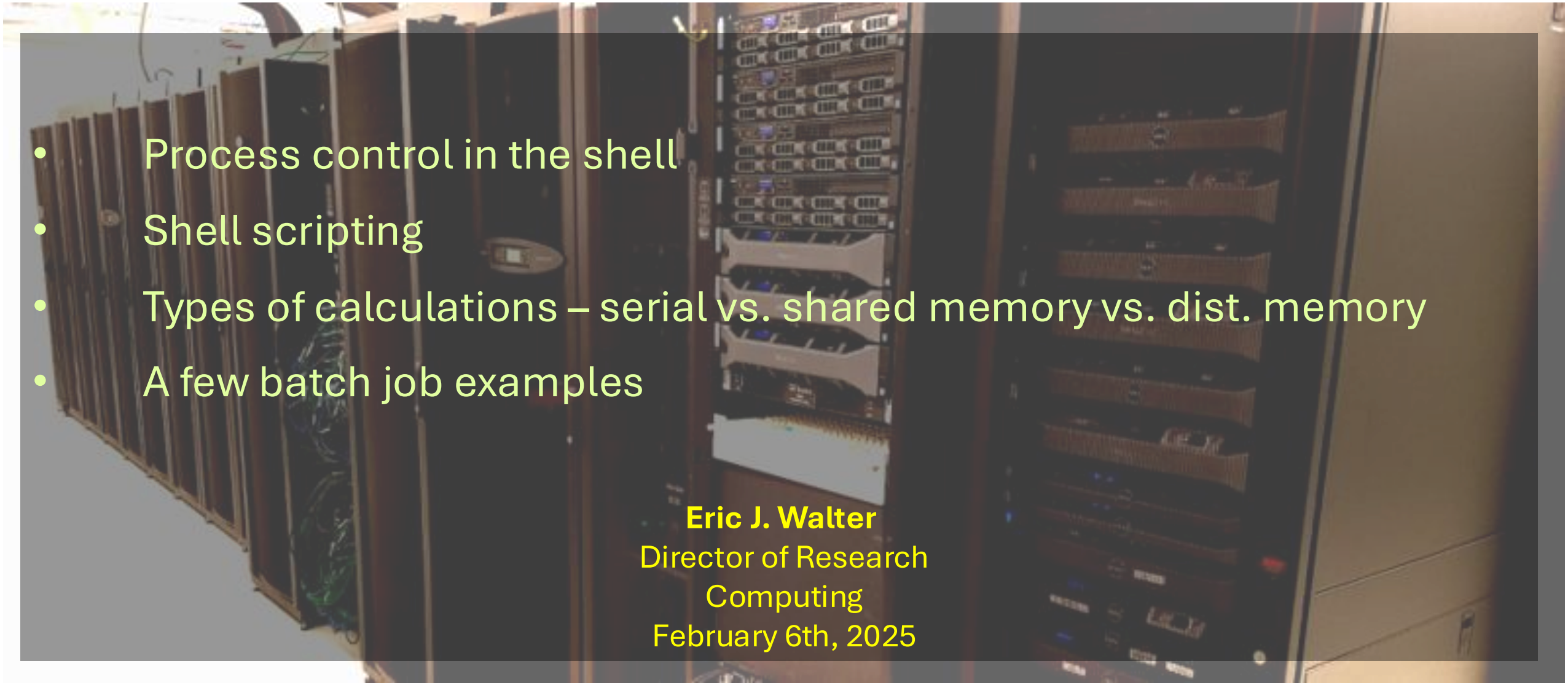
Using Computational Resources at W&M/VIMS



- Process control in the shell
- Shell scripting
- Types of calculations – serial vs. shared memory vs. dist. memory
- A few batch job examples

Eric J. Walter
Director of Research
Computing
February 6th, 2025

Process Control

control key

Sometimes, you will run a command and it takes too long / you want to kill it for some reason: ^c
Sometimes (interactive job session) you want to run a calculation in the "background": &, jobs, kill

```
1 [bora] find . -name hello &
[1] 14449

2 [bora] jobs
[1]+  Running                 find . -name hello &

3 [bora] kill %1

4 [bora]
[1]+  Terminated              find . -name hello
```

& - puts the calculation in the background
jobs – lists current running "jobs"
kill – kills the job (%1)
^c – cancel
^z – background

Occasionally, you must use "kill –9" (always try without –9 first!)

Process Control – cont. 2

You can also move a background job to the foreground and vice/versa:

```
1 [bora] find . -name hello
^z
[1]+  Stopped                 find . -name hello

2 [bora] bg
[1]+ find . -name hello &

3 [bora] jobs
[1]+  Running                 find . -name hello &

4 [bora] fg %1
find . -name hello
^c
```

Process Control – cont. 3

"job" control with %n ***only works for the shell that launched the process***. Any other shell need to use ***ps***.

`ps` **– lists the current processes**

```
ps -ef            (all proceses)
ps -fu ewalter    (processes for ewalter)
```

Can kill using process id:

```
1 [bora] ps -fu ewalter
UID          PID  PPID  C STIME TTY          TIME CMD
ewalter    20926 35637 23 19:46 pts/0    00:00:00 find . -name hello
ewalter    21014 35637  0 19:46 pts/0    00:00:00 ps -fu ewalter
ewalter    35636 35633  0 19:10 ?        00:00:00 sshd: ewalter@pts/0
ewalter    35637 35636  0 19:10 pts/0    00:00:00 -bash

2 [bora] kill 20926
```

`top` – shows you the current running processes on a computer interactively

Top

```
[2 ewalter@gt02 ~ ]$top

top - 09:58:42 up 107 days, 17 min,  1 user,  load average: 63.13, 62.95, 62.08
Tasks: 1666 total,   4 running, 1662 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.9 us,  1.0 sy,  0.0 ni, 95.4 id,  1.5 wa,  1.0 hi,  0.3 si,  0.0 st
MiB Mem : 515267.2 total, 364935.6 free,  38384.2 used, 115599.0 buff/cache
MiB Swap:   4096.0 total,   3947.2 free,    148.8 used. 476883.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
1910952 jrcalza+  20   0   23.3g   5.9g 185852 S  85.7   1.2   1607:43 python
1923561 ewalter   20   0   12316   5776   3316 R  19.0   0.0   0:00.09 top
1910962 jrcalza+  20   0  835612 301864 132280 S   4.8   0.1  15:41.49 python
1910974 jrcalza+  20   0  835612 301612 132044 S   4.8   0.1  16:34.16 python
1911020 jrcalza+  20   0  835616 303768 131944 S   4.8   0.1  16:17.75 python
1911029 jrcalza+  20   0  835616 303428 131628 R   4.8   0.1  15:35.23 python
1911031 jrcalza+  20   0  836636 300792 130220 S   4.8   0.1  15:51.29 python
1911032 jrcalza+  20   0  835740 304380 132168 S   4.8   0.1  14:09.83 python
1911040 jrcalza+  20   0  835744 301800 132056 S   4.8   0.1  14:08.38 python
1911051 jrcalza+  20   0  835616 304376 131864 S   4.8   0.1  11:06.48 python
1911062 jrcalza+  20   0  835612 302044 132028 S   4.8   0.1  16:33.09 python
1911064 jrcalza+  20   0  835612 302268 132696 S   4.8   0.1  14:36.01 python
1911069 jrcalza+  20   0  835616 304044 132288 S   4.8   0.1  16:13.66 python
1911077 jrcalza+  20   0  836640 301824 131140 S   4.8   0.1  15:56.05 python
      1 root      20   0  173776  10596   6360 S   0.0   0.0   6:44.94 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   1:23.19 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_flushwq
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-ev
     12 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
     13 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_kthre
     14 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_rude_
     15 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_trace
     16 root      20   0       0      0      0 S   0.0   0.0   0:44.87 ksoftirqd/0
     17 root      20   0       0      0      0 S   0.0   0.0   3:43.97 pr/ttyS0
     18 root      20   0       0      0      0 I   0.0   0.0  80:01.71 rcu_preempt
     20 root      rt   0       0      0      0 S   0.0   0.0   0:29.78 migration/0
     21 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/0
[3 ewalter@gt02 ~ ]$
```

top shows you the statuses of the processes running on the server.

PID – process id
USER – who owns process
VIRT – How much memory reserved for process
RES – How much memory the process is currently using

S – status column
R – Running
S – Sleeping
D – Disk activity
I – idle
Z – Zombie

%CPU – percent of one core it is using (can be > 100%)
%MEM – percent of the total system MEM in use
TIME - how much CPU time process has used
COMMAND – name of executable

Can use top to see what is happening with your job

Shell Scripting

- ***Shell scripting*** is essential to utilize Linux environment efficiently
- ***tcsh*** and ***bash*** are two different shell flavors
- All HPC users default to ***tcsh***
- **#** - comment character
- **#!** - NOT comment if on first line
- ***Linux is case sensitive***

```
#!/bin/tcsh
#  comment :   #! Is a script shell interpreter ("do this with tcsh syntax")
foreach i (1 2 3 4 5)
      echo $i
end
```

```
[68 ewalter@bora ~ ]$ chmod u+x test.csh
```

Change permissions to executable

```
[69 ewalter@bora ~ ]$./test.csh
1
2
3
4
5
```

. is not in your PATH, must add it explicitly

Shell Scripting – cont.

Say you have a file of all the inputs you want to run:

```
1 [bora] cat joblist
1.45
1.44
1.40
1.33
1.10
```

Say you run the job like this:

```
1 [bora] ./a.out -i 1.45

2 [bora] cat runjobs
#!/bin/tcsh
foreach i (`cat joblist`)
        echo $i
       ./a.out -i $i
end

3 [bora] chmod u+x runjobs
4 [bora] ./runjobs
```

`` (backtick) means "the result of this command"

***./runjobs* will run all parameters in joblist file**

Shell Scripting – cont. 2

Main tcsh constructs:

foreach loop

```
#!/bin/tcsh
foreach i (`cat joblist`)
        echo $i
       ./a.out -i $i
end
```

while loop

```
#!/bin/tcsh
set i = 1
while ($i < 5)
   echo "i is $i"
   @ i++
end
```

If-then-else

```
#!/bin/tcsh
set scen = b
if ($scen == a) then
  echo $scen
else
  echo "wrong value"
endif
```

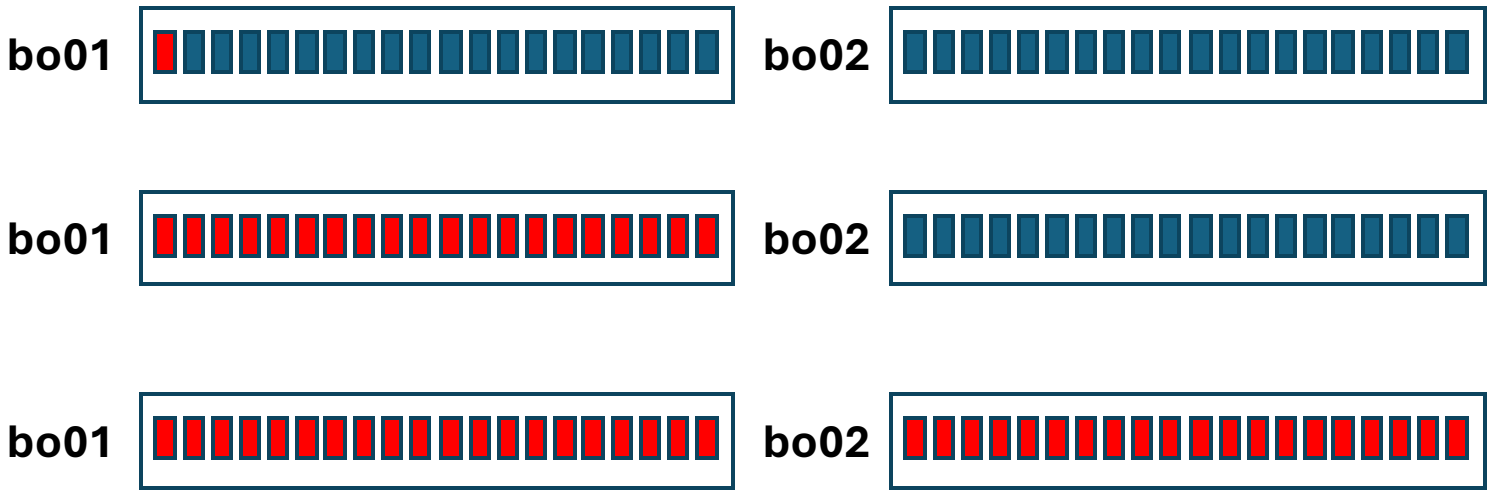
switch/case statement

```
switch ($myvar)
case 'foo':
     ./foo.csh
     breaksw
case 'bar':
     ./bar.csh
     breaksw
default:
     echo $usage
     breaksw
endsw
```

Even though you have a tcsh environment, you can still use bash shell scripts
Bash is considered more powerful for scripting / sometimes easier

Serial vs. Shared Memory vs. Distributed Memory

Two bora nodes



serial – only one core used

shared memory parallel –
one node, multi-core

distributed parallel –
multi-node, multi-core

How parallel the calculation can run is application dependent
How many cores a parallel application can use efficiently also varies
Try test calculations if not sure.

Job Memory Requirements

sacct - SLURM command to look at past jobs

```
[225 ewalter@bora ~ ]$sacct -o "JobID,Start,End,JobName,User,NodeList,Elapsed,CPUTime,State,AllocTRES%-90" -S 1/27/25 -X
JobID                    Start                 End    JobName      User        NodeList    Elapsed    CPUTime      State AllocTRES
------------ ------------------- ------------------- ---------- --------- --------------- ---------- ---------- ---------- ------------------------------------------
17383        2025-01-27T18:50:27 2025-01-27T19:01:43   get_stl2   ewalter       bo[15-17]   00:11:16   01:52:40 CANCELLED+ billing=10,cpu=10,mem=63000M,node=3
18847        2025-02-04T08:46:02 2025-02-04T08:46:14 interacti+   ewalter            bo03   00:00:12   00:00:24  COMPLETED billing=2,cpu=2,mem=12600M,node=1
18953        2025-02-04T18:34:05 2025-02-04T18:36:13 interacti+   ewalter            bo03   00:02:08   00:42:40     FAILED billing=20,cpu=20,mem=126000M,node=1
[226 ewalter@bora ~ ]$
```

seff - SLURM command to look efficiency of past job

```
[226 ewalter@bora ~ ]$seff 17383
Job ID: 17383
Cluster: bora
User/Group: ewalter/hpcf
State: CANCELLED (exit code 0)
Nodes: 3
Cores per node: 3
CPU Utilized: 00:26:28
CPU Efficiency: 23.49% of 01:52:40 core-walltime
Job Wall-clock time: 00:11:16
Memory Utilized: 49.22 GB
Memory Efficiency: 80.00% of 61.52 GB
[227 ewalter@bora ~ ]$
```

Remember man command:

```
>> man sacct
```

```
sacct(1)                                Slurm Commands                                sacct(1)

NAME
       sacct  - displays accounting data for all jobs and job steps in the Slurm job account-
       ing log or Slurm database

SYNOPSIS
       sacct [OPTIONS...]

DESCRIPTION
       Accounting information for jobs invoked with Slurm are either logged in  the  job  ac-
       counting  log  file  or  saved  to the Slurm database, as configured with the Account-
       ingStorageType parameter.

       The sacct command displays job accounting data stored in the job accounting  log  file
       or  Slurm database in a variety of forms for your analysis. The sacct command displays
       information on jobs, job steps, status, and exitcodes by default. You can  tailor  the
       output with the use of the --format= option to specify the fields to be shown.

       Job  records  consist of a primary entry for the job as a whole as well as entries for
       job steps. The Job Launch page has a more detailed description of  each  type  of  job
```

Jupyter Notebook / Interactive job

https://www.wm.edu/offices/it/services/researchcomputing/using/http/

Jupyter Notebook on HPC cluster:

```
[C 2025-02-06 12:24:27.170 ServerApp]

    To access the server, open this file in a browser:
        file:///sciclone/home/walter/.local/share/jupyter/runtime/jpserver-882779-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/tree?token=4979714940caa01cca8be5fcc7c680309f6acd9b560e7f66
        http://127.0.0.1:8888/tree?token=4979714940caa01cca8be5fcc7c680309f6acd9b560e7f66
```

Install jupyter notebook

```
1 [bora] pip install notebook
```

Will warn you that executable is installed in .local/bin folder and that this is not in your PATH.

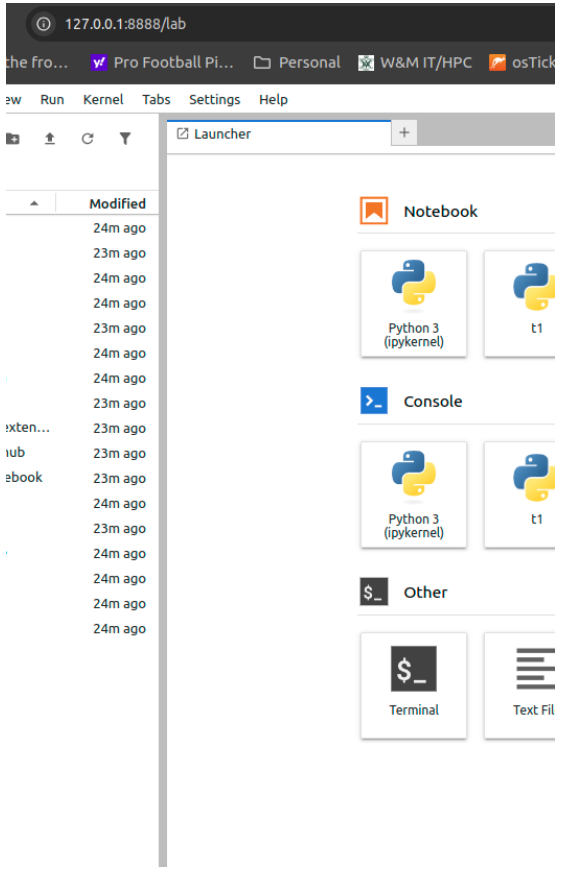
```
1 [bora] salloc -N1 -n1 -t 1:00:00
1 [bo03] cd .local/bin
2 [bo03] ./jupyter-notebook --no-browser --ip=\*
```

Need to tunnel http traffic from node to your local machine
(to use local web browser)

```
1 [my laptop] ssh -NL 8888:bo03:8888 ewalter@bora.sciclone.wm.edu
```

This command will hang until you kill it (`^c`)
Once both the tunnel and jupyter-notebook are started, you can put:
http://127.0.0.1:8888/tree?token=49797149…
into your local web browser

Change ***tree*** to ***lab*** in URL to get modern notebook



Batch job : Python

https://www.wm.edu/offices/it/services/researchcomputing/using/software/python/

Python widely used on HPC systems.

For simple things, you can simply use OS/built-in python
However, we have two modules that you should consider if building a python environment:

Should launch from scratch filesystem, not home directory: pwd = /sciclone/scr10/ewalter/job45

```
24 [bora] cat run
#!/bin/tcsh
#SBATCH --job-name=get_stl2
#SBATCH --nodes=1 --ntasks=1
#SBATCH --time=1:00:00
#SBATCH --gpus=1

module load miniforge3
conda activate testenv
which python

foreach i (`cat list`)
    echo $i > INPUT
    python run.py >& out.$i
end


25 [bora]
```

Python/3.12.7 - standard distribution of v3.12.7 from Python.org
miniforge3/24.9.2-0 - conda/mamba tools for installation

Can use:
python module to build ***virtual environment (venv)***
miniforge3 to build ***conda environments***

Load miniforge3 module
Activate 'testenv' conda environment
which python (will return path of the python executable)

foreach loop
`xxx` means evaluate xxx
Run the run.py script (takes INPUT as a parameter)
Also send the output of each run to out.<parameter>

Submit job with:
1 [bora] sbatch run

```
26 [bora] cat list
1.0
1.1
1.35
2.0
27 [bora]
```

Batch job : Parallel/MPI

srun instead of mpiexec, mpirun etc.
pwd = /sciclone/home/ewalter ; sending outputs to /sciclone/scr10/ewalter/myjoboutput
ckload – used to check that there are no rouge processes on nodes (useful if doing performance tests)
If the test fails, should send email to hpc-help@wm.edu to clean up node/nodes

```
#!/bin/tcsh                                          "hash bang"   which shell syntax to run (here tcsh)
#SBATCH --job-name=femto                             Job Name
#SBATCH --nodes=4 --ntasks-per-node 32               # nodes , # cores per node
#SBATCH --time=1-0                                   walltime (1 day)

module load netcdf-c/intel-2024.0/4.9.2_intelmpi     Load needed software modules
module load netcdf-fortran/intel-2024.0/4.6.1_intelmpi

ckload 0.05                                          Check load :  all load on all my servers should be <0.05

srun myjob.exe >& /sciclone/scr10/ewalter/myjoboutput/outputs
                                                     Run the parallel program with srun (passes topology)
                                                     Also redirect stdout, stderr to a file named "output" in
```

Getting Help

RC/HPC website

USING HPC

- Prerequisites
- Logging in to HPC systems
- Environment modules
- Files & Filesystems
- Running HPC jobs with SLURM
- Compiling & MPI software
- Tutorials
- Software
- Tunneling HTTP for Jupyter

Departments & Offices / ... / Using / Prerequisites

HPC Prerequisites

William & Mary's HPC clusters run on a mixture of Red Hat Enterprise Linux and its derivative CentOS, so you will need basic Unix/Linux knowledge to use the university's HPC systems. If you are unfamiliar with the Unix/Linux command-line, please avail yourself of one or more of the following resources:

- Unix—the Bare Minimum
- UNIX / Linux Tutorial for Beginners
- Writing tcsh shell scripts
- The Linux command line for beginners (Ubuntu focused)

W&M users also have access to many relevant technical e-books through Swem Library, including Unix Power Tools (also available in print), Learning the Unix Operating System (also available in print), Using csh & tcsh, Linux Pocket Guide: Essential Commands, and Unix in a Nutshell.

Text editors

As part of your command-line proficiency, you will want to be familiar with some kind of "plain text" editor. Every W&M HPC login server has at least `vim`, `nano`, and `emacs`, of which `nano` is the easiest for a beginner (but ultimately least powerful). Alternatively, some users prefer to do their editing on their desktop or laptop computers (with the text editor or IDE of their choice), and then use a file transfer utility such as FileZilla, PuTTY, WinSCP, Fetch, rsync, or sftp to copy files to and from the clusters.

HPC webpage: https://www.wm.edu/it/rc

HPC ticket system mail: ***hpc-help@wm.edu***