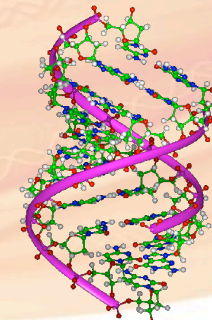


Bioinformatics Computational Methods 1 - BIOL 6308



September 12th 2013

<http://155.33.203.128/cleslin/home/teaching6308F2013.php>

Last Time

- Different Types OS
- Linux Overview
- Command Line Interface – CLI
- Linux Command Documentation – `man` pages
- Structure of Linux (Files and Directories)
- Types of Files
- Permissions
- System Commands
- Linking Files
- Questions?
- **Introduction of the shell**
- Was everyone successful using `scp`?

Bash – Bourne Again Shell

- Supports Basic Regular Expressions
 - E.x. `ls myfile.*` or `ls data4?`
 - `*` matches 0+ of any character (wildcard)
 - `?` matches 1 of any character
 - `[abc]` will match a, b, or c.
- Has a powerful scripting language
 - Bash scripting
 - You'll use this in Operating Systems

```
[cleslin@fisher letters]$ ls aa[ac]
aaa  aac
[cleslin@fisher letters]$ ls aa[a-c]
aaa  aab  aac
```

Linux: shell Path Basics

- Modifying *environment variables*

sh: `PAGER=/usr/bin/less; export PAGER`

bash: `export PAGER=/usr/bin/less`

- Execute an external command (sh)

`$ somecommand`

`somecommand: command not found`

`$ echo $PATH`

`/usr/lib64/qt-3.3/bin:/usr/kerberos/bin:/usr/lib64/
ccache:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/
usr/sbin:/sbin:/usr/local/blast/blast-2.2.18/bin:/
usr/local/fasta-35.4.7/bin:/home/cleslin/bin`

`$ pwd`

`/home/user_name`

`$ /data/METHODS/Fall/test/somecommand`

`Hello world!`

`$ cd /home/test`

`$ PATH=$PATH:`pwd`; export PATH`

`$ somecommand`

`Hello world!`

Login Scripts

- You don't want to enter aliases, set environment variables, set up command line editing, etc. **every time you log in**
- All of these things can be done in a script that is run each time the shell is started
- Location
 - For bash:
 - `~/ .bashrc`
 - For tcsh
 - `~/ .cshrc`
- Whatever you do:
 - `$ cp .bashrc .bashrc.orig #before you do anything with this file!!!!`

Just in case you screw up!!!

```
/bin/cp .bashrc.orig .bashrc
```

Edit Example .bashrc (partial)

```
alias fisher='ssh fisher.neu.edu'
alias 'rm'='rm -i'
alias 'l'='less'
#Use human-readable filesizes
alias du="du -h"
alias df="df -h"
alias today='date +"%A, %B %-d, %Y"'
#I'm lazy and don't want to remember, so I made a function!!
function dict() {

    curl dict://dict.org/d:"$@" ;

}
PATH=$PATH:/home/cleslin/test; export PATH

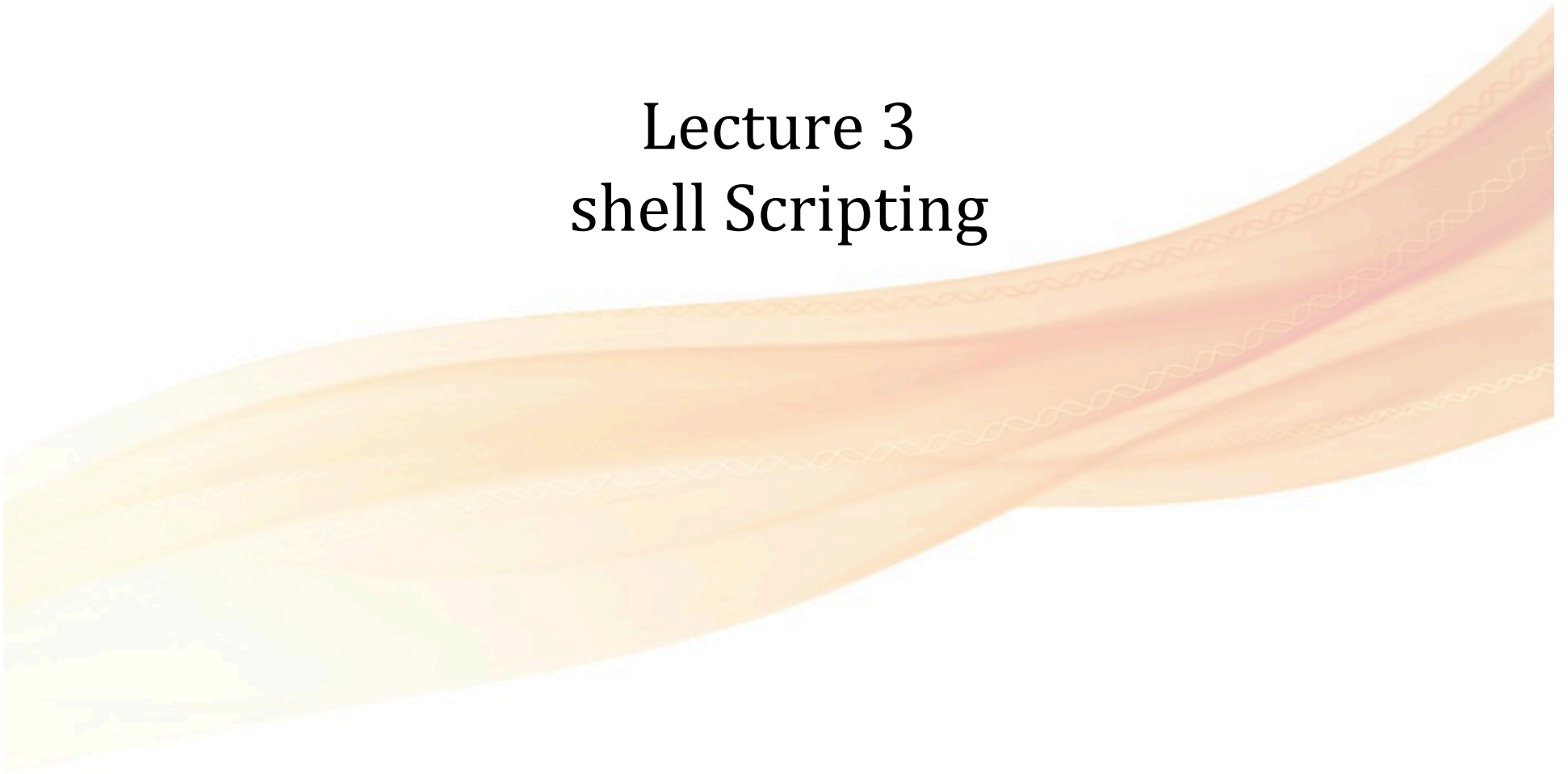
echo "Remember Chess... Make it a great day!"
```

Next time you login it will be set, or you can do a:

```
$ source .bashrc
```

Lecture 3

shell Scripting



Why Use shell Scripting

- Combine lengthy and repetitive sequences of commands into a single, simple command
- Create new commands using combinations of utilities **in ways original authors never thought of**
- shell scripts allow several commands that would be entered manually at a CLI
 - To be executed **automatically**
 - Without having to wait for a user to trigger each stage of the sequence
- Great for:
 - file manipulation
 - program execution
 - printing text
- shell scripting should be for small quick tricks
 - For projects use - higher level language like Perl

Linux: Bourne shell Script Syntax

- The first line of a *sh* script **should** begin as follows:
 #!/bin/sh #can use bash here
- Any unquoted # is treated as comment until end-of-line
- Every line **first** parsed for shell *metacharacters*.
 - Characters shell will do something with and **include**:
 # ' " & > < \$ % * [] ? ! ` ~ ; | , { }
- Distinct commands may be separated by end-of-line, semicolon, or comma
 - the ";"
 - Separator **between** words on a line - **interpreted** as a **newline**
 - **Multiple commands** on a **single line**
- "Back-tick" subshells are expanded and **executed**
- \
 - as the last character on a line causes the following line to be logically joined before interpretation (we'll soon see)

<http://tldp.org/LDP/abs/html/subshells.html>

ksh/bash vs sh

- **ksh** and **bash** are both supersets of **sh**
- For maximum portability, even to very old computers, you should stick to the commands found in **bash**

Table 6-1 Linux shells

Shell Name	Original Shell from Which Derived	Description in Terms of Shell Programming
Bash	Bourne and Korn shells	Offers strong scripting and programming language features, such as shell variables, logic structures, and math/logic expressions; combines the best features of the Bourne and Korn shells
cshtcsh	C shell	Conforms to a scripting and programming language format; shell expressions use operators similar to those found in the C programming language
kshzsh	Korn shell	Is similar to the Bash shell in many respects, but also has syntax similar to that of C programming; useful if you are familiar with older Korn shell scripts

All Linux versions use the Bash shell as the default

What is a shell Script?

```
$ cat > hello.sh <<MY_PROGRAM
#!/bin/sh
echo 'Hello, world'
MY_PROGRAM
```

} Generate file
named hello.sh

```
$ chmod +x hello.sh
$ ./hello.sh
Hello, world
```

} give right
privileges and
run

} output

Using UNIX/Linux shell Scripts

- After creating shell script:
 - The OS is instructed that the file is an executable shell script via the **chmod** command
- Script files can be run in several ways:
 - Set the path variable and type the script name at the command prompt
 - Type **./filename** if script is in current directory
 - Type the script name preceded by the full path

What is a shell Script? A Text File

```
$ cat > hello.sh <<MY_PROGRAM
```

```
#!/bin/sh
```

```
echo 'Hello, world'
```

```
MY_PROGRAM
```

```
$ chmod +x hello.sh
```

```
$ ./hello.sh
```

```
Hello, world
```

A decorative graphic consisting of several overlapping, wavy, translucent bands in shades of orange, yellow, and light green, flowing from the bottom left towards the top right, partially obscuring the lower portion of the terminal output.

What is a shell Script? How To Run

```
$ cat > hello.sh <<MY_PROGRAM
#!/bin/sh
echo 'Hello, world'
MY_PROGRAM
$ chmod +x hello.sh
$ ./hello.sh
Hello, world
```

This first line indicates what interpreter to use when running the script

What is a shell Script? What To Do

```
$ cat > hello.sh <<MY_PROGRAM
#!/bin/sh
echo 'Hello, world'
MY_PROGRAM
$ chmod +x hello.sh
$ ./hello.sh
Hello, world
```



What is a shell Script? Executable

```
$ cat > hello.sh <<MY_PROGRAM
#!/bin/sh
echo 'Hello, world'
MY_PROGRAM
$ chmod +x hello.sh
$ ./hello.sh
Hello, world
```

A decorative graphic consisting of several overlapping, wavy, translucent bands in shades of orange, yellow, and light pink, flowing from the bottom left towards the top right of the slide.

What is a shell Script? **Running it**

```
% cat > hello.sh <<MY_PROGRAM
#!/bin/sh
echo 'Hello, world'
MY_PROGRAM
% chmod +x hello.sh
% ./hello.sh
Hello, world
```

A decorative graphic consisting of several overlapping, wavy, translucent bands in shades of orange, yellow, and light green, flowing from the bottom left towards the top right of the slide.

Remember: The Program PATH

- `$ zd`
- `$ -bash: zd: command not found`
- `echo` vs. `/bin/echo`
- `$ echo $PATH`
`/usr/lib64/qt-3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr/local/trinityrnaseq_r2013_08_14:/usr/local/velvet_1.2.08:/usr/local/MUMmer3.23:/usr/local/fasta-36.3.5e/bin:/usr/local/clustalw-2.1:/usr/local/bowtie2-2.0.5:/usr/lib64/openmpi/bin:/usr/local/lib:/usr/local/FastX_ToolKit_0.0.13:/usr/local/ncbi-blast-2.2.27+/bin:/usr/local/stacks-0.99994/scripts:/usr/local/prinseq-lite-0.20.1:/usr/local/bin/fastx:/usr/local/bin/fastqc:/usr/local/bin/oases:/usr/local/bin/cmpfastq:/usr/local/bin/picard:/usr/local/bin/soap-2.04:/usr/local/expr4.0:/home/cleslin/bin`
- `$ which echo`
`/bin/echo`

On fisher

Variables

- Variables are symbolic names that represent values stored in memory
- Three types of variables:
 - **Configuration variables** store information about the setup of the OS
 - **Environment variables** hold information about your login session
 - **shell variables** are created at the command prompt or in shell scripts and are used to temporarily store information
 - You can **define** and **manipulate** for use with program commands in a shell
 - Observe basic guidelines for handling and naming shell variables
 - I recommend that you use all UPPERCASE characters when naming your variables

Variables and the Environment

```
$ hello.sh  
-bash: hello.sh: command not found  
$ PATH="$PATH:."  
$ hello.sh  
Hello, world
```

So what did we change here?

A decorative graphic consisting of several overlapping, wavy bands of color in shades of orange, yellow, and light pink, flowing from the bottom left towards the top right of the slide.

Continuing Lines: \

```
$ echo This \  
Is \  
  A \  
Very \  
Long \  
  Command Line
```

```
This Is A Very Long Command Line
```

I told you we come back to the \

exit Status

- Every command (program) has a value or `exit` status which it returns to the calling program
 - Separate from any output generated
 - Can be explicitly set using `exit N`, or defaults to the value of the last command run
- `$?` inside a script
- 0 is True, anything else false


```
$ df -x
df: illegal option - x
$ echo $?
64
$ echo $?
0
```

OS dependent, try it on your Mac,
and see what the return value is. Is it the same?

<http://www.faqs.org/docs/abs/HTML/exit-status.html>

Exit Status: Setting the Status of `exit`

```
$ cat > test.sh <<_TEST_  
exit 3  
_TEST_  
$ chmod +x test.sh  
$ ./test.sh  
$ echo $?  
3
```

A decorative graphic consisting of several overlapping, wavy, translucent bands in shades of orange, yellow, and light green, flowing from the bottom left towards the top right of the slide.

Passing Arguments

```
$ cat > test.sh <<_TEST_  
echo "Your name is \"$1 \"$2"  
_TEST_
```

```
$ chmod +x test.sh  
$ ./test.sh Chesley Leslin ignore-this  
Your name is Chesley Leslin
```

Logic: test

- test - check file types and compare values

```
$ test 1 -lt 10
```

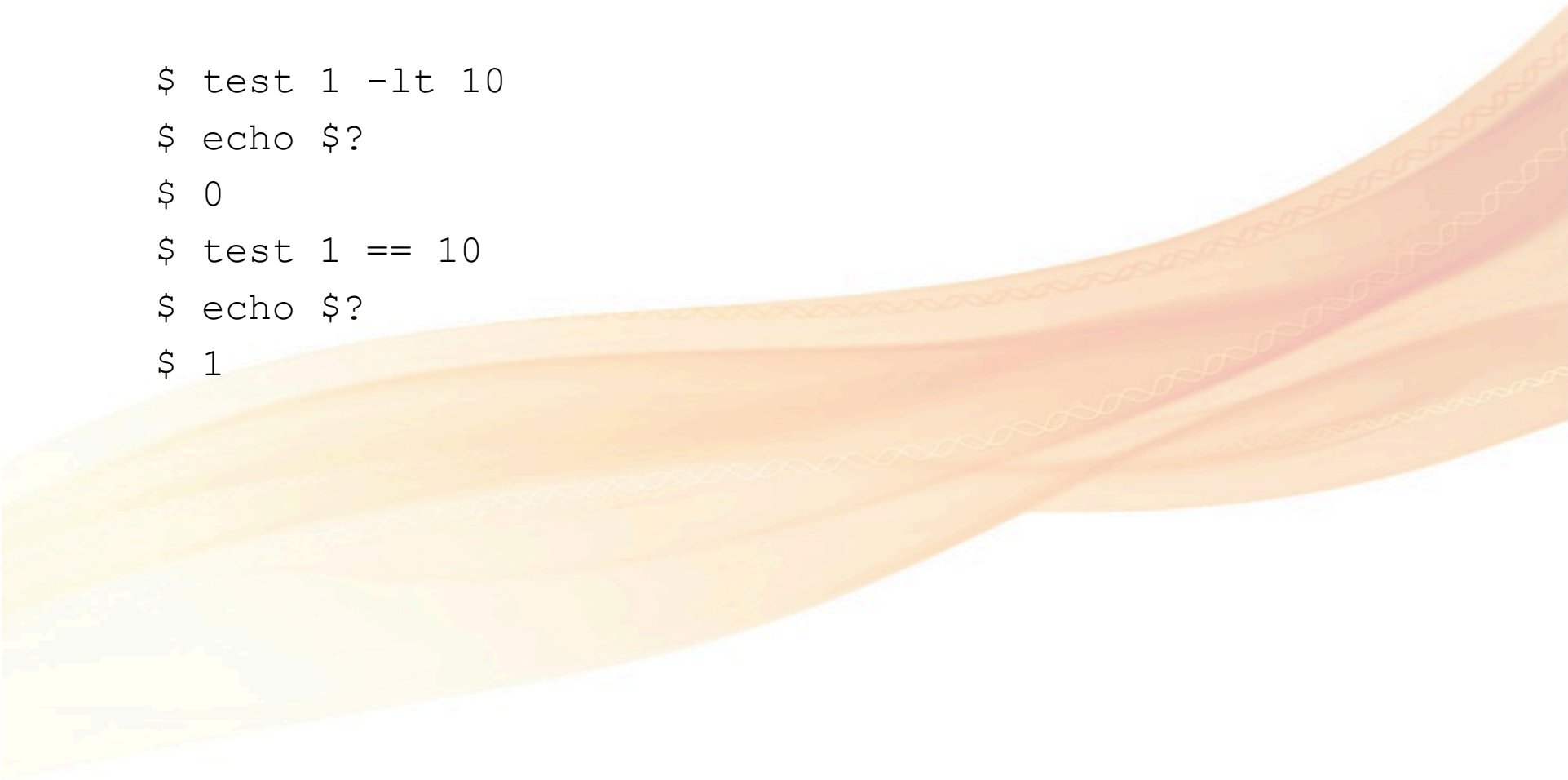
```
$ echo $?
```

```
$ 0
```

```
$ test 1 == 10
```

```
$ echo $?
```

```
$ 1
```

A decorative graphic consisting of several overlapping, wavy, horizontal bands in shades of orange, yellow, and light green, creating a sense of motion or a stylized landscape. The bands are slightly curved and have a soft, painterly texture.

Logic: test Inside a shell Script

- `test` is built into the shell
- `[]` - How we test a condition - comparison expressions or file tests, this is the built in test

```
[ 1 -lt 10 ] #1 < 10
```

```
[ 5 -gt 1 ] # 5 > 1
```

```
[ -e file] #does a file exist
```

- `(())` - construct evaluates and tests numerical expressions

```
(( 1 < 10 ))
```

```
(( 20 >= 10 ))
```

http://bash.cyberciti.biz/guide/Test_command

http://en.wikipedia.org/wiki/Test_%28Unix%29

shell - Special Variables

Variable	Description
\$0	The filename of the current script.
\$n	These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is \$1, the second argument is \$2, and so on).
\$#	The number of arguments supplied to a script.
\$@	All the arguments are individually double quoted. If a script receives two arguments, \$@ is equivalent to \$1 \$2.
\$?	The exit status of the last command executed.
\$\$	The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
#!	The process number of the last background command.

Simplest shell Scripts

- The most basic shell script is a list of commands exactly as could be typed interactively, prefaced by the # !
- All the parsing rules, filename wildcards, \$PATH searches etc., which were summarized thus far, apply

```
#!/bin/sh
```

```
echo Disk usage summary for $USER on `date`
```

```
echo These are my files
```

```
# List the files in columns
```

```
ls -C
```

```
# Summarize the disk usage
```

```
echo
```

```
echo Disk space usage for /home/$USER/MUSCLE
```

```
du -s /home/$USER/MUSCLE
```

```
exit 0
```

/data/METHODS/Fall/test/test1.sh

shell Scripting: while Loops

- Logic: while
while something
do

... .

done

- *sh* example:

```
#!/bin/sh
```

```
a=0; LIMIT=10
```

```
while [ $a -lt $LIMIT ] #while loop
```

```
do
```

```
    echo -n "$a "
```

```
    a=$(( a + 1 ))
```

```
done
```

Here's a way to add, but
we'll see a better way,
This only works for integers

what does the -n flag do in the
echo call?

shell Scripting: `for` loops

- These are useful when you want to run program in sequence with different filenames:
- Logic: while

```
for i in 1 2 3
do
    echo $i
done
```

- *sh* example:

```
#!/bin/sh
for VAR in /data/METHODS/Fall/test/project1.sh /data/METHODS/Fall/test/project2.sh; #for loop
do
    wc $VAR #show the word count
done
```

test2.sh

Counters

```
#!/bin/sh
cd /data/METHODS/Fall/test/files
COUNTER=2
FILE='test'
while [ -e "$FILE.$COUNTER.txt" ]
do
    ls "$FILE.$COUNTER.txt"
    COUNTER=`expr $COUNTER + 1` ##add 1
done
```

← check if file
exists

Email Notification

- Sometimes, when a program is done running it might be useful to send an email so you know to check up on your output
- Depending on the Linux setup, this is simple:

```
#!/bin/sh  
echo "Your Blast Run is complete" | \  
mail -s "Put Your Subject Here" \  
myEmailHere@neu.edu
```

Try this on your own Mac or Linux Box, but use your own email!

mail is not setup on fisher, so this will not work

Process Management - Linux Job Control

- Start a background process:

```
$ ./sleep.sh &  
    or, you can do this  
$ ./sleep.sh  
Hit CTRL-Z  
bg
```

- Where did it go?

```
$ jobs  
[1]-  Running                  ./sleep.sh &  
[2]+  Running                  ./sleep.sh &
```

```
ps waux | grep 'cleslin' #check programs running
```

- Terminate the job: kill it

```
kill %jobid  
kill pid  
killall The program is specified by command name
```

sleep.sh

Process Management - via top

- Process is a unit of running program
- Each process has some information, like process ID, owner, priority, etc

Example: Output of `top` command

```
cleslin — cleslin@fisher:~ — ssh — 80x24
top - 15:08:34 up 25 days, 5:06, 3 users, load average: 0.05, 0.03, 0.00
Tasks: 1271 total, 1 running, 1270 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.1%sy, 0.0%ni, 99.9%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 65925588k total, 53041304k used, 12884284k free, 184252k buffers
Swap: 33030136k total, 216k used, 33029920k free, 50901360k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
26758	kaluziak	20	0	15960	2228	968	S	4.6	0.0	8:17.14	top
28888	cleslin	20	0	15968	2204	952	R	1.0	0.0	0:00.12	top
259	root	20	0	0	0	0	S	0.3	0.0	1:00.38	events/0
2176	root	20	0	0	0	0	S	0.3	0.0	0:35.89	edac-poller
1	root	20	0	19352	1532	1224	S	0.0	0.0	0:07.47	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.78	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:16.89	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	0:23.57	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
6	root	RT	0	0	0	0	S	0.0	0.0	0:01.50	watchdog/0
7	root	RT	0	0	0	0	S	0.0	0.0	0:17.47	migration/1
8	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/1
9	root	20	0	0	0	0	S	0.0	0.0	0:34.19	ksoftirqd/1
10	root	RT	0	0	0	0	S	0.0	0.0	0:01.66	watchdog/1
11	root	RT	0	0	0	0	S	0.0	0.0	0:18.90	migration/2
12	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/2
13	root	20	0	0	0	0	S	0.0	0.0	0:26.21	ksoftirqd/2

Things To Do For Tuesday

- Reading Material
 - Read this, for a better understanding of why we use Perl and not shell scripting
 - http://www.perlmonks.org/?node_id=668481
 - Perl Readings
 - http://155.33.203.128/teaching/BIOL6308-Fall2011/local/Perl_Programming.html
 - [First steps in Perl](#)
 - [Working with Simple Values](#)
- Complete the Unix/Linux Lab 1
- Know the Unix Cheat Sheet
 - <http://155.33.203.128/teaching/BIOL6308-Fall2011/local/unixCheatSheet.html>