



# GW COLONIAL ONE



# HPC Workshop 3

---

What we're covering:

- Simple bash scripting
- Virtualenv Python Demo
- Install R modules in your home directory
- Parallel module
- Installing software in your home directory

Files for today's workshop are located in:

`/groups/hpcworkshop/ws3`



# Bash/Shell Scripting

---

**Overview:** A shell script is a computer program designed to be run by the Unix/Linux shell, a command-line interpreter.

- Shell scripts are just executable text files
- Use shell scripts to perform repetitive tasks
- Shell scripts can include any command you run on the command line
- Scripting allows you to use programming functions – such as ‘for’ loops, if/then/else statements, etc.
- You can use command substitution in a shell script, such as using the 'date' command to name a file.



# Bash/Shell Scripting

**Create a simple shell script file to make a directory for your today's workshop and copy the workshop files into the directory:**

- `$ cd ~`
- `$ nano workshop_copy.sh`
- Type:

```
cd ~
mkdir ~/hpcworkshop_3
cp /groups/hpcworkshop/ws3/* ~/hpcworkshop_3
```

- **Make your script executable:** `chmod +x workshop_copy.sh`
- **Execute the script:** `./workshop_copy.sh`
- **Is the new directory to see the files:** `ls /home/<username>/hpcworkshop_3`



# Bash/Shell Scripting

---

## Simple Addition Script

Text is located in ~/hpcworkshop\_3/add.sh

```
#!/bin/bash
sum=$(( $1 + $1 ))
echo $sum
```



# Bash/Shell Scripting

## For Loop Example

Text is located in ~/hpcworkshop\_3/for\_loop.sh

```
#!/bin/bash
for i in 1 2 3 4 5
do
    echo "Welcome $i times"
done
```

Make your script executable, then run it:

```
$ chmod +x for_loop.sh
./for_loop.sh
```



# Bash/Shell Scripting

## For Loop Example Continued ... looping over an array

```
LIST=(`ls /home/$USER`)  
for NAME in ${LIST[@]}; do echo $NAME ; done  
for NAME in ${LIST[@]}; do echo $NAME | tr '[a-z]' '[A-Z]'; done
```

## For More Complicated Loop Example

```
LIST=(`ls /home/$USER/hpcworkshop_3/*sh`)  
for NAME in ${LIST[@]}; do echo $NAME | awk '{split($0,a,"\\.sh") ;print  
a[1]".bash"}' ; done
```



# Bash/Shell Scripting

**Shell scripts that call modules should be loaded with the "source" command:**

Text is located in `~/hpcworkshop_3/load_python.sh`

- `nano loadpython.sh`
- Type:

```
#!/bin/bash
module load python/3.4.2
python
```

- **Make your script executable:** `chmod +x loadpython.sh`
- **Execute module script with the "source" command:** `source loadpython.sh`
- Type `quit()` to exit the Python command line





# Bash/Shell Scripting

**You can use shell scripts to load modules for specific tasks, like compiling a large program**

Text is located in `~/hpcworkshop_3/load_paraview.sh`

To load the Paraview build environment:

```
#!/bin/bash
module load openmpi/current
module load python/2.7.6
module load cmake/3.3.1
module load openblas/openblas
module load qt
```

Save the text in a file and you can load all those modules with one command:

```
$ source load_paraview.sh # source for persistence
```



# Bash/Shell Scripting

---

## Questions & Discussion



# Build your own modules

You can build your own modulefiles if you want to customize environment variables.

Create a folder called "modulefiles" in your home directory:

```
$ cd ~  
$ mkdir modulefiles  
$ cp ~/hpcworkshop_3/mypython ~/modulefiles/  
$ cd ~/modulefiles/
```

Look at the text file for your new module:

```
$ nano mypython
```

# Build your own modules

The modulefile:

```
##Module1.0#####  
  
module-whatismodule "Adds python 2.7.5 to your environment"  
  
setroot /c1/apps/python/2.7.5  
prepend-pathC_INCLUDE_PATH $root/include  
prepend-pathLD_LIBRARY_PATH $root/lib  
prepend-pathLIBRARY_PATH $root/lib  
prepend-pathPATH $root/bin
```

CTRL+o to save  
CTRL+x to exit



# Build your own modules

---

Configure the module system to use your new module directory:

```
$ module use /home/<username>/modulefiles (tab complete does not work)
```

List available modules:

```
$ module avail
```

Load your new module:

```
$ module load mypython
```

Confirm that your module loaded:

```
$ module list
```



# The parallel module

---

Parallel is a module that lets you run a single process across multiple cores or CPUs.

Basic usage:

```
parallel [options] [command [arguments]] < list_of_arguments
```

```
parallel [options] [command [arguments]] (::: arguments|::: argfile(s))...
```

The command can be one command or a script you've written. Parallel will launch concurrent instances of the command.

Parallel is a module:

```
$ module load parallel
```



# The parallel module

---

Parallel will automatically try to use as many threads as necessary based on the amount of input arguments:

```
$ parallel echo ::: A B C D E
```

A

B

C

D

E



# The parallel module

You can specify the total number of processes (or job slots) to use at the same time with the `-j` option. the `{%}` string represents the Job slot.

```
$ parallel -j 2 echo {%} {} ::: A B C D E
```

```
1 A
```

```
2 B
```

```
1 C
```

```
2 D
```

```
1 E
```





# The parallel module

Parallel will index the total number of processes run. The `{#}` string represents the job number.

```
$ parallel -j 2 echo {#} {} ::: A B C D E
```

```
1 A
```

```
2 B
```

```
3 C
```

```
4 D
```

```
5 E
```



# The parallel module

---

Run parallel with your add.sh script:

```
$ parallel /home/username/add.sh ::: 42 34 99 180
```

84

68

198

360



# The parallel module

---

Serial Fibonacci number example. Use the `fibonacci_serial` program to find specific numbers in a Fibonacci sequence:

```
$ parallel ~/hpcworkshop_3/fibonacci_serial ::: 42 43 44 45
```

Result:

```
42 267914296
43 433494437
44 701408733
45 1134903170
```



# The parallel module

---

You can use a text file with the same arguments instead of typing them out on the command line:

```
$ nano input.txt
```

```
Type:
```

```
42
```

```
43
```

```
44
```

```
45
```

```
$ parallel ~/hpcworkshop_3/fibonacci_serial < input.txt
```

```
Result:
```

```
42 267914296
```

```
43 433494437
```

```
44 701408733
```

```
45 1134903170
```



# The parallel module

---

If you wish to learn more about parallel, there is a massive tutorial located here:

[https://www.gnu.org/software/parallel/parallel\\_tutorial.html](https://www.gnu.org/software/parallel/parallel_tutorial.html)



# Installing your own software

---

## You can install software in your home directory on Colonial One

- Users do not have root on Colonial One, so software with many dependences should be installed by the C1 team
- You can compile programs directly in your home directory
- You can also download precompiled utilities and run them from your home directory



# Installing your own software

## Figlet Example

Text is located in `~/hpcworkshop_3/figlet_example`

- `$ cd ~`
- `$ mkdir figlet; cd figlet`
- `$ wget ftp://ftp.figlet.org/pub/figlet/program/unix/figlet-2.2.5.tar.gz`
- `$ tar -xvf figlet-2.2.5.tar.gz`
- `$ cd figlet-2.2.5`
- `$ nano Makefile`
  - `prefix = /home/<username>/figlet`
  - `DEFAULTFONTDIR = /home/<username>/figlet/figlet-2.2.5/fonts`
- `$ make all`
- `./figlet` (test out your program!)
- Type something and hit enter
- CTRL+C to exit



# Questions?

---

Please fill out the survey for the workshop. Thank you!

<https://goo.gl/Gpkvua>