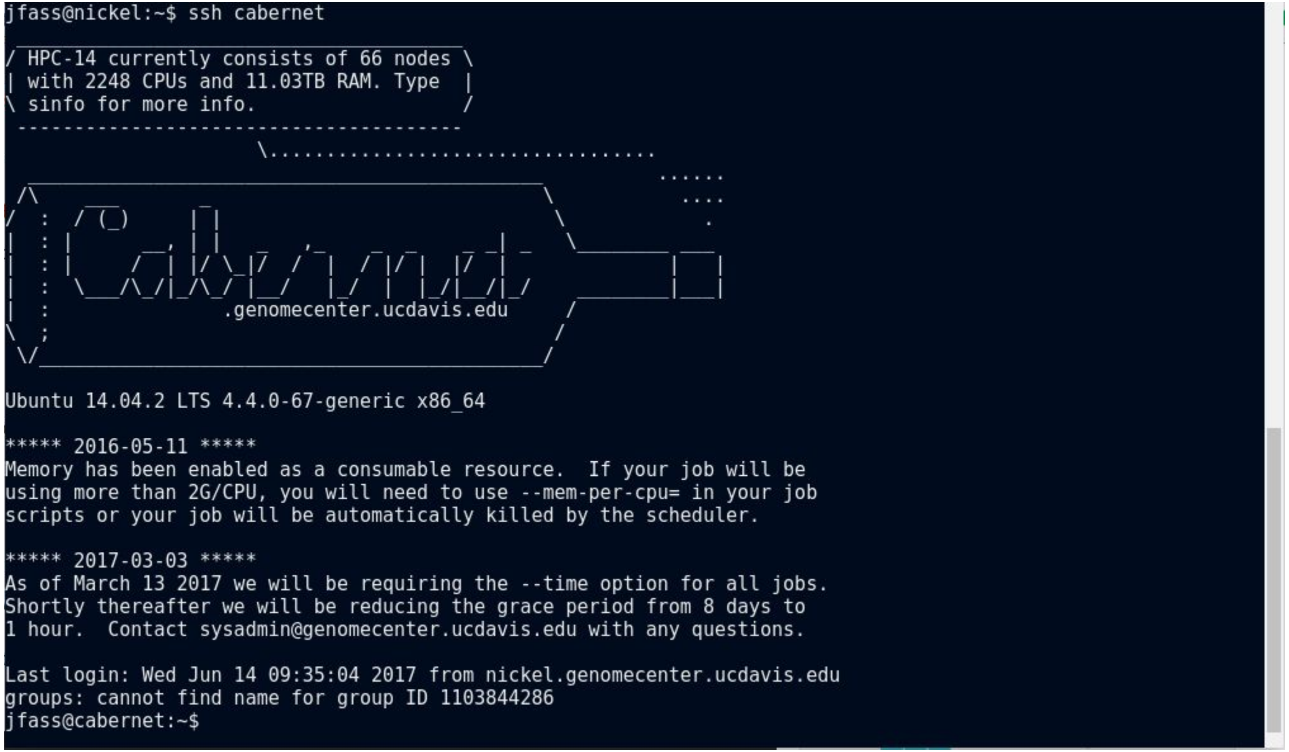
Intro to the Command Line

J Fass | 19 June 2017

The Unix Philosophy

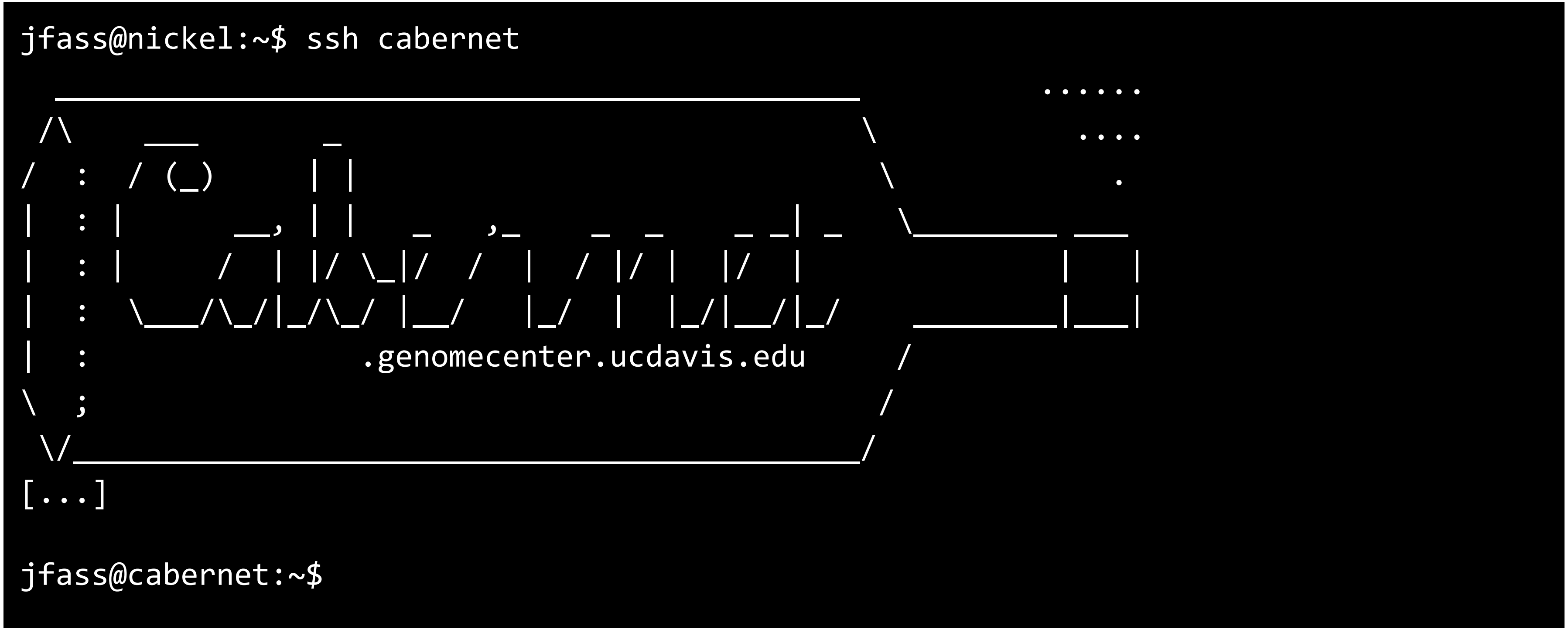
Small, sharp tools … each of which does a well defined job very well.

The Terminal



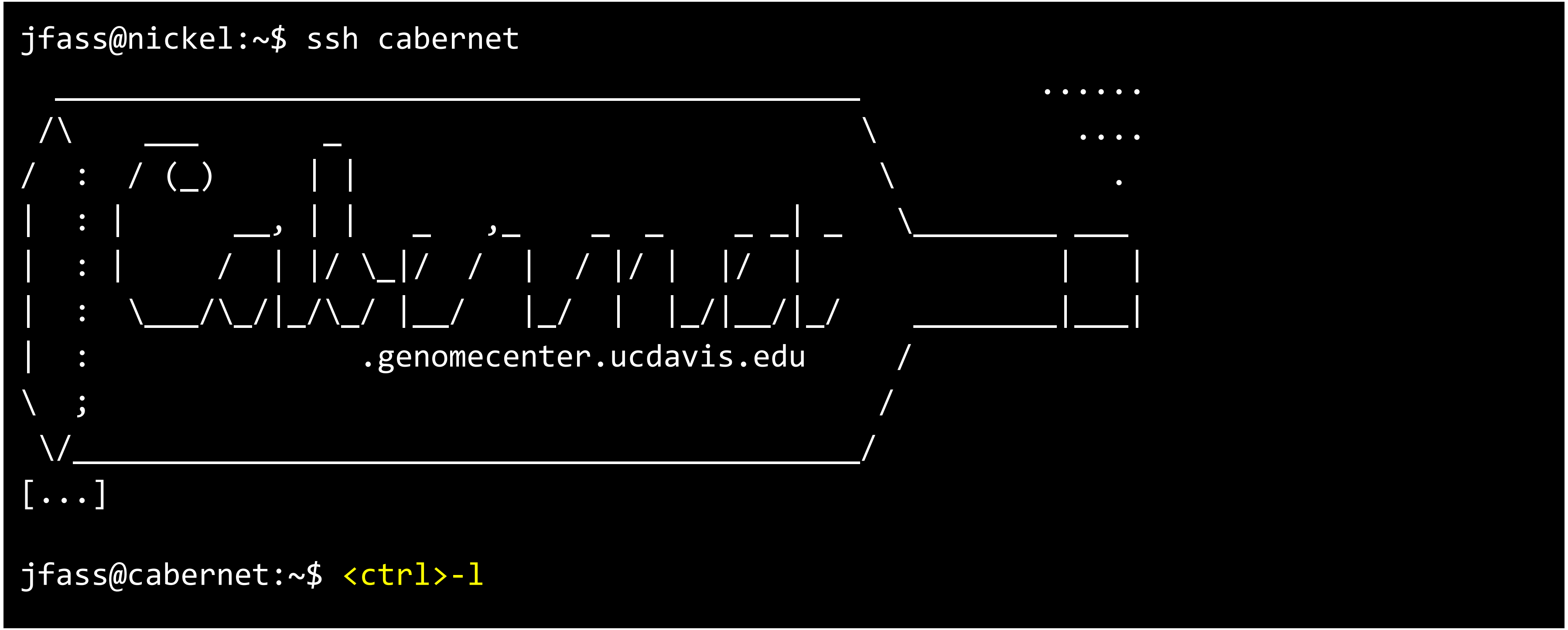
```
jfass@nickel:~$ ssh cabernet
 _______________________________________
/ HPC-14 currently consists of 66 nodes \
| with 2248 CPUs and 11.03TB RAM. Type  |
\ sinfo for more info.                  /
 ---------------------------------------
                      \..................................
  _____________________________________________          ......
 /\    ___       _                             \           ....
/  :  / (_)     | |                             \            .
|  : |      __, | |   _   ,_    _  _    _ _|_    \________ ___
|  : |     /  | |/ \_|/  /  |  / |/ |  |/  |              |   |
|  :  \___/\_/|_/\_/ |__/   |_/  |  |_/|__/|_/    ________|___|
|  :               .genomecenter.ucdavis.edu     /
\  ;                                            /
 \/____________________________________________/

Ubuntu 14.04.2 LTS 4.4.0-67-generic x86_64

***** 2016-05-11 *****
Memory has been enabled as a consumable resource.  If your job will be
using more than 2G/CPU, you will need to use --mem-per-cpu= in your job
scripts or your job will be automatically killed by the scheduler.

***** 2017-03-03 *****
As of March 13 2017 we will be requiring the --time option for all jobs.
Shortly thereafter we will be reducing the grace period from 8 days to
1 hour.  Contact sysadmin@genomecenter.ucdavis.edu with any questions.

Last login: Wed Jun 14 09:35:04 2017 from nickel.genomecenter.ucdavis.edu
groups: cannot find name for group ID 1103844286
jfass@cabernet:~$
```

The Terminal



```
jfass@nickel:~$ ssh cabernet

  _____________________________________________          ......
 /\    ___       _                             \           ....
/  :  / (_)     | |                             \            .
|  : |      __, | |   _   ,_    _  _    _ _| _   \________ ___
|  : |     /  | |/ \_|/  /  |  / |/ |  |/  |              |   |
|  :  \___/\_/|_/\_/ |__/   |_/  |  |_/|__/|_/    ________|___|
|  :               .genomecenter.ucdavis.edu     /
\  ;                                            /
 \/____________________________________________/
[...]

jfass@cabernet:~$
```

The Terminal



```
jfass@nickel:~$ ssh cabernet

  _____________________________________________          ......
 /\    ___       _                             \           ....
/  :  / (_)     | |                             \            .
|  : |      __, | |   _   ,_    _  _    _ _| _   \________ ___
|  : |     /  | |/ \_|/  /  |  / |/ |  |/  |              |   |
|  :  \___/\_/|_/\_/ |__/   |_/  |  |_/|__/|_/    ________|___|
|  :               .genomecenter.ucdavis.edu     /
\  ;                                            /
 \/____________________________________________/
[...]

jfass@cabernet:~$ <ctrl>-l
```

The Terminal

```
jfass@cabernet:~$
```

`<ctrl>-l or -k …`
`Clears terminal, start at top`

`See also: reset`

The Terminal

```
jfass@cabernet:~$
```

<-- prompt (includes $ and one space after)

Huge # of possible configurations; in this case:

`<uname>@<hostname>:<pwd>$<space>`

(pwd = **p**resent **w**orking **d**irectory)

Command Line Basics - Don’t work on head node!

```
jfass@cabernet:~$ srun -t 1440 -n 1 --mem 8000
--reservation workshop --pty /bin/bash

srun: job 5461103 queued and waiting for resources

srun: error: Lookup failed: Unknown host

srun: job 5461103 has been allocated resources

jfass@c4-1:~$
```

This logs you into an interactive session on one of the cluster nodes, so you don’t all play on the head node.

More on this later. Just log into cabernet, then enter the above and press <enter>.

From now on, I’ll omit “jfass@hostname:” for brevity!

Command Line Basics

```
~$ <type command here>

~$ pwd<enter>

/home/jfass
```

Follow command with <enter>

E.g. “pwd” … lists your **p**resent **w**orking **d**irectory

Command Line Basics - Getting Out!

```
~$ command fubar <control-c>

~$ # looks like this:

~$ command fubar ^C

~$

~$ sleep 1000

^C

~$
```

`<control-c>`: escape from entering a command …

(notice: “#” prevent interpretation of text that follows)

… kill a running command (“sleep” actively counts off the specified number of seconds before letting you do anything else)

Command Line Basics - Getting Out!

```
~$ R

[...]

> <control-d>

Save workspace image? [y/n/c]: n

~$
```

<control-d> … escape from some interactive sessions (R, python, …)

(R is a powerful data-centered, statistical computing language)

Command Line Basics - Getting Out!

```
~$ yes | more

[...]

<q>

~$
```

q = **quit**:

Escape from paginators!
(less, man, etc.)

(“yes” says “y” until killed
… it’s a dinosaur)

(“|” is the pipe character …
we’ll explore it more soon)

(“more” shows you a page of
text, then waits for you to
hit <space> to show another,
or exit with “q”)

Command Line Basics - Getting Out!

```
~$ exit
```

“exit” kills the current shell: the program that’s interpreting your commands for the operating system.

Command Line Basics - Where Am I?

```
~$ ls

[...]

~$ pwd

[...]
```

list file in the pwd

present **w**orking **d**irectory

Command Line Basics - Options!

```
~$ ls -R

[...]

[...]

<control-c>
```

list recursively

What did I just do???

Command Line Basics - Read The Manual (RTM)!

```
~$ man ls

[...]

<up, down arrows>

[...]

<q>
```

man <command> consults the **man**ual that exists for basic, OS commands. Any software author can write a “man page” for their software, but most scientific software authors don’t.

Command Line Basics - Options, options, options!

```
~$ ls -l

~$ ls -a

~$ ls -l -a

~$ ls -la

~$ ls -ltrha
```

`man ls …`

Can combine single letter options …

li**s**t **a**ll files (in pwd), in **l**ong format, in **r**everse **t**ime order with **h**uman readable file sizes

Command Line Basics - Options, options, options!

```
~$ alias

[...]

alias ll='ls -alFtrh'

[...]

~$ ll

[...]
```

“alias” allows you to define your own commands. “alias” by itself only *lists* the aliases defined in your bash configuration files. Try one on for size …

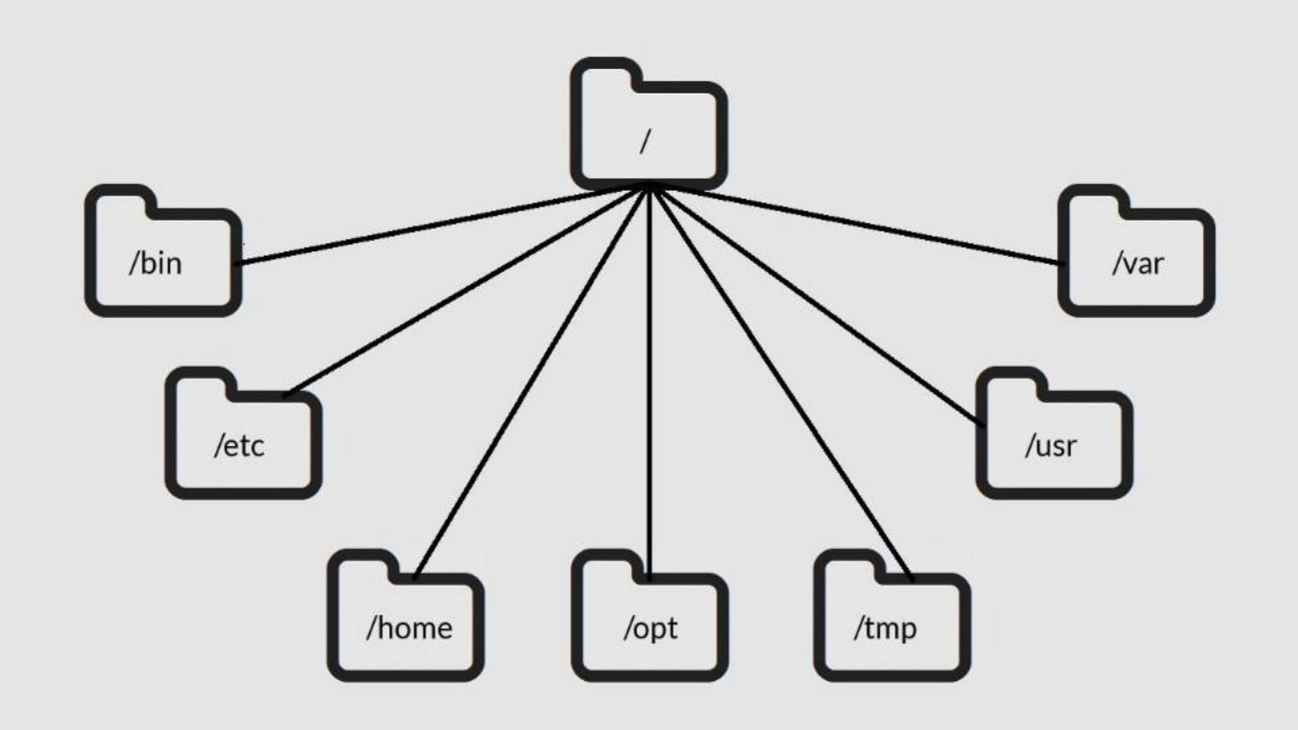
Command Line Basics - Directory Structure

```
~$ ls -R

[...]

./R/x86_64-pc-linux-gnu-library/3.3/BH/include

[...]

<control-c>
```

‘/’ separates directories

Names can include many characters, but avoid spaces and other weird stuff.

Command Line Basics - Directory Structure



www.linuxtrainingacademy.com

Command Line Basics - ‘.’ and ‘..’

```
~$ ls -a

.

..

.bash_history

[...]
```

“.” = pwd

“..” = up one level

Don’t be confused between use of “.” and filenames that start with “.” … the latter are valid filenames, that are just “hidden” unless you use the “ls” command’s “-a” option.

Command Line Basics - Absolute/Relative Address

```
~$ ls /home/jfass/

[...]

~$ ls ./

[same ...]

~$ ls ../jfass/

[same ...]

~$

[yup, same ...]
```

“.” = pwd

“..” = up one level

Command Line Basics - <Tab> Completion

```
~$ ls /home/jfas<tab>

~$ ls /home/jfass/
```

<tab> will literally save your life. Hours of it.

A single <tab> auto-completes when it’s possible (when only a single possible completion exists).

Command Line Basics - <Tab> Completion

```
~$ ls /home/j<tab>

~$ ls /home/j

~$ ls /home/j<tab>

jacob/     [...]
jagadish/
jagomez/                                         jwbucha/

[...]
```

`<tab> will literally save your life. Hours of it.`

`Two <tab>s in a row will show you all the possible completions, when there wasn’t a` *`single`* `one for the single <tab> to use.`

Command Line Basics - <Tab> Completion

```
~$ ls /h<tab>

~$ ls /home/<tab>

~$ ls /home/<tab>

~$ ls /home/j<tab>

~$ ls /home/j<tab>

~$ ls /home/jf<tab>

~$ ls /home/jfass/
```

Use it!

Watch out for RSI ...

Command Line Basics - Create and Destroy

```
~$ mkdir temp

~$ cd temp/

~/temp$ echo “Hello, world!” > first.txt

~/temp$ cat first.txt

~/temp$ rm first.txt

~/temp$ cd ../

~$ rmdir temp
```

Create a directory

Change directories

Put text into a file

Con**cat**enate file to screen

Remove file

Up and out

Remove (empty) directory

Command Line Basics - Pipe and Redirect

```
~$ mkdir CLB; cd CLB/

~/CLB$ echo “first” > test.txt

~/CLB$ echo “second” > test.txt

~/CLB$ cat test.txt

~/CLB$ echo “third” >> test.txt

~/CLB$ cat test.txt
```

“>” redirects the output from one command to a file, instead of the screen.

“>” replaces

“>>” appends

Command Line Basics - Pipe and Redirect

```
~/CLB$ cut -c 1-3 test.txt

~/CLB$ cat test.txt | cut -c 1-3



~/CLB$ cat test.txt > cut -c 1-3
```

“cut” cuts lines of text.

“|” pipes output from one command to be the input of another command.

“>” is wrong here … what does this command do?

Command Line Basics - Pipe and pipe and pipe …

```
~/CLB$ cat test.txt

~/CLB$ cat test.txt | cut -c1-3

~/CLB$ cat test.txt | cut -c1-3 | grep s
```

Pipes allow us to build up compound operations, filtering and changing data as we go.

(“grep” searches for matches to **r**egular **e**xpressions … patterns)

Command Line Basics - History

```
~/CLB$ history

~/CLB$ history | head

~/CLB$ history | tail

~/CLB$ history | tail -n 15

~/CLB$ history | less
```

Since we often develop long commands through trial and error, it helps to see and access what we’ve done.

In “less,” up and down arrows, pgUp, pgDn, and “q” to exit. Also, “/pattern” searches for pattern each <enter>. “n” and “N” for next and previous match, “g” and “G” for beginning and end of file / stream.

Command Line Basics - History

```
  558  cat test.txt | cut -c1-3 | grep s
  559  history
  560  history

~/CLB$ !560
```

“!#” repeats command # from your history.

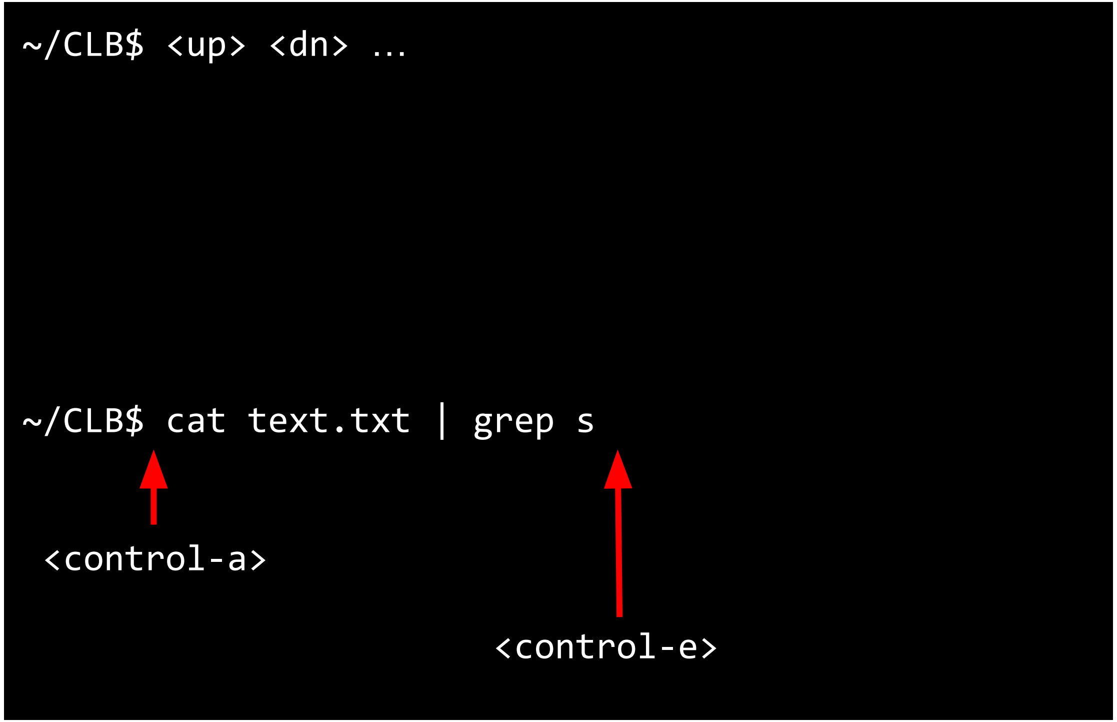
Command Line Basics - History Search

```
~/CLB$ <control-r>

(reverse-i-search)`first': echo "first" > test.txt
```

`<control-r>text` triggers a recursive search for “text” from your history. After finding the most recent match that you like, use `<control-r>` again to get to an earlier match (and again, and …).

`<enter>` executes the command; left or right arrow fills the command line with the command but allows you to edit it before running it.

Command Line Basics - History Search



```
~/CLB$ <up> <dn> …

~/CLB$ cat text.txt | grep s
```

And, by the way, the up and down arrows take you backwards and forwards through your history of commands. Reach one you like, and start editing.

Also, by the way, <control-a> and <control-e> bring you to the beginning and end of your command.

Command Line Basics - Editing Commands

```
~/CLB$ blah blah blah

~/CLB$ blah blah <control-k>blah

~/CLB$ blah blah

~/CLB$ blah blah <control-w>

~/CLB$ blah
```

`Left arrow to before the last “blah,” then <control-k> … kills text from here ‘til the end of the line.`

`Now, <control-w> … kills` *`preceding`* `word.`

Command Line Basics - Compression

```
~/CLB$ gzip test.txt

~/CLB$ file text.txt.gz

~/CLB$ gunzip test.txt.gz

~/CLB$ bzip2 test.txt; bunzip2 test.txt.bz2
```

Compress big files using “gzip,” “bzip2.” Bzip2 compresses smaller, but takes longer.

(“file” gives you info about the *type* of file you’re looking at)

Command Line Basics - Archives

```
~/CLB$ wget
ftp://igenome:G3nom3s4u@ussd-ftp.illumina.com/PhiX/I
llumina/RTA/PhiX_Illumina_RTA.tar.gz

~/CLB$ tar -xzvf PhiX_Illumina_RTA.tar.gz
```

Large directory trees may be compressed as “tarballs” … see “tar.”

Let’s grab one and expand it.

Command Line Basics - Forced Removal

```
~/CLB$ rm -rf PhiX
```

This is a dangerous one. Remove a file / directory, do it **r**ecursively to all sub-directories, and **f**orce removal (by-pass confirmation questions).

Caution is warranted. There’s no Trash Bin, and no gauranteed way to recover deleted files.

Command Line Basics - Wildcard Characters

```
~/CLB$ tar -xzvf PhiX_Illumina_RTA.tar.gz

~/CLB$ ls PhiX/Illumina/RTA/Sequence/*/*.fa

[...]
```

Let’s re-unarchive that tarball, to have something to look at.

List all files a few directories down that end in “.fa” …

Command Line Basics - Wildcards and Find

```
~/CLB$ find . -name “*.fa”

[...]




~/CLB$ find . -name “*.f?”
```

“*” can fill in for anything in a filename, *except* “/” … there’s a more appropriate command to use when you don’t know which directory level the files you’re looking for are at: “find”

“?” is like “*,” except only fills in for a single character.

Command Line Basics - Fasta example

```
~/CLB$ curl -k
https://bioshare.bioinformatics.ucdavis.edu/bioshare
/download/pdhqicmfgw2bra8/variant.neighborhoods.fa >
regions.fa

~/CLB$ wc -l *

~/CLB$ grep -c “>” regions.fa
```

Grab a sequence file from the web (“curl” copies from a **url**) … see also “wget”

“wc” counts words, etc.; “-l” only gives line count

“-c” causes grep to count matches instead of printing them

Command Line Basics - Fasta example … Pipes!

```
~/CLB$ grep “>” regions.fa | cut -c 2-

~/CLB$ grep ">" regions.fa | cut -c 2- | cut -f1 -d:

~/CLB$ grep ">" regions.fa | cut -c 2- | cut -f1 -d:
| sort

~/CLB$ grep ">" regions.fa | cut -c 2- | cut -f1 -d:
| sort | uniq -c

~/CLB$ grep ">" regions.fa | cut -c 2- | cut -f1 -d:
| sort | uniq -c | sort -rn -k1,1
```

Follow each step of these commands … the entire final command counts how many sequences come from each contig.

“sort,” “uniq” are self-explanatory; “uniq -c” adds the counts (must be sorted first), and “sort -rn -k1,1” sorts lines in **r**everse **n**umerical order based only on the first column (the counts).

Command Line Basics - Symbolic Links

```
~/CLB$ ln -s PhiX/Illumina/RTA/Sequence/WholeGenomeFasta/genome.fa .

~/CLB$ ls -ltrha

[...] genome.fa -> PhiX/Illumina/RTA/Sequence/WholeGenomeFasta/genome.fa

~/CLB$ grep -c ">" genome.fa

1
```

“`ln -s [target] [link location/name]`” creates a “shortcut” to the file.

The target file can be deleted, which leaves a broken link. The link can be deleted without affecting the file.

Otherwise, addressing the link addresses the file.

Command Line Basics - Bioinformatics At Last?

```
~/CLB$ bwa mem genome.fa regions.fa > aln.sam

[E::bwa_idx_load] fail to locate the index files

~/CLB$ cat aln.sam

~/CLB$
```

Let’s align sequences to a sequence!

Woops … what went wrong?

I redirected output to a file, why didn’t it go there? (Even if I wouldn’t want that kind of stuff to end up in an alignment file!)

Command Line Basics - STDOUT & STDERR

```
~/CLB$ bwa mem genome.fa regions.fa 1> aln.sam 2>
aln.err

~/CLB$ cat aln.sam

~/CLB$ cat aln.err

[E::bwa_idx_load] fail to locate the index files
```

Programs can write to two separate output streams: standard out, and standard error. Former mostly used for output, latter mostly used for error messages.

(“1>” is equivalent to “>”)

Now we can store error messages from many jobs run at once, to separate files.

Command Line Basics - Loops

```
~/CLB$ for i in {1..21}; do echo $i >> a; done

for name in {list}; do

    commands

done

~/CLB$ cat a

[...]
```

For loop, for a defined number of steps.

Command Line Basics - Loops

```
~/CLB$ grep “>” regions.fa | cut -c2- | while read
header; do echo “contig_$header” >> b; done

while {condition}; do

    commands

done

~/CLB$ cat b

[...]
```

While loop, for stopping on a condition.

Command Line Basics - Paste

```
~/CLB$ paste a b

[...]
```

“paste” jams columns of equal size together. See also “join” …

Command Line Basics - Running in the Background

```
~/CLB$ sleep 100000

[...]

<control-z>

^Z

[1]+  Stopped                 sleep 100000

bg

[1]+ sleep 100000 &
```

When a long job is running, use “<control-z>”, then “bg” to suspend it, then run it in the background. If you want to kill it before it finisheds on its own, either “kill %1” (or %2, etc., if multiple jobs), or “fg,” then <control-c>.

Command Line Basics - Running in the Background

```
~/CLB$ sleep 100000 &

[1] 2166

~/CLB$ kill %1

~/CLB$

[1]+  Terminated                    sleep 100000
```

To start a job already running in the background, append “&” to the command.

Command Line Basics - Running in the Background

```
~/CLB$ nohup sleep 100000 &

[1] 2178

~/CLB$ nohup: ignoring input and appending output to
‘nohup.out’



~/CLB$ jobs

[1]+  Running                 nohup sleep 100000 &
```

To make a background job resistant to a lost connection or terminal problems, use “nohup.”

“jobs” will list jobs running in the background.

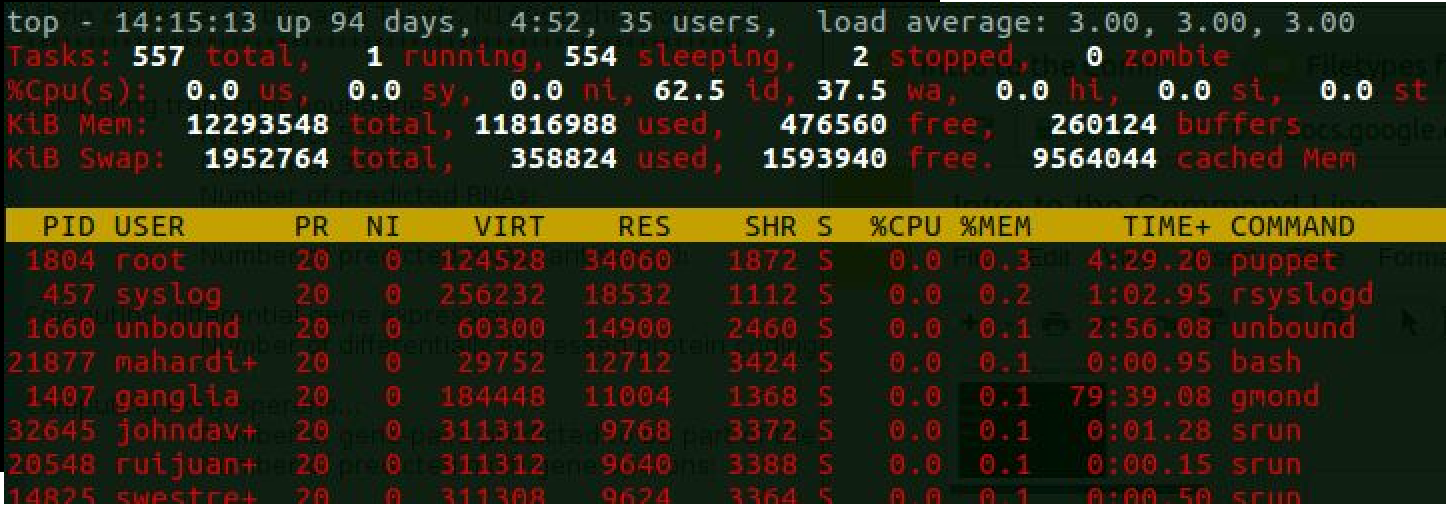
With nohup, you should be able to exit from the shell, and the job keeps running.

(Not as useful in a cluster environment?)

Command Line Basics - Table of Processes

```
~/CLB$ top
```

“Top” prints a self-updating table of running processes and system stats. Try “M,” “P,” and “u” then uname …



```
top - 14:15:13 up 94 days,  4:52, 35 users,  load average: 3.00, 3.00, 3.00
Tasks: 557 total,   1 running, 554 sleeping,   2 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni, 62.5 id, 37.5 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  12293548 total, 11816988 used,   476560 free,   260124 buffers
KiB Swap:  1952764 total,   358824 used,  1593940 free.  9564044 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1804 root      20   0  124528  34060   1872 S   0.0  0.3   4:29.20 puppet
  457 syslog    20   0  256232  18532   1112 S   0.0  0.2   1:02.95 rsyslogd
 1660 unbound   20   0   60300  14900   2460 S   0.0  0.1   2:56.08 unbound
21877 mahardi+  20   0   29752  12712   3424 S   0.0  0.1   0:00.95 bash
 1407 ganglia   20   0  184448  11004   1368 S   0.0  0.1  79:39.08 gmond
32645 johndav+  20   0  311312   9768   3372 S   0.0  0.1   0:01.28 srun
20548 ruijuan+  20   0  311312   9640   3388 S   0.0  0.1   0:00.15 srun
14825 swestre+  20   0  311308   9624   3364 S   0.0  0.1   0:00.50 srun
```

Command Line Basics - Shell Scripts

```
#!/bin/bash

grep -o . $1 | \

    sort | \

    uniq -c | \

    sort -rn -k1,1

<control-o><control-x>
```

“Nano” is a simple text editor; open it with “nano test.sh” and type / copy in this … your first shell script! Save and exit as shown at the bottom, via <control-o><control-x>.

Command Line Basics - Permission Needed

```
~/CLB$ ll test.sh

-rw-rw-r-- 1 jfass biocore 79 Jun 18 19:00 test.sh

~/CLB$ chmod u+x test.sh

~/CLB$ ll test.sh

-rwxrw-r-- 1 jfass biocore 79 Jun 18 19:00 test.sh*
```

Though you can run the commands in test.sh in several ways, to really make it a script you need to give yourself permission to execute it.

Permissions now are to read and write for user and group, and only read for the world (ignore first dash, then rw- for user, rw- for group, and r-- for world). Add execute for user as shown.

Command Line Basics - Permission Needed

```
~/CLB$ ./test.sh genome.fa

   1686 T
   1292 A
   1253 G
   1155 C
      1 x
      1 p
      1 i
      1 h
      1 >
```

Execute by addressing the shell script with its path, and feed it an argument as required in the script.

Congrats! You've now counted all characters in the phiX genomic fasta file.

Command Line Basics - Installing (Simple) Software

```
~/CLB$ cd ../

~$ mkdir tools

~$ cd tools/

~$ git clone https://github.com/lh3/minimap.git

~$ cd minimap/

~$ make

~$ ./minimap
```

Simple software installations require “make” … or maybe “./configure” then “make” … and yield an executable in the main software directory. Other software can be massively painful to install. Talk to your sys admins …

Questions … comments … confusion?