THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

CCAS and SEAS Computing Facility

HPC Workshop 3

For new Colonial One users!!!

Login to Colonial One using your GW NetID/email address & password

```
 $ ssh <netid>@login.colonialone.gwu.edu

cd ~
cd skel
cp .bash* ../
cd ~
```

HPC Workshop 3

What we're covering:

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

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, while loops, 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:

- 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>/workshop_3`

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
```

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

If Then Example

Text is located in ~/hpcworkshop_3/if_then.sh

```
#!/bin/bash
T1=$1
T2=$2
if [ "$T1" = "$T2" ]; then
  echo expression evaluated as true
else
  echo expression evaluated as false
fi
```

Make your script executable, then run it:

```
chmod +x if_then.sh
usage: ./if_then <argument> <argument>
```

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`

Your own Python Environment

Create your own Python environment with Virtualenv

- Regular Colonial One users do not have permission to alter the system-wide Python installations
- Virtualenv creates your own custom python build inside a directory you choose
- You can install additional python libraries without going through the C1 team
- You can build multiple Python environments for different projects, including different versions of the Python executable and different libraries for each project

Your own Python Environment

Virtualenv Example

Text is located in ~/hpcworkshop_3/if_then.sh

```
1.  cd ~
2.  mkdir my_virtual_env
3.  cd my_virtual_env
4.  module load python/3.4.2 < --- The environment will build
    based off the version of Python you've loaded
5.  virtualenv test_env
6.  source ~/my_virtual_env/test_env/bin/activate
7.  pip install numpy, scipy, times, etc.
```

You can unload the virtual environment by deactivating it:

```
deactivate
```

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
```

Create a text file for your new module:

```
nano mypython
```

Build your own modules

Copy and paste this text into your document:

```
#%Module1.0#################################################

module-whatis   "Adds python 2.7.5 to your environment"

set             root            /c1/apps/python/2.7.5
prepend-path    INCLUDE         $root/include
prepend-path    LD_LIBRARY_PATH $root/lib
prepend-path    LIBRARY_PATH    $root/lib
prepend-path    PATH            $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
```

List available modules:

```
module avail
```

Load your new module:

```
module load mypython
```

Confirm that your module loaded:

```
module list
```

Your own R libraries

Installing R libraries in your home directory is very simple - you create a directory, set an environment variable, and that's it!

```
cd ~; mkdir R_libs
export R_LIBS="/home/<username>/R_libs"
module load R
R
install.packages('sqldf',repos="http://cran.r-project.org")
quit()
```

Check your R_libs directory: `ls ~/R_libs`

To make the change permanent, add the export command to your .bashrc file:

```
echo export R_LIBS="/home/<username>/R_libs" >> ~/.bashrc
```

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.

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 to use at the same with 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 total process number.

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

```
1 A
2 B
3 C
4 D
5 E
```

The parallel module

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

```
parallel /groups/hpcworkshop/ws3/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
```

Questions?

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

https://goo.gl/HiGqZC