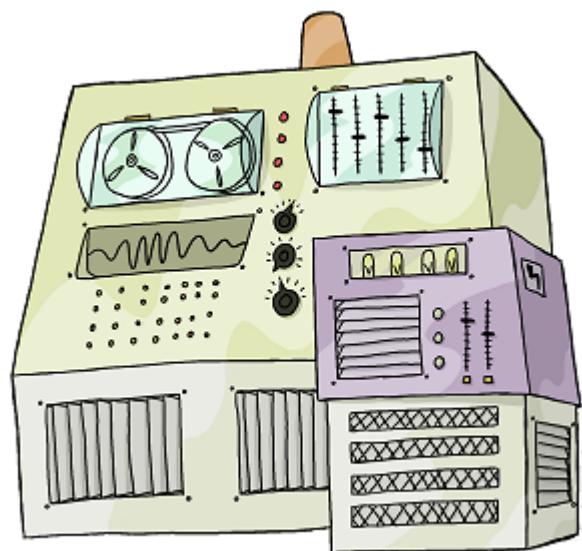
shell

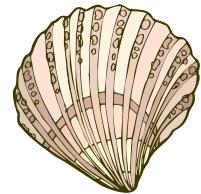




shell

The shell is a program

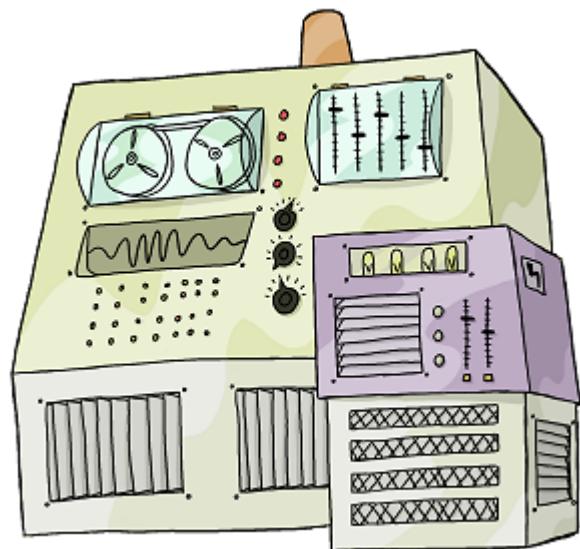


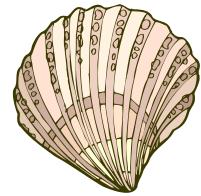


shell

The shell is a program

It has variables



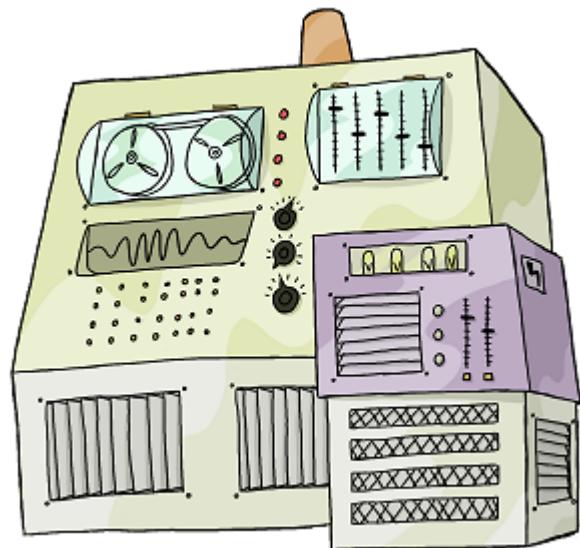


shell

The shell is a program

It has variables

Changing their values  
changes its behavior



**\$ set**

*COMPUTERNAME=TURING*

*HOME=/home/vlad*

*HOMEDRIVE=C:*

*HOSTNAME=TURING*

*HOSTTYPE=i686*

*MANPATH=/usr/local/man:/usr/share/man:/usr/man*

*NUMBER\_OF\_PROCESSORS=4*

*OS=Windows\_NT*

*PATH=/usr/local/bin:/usr/bin:/bin:/cygdrive/c/Windows/system32:*

*/cygdrive/c/Windows:/cygdrive/c/bin:/cygdrive/c/Python27*

*PWD=/home/vlad*

*UID=1000*

*USERNAME=vlad*

\$ **set**

With no arguments, shows all variables and their values

*COMPUTERNAME=TURING*  
*HOME=/home/vlad*  
*HOMEDRIVE=C:*  
*HOSTNAME=TURING*  
*HOSTTYPE=i686*  
*MANPATH=/usr/local/man:/usr/share/man:/usr/man*  
*NUMBER\_OF\_PROCESSORS=4*  
*OS=Windows\_NT*  
*PATH=/usr/local/bin:/usr/bin:/bin:/cygdrive/c/Windows/system32:*  
*/cygdrive/c/Windows:/cygdrive/c/bin:/cygdrive/c/Python27*  
*PWD=/home/vlad*  
*UID=1000*  
*USERNAME=vlad*

\$ set

*COMPUTERNAME=TURING*

*HOME=/home/vlad*

*HOMEDRIVE=C:*

*HOSTNAME=TURING*

*HOSTTYPE=i686*

*MANPATH=/usr/local/man:/usr/share/man:/usr/man*

*NUMBER\_OF\_PROCESSORS=4*

*OS=Windows\_NT*

*PATH=/usr/local/bin:/usr/bin:/bin:/cygdrive/c/Windows/system32:*

*/cygdrive/c/Windows:/cygdrive/c/bin:/cygdrive/c/Python27*

*PWD=/home/vlad*

*UID=1000*

*USERNAME=vlad*

Standard to use upper-case names

```
$ set  
COMPUTERNAME=TURING  
HOME=/home/vlad  
HOMEDRIVE=C:  
HOSTNAME=TURING  
HOSTTYPE=i686  
MANPATH=/usr/local/man:/usr/share/man:/usr/man  
NUMBER_OF_PROCESSORS=4  
OS=Windows_NT  
PATH=/usr/local/bin:/usr/bin:/bin:/cygdrive/c/Windows/system32:  
/cygdrive/c/Windows:/cygdrive/c/bin:/cygdrive/c/Python27  
PWD=/home/vlad  
UID=1000  
USERNAME=vlad
```

All values are strings

\$ set

*COMPUTERNAME=TURING*

*HOME=/home/vlad*

*HOMEDRIVE=C:*

*HOSTNAME=TURING*

*HOSTTYPE=i686*

*MANPATH=/usr/local/man:/usr/share/man:/usr/man*

*NUMBER\_OF\_PROCESSORS=4*

*OS=Windows\_NT*

*PATH=/usr/local/bin:/usr/bin:/bin:/cygdrive/c/Windows/system32:*

*/cygdrive/c/Windows:/cygdrive/c/bin:/cygdrive/c/Python27*

*PWD=/home/vlad*

*UID=1000*

*USERNAME=vlad*

All values are strings

Programs must convert to other types when/as necessary

\$ set

*COMPUTERNAME=TURING*

*HOME=/home/vlad*

*HOMEDRIVE=C:*

*HOSTNAME=TURING*

*HOSTTYPE=i686*

*MANPATH=/usr/local/man:/usr/share/man:/usr/man*

*NUMBER\_OF\_PROCESSORS=4*

*OS=Windows\_NT*

*PATH=/usr/local/bin:/usr/bin:/bin:/cygdrive/c/Windows/system32:*

*/cygdrive/c/Windows:/cygdrive/c/bin:/cygdrive/c/Python27*

*PWD=/home/vlad*

*UID=1000*

*USERNAME=vlad*

int(string) for numbers

\$ set

*COMPUTERNAME=TURING*

*HOME=/home/vlad*

*HOMEDRIVE=C:*

split(':') for lists

*HOSTNAME=TURING*

*HOSTTYPE=i686*

*MANPATH=/usr/local/man:/usr/share/man:/usr/man*

*NUMBER\_OF\_PROCESSORS=4*

*OS=Windows\_NT*

*PATH=/usr/local/bin:/usr/bin:/bin:/cygdrive/c/Windows/system32:*

*/cygdrive/c/Windows:/cygdrive/c/bin:/cygdrive/c/Python27*

*PWD=/home/vlad*

*UID=1000*

*USERNAME=vlad*

# PATH controls where the shell looks for programs

PATH controls where the shell looks for programs

`$ ./analyze`



Run the analyze program  
in the current directory

PATH controls where the shell looks for programs

`$ ./analyze`

`$ /bin/analyze`



Run the analyze program  
in the /bin directory

PATH controls where the shell looks for programs

`$ ./analyze`

`$ /bin/analyze`

`$ analyze`

PATH controls where the shell looks for programs

`$ ./analyze`

`$ /bin/analyze`

`$ analyze`



directories = split(PATH, ':')

for each directory:

if directory/analyze exists, run it

# PATH controls where the shell looks for programs

`$ ./analyze`

`$ /bin/analyze`

`$ analyze`

*/usr/local/bin*

*/usr/bin*

*/bin*

*/cygdrive/c/Windows/system32*

*/cygdrive/c/Windows*

*/cygdrive/c/bin*

*/cygdrive/c/Python27*

# PATH controls where the shell looks for programs

\$ ./analyze

\$ /bin/analyze

\$ analyze

*/usr/local/bin*

*/usr/bin*

*/bin*                           */bin/analyze*

*/cygdrive/c/Windows/system32*

*/cygdrive/c/Windows*

*/cygdrive/c/bin*                           */cygdrive/c/bin/analyze*

*/cygdrive/c/Python27*

*/users/vlad/analyze*

# PATH controls where the shell looks for programs

`$ ./analyze`

`$ /bin/analyze`

`$ analyze`

*/usr/local/bin*

*/usr/bin*

*/bin*                           */bin/analyze*

*/cygdrive/c/Windows/system32*

*/cygdrive/c/Windows*

*/cygdrive/c/bin*                   */cygdrive/c/bin/analyze*

*/cygdrive/c/Python27*

*/users/vlad/analyze*



PATH controls where the shell looks for programs

`$ ./analyze`

`$ /bin/analyze`

`$ analyze`

*/usr/local/bin*

*/usr/bin*

*/bin*                            */bin/analyze*

*/cygdrive/c/Windows/system32*

*/cygdrive/c/Windows*

*/cygdrive/c/bin*                            */cygdrive/c/bin/analyze*

*/cygdrive/c/Python27*

*/users/vlad/analyze*

# echo prints its arguments

echo prints its arguments

Use it to show variables' values

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania!
```

*hello transylvania!*

```
$
```

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania!
```

*hello transylvania!*

```
$ echo HOME
```

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania!
```

*hello transylvania!*

```
$ echo HOME
```

*HOME*

```
$
```

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania!
```

*hello transylvania!*

```
$ echo HOME
```

*HOME*

```
$ echo $HOME
```

*/home/vlad*

```
$
```

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania!
```

*hello transylvania!*

```
$ echo HOME
```

*HOME*

```
$ echo $HOME
```

*/home/vlad*

```
$
```

Ask shell to replace variable name  
with value before program runs

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania!
```

*hello transylvania!*

```
$ echo HOME
```

*HOME*

```
$ echo $HOME
```

*/home/vlad*

```
$
```

Ask shell to replace variable name  
with value before program runs  
Just like \* and ? are expanded  
before the program runs

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania!
```

*hello transylvania!*

```
$ echo HOME
```

*HOME*

```
$ echo $HOME
```



```
echo /home/vlad
```

*/home/vlad*

```
$
```

# Create variable by assigning to it

Create variable by assigning to it

Change values by reassigning to existing variables

Create variable by assigning to it

Change values by reassigning to existing variables

```
$ SECRET_IDENTITY=Dracula
```

```
$ echo $SECRET_IDENTITY
```

*Dracula*

```
$ SECRET_IDENTITY=Camilla
```

```
$ echo $SECRET_IDENTITY
```

*Camilla*

```
$
```

Assignment only changes variable's value  
in *this* shell

Assignment only changes variable's value  
in *this* shell

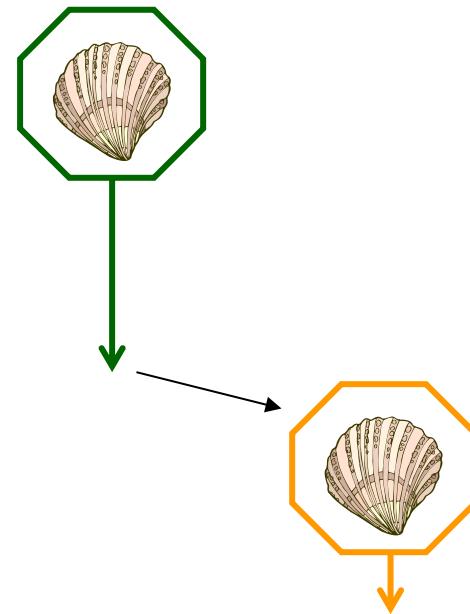
```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$
```

Assignment only changes variable's value  
in *this* shell

```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$ bash
$
```

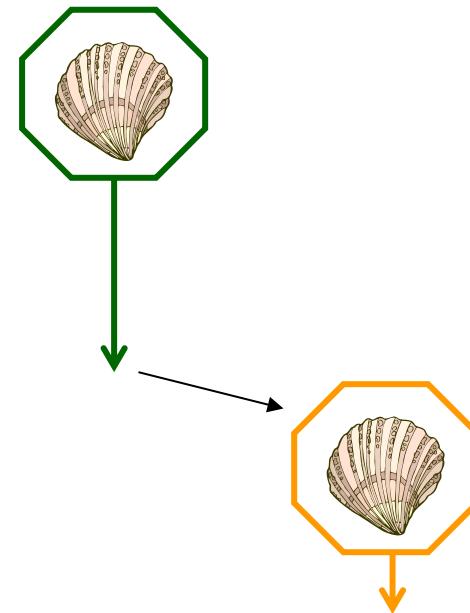
Assignment only changes variable's value  
in *this* shell

```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$ bash
$
```



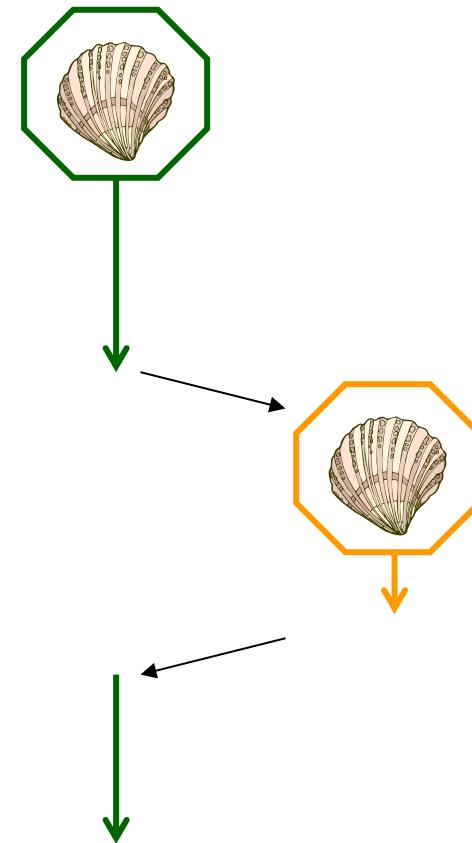
Assignment only changes variable's value  
in *this* shell

```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$ bash
$ echo $SECRET_IDENTITY
$
```



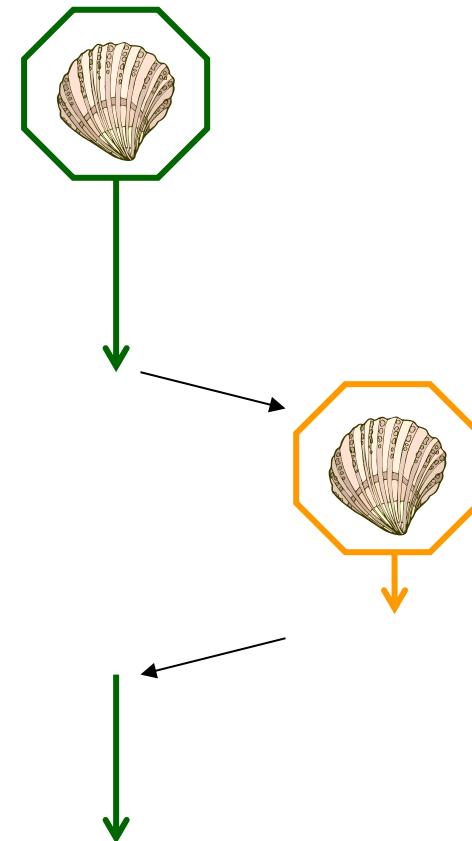
Assignment only changes variable's value  
in *this* shell

```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$ bash
$ echo $SECRET_IDENTITY
$ exit
$
```



Assignment only changes variable's value  
in *this* shell

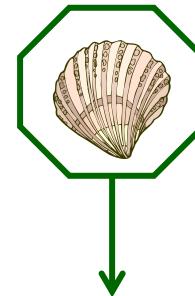
```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$ bash
$ echo $SECRET_IDENTITY
$ exit
$ echo $SECRET_IDENTITY
Dracula
$
```



Use `export` to signal that the variable should be visible to subprocesses

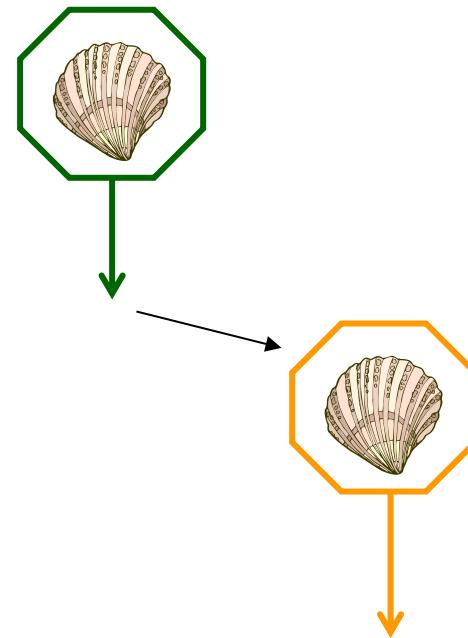
Use export to signal that the variable should be  
visible to subprocesses

```
$ SECRET_IDENTITY=Dracula  
$ export SECRET_IDENTITY  
$
```



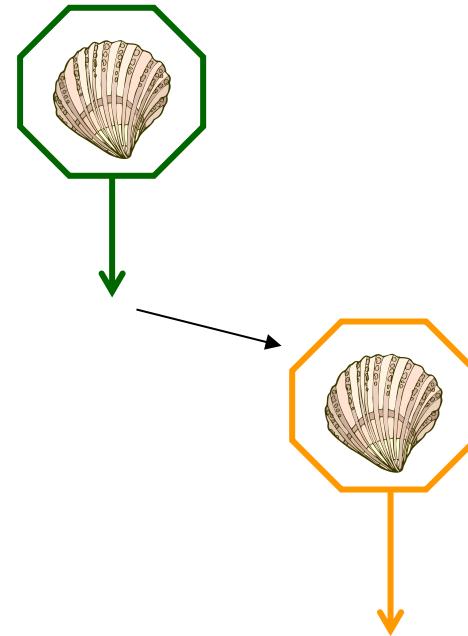
Use export to signal that the variable should be visible to subprocesses

```
$ SECRET_IDENTITY=Dracula
$ export SECRET_IDENTITY
$ bash
$
```



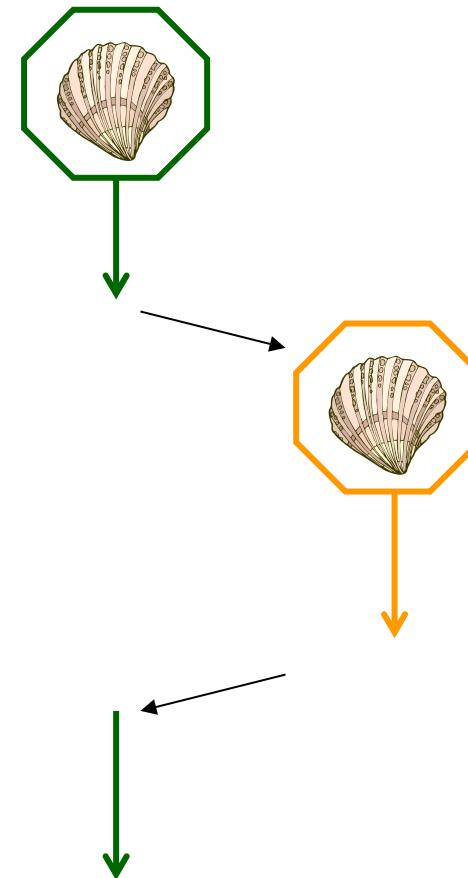
Use export to signal that the variable should be visible to subprocesses

```
$ SECRET_IDENTITY=Dracula
$ export SECRET_IDENTITY
$ bash
$ echo $SECRET_IDENTITY
Dracula
$
```



Use export to signal that the variable should be visible to subprocesses

```
$ SECRET_IDENTITY=Dracula
$ export SECRET_IDENTITY
$ bash
$ echo $SECRET_IDENTITY
Dracula
$ exit
$
```



Commands in `~/.bashrc` are executed  
when shell starts

Commands in \$HOME/.bashrc are executed  
when shell starts

```
export SECRET_IDENTITY=Dracula  
export BACKUP_DIR=$HOME/backup
```

/home/vlad/.bashrc

Commands in \$HOME/.bashrc are executed  
when shell starts

```
export SECRET_IDENTITY=Dracula  
export BACKUP_DIR=$HOME/backup
```

Also common to use alias to create shortcuts

Commands in `$HOME/.bashrc` are executed  
when shell starts

```
export SECRET_IDENTITY=Dracula
export BACKUP_DIR=$HOME/backup
```

Also common to use alias to create shortcuts

```
alias backup=/bin/zarble -v --nostir -R 20000 $HOME $BACKUP_DIR
```

Commands in `$HOME/.bashrc` are executed  
when shell starts

```
export SECRET_IDENTITY=Dracula  
export BACKUP_DIR=$HOME/backup
```

Also common to use alias to create shortcuts

```
alias backup=/bin/zarble -v --nostir -R 20000 $HOME $BACKUP_DIR
```

Not something you want to type over and over



# The Unix Shell

## The Secure Shell



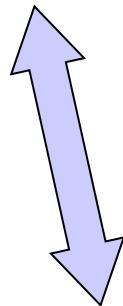
Copyright © Software Carpentry 2011

This work is licensed under the Creative Commons Attribution License

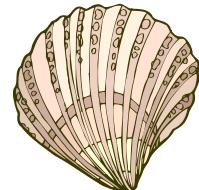
See <http://software-carpentry.org/license.html> for more information.



\$ pwd

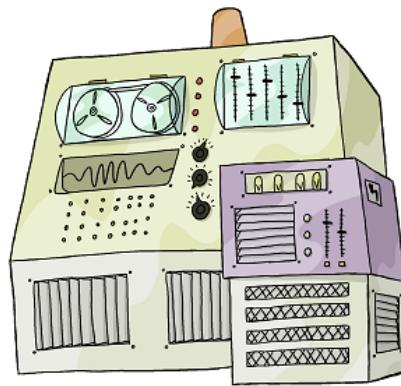
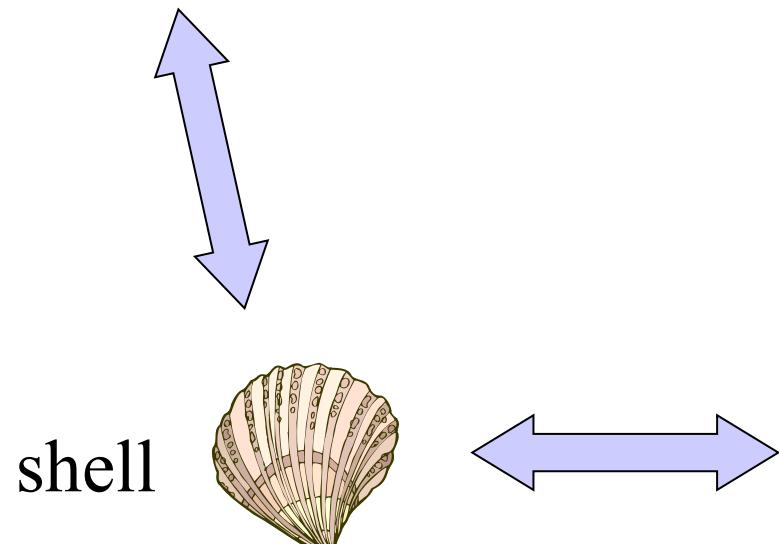


shell



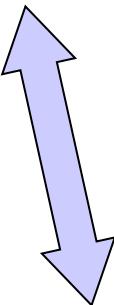


```
$ pwd  
/users/vlad  
$
```

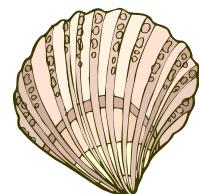




**login as: vlad**  
**Password: \*\*\*\*\***



**shell**

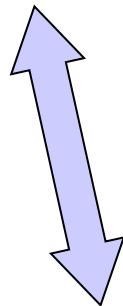




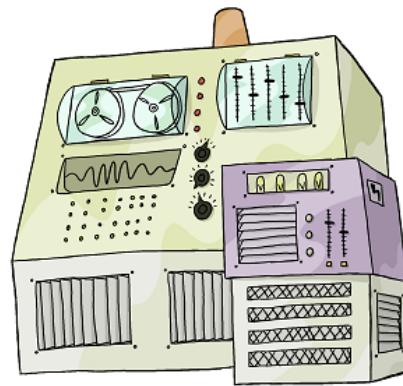
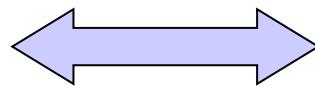
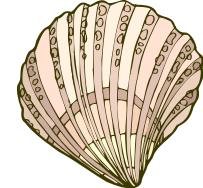
**login as: vlad**

**Password: \*\*\*\*\***

**\$**

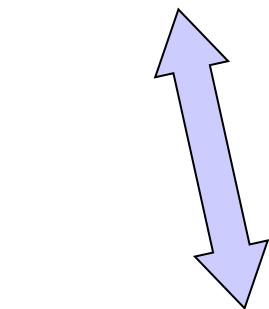


**shell**

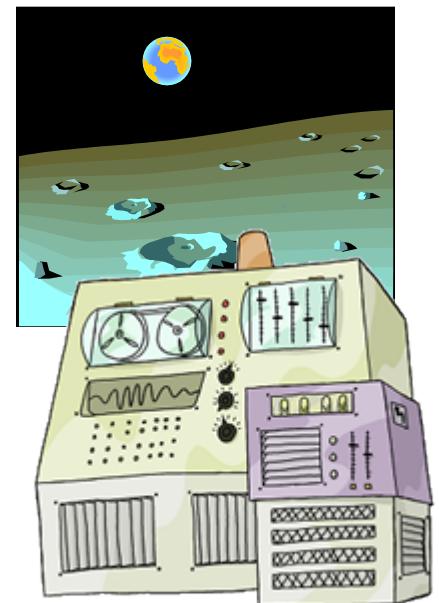
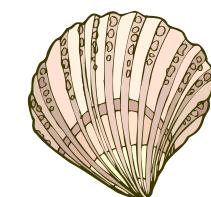
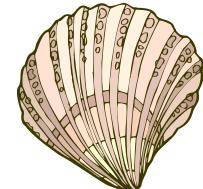




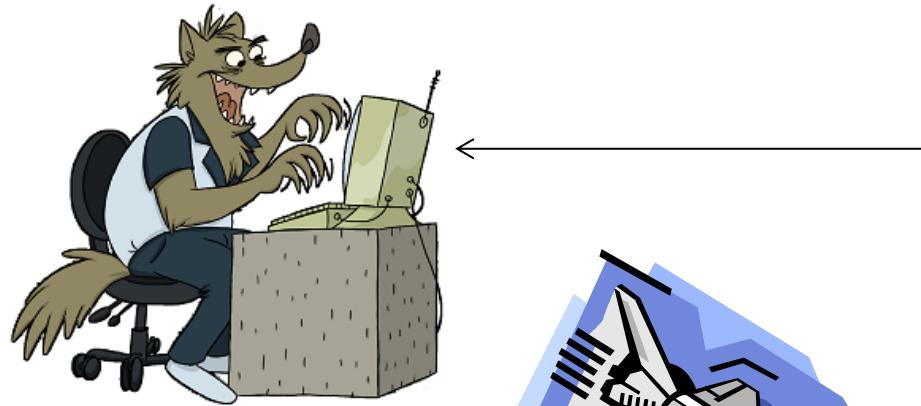
```
login as: vlad  
Password: *****  
moon>
```



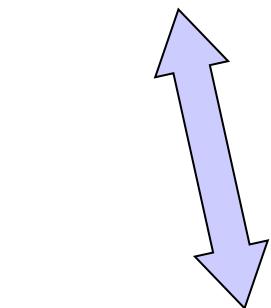
shell



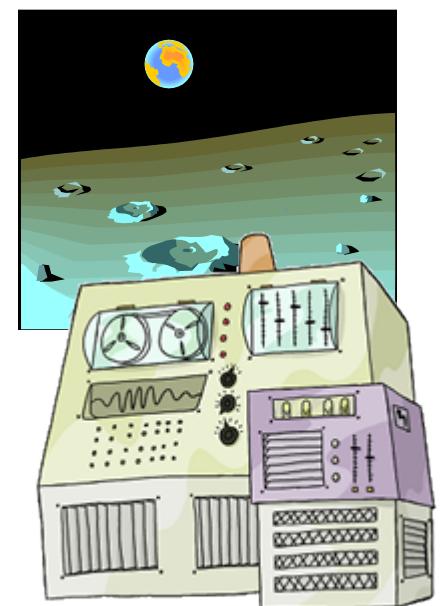
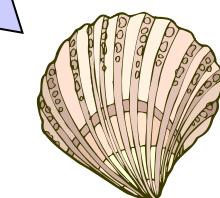
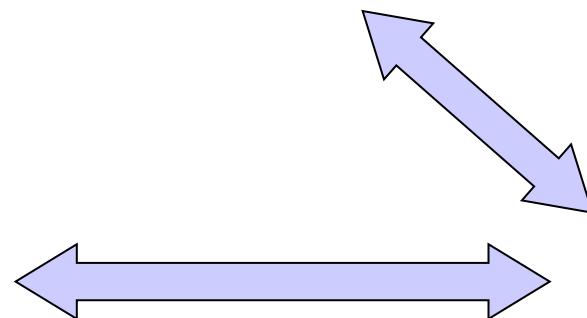
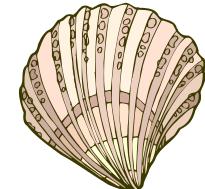
remote shell



login as: vlad  
Password: \*\*\*\*\*  
moon>



shell



remote shell

```
$ pwd
```

```
/users/vlad
```

```
$ ssh vlad@moon
```

```
Password:
```

\$ `pwd`

*/users/vlad*

\$ `ssh vlad@moon`

**Password:** \*\*\*

*Access denied*

**Password:**

```
$ pwd
```

*/users/vlad*

```
$ ssh vlad@moon
```

**Password:** \*\*\*

*Access denied*

**Password:** \*\*\*\*\*

```
moon> pwd
```

*/home/vlad*

```
moon> ls -F
```

*bin/ cheese.txt dark\_side/ rocks.cfg*

\$ **pwd**

*/users/vlad*

\$ **ssh vlad@moon**

**Password:** \*\*\*

*Access denied*

**Password:** \*\*\*\*\*

**moon> pwd**

*/home/vlad*

**moon> ls -F**

*bin/ cheese.txt dark\_side/ rocks.cfg*

**moon> exit**

\$ **pwd**

*/users/vlad*

\$ ssh vlad@moon

**Password:** \*\*\*\*\*

**moon>** pwd

*/home/vlad*

**moon>** ls -F

*bin/ cheese.txt dark\_side/ rocks.cfg*

**moon>** exit

\$ pwd

*/users/vlad*

\$ ls -F

*bin/ data/ mail/ music/*  
*notes.txt papers/ pizza.cfg solar/*  
*solar.pdf swc/*

```
$ scp vlad@moon:/home/vlad/cheese.txt
```

```
vlad@earth:/users/vlad
```

source file...

```
$ scp vlad@moon:/home/vlad/cheese.txt
```

vlad@earth:/users/vlad

source file...

...to destination directory

```
$ scp vlad@moon:/home/vlad/cheese.txt
```

```
vlad@earth:/users/vlad
```

source file...

...to destination directory

source and destination are written as

user@computer:path

```
$ scp vlad@moon:/home/vlad/cheese.txt
```

```
vlad@earth:/users/vlad
```

**Password:** \*\*\*\*\*

```
cheese.txt      100% 9 1.0 KB/s 00:00
```

```
$ scp vlad@moon:/home/vlad/cheese.txt
```

```
    vlad@earth:/users/vlad
```

```
$ scp -r vlad@moon:/home/vlad/dark_side
```

```
    vlad@earth:/users/vlad
```

-r indicates a directory and its contents

```
$ scp -r vlad@moon:/home/vlad/dark_side
```

```
    vlad@earth:/users/vlad
```

```
$ scp -r vlad@moon:/home/vlad/dark_side
```

```
    /users/vlad
```

```
$ pwd
```

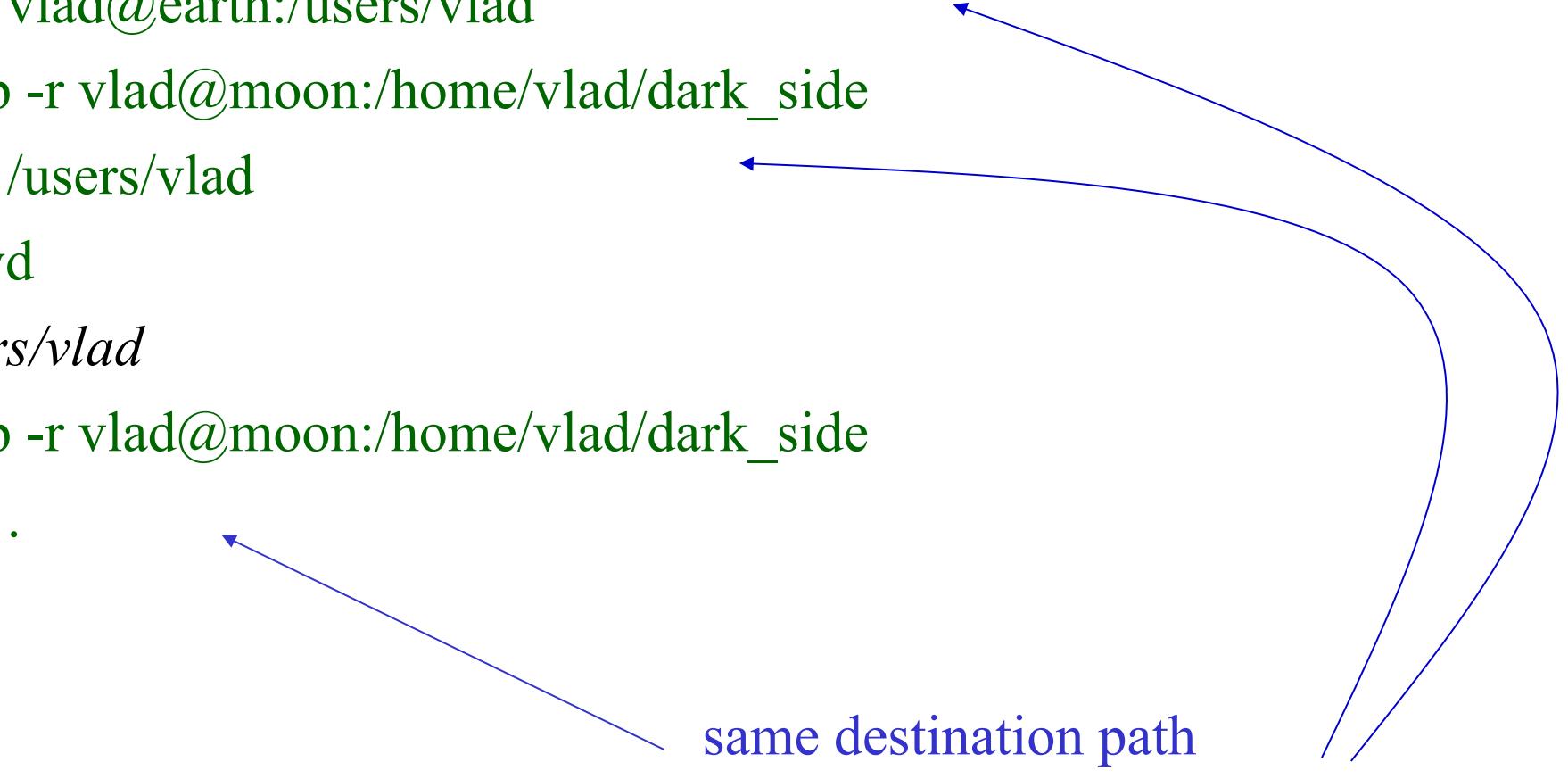
```
/users/vlad
```

```
$ scp -r vlad@moon:/home/vlad/dark_side
```

```
.
```

```
.
```

same destination path



The diagram consists of three blue arrows pointing from the destination paths in the first two scp commands to the final destination path in the third command. The first arrow points from 'vlad@earth:/users/vlad' to 'users/vlad'. The second arrow points from '/users/vlad' to 'users/vlad'. The third arrow points from the final '.' in the third command to 'users/vlad'.

```
$ ssh vlad@moon
```

**Password:** \*\*\*\*\*

```
moon> df -h
```

<i>Filesystem</i>	<i>Size</i>	<i>Used</i>	<i>Avail</i>	<i>Use%</i>	<i>Mounted On</i>
-------------------	-------------	-------------	--------------	-------------	-------------------

<i>/dev/sda1</i>	<i>7.9G</i>	<i>2.1G</i>	<i>5.5G</i>	<i>28%</i>	<i>/</i>
------------------	-------------	-------------	-------------	------------	----------

<i>/dev/sda2</i>	<i>791G</i>	<i>150G</i>	<i>642G</i>	<i>19%</i>	<i>/home</i>
------------------	-------------	-------------	-------------	------------	--------------

```
moon> df -h > usage.txt
```

```
moon> exit
```

```
$ scp vlad@moon:/home/vlad/usage.txt .
```

**Password:** \*\*\*\*\*

```
usage.txt      100% 134 1.0 KB/s 00:00
```

```
$ ssh vlad@moon 'df -h'
```

**Password:** \*\*\*\*\*

<i>Filesystem</i>	<i>Size</i>	<i>Used</i>	<i>Avail</i>	<i>Use%</i>	<i>Mounted On</i>
<i>/dev/sda1</i>	<i>7.9G</i>	<i>2.1G</i>	<i>5.5G</i>	<i>28%</i>	<i>/</i>
<i>/dev/sda2</i>	<i>791G</i>	<i>150G</i>	<i>642G</i>	<i>19%</i>	<i>/home</i>

```
$ ssh vlad@moon 'df -h'
```

**Password:** \*\*\*\*\*

<i>Filesystem</i>	<i>Size</i>	<i>Used</i>	<i>Avail</i>	<i>Use%</i>	<i>Mounted On</i>
/dev/sda1	7.9G	2.1G	5.5G	28%	/
/dev/sda2	791G	150G	642G	19%	/home

```
$ ssh vlad@moon 'df -h' >> usage.log
```

**Password:** \*\*\*\*\*

```
$ ls -F
```

```
bin/      data/    mail/    music/  
notes.txt  papers/   pizza.cfg  solar/  
solar.pdf  swc/     usage.log  usage.txt
```

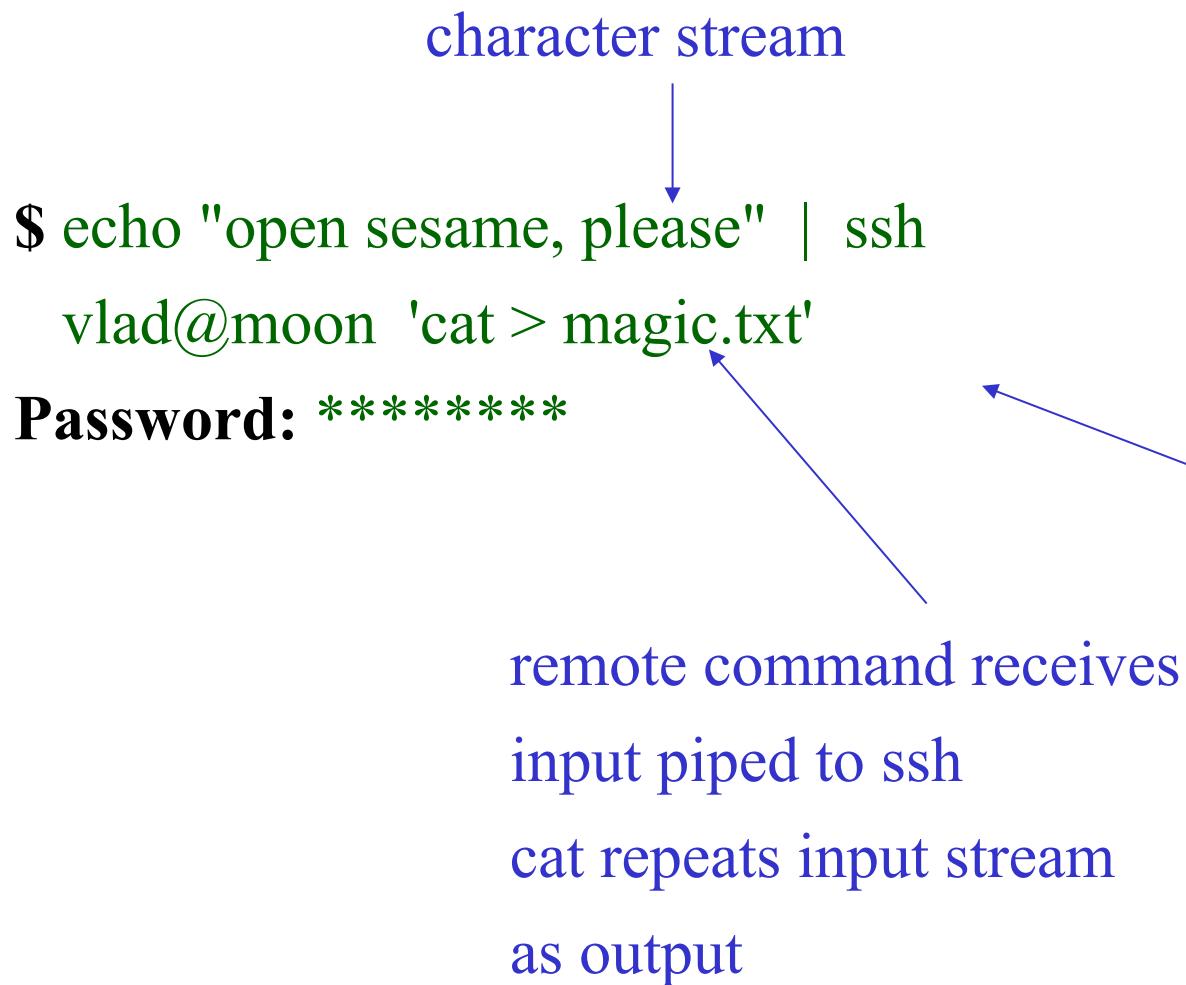
same result

character stream



```
$ echo "open sesame, please" | ssh  
vlad@moon 'cat > magic.txt'
```

**Password:** \*\*\*\*\*



```
$ ssh vlad@moon 'ls -F /home/vlad'
```

**Password:** \*\*\*\*\*

*bin/ cheese.txt dark\_side/ rocks.cfg*

```
$ echo "open sesame, please" | ssh
```

vlad@moon 'cat > magic.txt'

**Password:** \*\*\*\*\*

```
$ ssh vlad@moon 'ls -F /home/vlad'
```

**Password:** \*\*\*\*\*

*bin/ cheese.txt dark\_side/ magic.txt*

*rocks.cfg*

before

after

```
$ ssh vlad@moon 'ls -F /home/vlad'
```

**Password:** \*\*\*\*\*

*bin/ cheese.txt dark\_side/ rocks.cfg*

```
$ echo "open sesame, please" | ssh
```

    vlad@moon 'cat > magic.txt'

**Password:** \*\*\*\*\*

```
$ ssh vlad@moon 'ls -F /home/vlad'
```

**Password:** \*\*\*\*\*

*bin/ cheese.txt dark\_side/ magic.txt*

*rocks.cfg*

```
$ scp vlad@moon:/home/vlad/magic.txt .
```

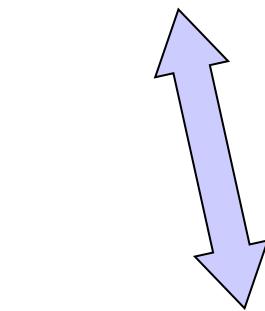
**Password:** \*\*\*\*\*

before

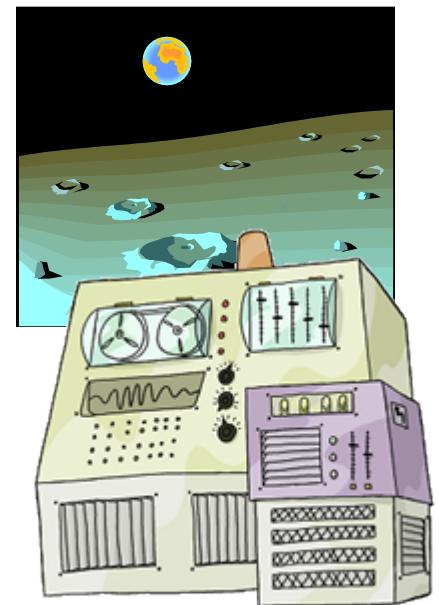
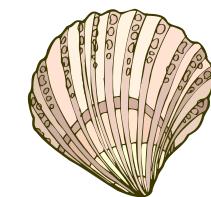
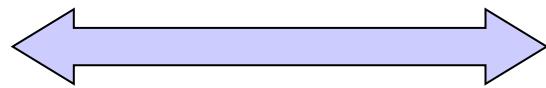
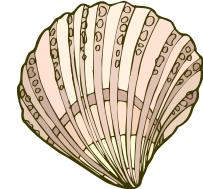
after



**login as: vlad**  
**Password: \*\*\*\*\***



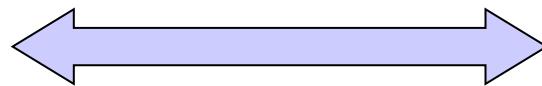
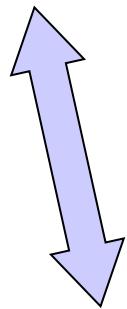
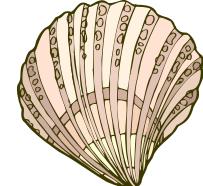
**shell**



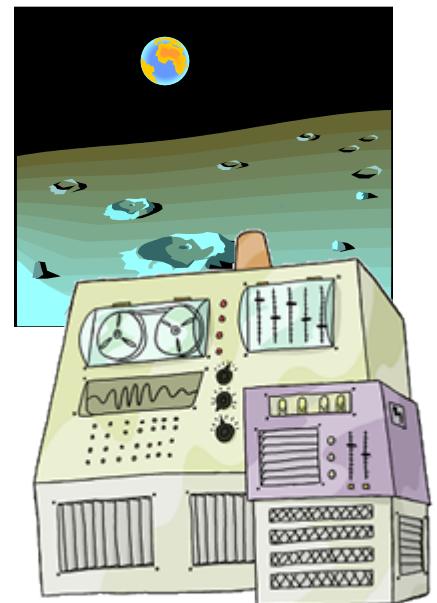
**remote shell**



shell



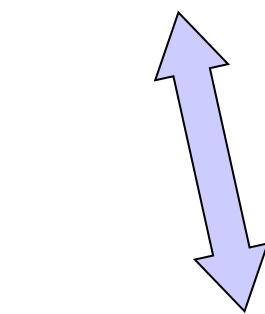
**login as: vlad**  
**Password: thriller**



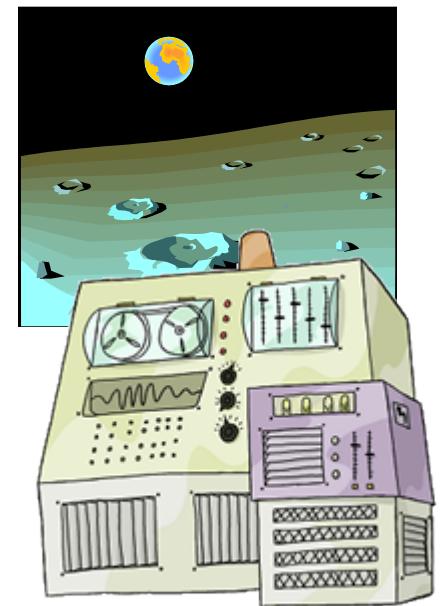
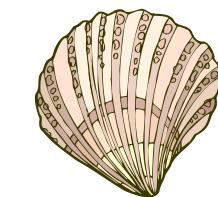
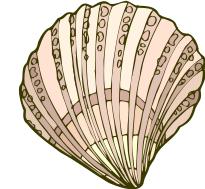
remote shell



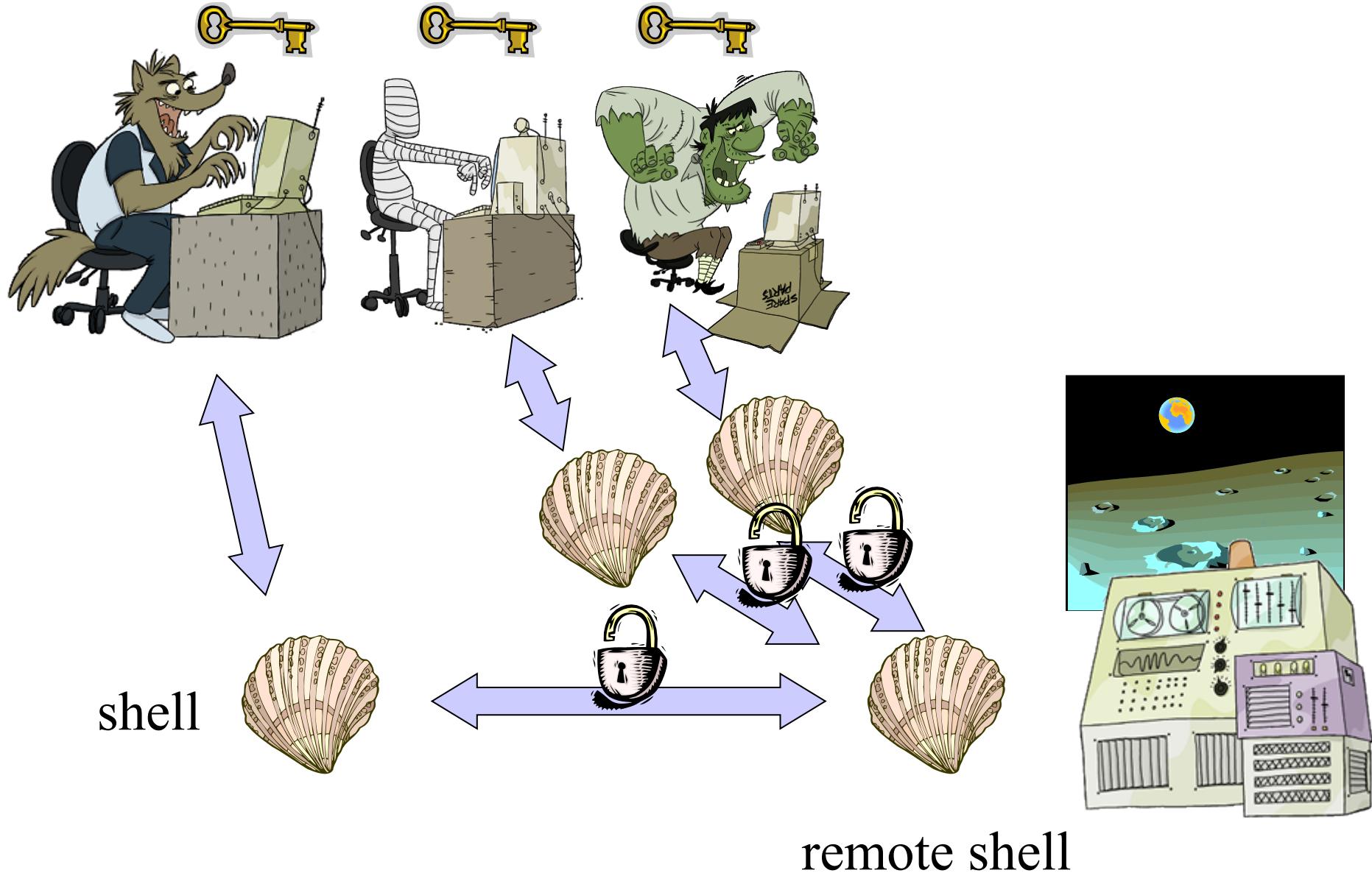
**login as: vlad**  
**Password: thriller**

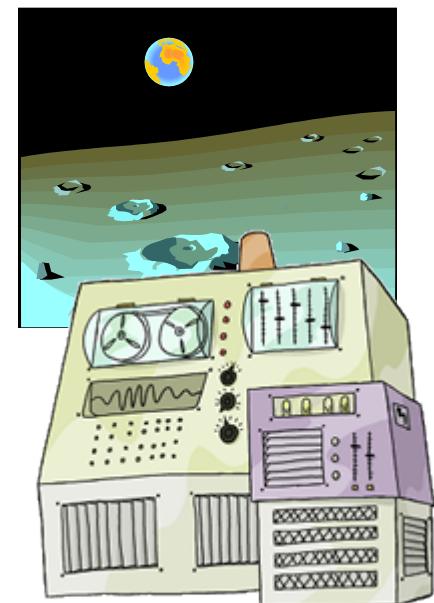
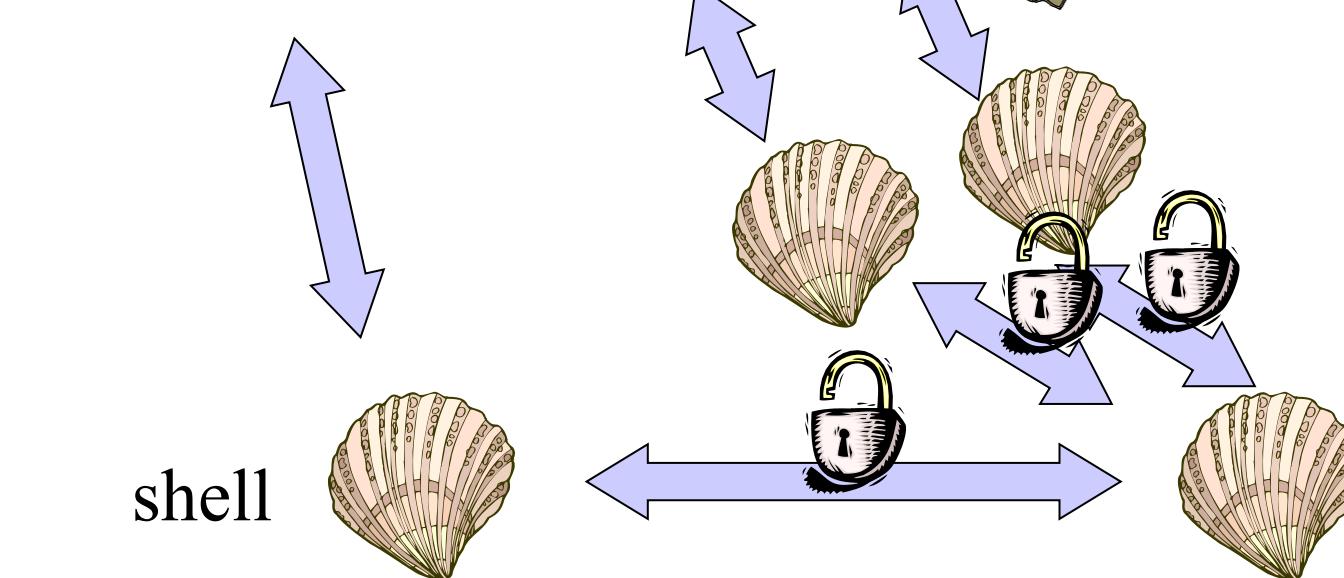
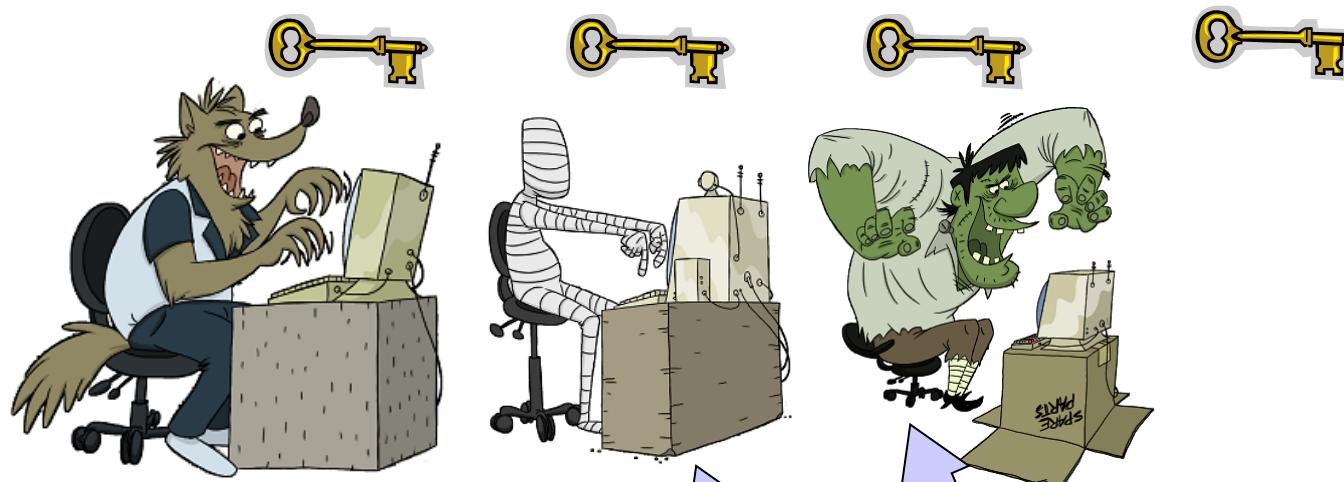


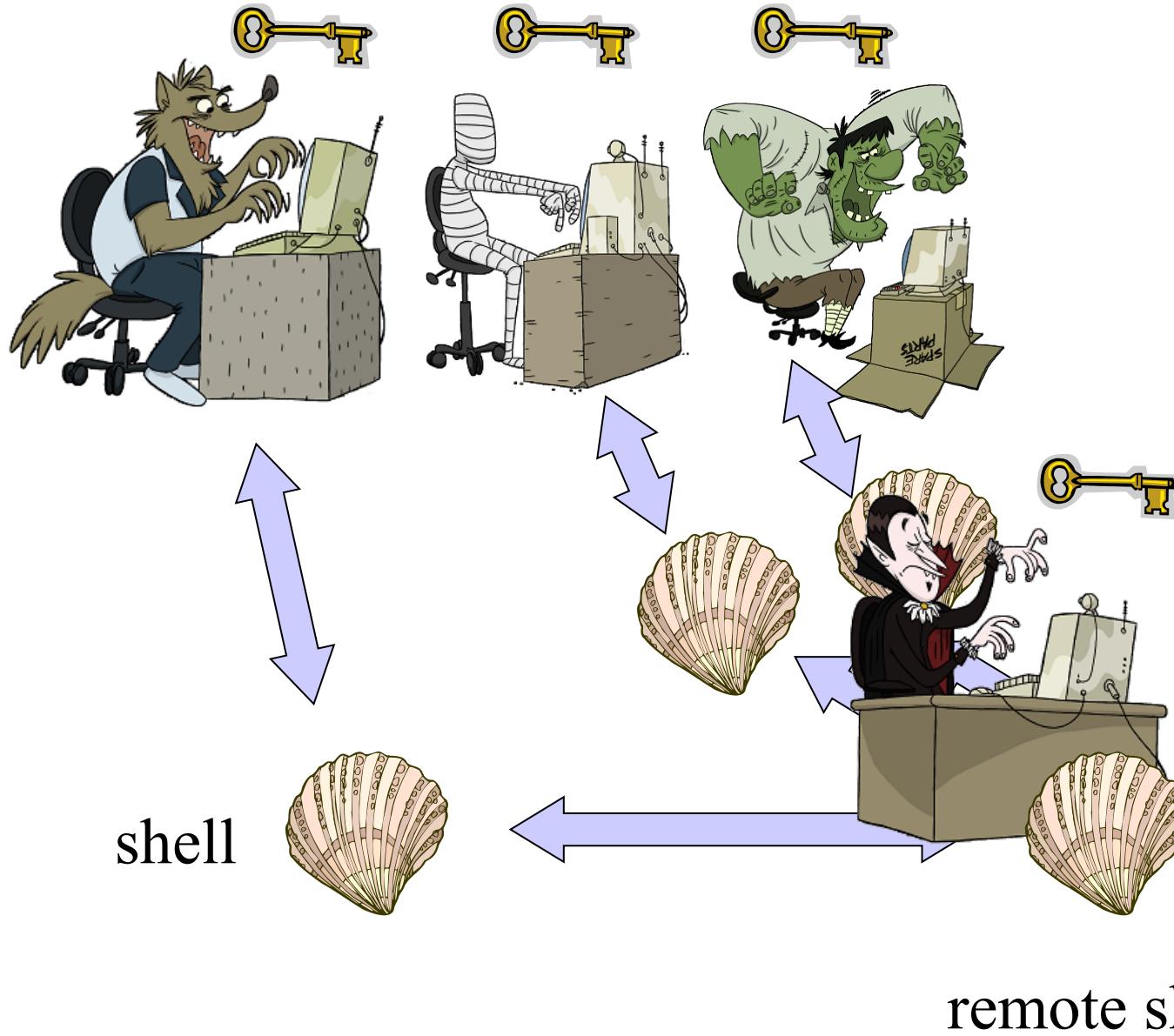
**shell**

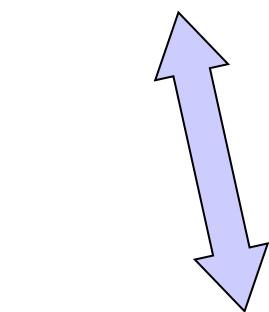


**remote shell**

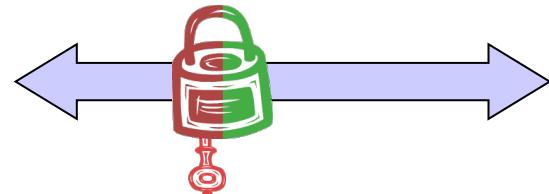
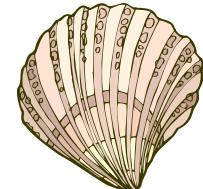




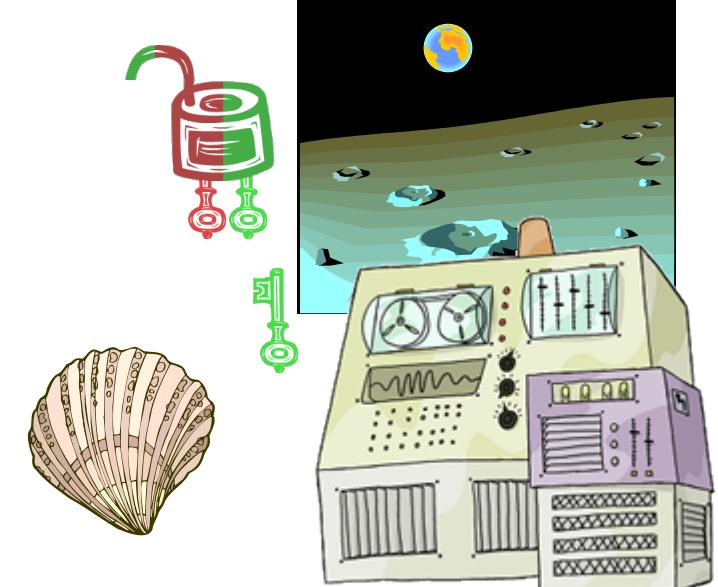


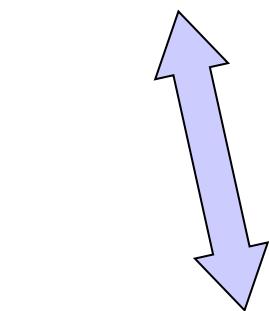


shell

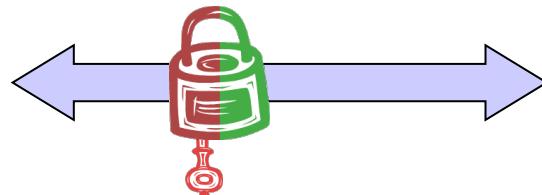
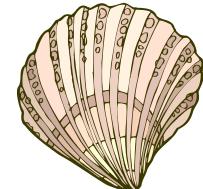


remote shell

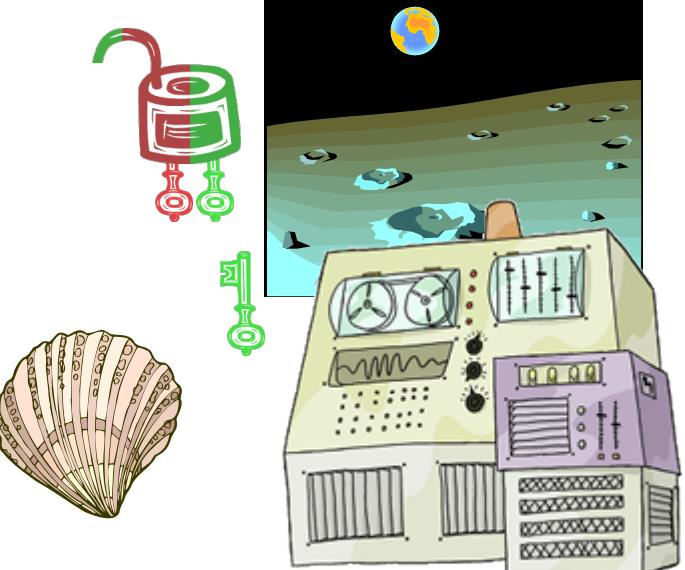


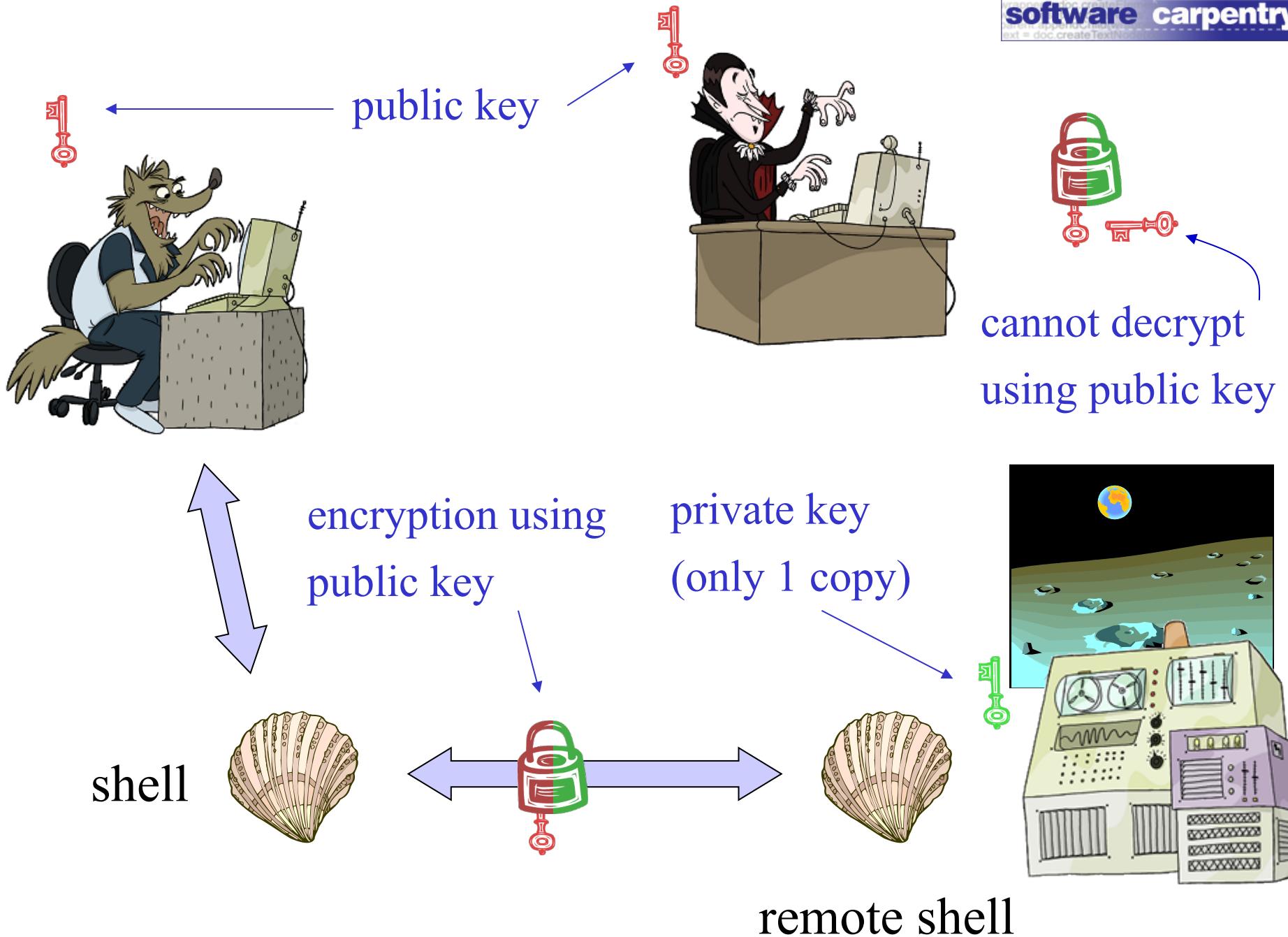


shell



remote shell



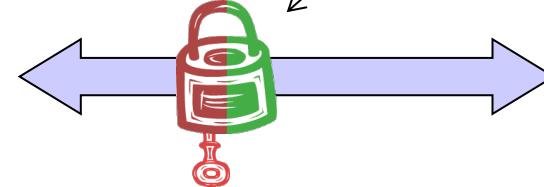
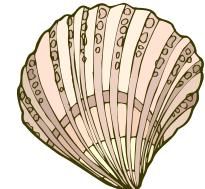


login as: vlad

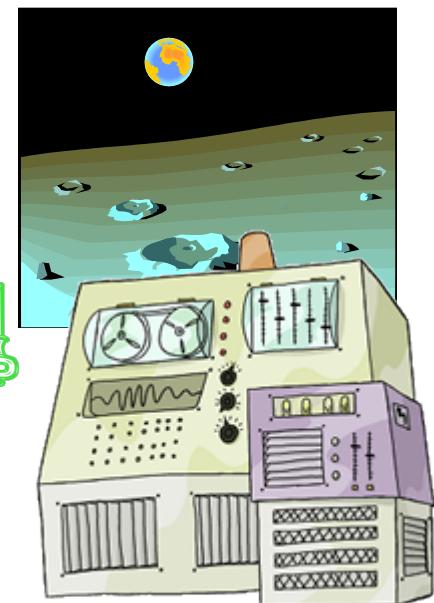
Password: \*\*\*\*\*



shell



huxyo ew: xdvw  
uqfcmjbn: lhiujdbj



remote shell

\$ ssh vlad@moon

The authenticity of host 'moon (10.1.2.3)'  
can't be established.

RSA key fingerprint is

f1:68:f5:90:47:dc:a8:e9:62:df:c9:21:f0:8b:c5:39.

**Are you sure you want to continue connecting  
(yes/no)?** yes

Warning: Permanently added 'moon,10.1.2.3' (RSA)  
to the list of known hosts.

**Password:** \*\*\*\*\*

moon>

```
while true:
```

```
...
```

```
if time.mins == 30:
```

```
    ssh vlad@moon 'df -h' >> usage.log
```

```
...
```

```
while true:
```

```
...
```

```
if time.mins == 30:
```

```
    ssh vlad@moon 'df -h' >> usage.log
```

```
...
```

```
$ ssh vlad@moon 'df -h' >> usage.log
```

**Password:**

*Connection closed by 10.1.2.3*

← waited too long

```
$
```



user key pair 1

host key pair

only 1 copy

default if user has no user key pair

```
$ ssh-keygen -t rsa
```

*Generating public/private rsa key pair.*

**Enter file in which to save the key**

(/users/vlad/.ssh/id\_rsa):

← press enter

**Enter passphrase (empty for no  
passphrase): \*\*\*\*\***

**Enter same passphrase again: \*\*\*\*\***

Your identification has been saved in  
/users/vlad/.ssh/id\_rsa.

Your public key has been saved in  
/users/vlad/.ssh/id\_rsa.pub.

The key fingerprint is:

d3:1a:27:38:aa:54:e8:a5:03:db:79:2f:b2:c3:c9:3d

```
$ ssh vlad@moon
```

Enter passphrase for key

```
'/users/vlad/.ssh/id_rsa': *****
```

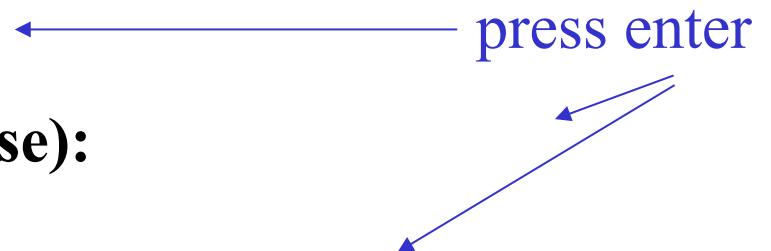
```
moon>
```

```
$ ssh-keygen -t rsa
```

*Generating public/private rsa key pair.*

**Enter file in which to save the key**

(/users/vlad/.ssh/id\_rsa):



**Enter passphrase (empty for no passphrase):**

**Enter same passphrase again:**

Your identification has been saved in

/users/vlad/.ssh/id\_rsa.

Your public key has been saved in

/users/vlad/.ssh/id\_rsa.pub.

The key fingerprint is:

d3:1a:27:38:aa:54:e8:a5:03:db:79:2f:b2:c3:c9:3d

```
$ scp ~/.ssh/id_rsa.pub vlad@moon
```

**Password:** \*\*\*\*\*

```
$ ssh vlad@moon
```

**Password:** \*\*\*\*\*

```
moon> cat id_rsa.pub >> ~/.ssh/authorized_keys
```

```
moon> exit
```

```
$ cat ~/.ssh/id_rsa.pub | ssh vlad@moon
```

```
'cat >> ~/.ssh/authorized_keys'
```

**Password:** \*\*\*\*\*

```
$ ssh-copy-id vlad@moon
```

**Password:** \*\*\*\*\*

```
$ ssh vlad@moon
```

```
moon>
```



```
while true:
```

```
...
```

```
if time.mins == 30:
```

```
    ssh vlad@moon 'df -h' >> usage.log
```

```
...
```

# software carpentry

created by

# Elango Cheran

February 2011



Copyright © Software Carpentry 2011

This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.



# The Unix Shell

## Advanced Shell Tricks



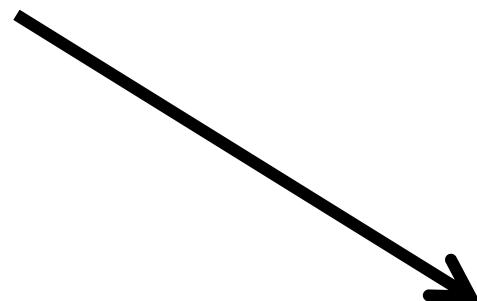
Copyright © The University of Southampton 2011

This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.



“How should I do  
this?”



**Some  
technical  
problem...**



“How should I do  
this?”

With smartphones, you’ll often hear  
people say something like

*“There’s an  
app for that...  
check this out!”*





“How should I do this?”

With smartphones, you’ll often hear people say something like

*“There’s an app for that... check this out!”*



Whereas Unix shell programmers will say

*“There’s a shell trick for that... check this out!”*

In previous episodes, we've seen how to:

- Combine existing programs using pipes & filters

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

In previous episodes, we've seen how to:

- Combine existing programs using pipes & filters
- Redirect output from programs to files

```
$ wc -l *.pdb > lengths
```

In previous episodes, we've seen how to:

- Combine existing programs using pipes & filters
- Redirect output from programs to files
- Use variables to control program operation

```
$ SECRET_IDENTITY=Dracula
```

```
$ echo $SECRET_IDENTITY
```

*Dracula*

In previous episodes, we've seen how to:

- Combine existing programs using pipes & filters
- Redirect output from programs to files
- Use variables to control program operation

Very powerful when used together

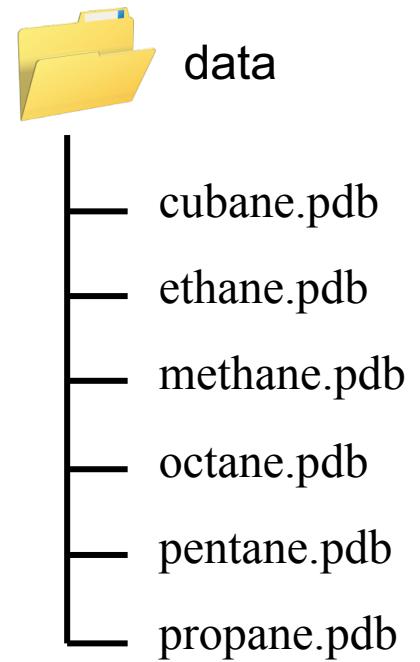
In previous episodes, we've seen how to:

- Combine existing programs using pipes & filters
- Redirect output from programs to files
- Use variables to control program operation

Very powerful when used together

But there are other useful things we can do with these – let's take a look...

First, let's revisit redirection...



First, let's revisit redirection...

```
$ ls *.pdb > files
```

← list all pdb files  
redirect to a file



data

```
└── cubane.pdb
    ├── ethane.pdb
    ├── methane.pdb
    ├── octane.pdb
    ├── pentane.pdb
    └── propane.pdb
```

First, let's revisit redirection...

```
$ ls *.pdb > files
```

← list all pdb files  
redirect to a file

*The 'redirection'  
operator*



data

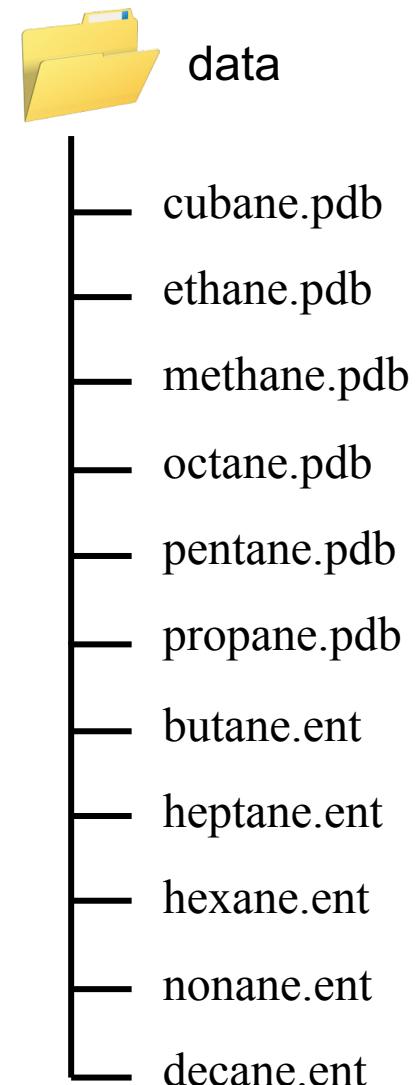
```
└── cubane.pdb
    ├── ethane.pdb
    ├── methane.pdb
    ├── octane.pdb
    ├── pentane.pdb
    └── propane.pdb
```

First, let's revisit redirection...

```
$ ls *.pdb > files
```

← list all pdb files  
redirect to a file

But what about adding this together with other  
results generated later?



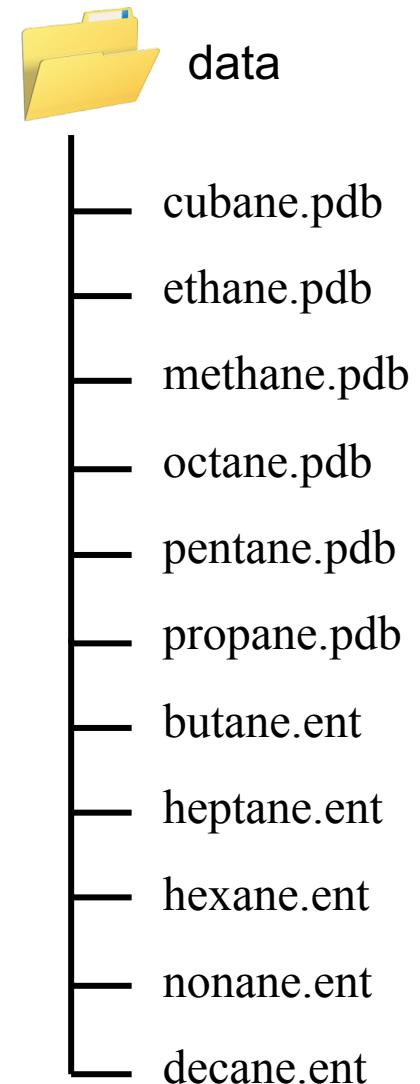
First, let's revisit redirection...

```
$ ls *.pdb > files
```

← list all pdb files  
redirect to a file

But what about adding this together with other  
results generated later?

```
$ ls *.ent > more-files
```



First, let's revisit redirection...

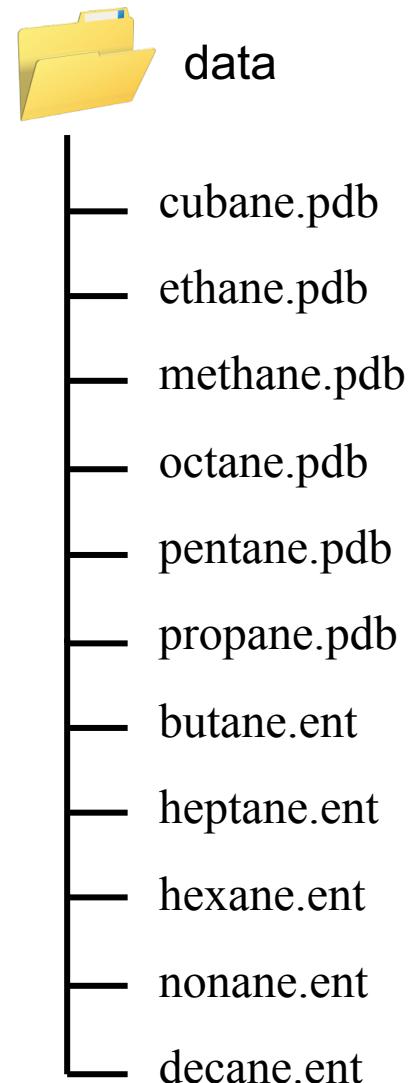
```
$ ls *.pdb > files
```

← list all pdb files  
redirect to a file

But what about adding this together with other  
results generated later?

```
$ ls *.ent > more-files
```

*We just want  
the ent files*



First, let's revisit redirection...

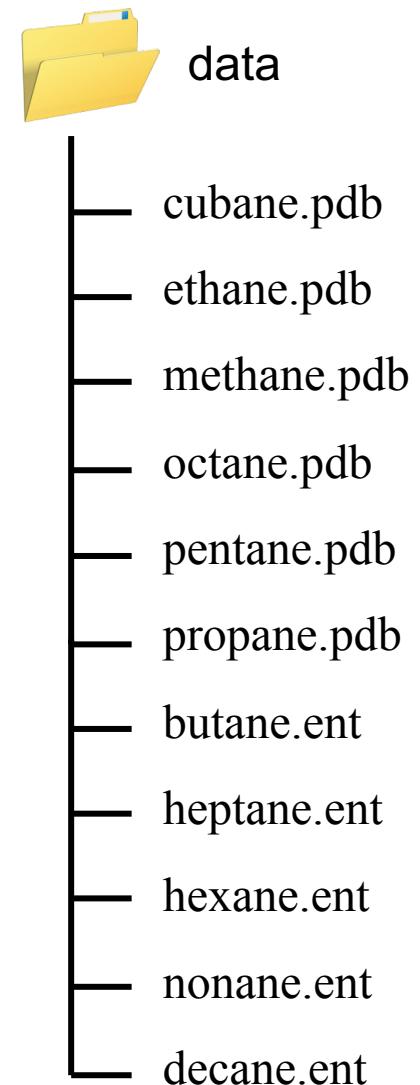
```
$ ls *.pdb > files
```

← list all pdb files  
redirect to a file

But what about adding this together with other  
results generated later?

```
$ ls *.ent > more-files  
$ cat files more-files > all-files
```

← append files  
into a single  
new file



First, let's revisit redirection...

```
$ ls *.pdb > files
```

← list all pdb files  
redirect to a file

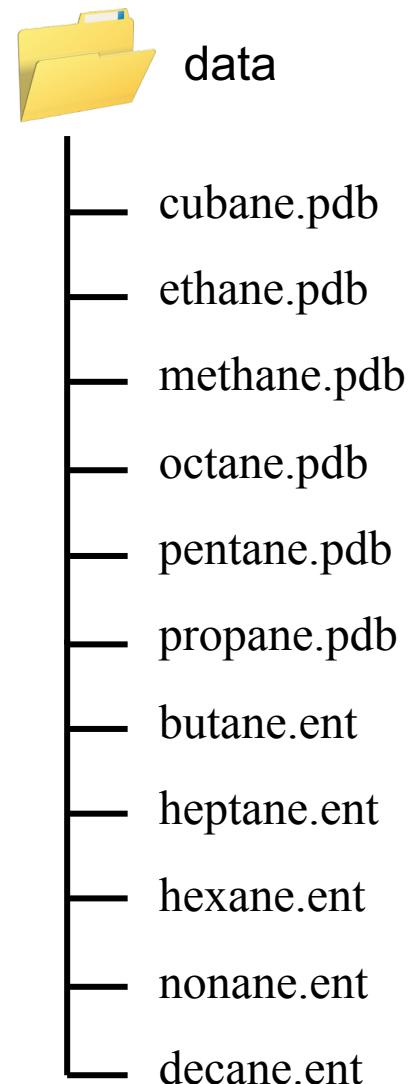
But what about adding this together with other  
results generated later?

```
$ ls *.ent > more-files  
$ cat files more-files > all-files
```

← append files  
into a single  
new file

Instead, we can do...

```
$ ls *.ent >> files
```



First, let's revisit redirection...

```
$ ls *.pdb > files
```

← list all pdb files  
redirect to a file

But what about adding this together with other  
results generated later?

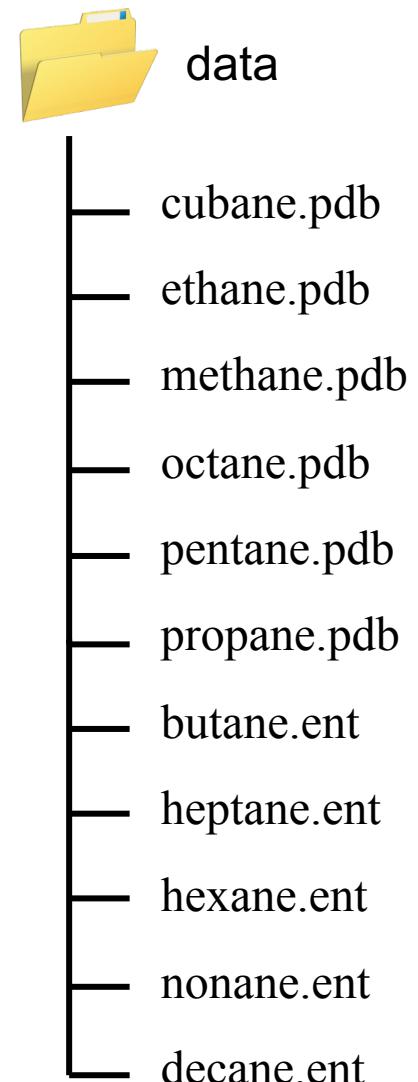
```
$ ls *.ent > more-files  
$ cat files more-files > all-files
```

← append files  
into a single  
new file

Instead, we can do...

```
$ ls *.ent >> files
```

*Note the double >'s – the  
append' operator*

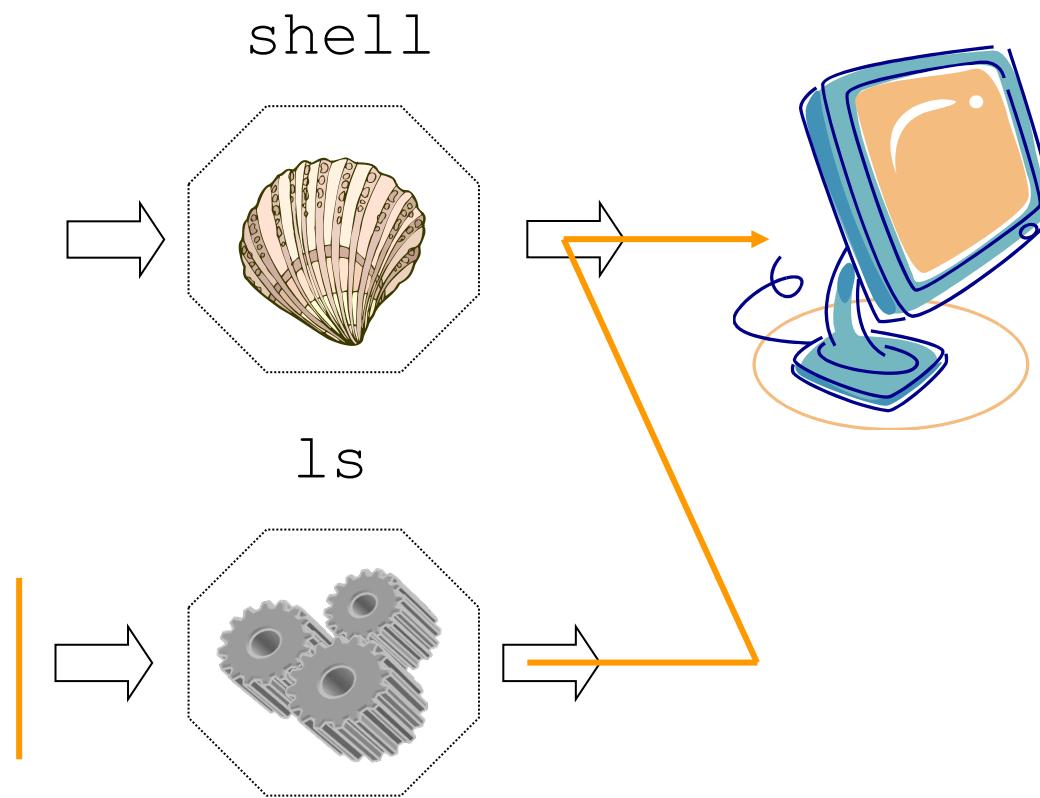


We know that...

Normally, standard output is directed to a display:

We know that...

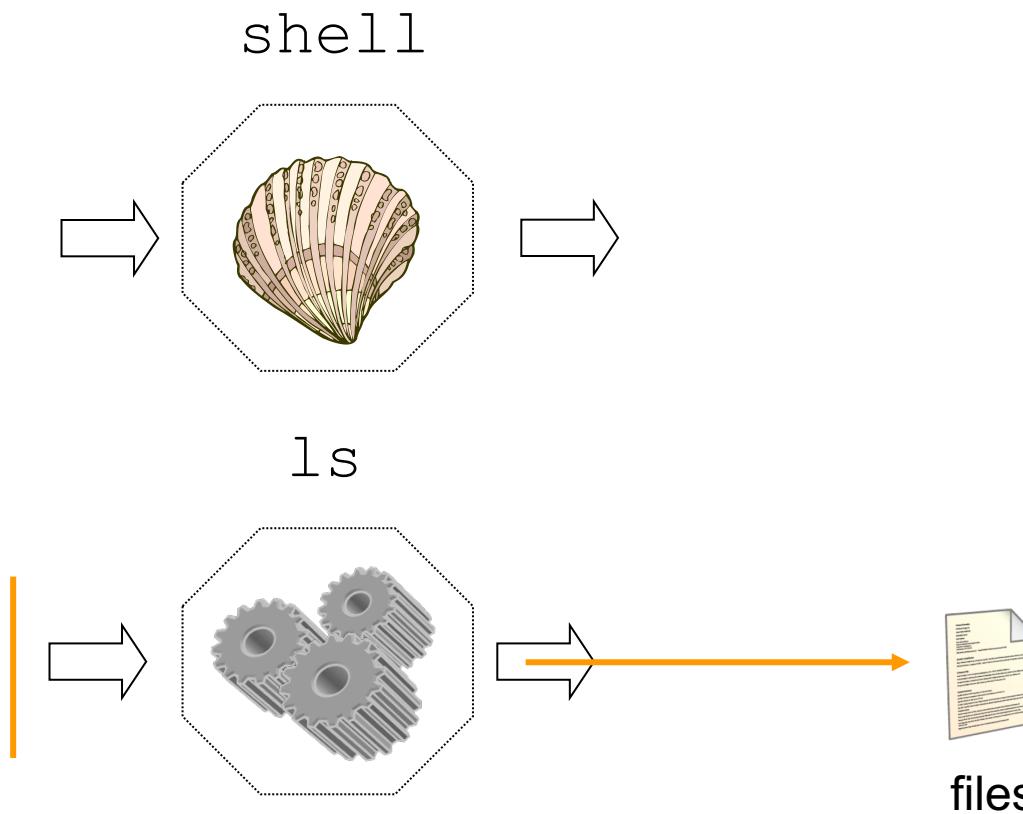
Normally, standard output is directed to a display:



We know that...

Normally, standard output is directed to a display:

But we have redirected it to a file instead:



# But what happens with error messages?

But what happens with error messages?

For example...

```
$ ls /some/nonexistent/path > files
ls: /some/nonexistent/path: No such file or directory
```

But what happens with error messages?

For example...

```
$ ls /some/nonexistent/path > files
ls: /some/nonexistent/path: No such file or directory
```

No files are listed in *files*, as you might expect.

But what happens with error messages?

For example...

```
$ ls /some/nonexistent/path > files
```

```
ls: /some/nonexistent/path: No such file or directory
```

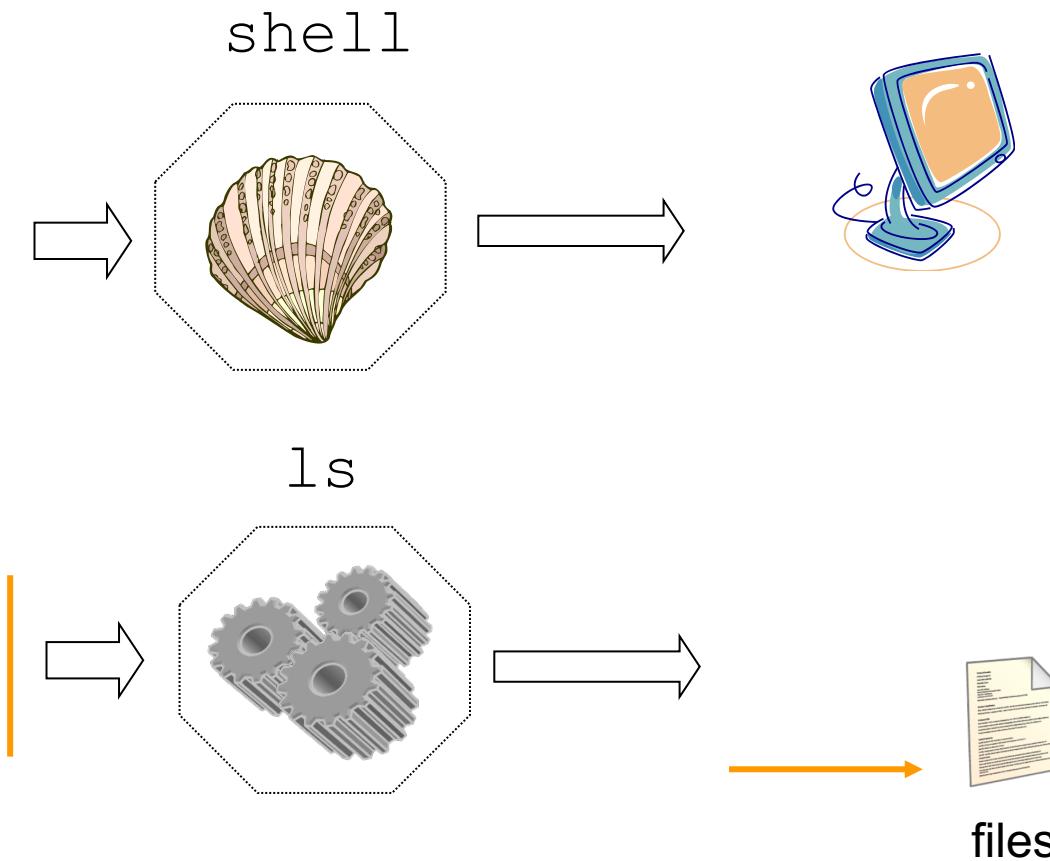
No files are listed in *files*, as you might expect.

But why isn't the error message in *files*?

This is because error messages are sent to the *standard error* (stderr), separate to stdout

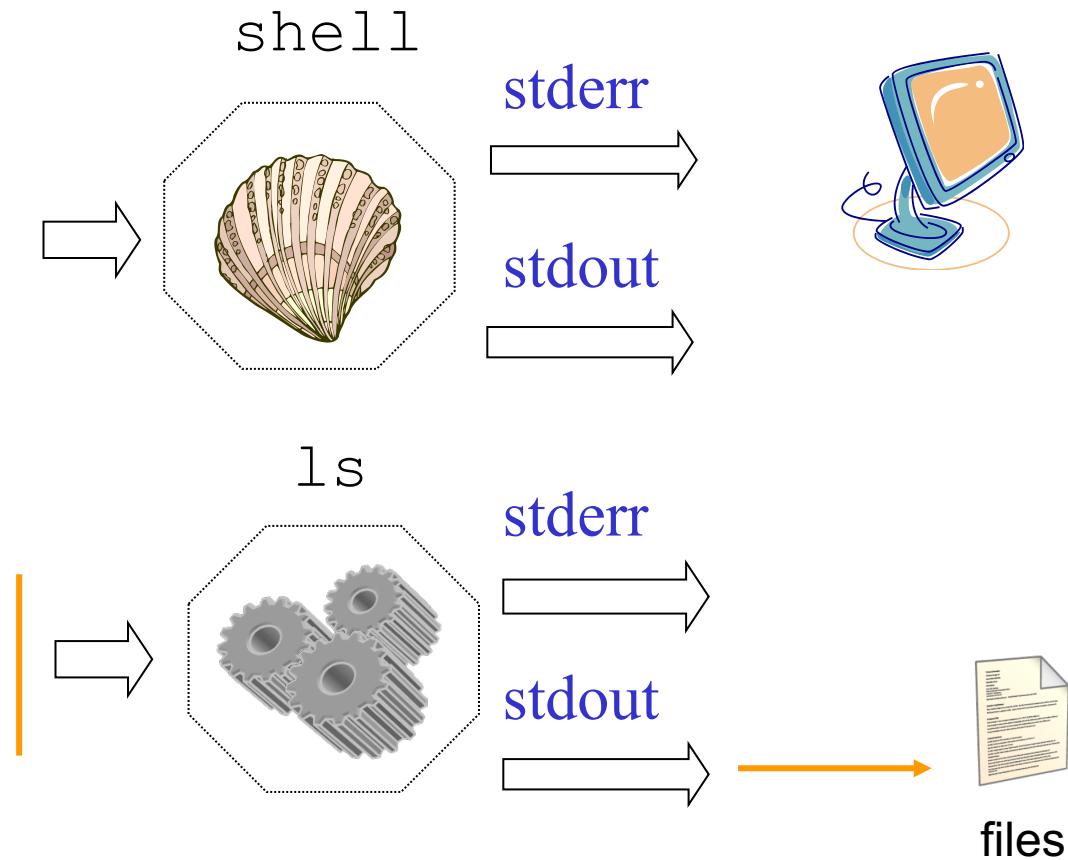
This is because error messages are sent to the *standard error* (stderr), separate to stdout

So what was happening with the previous example?



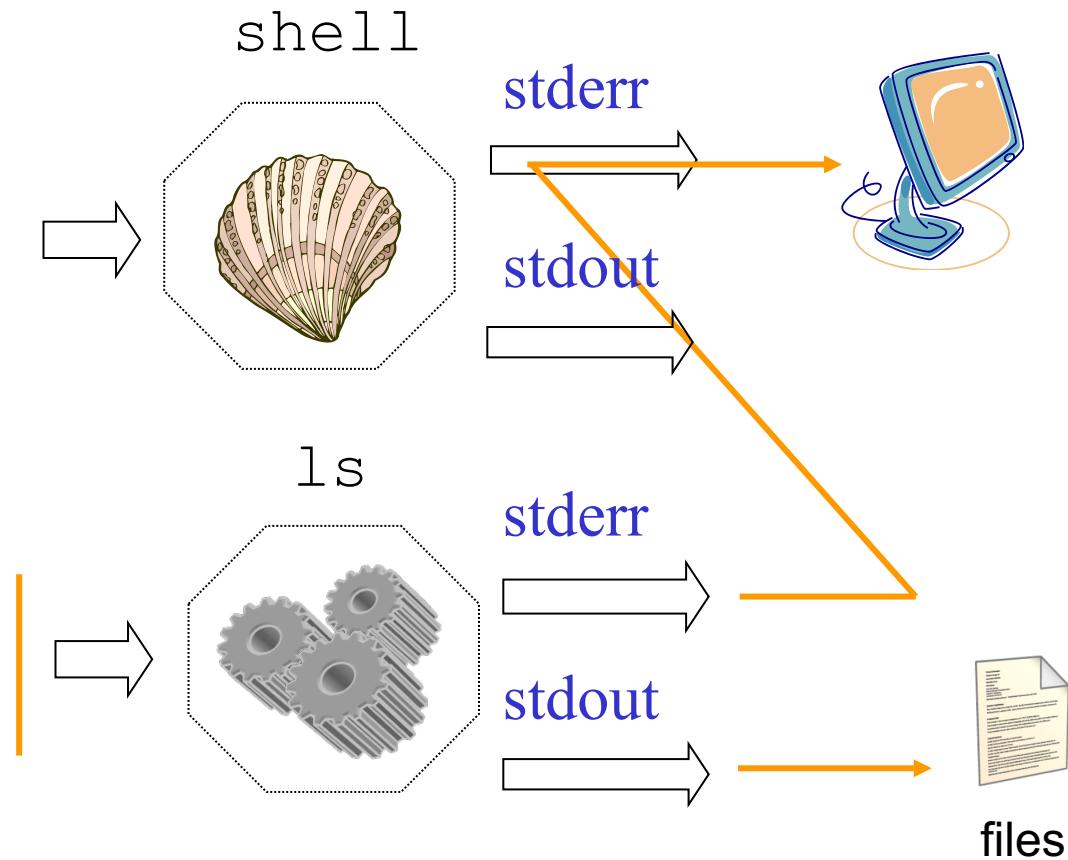
This is because error messages are sent to the *standard error* (stderr), separate to stdout

So what was happening with the previous example?



This is because error messages are sent to the *standard error* (stderr), separate to stdout

So what was happening with the previous example?



We can capture standard error as well as standard output

We can capture standard error as well as standard output

To redirect the standard error to a file, we can do:

```
$ ls /some/nonexistent/path 2> error-log
```



*Redirect as before,  
but with a slightly  
different operator*

We can capture standard error as well as standard output

To redirect the standard error to a file, we can do:

```
$ ls /some/nonexistent/path 2> error-log
```

Now we have any error messages stored in *error-log*

We can capture standard error as well as standard output

To redirect the standard error to a file, we can do:

```
$ ls /some/nonexistent/path 2> error-log
```

Now we have any error messages stored in *error-log*

To redirect both stdout and stderr, we can then do:

```
$ ls /usr /some/nonexistent/path > files 2> error-log
```

We can capture standard error as well as standard output

To redirect the standard error to a file, we can do:

```
$ ls /some/nonexistent/path 2> error-log
```

Now we have any error messages stored in *error-log*

To redirect both stdout and stderr, we can then do:

```
$ ls /usr /some/nonexistent/path > files 2> error-log
```

*We can use both stdout and stderr  
redirection – at the same time*

We can capture standard error as well as standard output

To redirect the standard error to a file, we can do:

```
$ ls /some/nonexistent/path 2> error-log
```

Now we have any error messages stored in *error-log*

To redirect both stdout and stderr, we can then do:

```
$ ls /usr /some/nonexistent/path > files 2> error-log
```

Which would give us contents of */usr* in *files* as well.

So why a ‘2’ before the ‘>’ ?

So why a ‘2’ before the ‘>’ ?

Both stdout and stderr can be referenced by numbers:

```
$ ls /usr /some/nonexistent/path 1> files 2> error-log
```

So why a ‘2’ before the ‘>’ ?

Both stdout and stderr can be referenced by numbers:

```
$ ls /usr /some/nonexistent/path 1> files 2> error-log
```

*Refers to  
stdout*

*Refers to  
stderr*

So why a ‘2’ before the ‘>’ ?

Both stdout and stderr can be referenced by numbers:

```
$ ls /usr /some/nonexistent/path 1> files 2> error-log
```

To just redirect both to the same file we can also do:

```
$ ls /usr /some/nonexistent/path &> everything
```

With ‘&’ denoting both stdout and stderr

So why a ‘2’ before the ‘>’ ?

Both stdout and stderr can be referenced by numbers:

```
$ ls /usr /some/nonexistent/path 1> files 2> error-log
```

To just redirect both to the same file we can also do:

```
$ ls /usr /some/nonexistent/path &> everything
```

With ‘&’ denoting both stdout and stderr

We can also use append for each of these too:

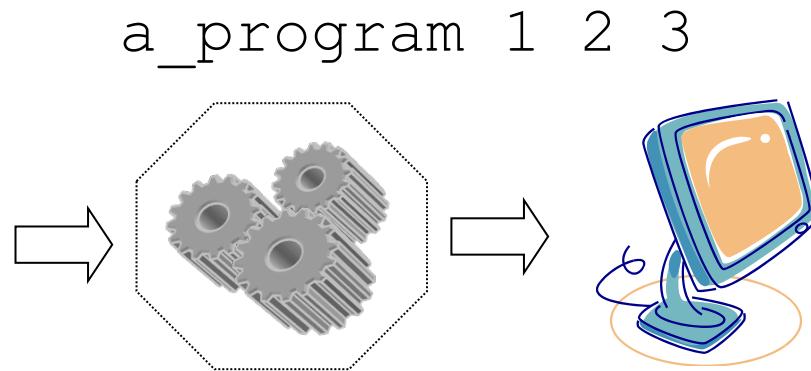
```
$ ls /usr /some/nonexistent/path 1>> files 2>> error-log
```

>	1>	Redirect stdout to a file
	2>	Redirect stderr to a file
	&>	Redirect both stdout and stderr to the same file

>	1>	Redirect stdout to a file
	2>	Redirect stderr to a file
	&>	Redirect both stdout and stderr to the same file
>>	1>>	Redirect and append stdout to a file
	2>>	Redirect and append stderr to a file
	&>>	Redirect and append both stdout and stderr to a file

We've seen how pipes and filters work with using a single program on some input data...

We've seen how pipes and filters work with using a single program on some input data...

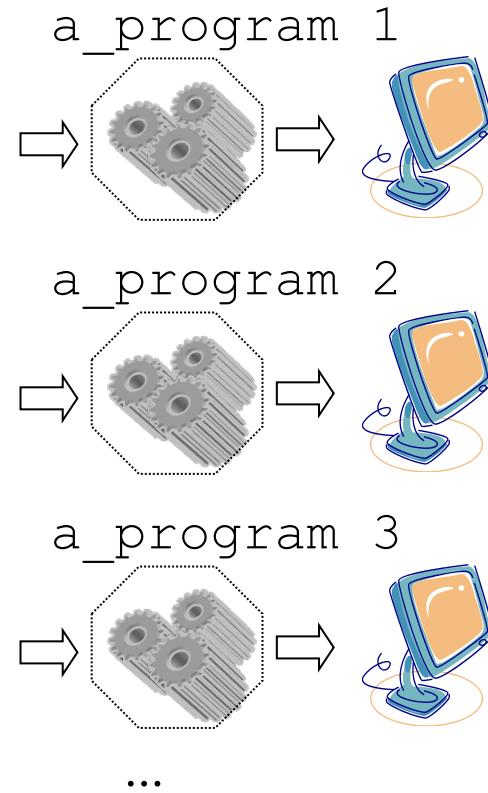


We've seen how pipes and filters work with using a single program on some input data...

But what about running the same program *separately*, for each input?

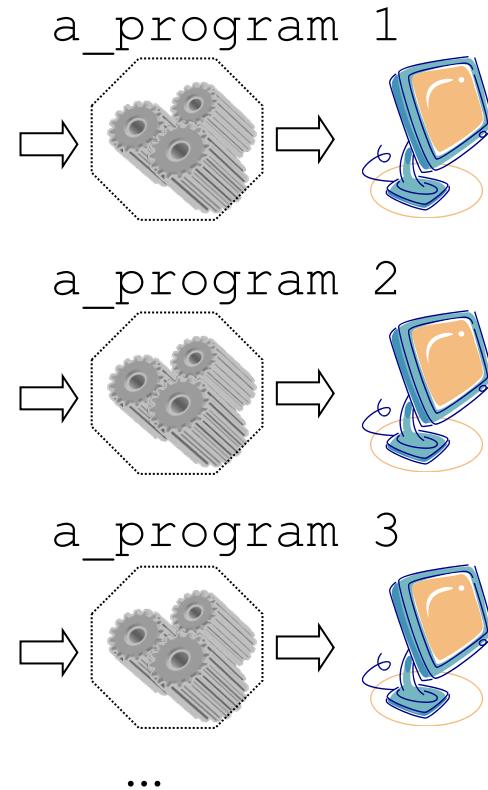
We've seen how pipes and filters work with using a single program on some input data...

But what about running the same program *separately*, for each input?



We've seen how pipes and filters work with using a single program on some input data...

But what about running the same program *separately*, for each input?



We can use *loops* for this...

# So what can we do with loops?

## So what can we do with loops?

Let's go back to our first set of pdb files, and assume we want to compress each of them



data

- cubane.pdb
- ethane.pdb
- methane.pdb
- octane.pdb
- pentane.pdb
- propane.pdb

## So what can we do with loops?

Let's go back to our first set of pdb files, and assume we want to compress each of them

We could do the following for each:

```
$ zip cubane.pdb.zip cubane.pdb  
adding: cubane.pdb (deflated 73%)
```



data

- cubane.pdb
- ethane.pdb
- methane.pdb
- octane.pdb
- pentane.pdb
- propane.pdb

## So what can we do with loops?

Let's go back to our first set of pdb files, and assume we want to compress each of them

We could do the following for each:

```
$ zip cubane.pdb.zip cubane.pdb  
adding: cubane.pdb (deflated 73%)
```

← typical output  
from the zip  
command



data

cubane.pdb

ethane.pdb

methane.pdb

octane.pdb

pentane.pdb

propane.pdb

## So what can we do with loops?

Let's go back to our first set of pdb files, and assume we want to compress each of them

We could do the following for each:

```
$ zip cubane.pdb.zip cubane.pdb  
adding: cubane.pdb (deflated 73%)
```

*The zip file  
we wish to  
create*



data

- cubane.pdb
- ethane.pdb
- methane.pdb
- octane.pdb
- pentane.pdb
- propane.pdb

← typical output  
from the zip  
command

## So what can we do with loops?

Let's go back to our first set of pdb files, and assume we want to compress each of them

We could do the following for each:

```
$ zip cubane.pdb.zip cubane.pdb
```

*adding: cubane.pdb (deflated 73%)*

*The zip file  
we wish to  
create*

*The file(s)  
we wish to  
add to the  
zip file*

← typical output  
from the zip  
command



data

cubane.pdb

ethane.pdb

methane.pdb

octane.pdb

pentane.pdb

propane.pdb

## So what can we do with loops?

Let's go back to our first set of pdb files, and assume we want to compress each of them

We could do the following for each:

```
$ zip cubane.pdb.zip cubane.pdb  
adding: cubane.pdb (deflated 73%)
```

Not efficient for many files



data

- cubane.pdb
- ethane.pdb
- methane.pdb
- octane.pdb
- pentane.pdb
- propane.pdb

Using a loop, we can iterate over each file, and run *zip* on each of them:

```
$ for file in *.pdb; do zip $file.zip $file; done
```

Using a loop, we can iterate over each file, and run *zip* on each of them:

```
$ for file in *.pdb; do zip $file.zip $file; done
```

*For each pdb  
file in this  
directory...*

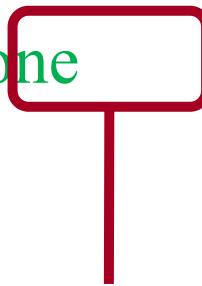
Using a loop, we can iterate over each file, and run *zip* on each of them:

```
$ for file in *.pdb; do zip $file.zip $file; done
```

*Run this command*

Using a loop, we can iterate over each file, and run *zip* on each of them:

```
$ for file in *.pdb; do zip $file.zip $file; done
```



*This is the end of the loop*

Using a loop, we can iterate over each file, and run *zip* on each of them:

```
$ for file in *.pdb; do zip $file.zip $file; done
```

*The semicolons  
separate each part of  
the loop construct*

Using a loop, we can iterate over each file, and run *zip* on each of them:

```
$ for file in *.pdb; do zip $file.zip $file; done
```

*This expands to a list  
of every pdb file*

Using a loop, we can iterate over each file, and run `zip` on each of them:

```
$ for file in *.pdb; do zip $file.zip $file; done
```

*This variable holds  
the next pdb file in  
the list*

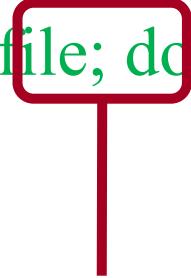
Using a loop, we can iterate over each file, and run *zip* on each of them:

```
$ for file in *.pdb; do zip $file.zip $file; done
```

*We reference the 'file' variable, and use '.' to add the zip extension to the filename*

Using a loop, we can iterate over each file, and run *zip* on each of them:

```
$ for file in *.pdb; do zip $file.zip $file; done
```



*We reference the  
'file' variable again*

Using a loop, we can iterate over each file, and run *zip* on each of them:

```
$ for file in *.pdb; do zip $file.zip $file; done
```

*adding: cubane.pdb (deflated 73%)*

*adding: ethane.pdb (deflated 70%)*

*adding: methane.pdb (deflated 66%)*

*adding: octane.pdb (deflated 75%)*

*adding: pentane.pdb (deflated 74%)*

*adding: propane.pdb (deflated 71%)*

Using a loop, we can iterate over each file, and run *zip* on each of them:

```
$ for file in *.pdb; do zip $file.zip $file; done
adding: cubane.pdb (deflated 73%)
adding: ethane.pdb (deflated 70%)
...
```

In one line, we've ended up with all files zipped

Using a loop, we can iterate over each file, and run *zip* on each of them:

```
$ for file in *.pdb; do zip $file.zip $file; done  
adding: cubane.pdb (deflated 73%)  
adding: ethane.pdb (deflated 70%)  
...
```

In one line, we've ended up with all files zipped

```
$ ls *.zip  
cubane.pdb.zip      methane.pdb.zip  pentane.pdb.zip  
ethane.pdb.zip      octane.pdb.zip    propane.pdb.zip
```

Now instead, what if we wanted to output the first line of each pdb file?

Now instead, what if we wanted to output the first line of each pdb file?

We could use `head -1 *.pdb` for that, but it would produce:

```
==> cubane.pdb <==  
COMPND    CUBANE
```

```
==> ethane.pdb <==  
COMPND    ETHANE
```

```
==> methane.pdb <==  
COMPND    METHANE
```

...

Now instead, what if we wanted to output the first line of each pdb file?

We could use `head -1 *.pdb` for that, but it would produce:

*==> cubane.pdb <==*  
*COMPND CUBANE* 

head produces this  
(it's not in the file)

*==> ethane.pdb <==*  
*COMPND ETHANE*

*==> methane.pdb <==*  
*COMPND METHANE*

...

Now instead, what if we wanted to output the first line of each pdb file?

We could use `head -1 *.pdb` for that, but it would produce:

*==> cubane.pdb <==*  
*COMPND CUBANE*

head produces this  
(it's not in the file)

*==> ethane.pdb <==*  
*COMPND ETHANE*

← this is actually the first  
line in this file!

*==> methane.pdb <==*  
*COMPND METHANE*

...

Now instead, what if we wanted to output the first line of each pdb file?

We could use `head -1 *.pdb` for that, but it would produce:

`==> cubane.pdb <==`  
`COMPND CUBANE`

head produces this  
(it's not in the file)

`==> ethane.pdb <==`  
`COMPND ETHANE`

← this is actually the first  
line in this file!

`==> methane.pdb <==`  
`COMPND METHANE`

...

Perhaps we only want the actual first lines...

However, using a loop:

However, using a loop:

```
$ for file in *.pdb; do head -1 $file; done
```

However, using a loop:

```
$ for file in *.pdb; do head -1 $file; done
```

*We use \$file as we  
did before, but this  
time with the head  
command*

However, using a loop:

```
$ for file in *.pdb; do head -1 $file; done
```

*COMPND CUBANE*

*COMPND ETHANE*

*COMPND METHANE*

*COMPND OCTANE*

*COMPND PENTANE*

*COMPND PROPANE*

What if we wanted this list sorted in reverse afterwards?

What if we wanted this list sorted in reverse afterwards?

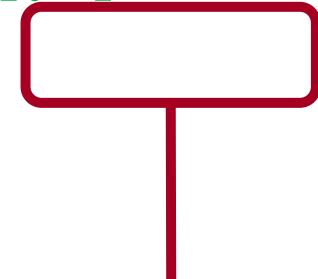
Simple!

```
$ (for file in ls *.pdb; do head -1 $file; done) | sort -r
```

What if we wanted this list sorted in reverse afterwards?

Simple!

```
$ (for file in ls *.pdb; do head -1 $file; done) | sort -r
```



*Using a pipe, we can just add this on the end*

What if we wanted this list sorted in reverse afterwards?

Simple!

```
$ (for file in ls *.pdb; do head -1 $file; done) | sort -r
```

<i>COMPND</i>	<i>PROPANE</i>
<i>COMPND</i>	<i>PENTANE</i>
<i>COMPND</i>	<i>OCTANE</i>
<i>COMPND</i>	<i>METHANE</i>
<i>COMPND</i>	<i>ETHANE</i>
<i>COMPND</i>	<i>CUBANE</i>

`zip`

Create a compressed zip file with other files in it

---

`for ...; do ... done;`

---

Loop over a list of data and run a command once for each element in the list

---



created by

Steve Crouch

July 2011



Copyright © Software Carpentry and The University of Southampton 2010-2011

This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.