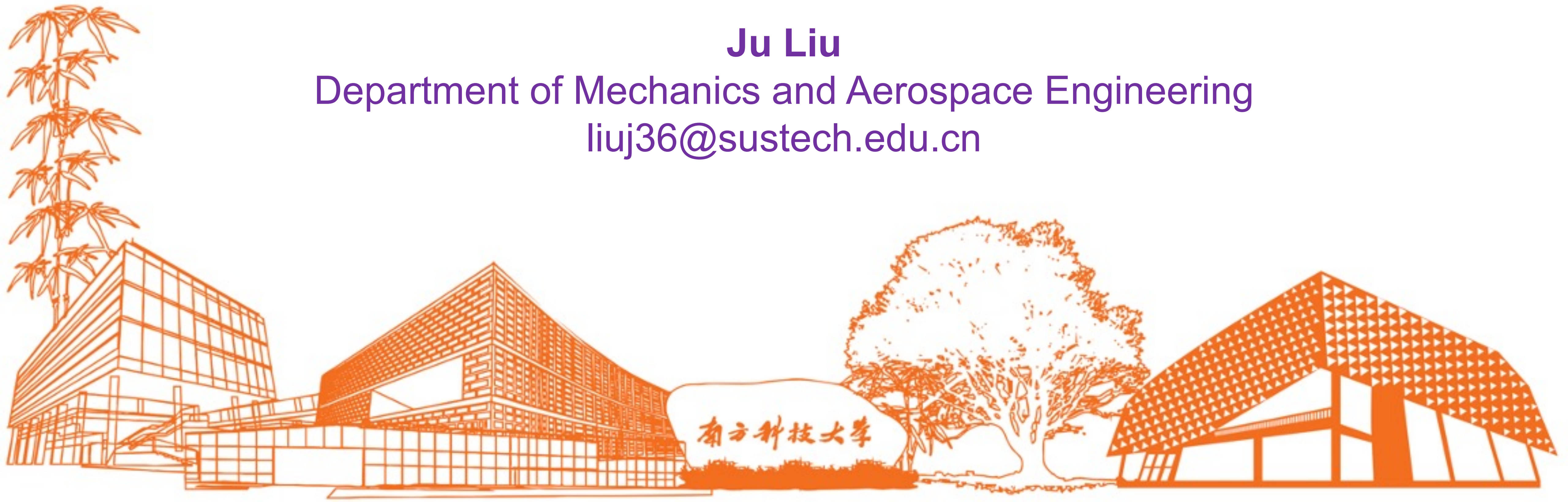
MAE 5032 High Performance Computing: Methods and Applications

Lecture 3: Unix/Linux - part 2

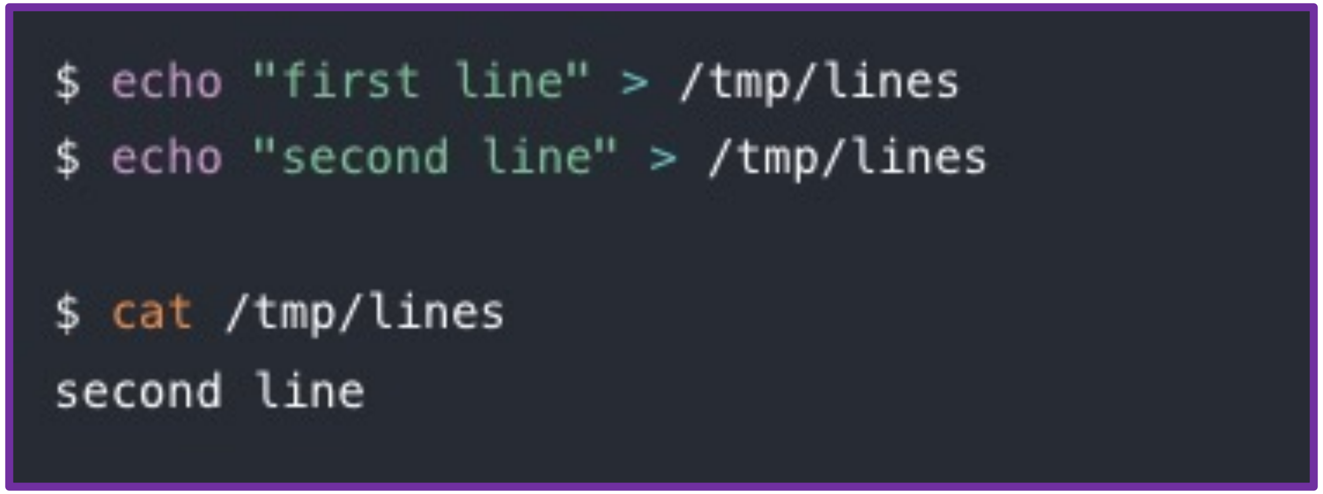
Ju Liu
Department of Mechanics and Aerospace Engineering
liuj36@sustech.edu.cn



1. Basic Commands
2. File attributes and permissions
3. Regular expressions
4. Interacting with the shell
5. Unix pipes
6. Job control
7. Unix environmental variables
8. Text editors
9. Shell scripting
10. Additional topics

Shell Stream Redirection

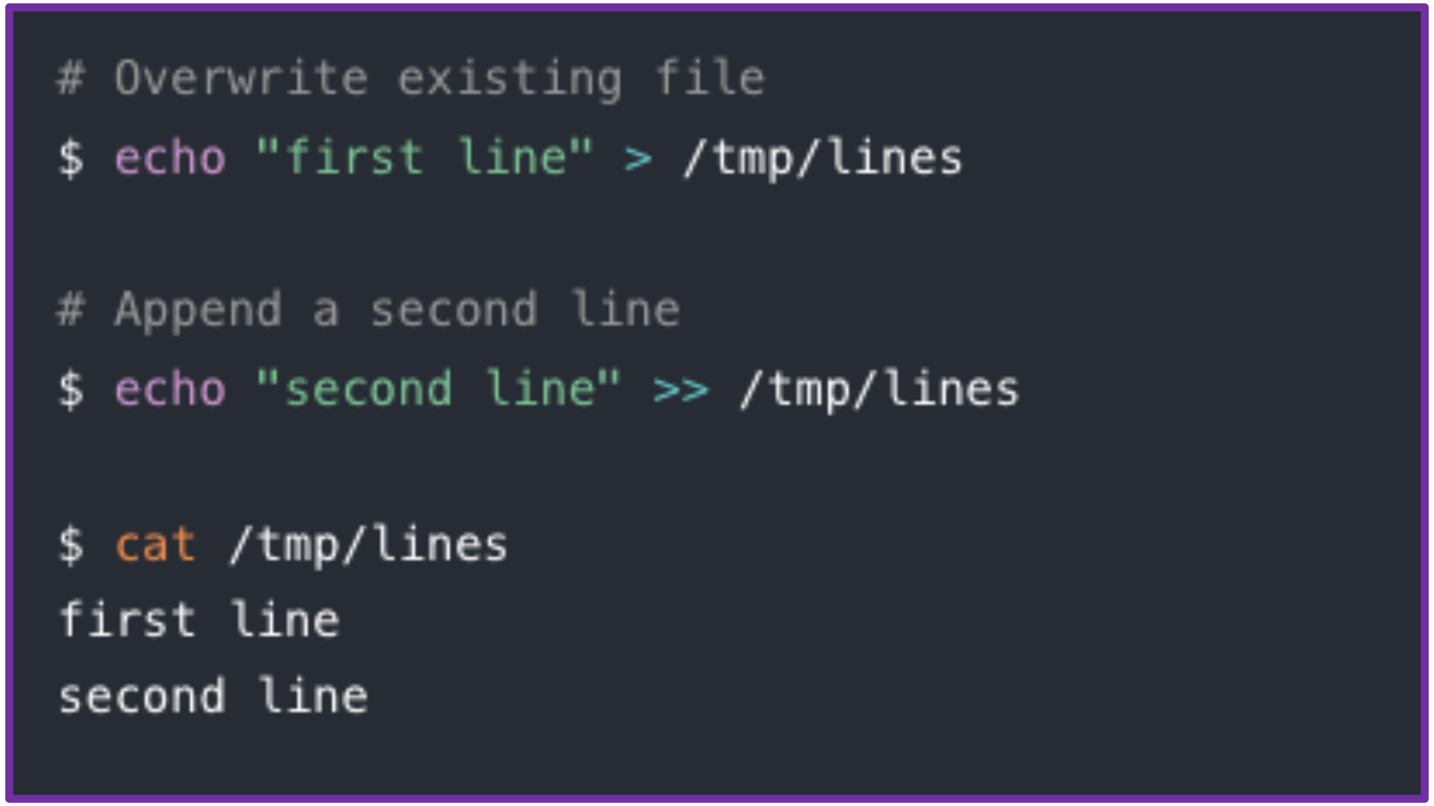
Truncate >:
1. Create specified file if it does not exist;
2. Truncate
3. Write to file



```
$ echo "first line" > /tmp/lines
$ echo "second line" > /tmp/lines

$ cat /tmp/lines
second line
```

Shell Stream Redirection

Append >>:
1. Create specified file if it does not exist;
2. Append file at the end of file

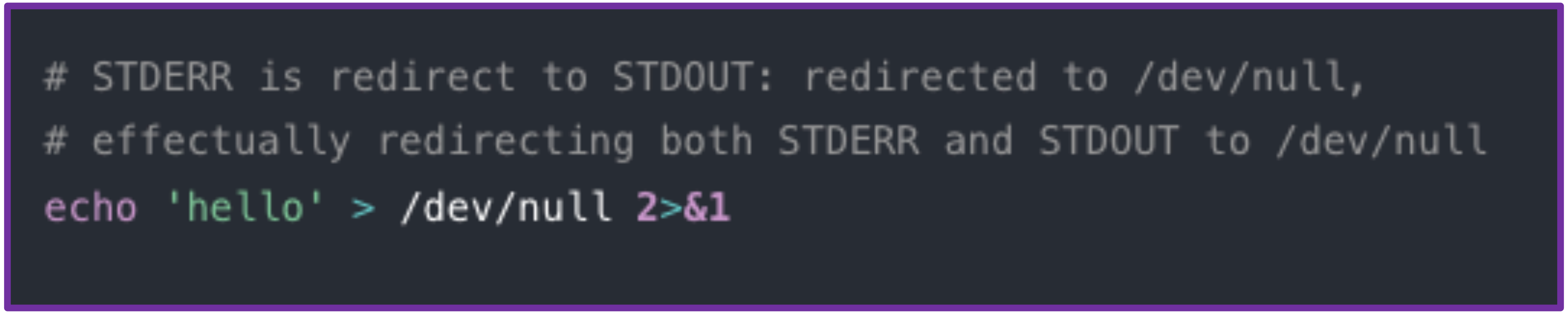


```
# Overwrite existing file
$ echo "first line" > /tmp/lines

# Append a second line
$ echo "second line" >> /tmp/lines

$ cat /tmp/lines
first line
second line
```

Shell Stream Redirection

We may use file descriptors to specify the stream
“>” redirects the stdout
“1>” = “>”
“2>” redirects the stderr
“&>” redirects stdout and stderr
/dev/null is a place of nowhere

Note: File descriptors are associated with each stream
0=STDIN
1=STDOUT
2=STDERR



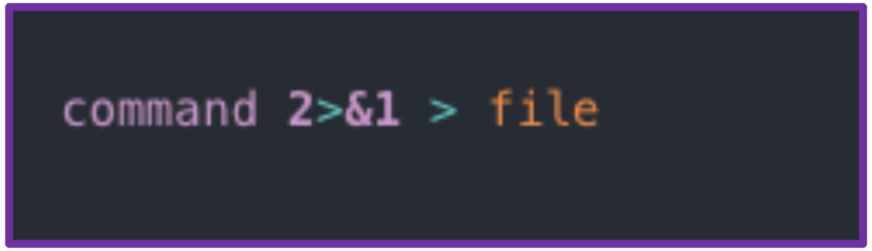
```
# STDERR is redirect to STDOUT: redirected to /dev/null,
# effectually redirecting both STDERR and STDOUT to /dev/null
echo 'hello' > /dev/null 2>&1
```

Shell Stream Redirection

We may use file descriptors to specify the stream
“>”   redirects the stdout
“1>” = “>”
“2>” redirects the stderr
“&>” redirects stdout and stderr
/dev/null is a place of nowhere (for garbage streams)

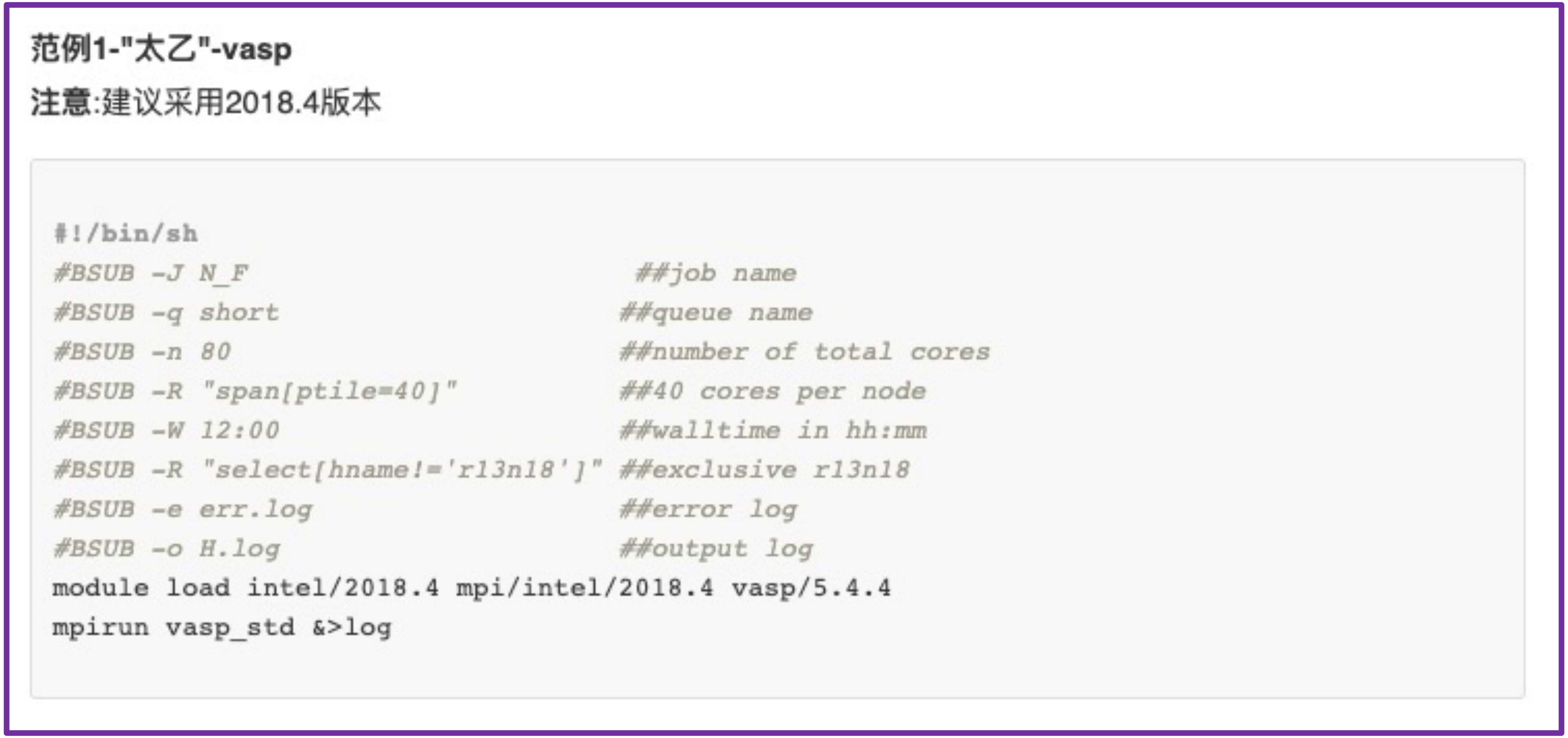
Note: File descriptors are associated with each stream
0=STDIN
1=STDOUT
2=STDERR

What will this do?



```
command 2>&1 > file
```

Shell Stream Redirection

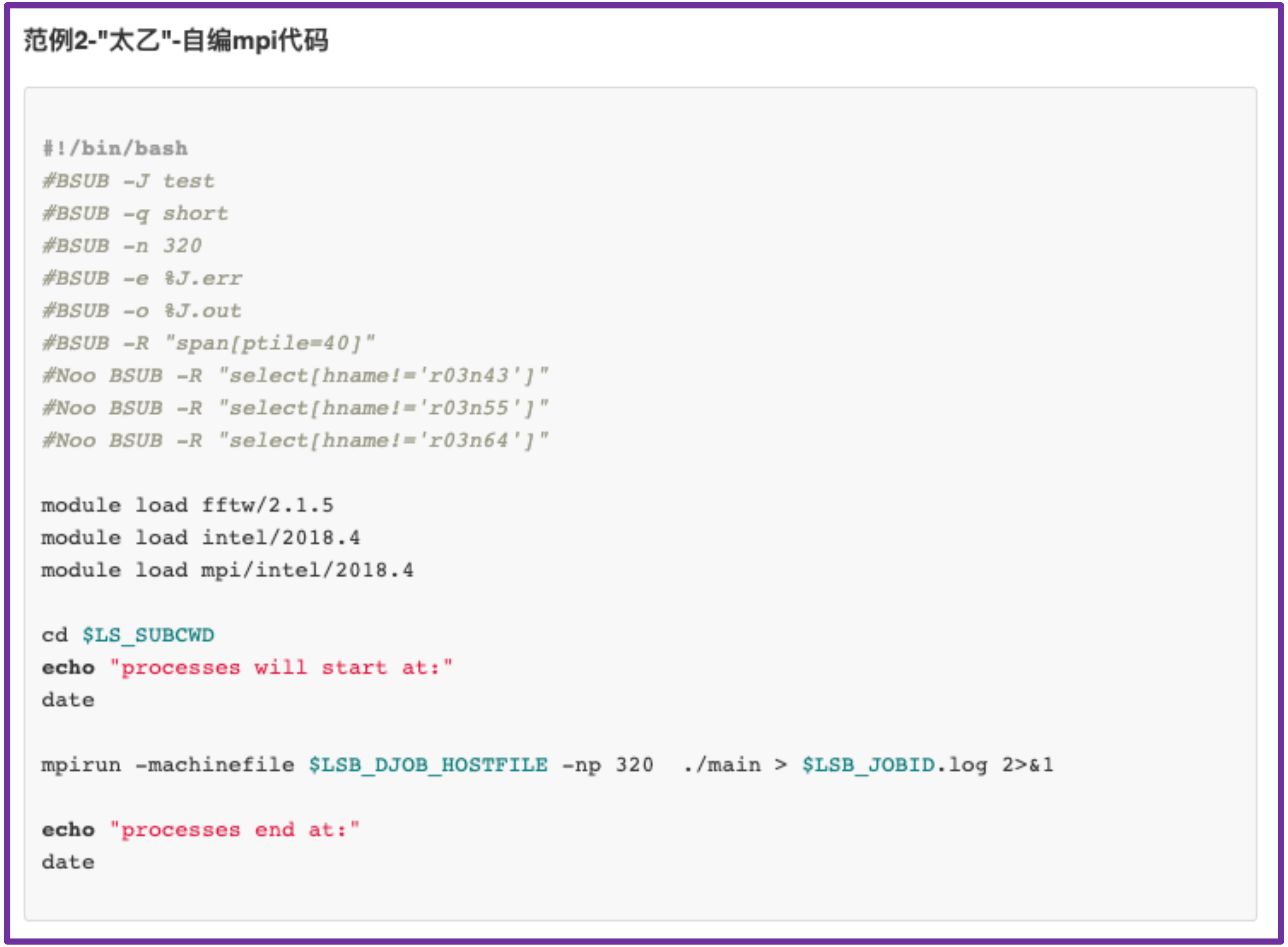


范例1-"太乙"-vasp

注意:建议采用2018.4版本

```
#!/bin/sh
#BSUB -J N_F                        ##job name
#BSUB -q short                     ##queue name
#BSUB -n 80                        ##number of total cores
#BSUB -R "span[ptile=40]"          ##40 cores per node
#BSUB -W 12:00                     ##walltime in hh:mm
#BSUB -R "select[hname!='r13n18']" ##exclusive r13n18
#BSUB -e err.log                   ##error log
#BSUB -o H.log                     ##output log
module load intel/2018.4 mpi/intel/2018.4 vasp/5.4.4
mpirun vasp_std &>log
```

Shell Stream Redirection



范例2-"太乙"-自编mpi代码

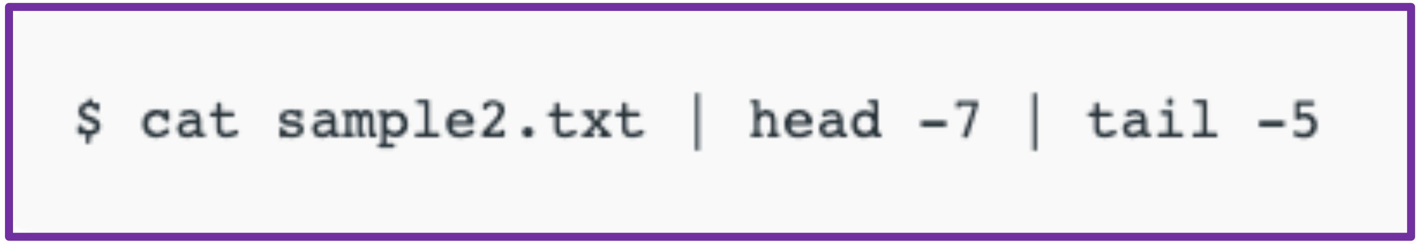
```
#!/bin/bash
#BSUB -J test
#BSUB -q short
#BSUB -n 320
#BSUB -e %J.err
#BSUB -o %J.out
#BSUB -R "span[ptile=40]"
#Noo BSUB -R "select[hname!='r03n43']"
#Noo BSUB -R "select[hname!='r03n55']"
#Noo BSUB -R "select[hname!='r03n64']"

module load fftw/2.1.5
module load intel/2018.4
module load mpi/intel/2018.4

cd $LS_SUBCWD
echo "processes will start at:"
date

mpirun -machinefile $LSB_DJOB_HOSTFILE -np 320  ./main > $LSB_JOBID.log 2>&1

echo "processes end at:"
date
```

Unix pipes

“|” connects the standard output of the first command to the second command

“|&” connects the standard output and error of the first command to the second command

What will be the output?



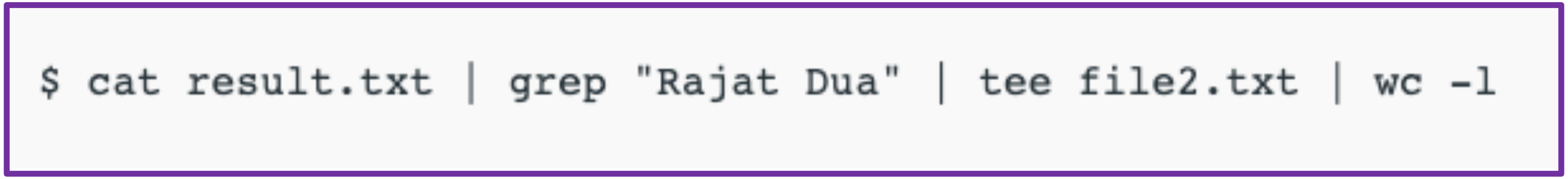
```
$ cat sample2.txt | head -7 | tail -5
```

Unix pipes

“|” connects the standard output of the first command to the second command

“|&” connects the standard output and error of the first command to the second command

What will be the output?



```
$ cat result.txt | grep "Rajat Dua" | tee file2.txt | wc -l
```

Unix pipes

- “;” is a command separator.
- “&&” is a logical AND.
- “||” is a logical OR.

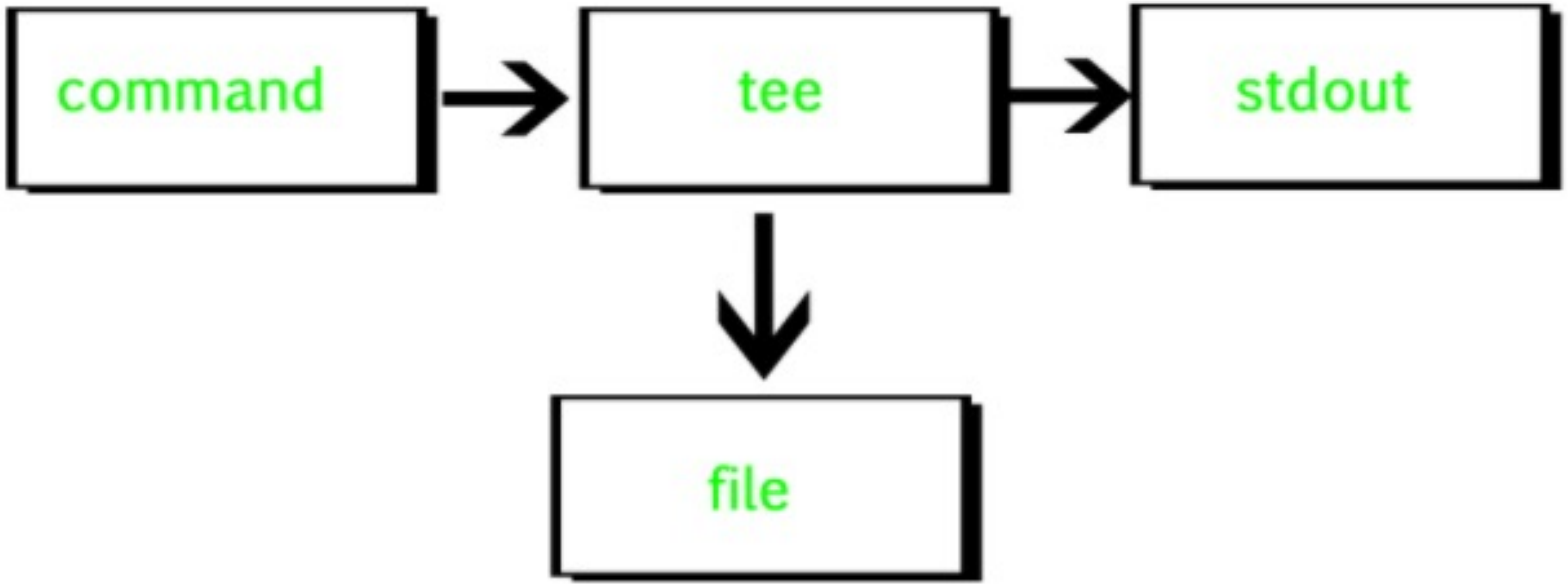
cmd1 ; cmd2 will execute cmd2 after cmd1.

cmd1 && cmd2 will execute cmd2 only if cmd1 is executed successfully.

cmd1 || cmd2 will execute cmd2 only if cmd1 fails.

Unix pipes

“tee” reads the standard input and writes it to both the standard output as well as one or more files.



Unix pipes

“tee” reads the standard input and writes it to both the standard output as well as one or more files.

Example: In the README.txt file for MPICH-3.2.1, it states how you shall install the MPICH on your computer. The configuration step is

```
./configure --prefix=/home/<USERNAME>/mpich-install 2>&1 | tee c.txt
```

and the make step is

```
make 2>&1 | tee m.txt
```

1. Basic Commands
2. File attributes and permissions
3. Regular expressions
4. Interacting with the shell
5. Unix pipes
6. Job control
7. Unix environmental variables
8. Text editors
9. Shell scripting
10. Additional topics

Job Control

Shell allows you to manage jobs:

- place jobs in the background
- move jobs to the foreground
- suspend a job
- kill a job

- Place “&” at the end of a command will place the job in the background.

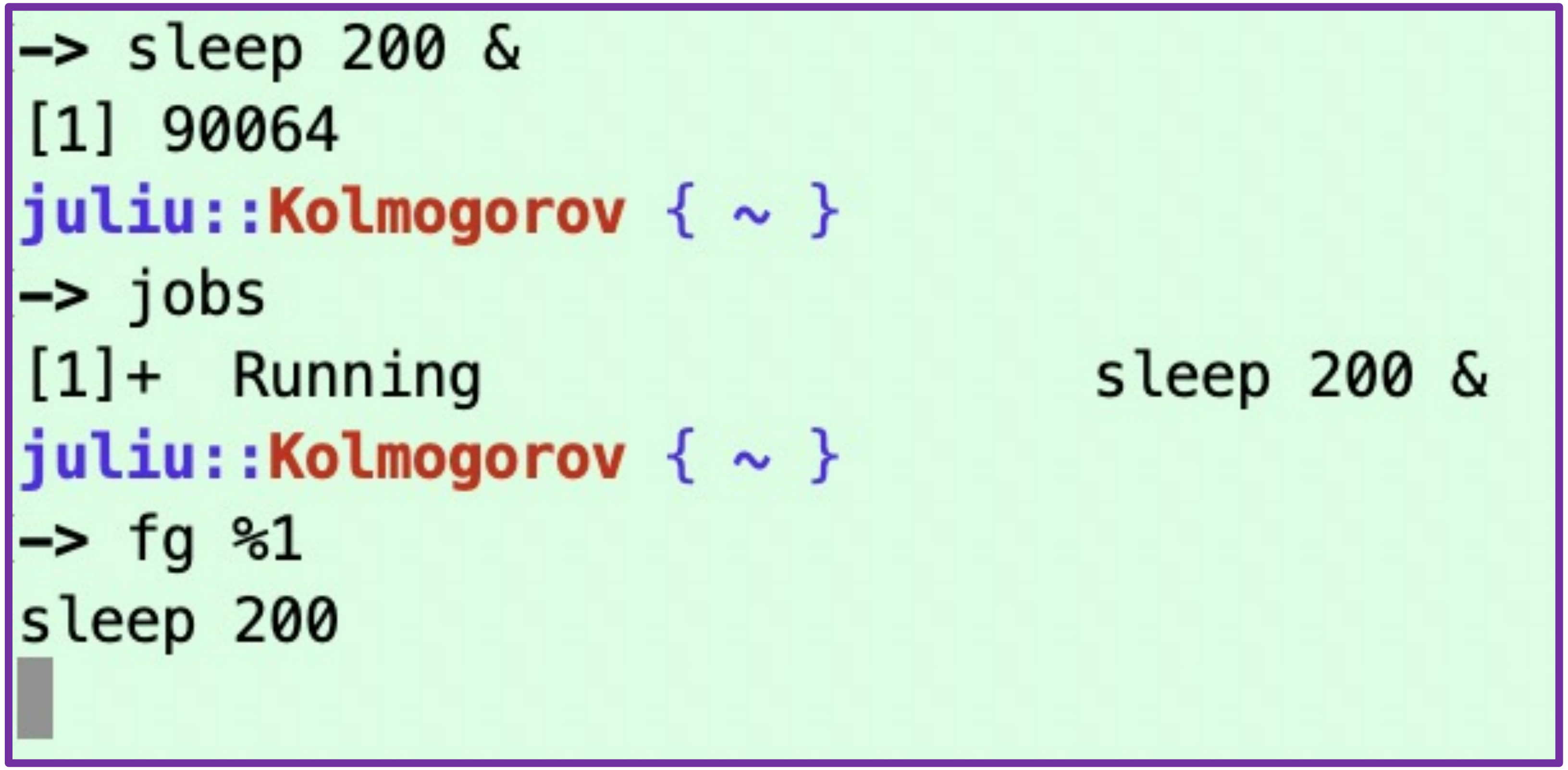
```
make all &> make.out &
```

- Place “nohup” at the beginning will keep the job running even if you close the terminal session.

```
nohup make all &> make.out &
```

Job Control

- “jobs” will list all background jobs.
- Shell assigns a number to each job.
- “fg” will bring the job to the foreground.

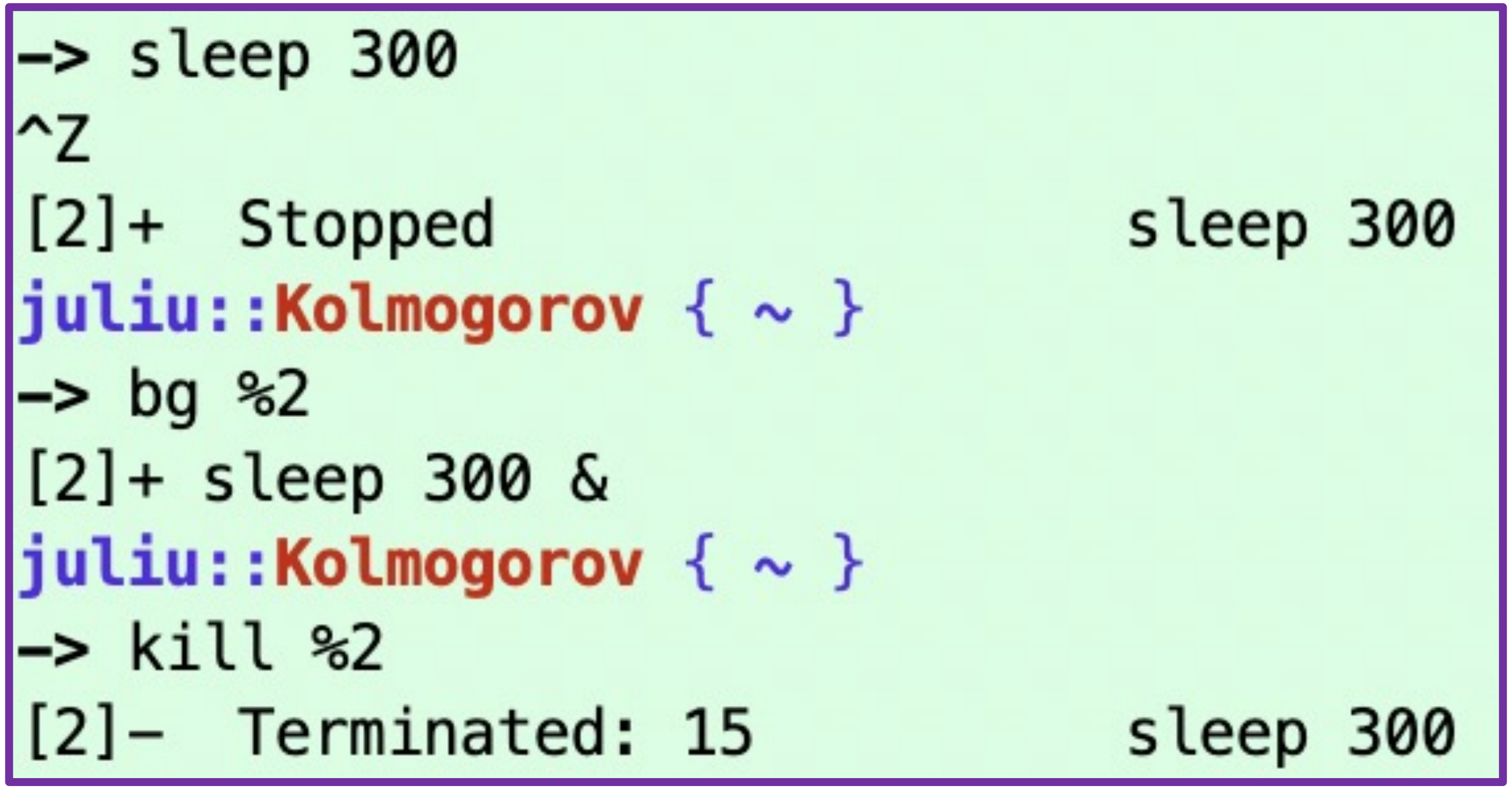


```
-> sleep 200 &
[1] 90064
juliu::Kolmogorov { ~ }
-> jobs
[1]+  Running                 sleep 200 &
juliu::Kolmogorov { ~ }
-> fg %1
sleep 200
```

Job Control

- Use ctrl-z to suspend the current foreground job.
- “bg” will bring the job to the background.
- “kill” will kill a job in the background.

Note: it’s important to include the “%” sign to reference a job number.

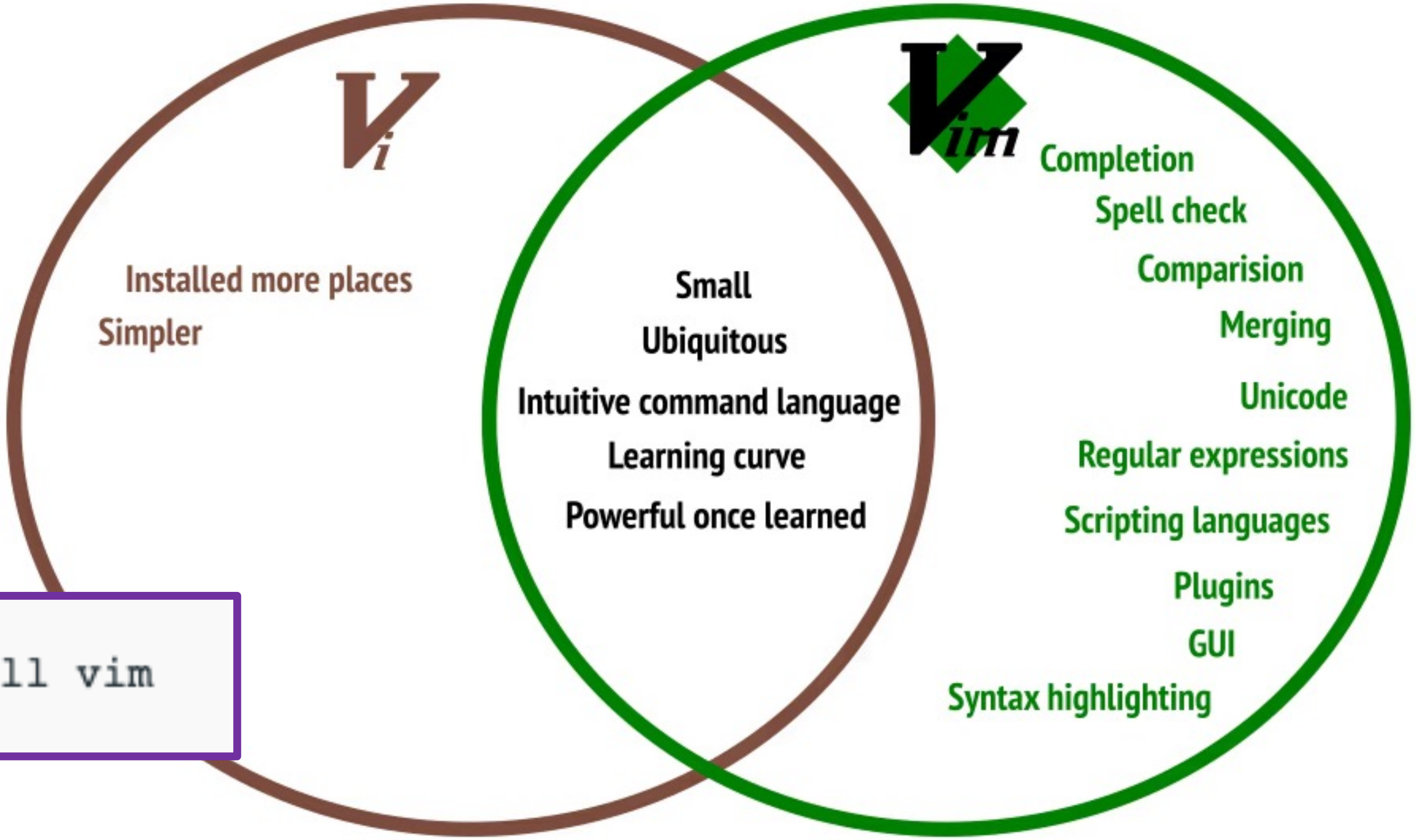


```
-> sleep 300
^Z
[2]+  Stopped                    sleep 300
juliu::Kolmogorov { ~ }
-> bg %2
[2]+ sleep 300 &
juliu::Kolmogorov { ~ }
-> kill %2
[2]-  Terminated: 15             sleep 300
```

1. Basic Commands
2. File attributes and permissions
3. Regular expressions
4. Interacting with the shell
5. Unix pipes
6. Job control
7. Unix environmental variables
8. Text editors
9. Shell scripting
10. Additional topics

Vi

- For programmers, it is necessary to use the available Unix/Linux text editors.
- The most popular and available editors are vi and emacs
- vim (Vi IMproved) is an enhanced version of vi with many powerful features.



```
sudo apt-get install vim
```

Vi

- vi is a modal editor
  Insert Mode: typed texts become part of the file
  Command Mode: keystrokes are interpreted as commands

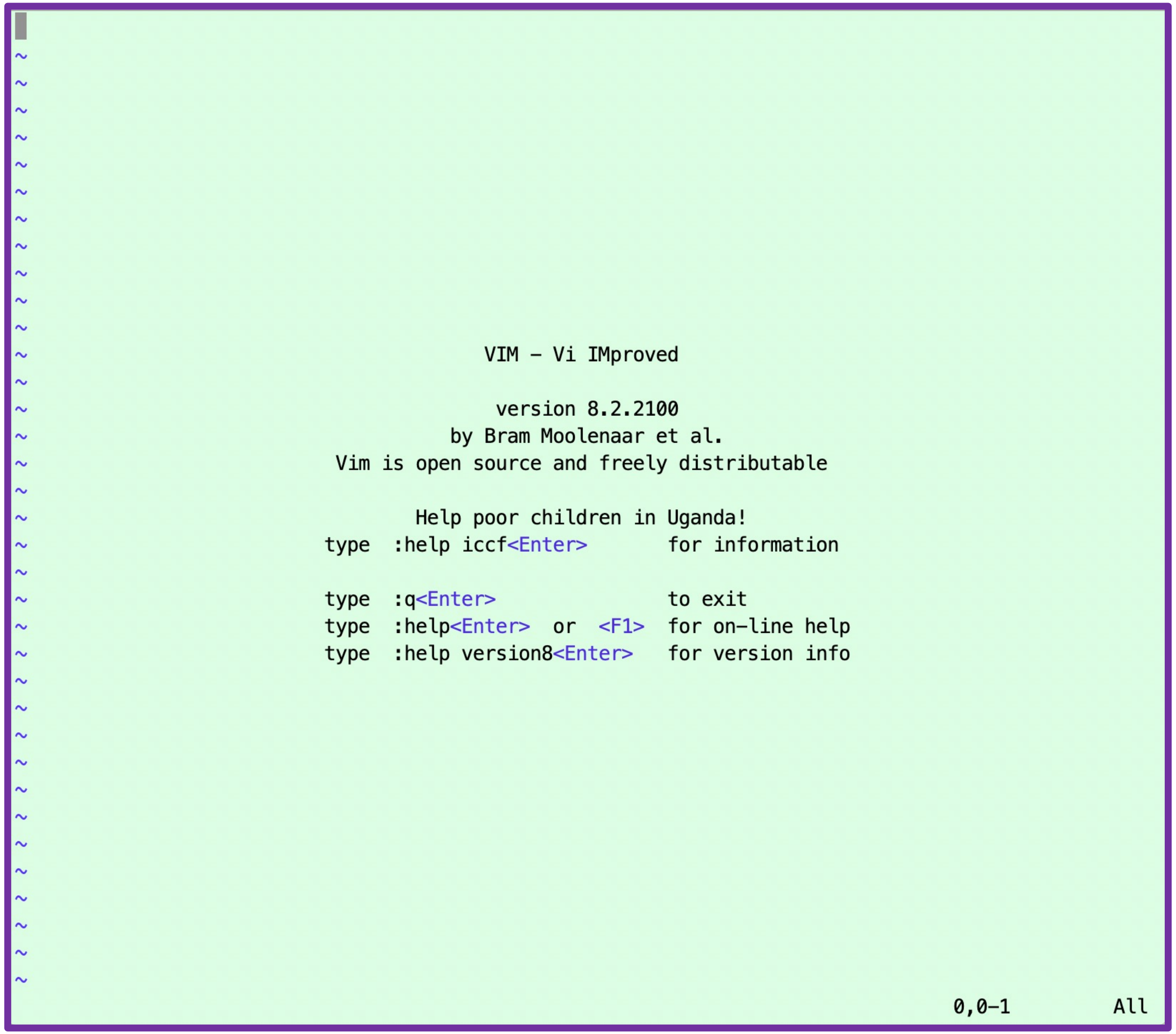
- Starting vi by
  vi (vi open an unnamed buffer)
  or vi filename (vi open a file)
  or vi [options] filename

- vi starts in the command mode by default

- Press i to enable insert mode

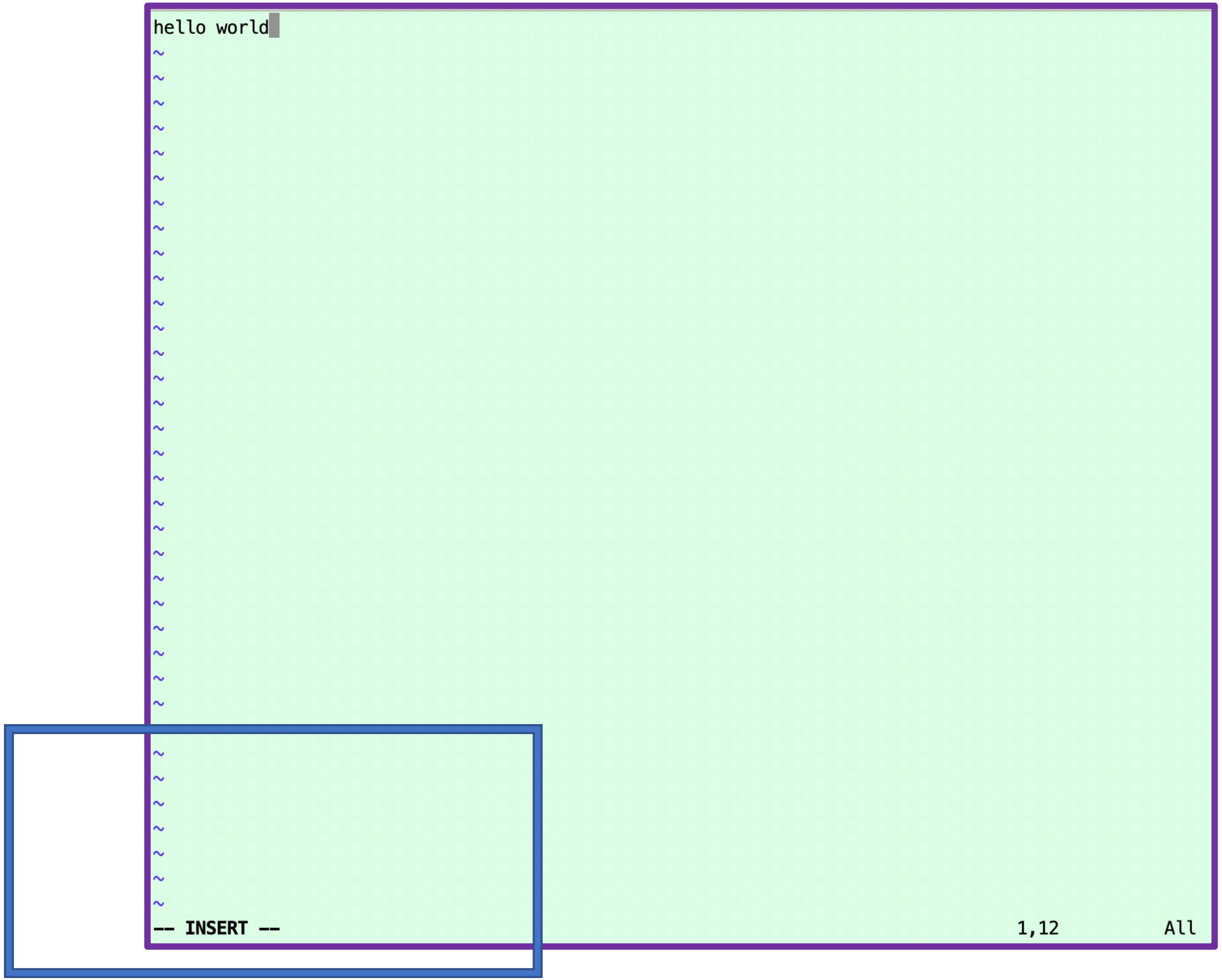
- Press Esc to switch back to command mode

Vi



```

~
~
~
~
~
~
~
~
~
~
~
~                                        VIM - Vi IMproved
~
~                                         version 8.2.2100
~                                     by Bram Moolenaar et al.
~                           Vim is open source and freely distributable
~
~                                  Help poor children in Uganda!
~                          type  :help iccf<Enter>       for information
~
~                          type  :q<Enter>               to exit
~                          type  :help<Enter>  or  <F1>  for on-line help
~                          type  :help version8<Enter>   for version info
~
~
~
~
~
~
~
~
~
~
~
~
                                                                                  0,0-1         All
```

Vi



```
hello world
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                                    1,12          All
```

Vi

- In the command mode

  press :x [enter] to save and quit
  press :q [enter] to quit
  press :q! [enter] to force quit (without saving)
  press :w <filename> to save the file
  press :w to save
  press :wq to save and exit
  press / <string> to search within the document
  press dd to delete the current line
  press yy to copy the current line
  press p to paste the last cut/deleted line
  press i to go to insert mode
  press :set number to show line numbers
  press :set spell to enable spell checking
  ……

1. Basic Commands
2. File attributes and permissions
3. Regular expressions
4. Interacting with the shell
5. Unix pipes
6. Job control
7. Unix environmental variables
8. Text editors
9. Shell scripting
10. Additional topics

Shell Scripting

- Shell scripting is “easy”: you just need to place all the Unix/Linux commands in a file as opposed to typing them interactively.
- Handy for automating certain tasks:

  staging your scientific applications

  performing postprocessing operations

  any repetitive operations on files

  …
- Shell scripts must begin with a specific line to indicate which shell to be used for executing the remaining commands in the file. This is known as “Shebang”

BASH:

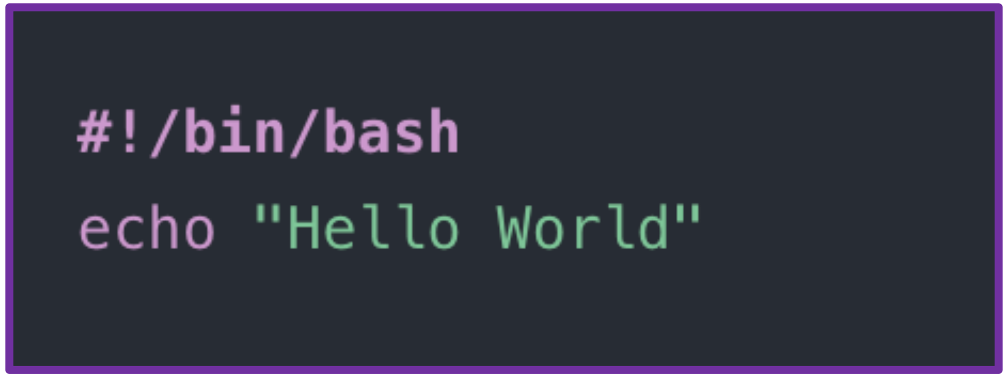
```
#!/bin/bash
```

TCSH:

```
#!/bin/tcsh
```

Shell Scripting

- Comment lines start with #
- In order to run a shell script, it must have the execute permission.



```
#!/bin/bash
echo "Hello World"
```

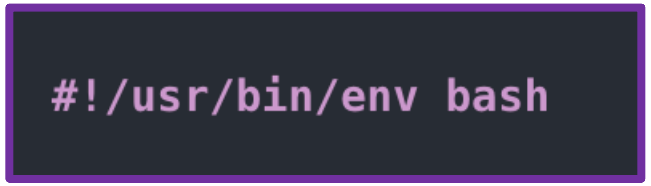
- Execute the script:
- ./hello-world.sh (recommended)

- /bin/bash hellow-world.sh

- bash hello-world.sh (assuming bin is in your PATH)

- sh hello-world.sh

A better shebang because sometimes bash is in other locations, such as /usr/bin/bash

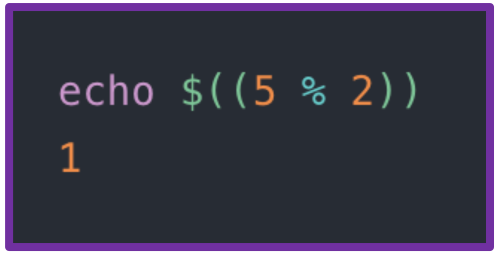


```
#!/usr/bin/env bash
```

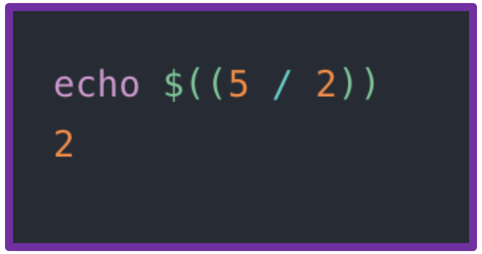
Accessing parameters

- Parameters passed to the script are named by their position: `$1` is the name of the first parameter, `$2` is the name of the second parameter.
- "$@" expands all parameters
- "$#" gets the number of parameters passed to the script.

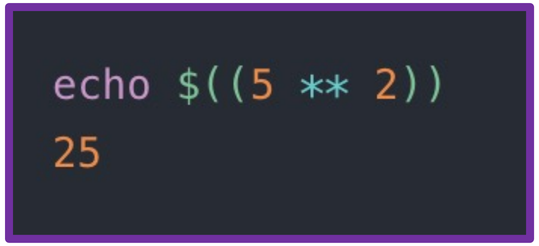
- Arithmetic computation can be done with (()).



```
echo $((5 % 2))
1
```



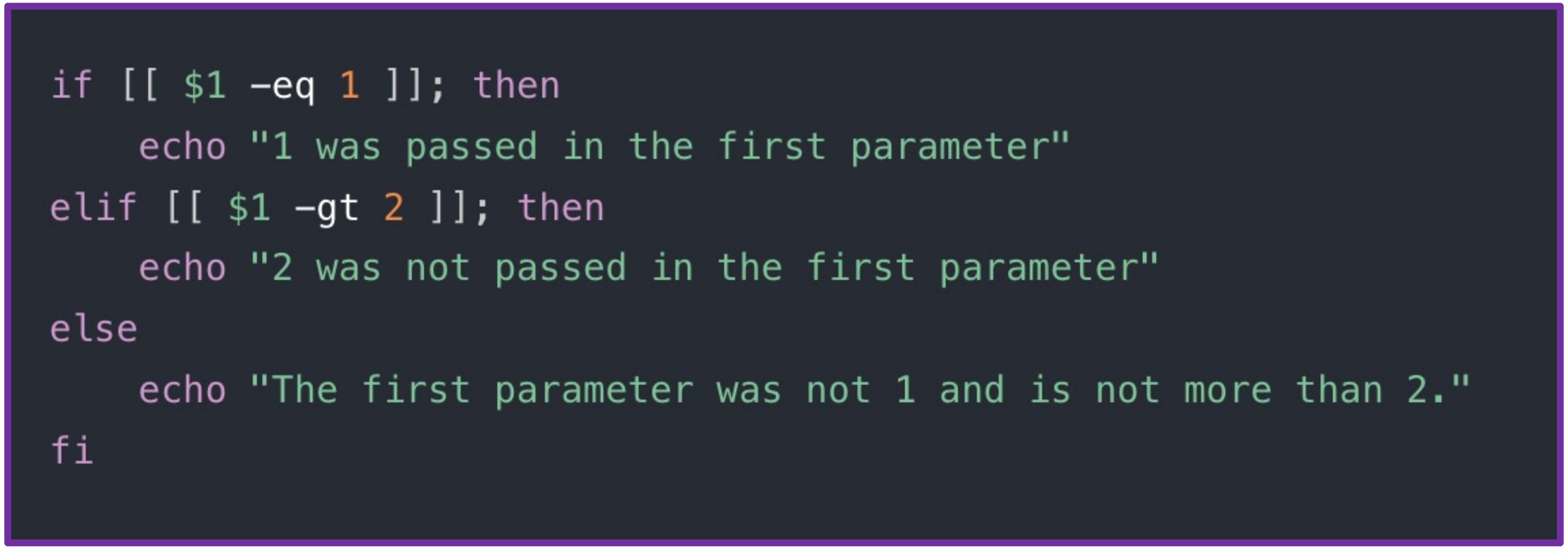
```
echo $((5 / 2))
2
```



```
echo $((5 ** 2))
25
```

Control Structures

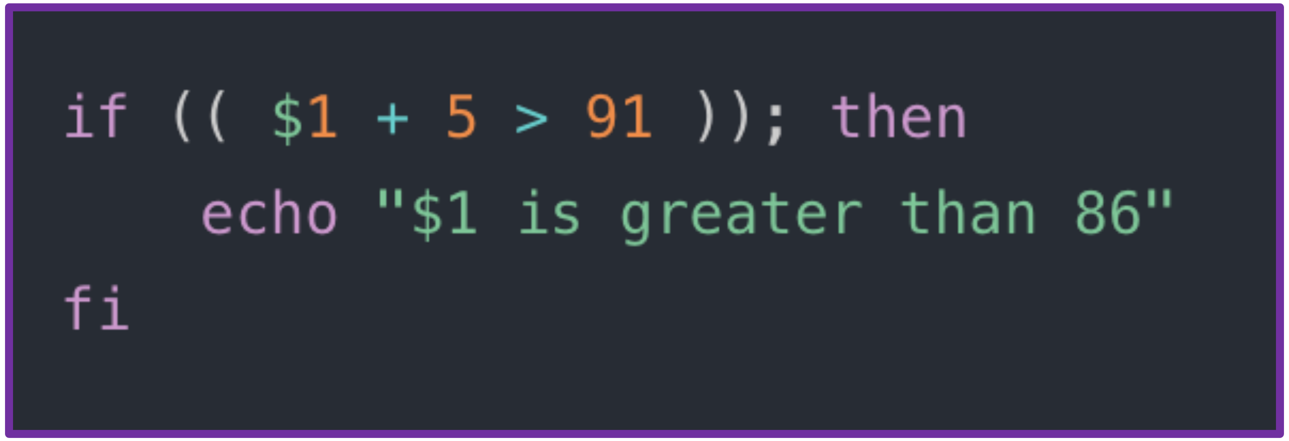
- Closing `fi` is necessary.
- `elif` and/or `else` are unnnecessary.
- `;` is the command connector that puts then in the first line. They can be omitted if one put then to the next line.



```
if [[ $1 -eq 1 ]]; then
    echo "1 was passed in the first parameter"
elif [[ $1 -gt 2 ]]; then
    echo "2 was not passed in the first parameter"
else
    echo "The first parameter was not 1 and is not more than 2."
fi
```

Control Structures

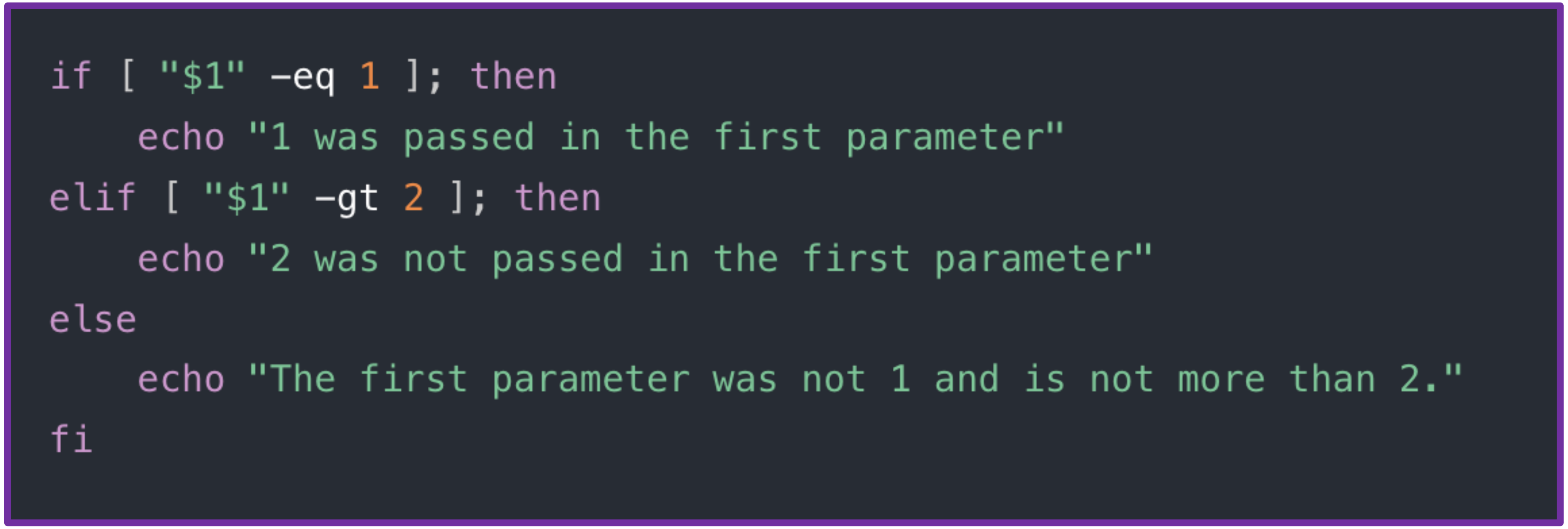
- ‘[[’ and ‘]]’ are commands that test the outcomes.
- Math expressions can be tested with double parentheses `((` and `))`
- ‘[’ and ‘]’ will also work just like the double brackets.



```
if (( $1 + 5 > 91 )); then
    echo "$1 is greater than 86"
fi
```

Control Structures

- ‘[[’ and ‘]]’ are commands that test the outcomes.
- remember to have a space between the condition and the brackets.
- Math expressions can be tested with double parentheses `((` and `))`
- ‘[’ and ‘]’ will also work just like the double brackets.



```
if [ "$1" -eq 1 ]; then
    echo "1 was passed in the first parameter"
elif [ "$1" -gt 2 ]; then
    echo "2 was not passed in the first parameter"
else
    echo "The first parameter was not 1 and is not more than 2."
fi
```

Control Structures

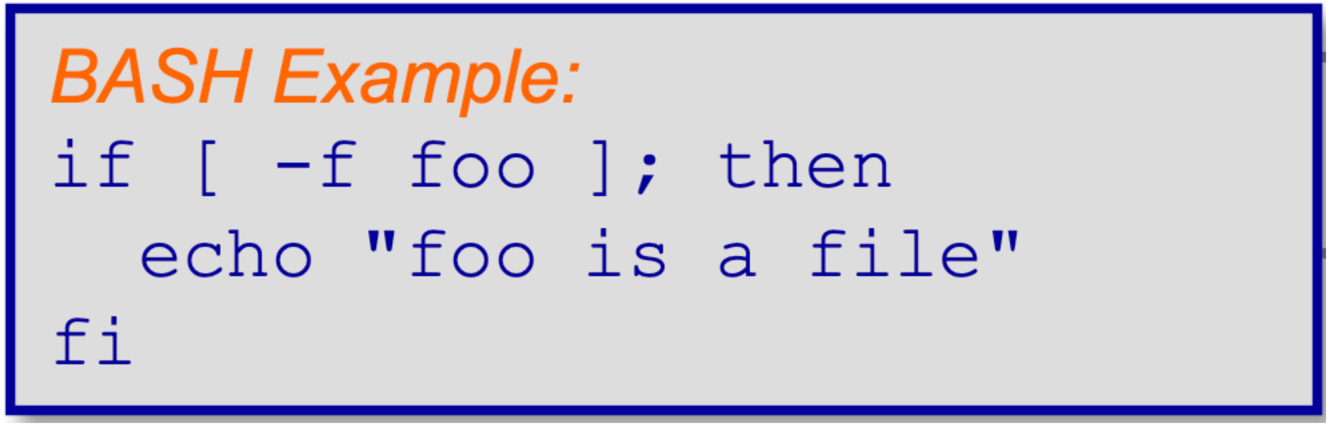
File operations:

-e “$file” returns true if the file exists

-d “$file” returns true if the file exists and is a directory

-f “$file” returns true if the file exists and is a regular file

-h “$file” returns true if the file exists and is a symbolic link



BASH Example:

```
if [ -f foo ]; then
  echo "foo is a file"
fi
```

Control Structures

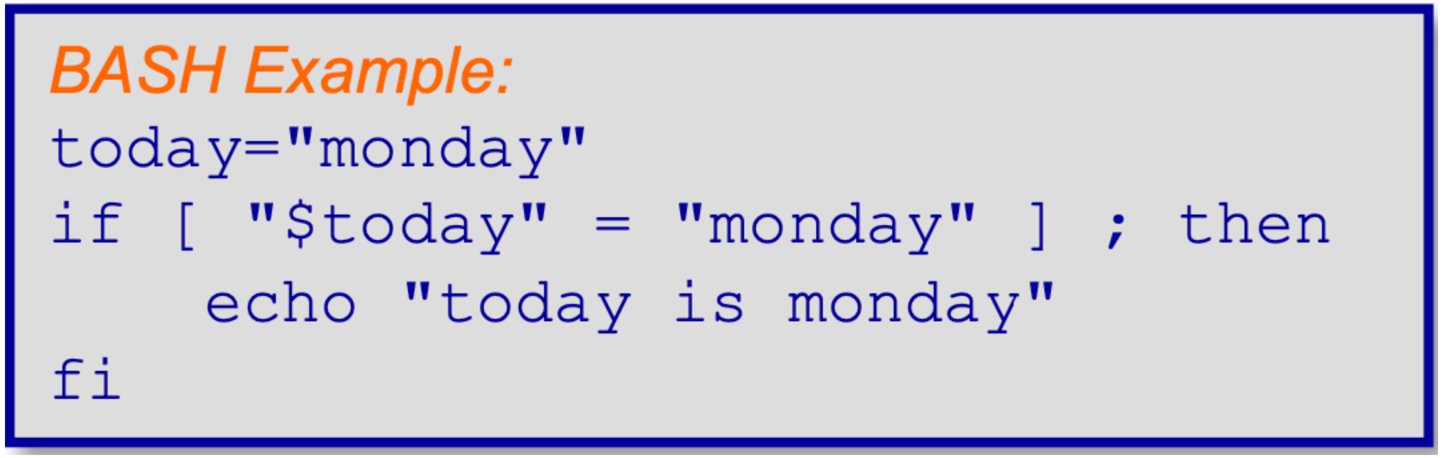
String comparisons:

-z “$str” returns true if string is zero

-n “$str” returns true if length of string is nonzero

“$str1” = “$str2” returns true if two strings match

“$str1” != “$str2” returns true if two strings are not equal



BASH Example:

```
today="monday"
if [ "$today" = "monday" ] ; then
    echo "today is monday"
fi
```

Control Structures

Integer Comparisions

“$int1” –eq “$int2”	returns true if the integers are equal
“$int1” –ne “$int2”	returns true if the integers are not equal
“$int1” –gt “$int2”	returns true if int1 is greater than int2
“$int1” –ge “$int2”	returns true if int1 is greater than or equal to int2
“$int1” –lt “$int2”	returns true if int1 is less than int2
“$int1” –le “$int2”	returns true if int1 is less than or equal to int2

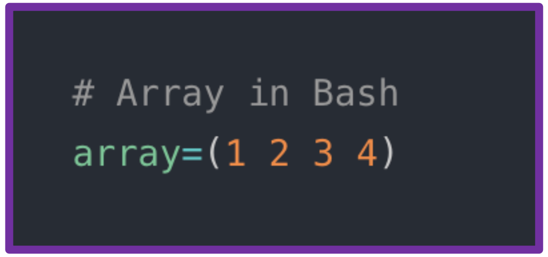
There are many more in bash. Search online.

BASH Example:

```
x=13
y=25
if [ $x -lt $y ]; then
  echo "$x is less than $y"
fi
```

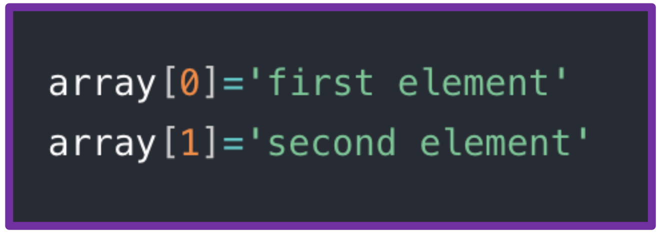
Arrays

Bash simply use space to separate array elements.

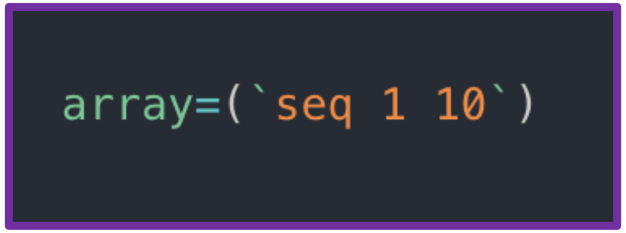


```
# Array in Bash
array=(1 2 3 4)
```

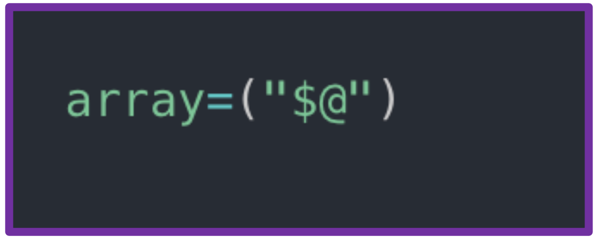
You may assign the array by indices, by seq command, or by script’s input



```
array[0]='first element'
array[1]='second element'
```



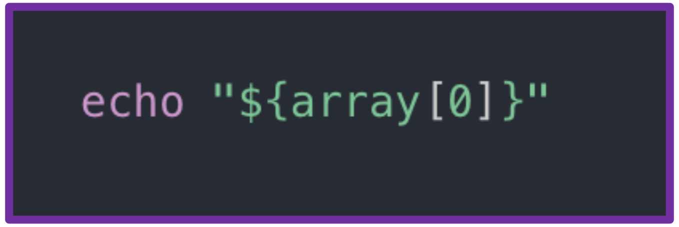
```
array=(`seq 1 10`)
```



```
array=("$@")
```

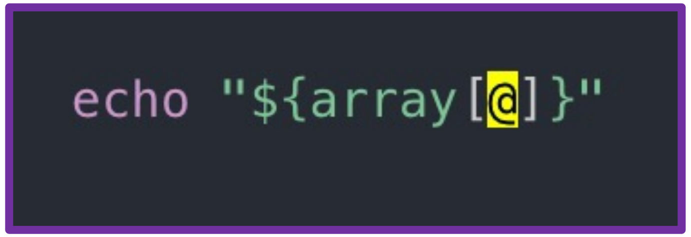
Arrays

Array element can be accessed with indices



```
echo "${array[0]}"
```

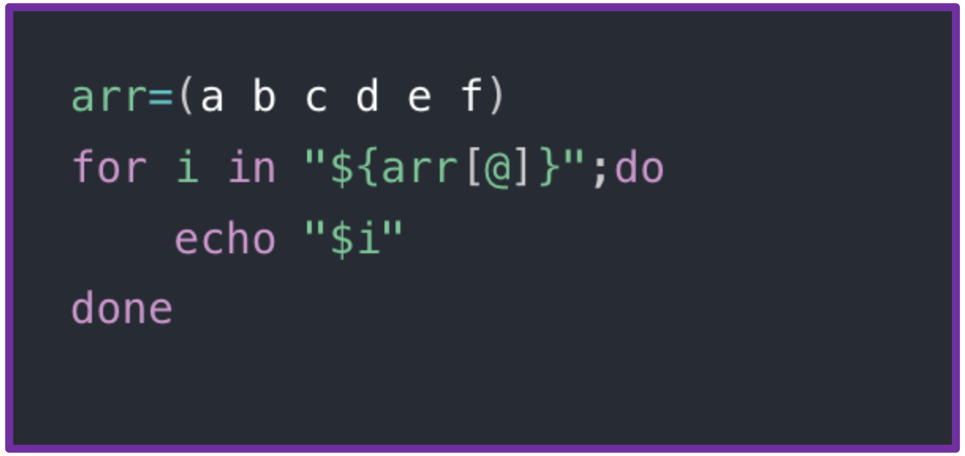
${array[@]}	all the items in the array
${!array[@]}	all the indices in the array
${#array[@]}	number of items in the array



```
echo "${array[@]}"
```

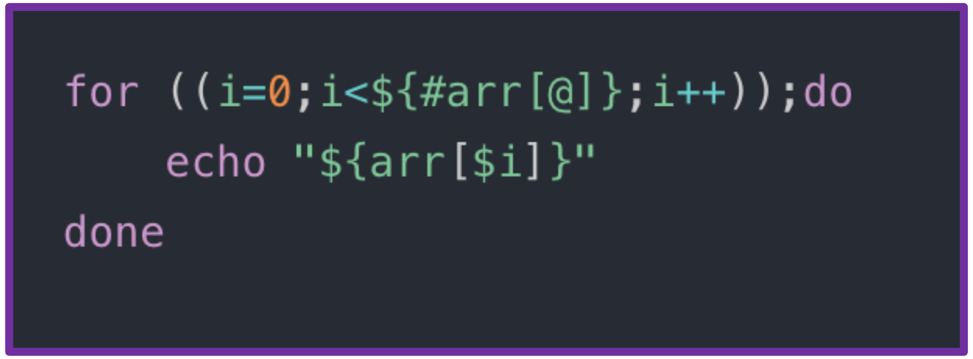
Loops

For loop can be written with the help of arrays



```
arr=(a b c d e f)
for i in "${arr[@]}";do
    echo "$i"
done
```

Or it can be written with the C-style syntax



```
for ((i=0;i<${#arr[@]};i++));do
    echo "${arr[$i]}"
done
```

1. Basic Commands
2. File attributes and permissions
3. Regular expressions
4. Interacting with the shell
5. Unix pipes
6. Job control
7. Unix environmental variables
8. Text editors
9. Shell scripting
10. Additional topics

More commands: tar

tar: tape archive is used to create Archive and extract the Archive files.

Archive is a single file that contains a collection of other files and/or directories. It can be easily compressed and transferred.

tar [-options] archive-file file_or_directory_to_be_archived

-c	create archive
-x	extract archive
-f	create archive with given filename
-v	display verbose information
-z	use gzip

More commands: tar

tar: tape archive is used to create Archive and extract the Archive files.

```
tar [-options] archive-file file_or_directory_to_be_archived
```

-c	create archive
-x	extract archive
-f	create archive with given filename
-v	display verbose information
-z	use gzip

Ex: gzip compression on the archive

```
tar –zcvf file.tar.gz folder
```

extract a gzip archive

```
tar –zxvf file.tar.gz
```

More commands: scp

scp: secure copy is a command that copies files to remove machines

scp [-options] file_source file_target

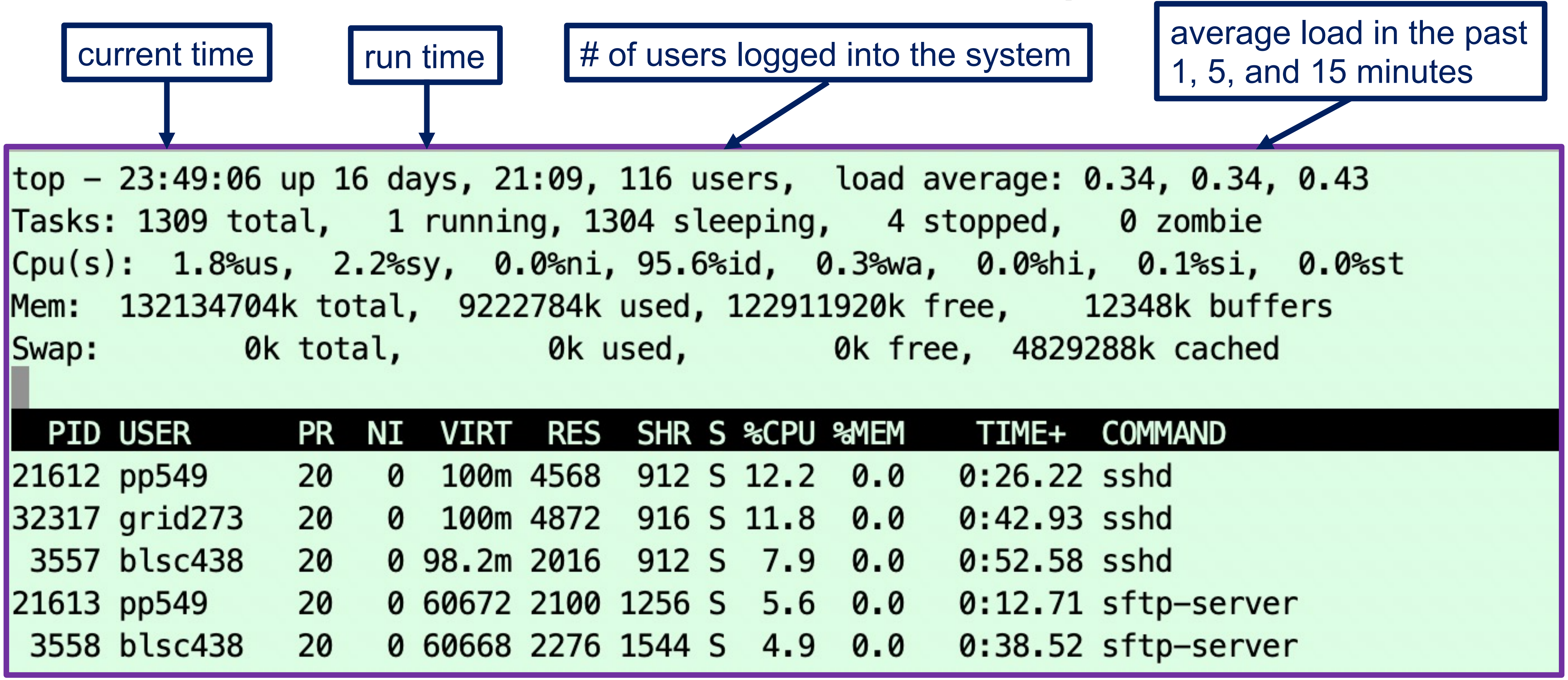
-r recursively (for directories)
-P port number
-l limit the bandwidth (in kB/s)

```
scp –r data_folder mae-liuj@172.18.6.175:/work/mae-liuj
scp –r mae-liuj@172.18.6.175:/work/mae-liuj/data_folder .
```

More commands: top

- top: task manager program
- its output contains the summary area and the task area.
- top updates every three second.

More commands: top



current time

run time

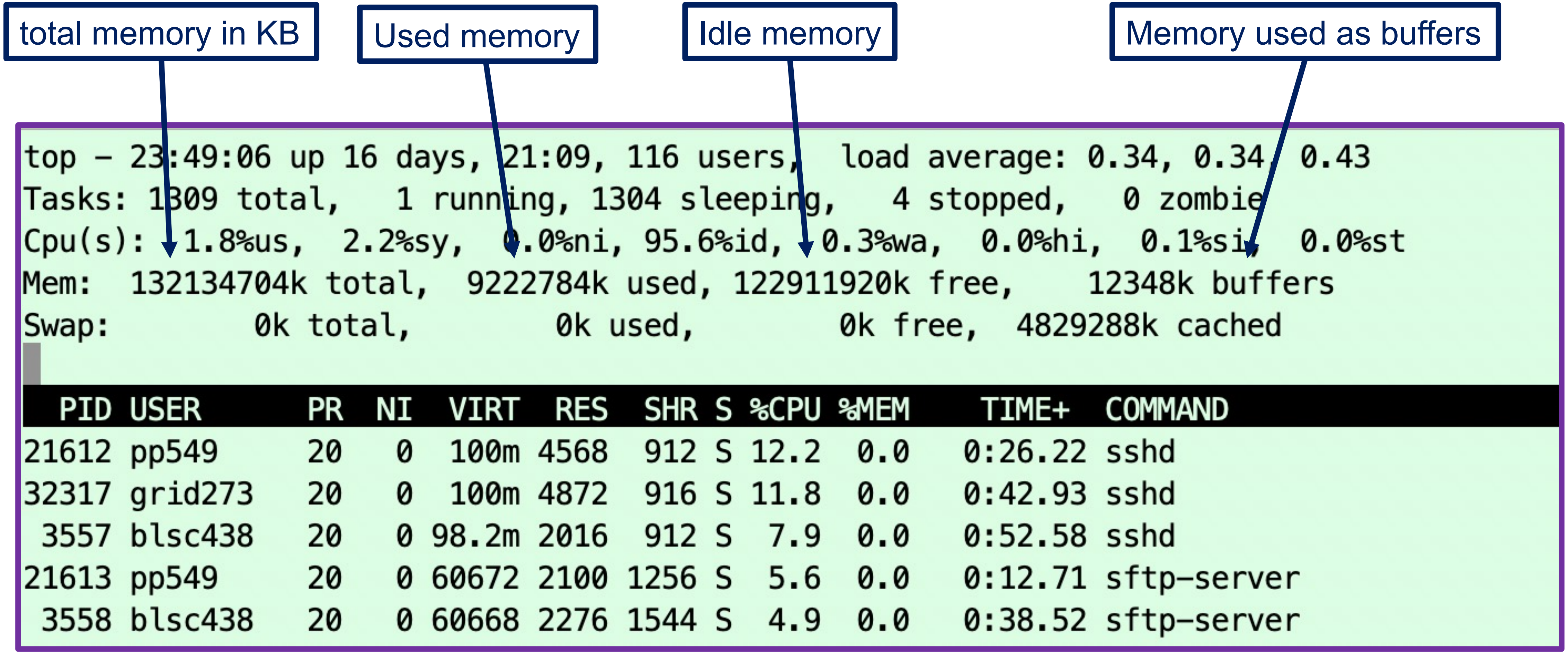
# of users logged into the system

average load in the past
1, 5, and 15 minutes

```
top - 23:49:06 up 16 days, 21:09, 116 users,  load average: 0.34, 0.34, 0.43
Tasks: 1309 total,   1 running, 1304 sleeping,   4 stopped,   0 zombie
Cpu(s):  1.8%us,  2.2%sy,  0.0%ni, 95.6%id,  0.3%wa,  0.0%hi,  0.1%si,  0.0%st
Mem:  132134704k total,  9222784k used, 122911920k free,    12348k buffers
Swap:        0k total,        0k used,        0k free,  4829288k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
21612 pp549     20   0  100m 4568  912 S 12.2  0.0   0:26.22 sshd
32317 grid273   20   0  100m 4872  916 S 11.8  0.0   0:42.93 sshd
 3557 blsc438   20   0 98.2m 2016  912 S  7.9  0.0   0:52.58 sshd
21613 pp549     20   0 60672 2100 1256 S  5.6  0.0   0:12.71 sftp-server
 3558 blsc438   20   0 60668 2276 1544 S  4.9  0.0   0:38.52 sftp-server
```

us : amount of time the CPU spends executing processes for people in user space
sy : amount of time spent running system
ni : amount of time spent executing processes with a manually set nice value

More commands: top



total memory in KB

Used memory

Idle memory

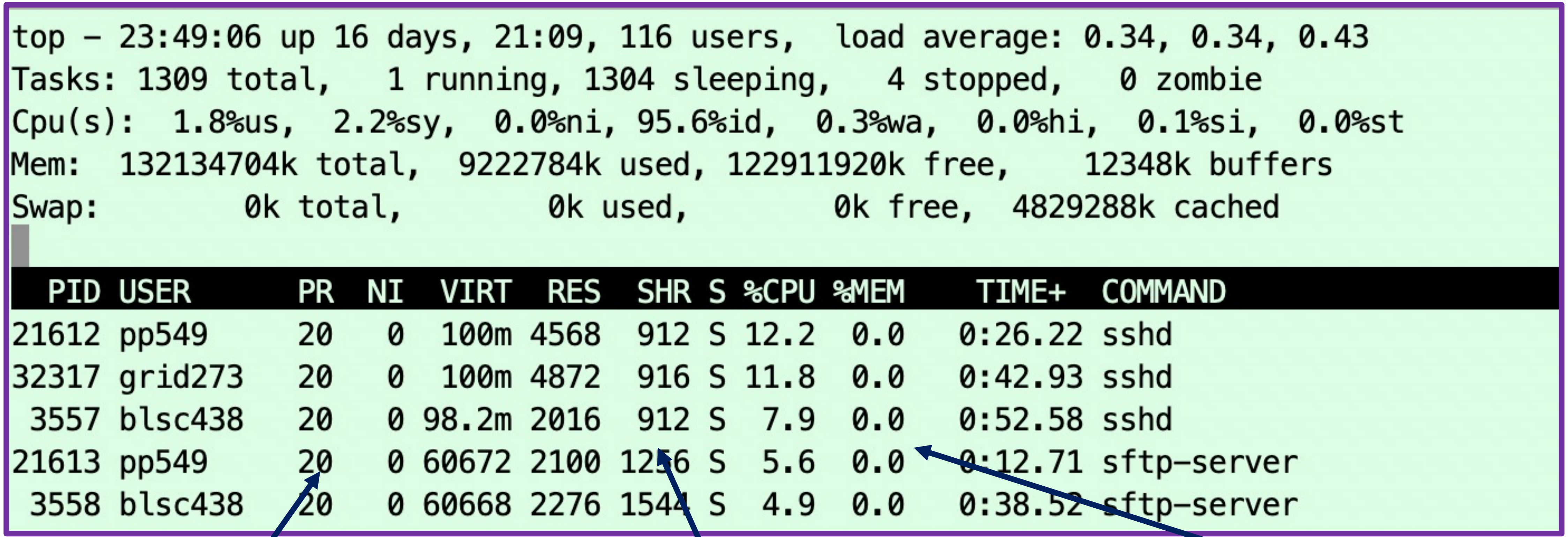
Memory used as buffers

```
top - 23:49:06 up 16 days, 21:09, 116 users,  load average: 0.34, 0.34, 0.43
Tasks: 1309 total,   1 running, 1304 sleeping,   4 stopped,   0 zombie
Cpu(s):  1.8%us,  2.2%sy,  0.0%ni, 95.6%id,  0.3%wa,  0.0%hi,  0.1%si,  0.0%st
Mem:  132134704k total,  9222784k used, 122911920k free,    12348k buffers
Swap:        0k total,        0k used,        0k free,  4829288k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
21612 pp549     20   0  100m 4568  912 S 12.2  0.0   0:26.22 sshd
32317 grid273   20   0  100m 4872  916 S 11.8  0.0   0:42.93 sshd
 3557 blsc438   20   0 98.2m 2016  912 S  7.9  0.0   0:52.58 sshd
21613 pp549     20   0 60672 2100 1256 S  5.6  0.0   0:12.71 sftp-server
 3558 blsc438   20   0 60668 2276 1544 S  4.9  0.0   0:38.52 sftp-server
```

id : amount of CPU idle time
wa: amount of time CPU spends waiting for I/O to complete
hi : amount of time spent servincing hardware interrupts

si : amount of time spent serviing software interrupts
st : amount of time lost due to running virtual machine

More commands: top



```
top - 23:49:06 up 16 days, 21:09, 116 users,  load average: 0.34, 0.34, 0.43
Tasks: 1309 total,   1 running, 1304 sleeping,   4 stopped,   0 zombie
Cpu(s):  1.8%us,  2.2%sy,  0.0%ni, 95.6%id,  0.3%wa,  0.0%hi,  0.1%si,  0.0%st
Mem:  132134704k total,  9222784k used, 122911920k free,    12348k buffers
Swap:        0k total,        0k used,        0k free,  4829288k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
21612 pp549     20   0  100m 4568  912 S 12.2  0.0   0:26.22 sshd
32317 grid273   20   0  100m 4872  916 S 11.8  0.0   0:42.93 sshd
 3557 blsc438   20   0 98.2m 2016  912 S  7.9  0.0   0:52.58 sshd
21613 pp549     20   0 60672 2100 1256 S  5.6  0.0   0:12.71 sftp-server
 3558 blsc438   20   0 60668 2276 1544 S  4.9  0.0   0:38.52 sftp-server
```

PR: priority smaller number represents higher priority

NI: the nice value of the process

RES: physical memory used by this task

S: status of the process S is sleeping R is running

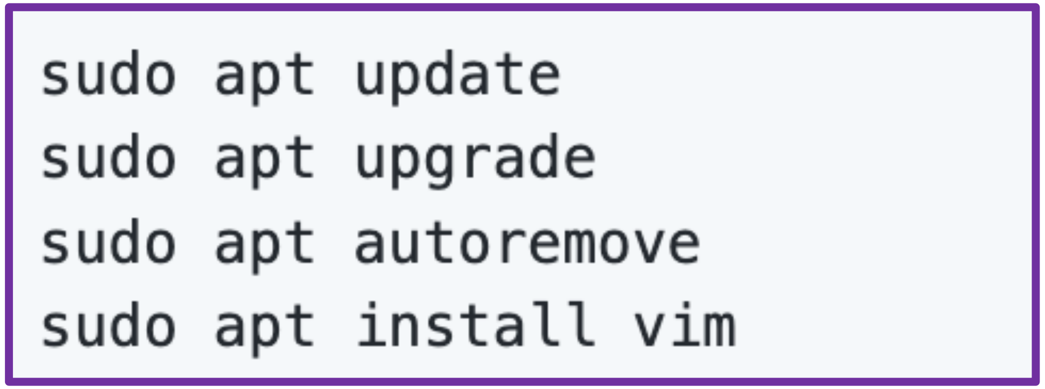
%CPU: the share of CPU time used by this process

%MEM: the share of physical memory by this process

More commands: sudo apt-get

- **sudo** : super-user-do is used to access restricted files and operations. By default, Linux restricts access to certain parts of the system preventing sensitive files from being compromised.

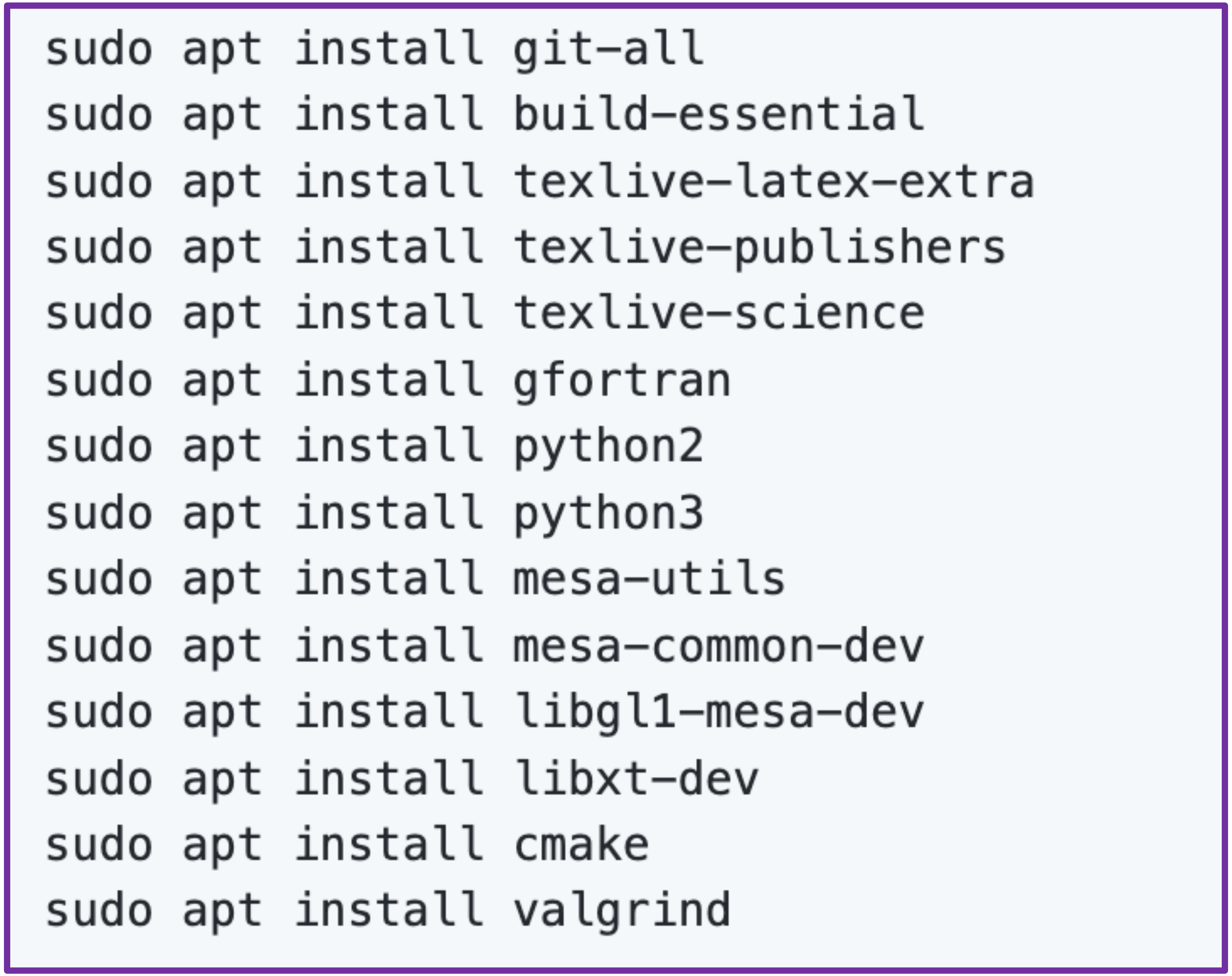
sudo [command]



```
sudo apt update
sudo apt upgrade
sudo apt autoremove
sudo apt install vim
```

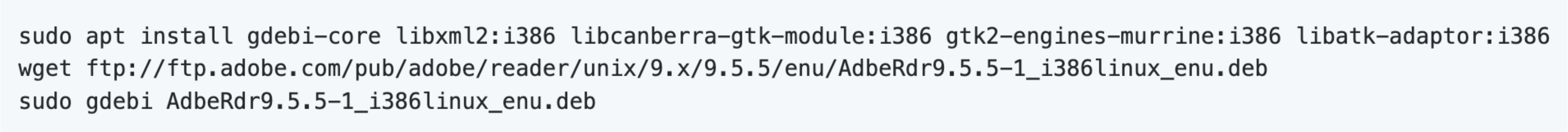
- **apt-get:** is a command used to help managing packages in Linux.
  - update: This command is used to synchronize the package index files from their sources again.
  - upgrade: This command is used to install the latest versions of the packages currently installed.
  - install: This command is used to install or upgrade packages.

More commands: sudo apt-get



```
sudo apt install git-all
sudo apt install build-essential
sudo apt install texlive-latex-extra
sudo apt install texlive-publishers
sudo apt install texlive-science
sudo apt install gfortran
sudo apt install python2
sudo apt install python3
sudo apt install mesa-utils
sudo apt install mesa-common-dev
sudo apt install libgl1-mesa-dev
sudo apt install libxt-dev
sudo apt install cmake
sudo apt install valgrind
```

Install Adobe Reader



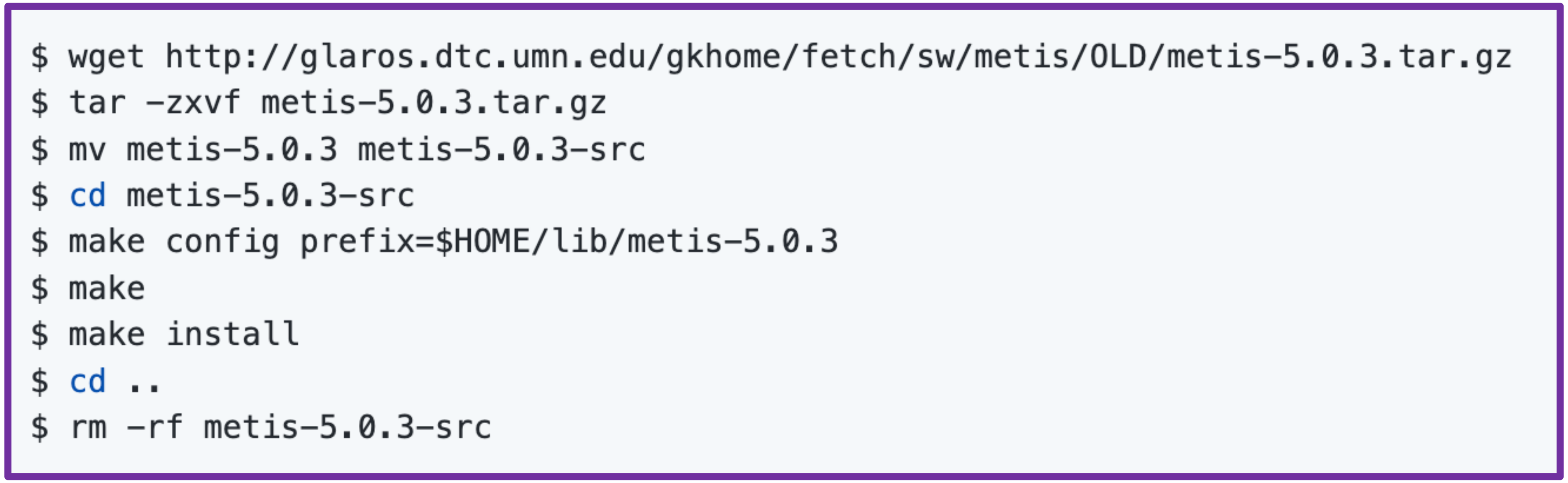
```
sudo apt install gdebi-core libxml2:i386 libcanberra-gtk-module:i386 gtk2-engines-murrine:i386 libatk-adaptor:i386
wget ftp://ftp.adobe.com/pub/adobe/reader/unix/9.x/9.5.5/enu/AdbeRdr9.5.5-1_i386linux_enu.deb
sudo gdebi AdbeRdr9.5.5-1_i386linux_enu.deb
```

Compile libraries without sudo permission

- Oftentimes, you may need to install external libraries on a machine without sudo permission.

- You will have to specify a location of the build by assigning prefix value in the configuration stage.

- The rest step will follow a typical manner of library install make && make install.



```
$ wget http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/OLD/metis-5.0.3.tar.gz
$ tar -zxvf metis-5.0.3.tar.gz
$ mv metis-5.0.3 metis-5.0.3-src
$ cd metis-5.0.3-src
$ make config prefix=$HOME/lib/metis-5.0.3
$ make
$ make install
$ cd ..
$ rm -rf metis-5.0.3-src
```

Summary

- We covered basics of Linux/Unix which should help one to get started on managing a HPC machine or a cluster.

- There are more commands that could be useful.
  Unix in a Nutshell, A Robbins, 2006

- Advanced bash scripting guide https://tldp.org/LDP/abs/html

- Bash reference manual http://www.faqs.org/docs/bashman/bashref.html

- Stack overflow https://stackoverflow.com/