

# Quick Start Guide

by **Burak Himmetoglu**

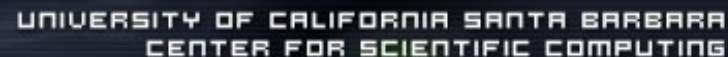
Supercomputing Consultant

Enterprise Technology Services &  
Center for Scientific Computing

**E-mail:** [bhimmetoglu@ucsb.edu](mailto:bhimmetoglu@ucsb.edu)

The logo for the Center for Scientific Computing (CSC) at the University of California Santa Barbara. It features the letters 'CSC' in a stylized, white, italicized font with a slight shadow effect, set against a dark background.

**CSC**

The logo for the University of California Santa Barbara Center for Scientific Computing. It consists of the text 'UNIVERSITY OF CALIFORNIA SANTA BARBARA' and 'CENTER FOR SCIENTIFIC COMPUTING' in a small, white, sans-serif font, arranged in two lines. The background of the logo area is dark with some blurred light effects.

UNIVERSITY OF CALIFORNIA SANTA BARBARA  
CENTER FOR SCIENTIFIC COMPUTING

# Contents

- User access, logging in
- Linux/Unix basics
- Running jobs
- Compiling & linking libraries

# User access

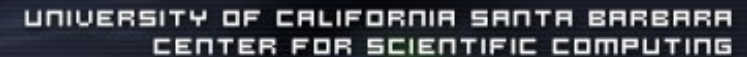
Requesting access:

<http://csc.cnsi.ucsb.edu/acct>

Accounts are only available to UCSB students / researchers / collaborators.

The logo for the Center for Scientific Computing (CSC) at UCSB, featuring the letters 'CSC' in a stylized, italicized white font.

**CSC**

The logo for the University of California Santa Barbara Center for Scientific Computing, featuring the text 'UNIVERSITY OF CALIFORNIA SANTA BARBARA' and 'CENTER FOR SCIENTIFIC COMPUTING' in a white, sans-serif font.

UNIVERSITY OF CALIFORNIA SANTA BARBARA  
CENTER FOR SCIENTIFIC COMPUTING

# Accessing machines

- `ssh [option] username@machinename.cnsi.ucsb.edu`
- `[options]`:
  - `-X` for graphics in Linux (`-Y` for Macs)

## Remote login:

- Use VPN if not in UCSB campus:

<http://www.ets.ucsb.edu/services/campus-vpn/get-connected>

- You may also need to download ssh client, Putty, X Window system if using Windows.

Putty: <http://www.chiark.greenend.org.uk/~sgtatham/putty/>

Xming: <http://sourceforge.net/projects/xming/>

# Linux/Unix basic commands

Basic command structure: `command -[options] some_argument`

<code>ls [-l, -rt, -h]</code>	List contents in the directory
<code>mkdir</code>	Make a new directory
<code>cd [directory]</code>	Change into a directory
<code>man [command_name]</code>	Open the manual for a command
<code>cp [-r]</code>	Copy file (or a directory with <code>-r</code> )
<code>mv</code>	Move/Rename file or directory
<code>rm [-r]</code>	Remove a file (or a directory with <code>-r</code> )

# Linux/Unix basic commands

<code>pwd</code>	Show the full path of current directory
<code>cat [file]</code>	View file, scrolling
<code>more</code>	View file, one screen at a time
<code>less</code>	Like more, more features
<code>diff [file1] [file2]</code>	Display differences between file1 & file2
<code>grep 'pattern' file</code>	Find a regular expression in a file

Example: Look for the pattern PBS in a file submit.job

```
$ grep 'PBS' submit.job
-->
$ #PBS -l nodes=1:ppn=4
$ #PBS -l walltime=2:00:00
```

# Pipes and redirection

Very useful feature of the command line.

<code>command &gt; file</code>	Redirect output of command to a file
<code>command &gt;&gt; file</code>	Redirect to file, append
<code>command &lt; file1 &gt; file2</code>	Get input from file1, direct output to file2
<code>command1   command2</code>	Pipe output of command1 to command2

Example: Take input from a data file, run an executable, and print to an output file

```
$ test.x < data.input > data.output
```



# Pipes and redirection

Example: Check your running jobs, and display only those which started on Wednesday

```
$ showq -u username | grep 'Wed'
```

→

```
1146997  username      Running      4  0:21:32:18  Wed Jan 02 08:36:49
1147924  username      Running      4  0:02:06:32  Wed Jan 02 11:11:03
```

If you use only `show -q`, you will see all the jobs including those that started on a different day than Wednesday.



# Some other useful aspects

Wildcards can replace a string of characters in the command line

- \* Match any string of characters, e.g. `ls foo*` will list all files starting with "foo".
- ~ Short for home directory, e.g. `cd ~` will change directory to home directory.
- .
- .. One directory up the tree, e.g. `cd ..` will change the directory one level up.

Using TAB key finishes the current command, filename, directory or shows any of that match the current string. Very useful for quick typing.

# Permissions

Permissions determine who can read a file or directory, write to it, and execute it.

```
-rwxr-xr-- 1 burak burak 656 Jul 14 10:05 qeinput.py
drwxr-xr-- 1 burak burak 656 Jul 14 10:05 inputs_folder
-rw-rw-r-- 1 burak burak 547 Jul 14 10:05 scf.in
```



-    - - -    - - -    - - -

- : is equal to d if directory
- - - : owner read, write, execute
- - - : group read, write, execute
- - - : others read, write, execute

# Permissions

Numerical values for each permission:

**Read = 4, Write = 2, Execute = 1**

Add the number value of the permissions you want to grant each group to make a three digit number, one digit each for the owner, the group, and the world. Use `chmod` command with the numerical value to assign the permission:

<code>chmod 600 filename</code>	<code>- rw- --- ---</code>	User can r&w (4+2)
<code>chmod 700 filename</code>	<code>- rwx --- ---</code>	User can r&w&x (4+2+1)
<code>chmod 644 filename</code>	<code>- rw- r-- r--</code>	User r&w (4+2), group r (4), others r (4)
<code>Chmod 755 filename</code>	<code>- rwx r-x r-x</code>	User r&w&x (4+2+1), group r&x (4+1), others r&x (4+1)

# Permissions

Another way:

Specifically change permissions with letters:

u = user      g = group      a = others  
r = read      w = write      x = execute

chmod u+rx filename      Give user r&x

chmod u+x filename      Give user x

chmod a+rw filename      Give others r&w

chmod a-x filename      Take x away from others

# Creating archives

- tar: Creating one archive from many files/folders. It does not lead to compression automatically
- zip: For compression
- bzip, gzip : compression algorithms

## Examples:

Create a gzipped tar archive `arch.tar.gz` from files `file1`, `file2` and the directory `directory0`

```
$ tar -czvf arch.tar.gz file1, file2, directory0
```

Extract `arch.tar.gz`

```
$ tar -xzvf arch.tar.gz
```

# Editors

- Several editors are available in Linux: vi, emacs, nano ...
- To edit a file, choose you favorite editor, e.g

```
$ vi file.txt
```

- Nano is recommended for beginners
- Vi and Emacs are more advanced

# File transfer

Several choices exist: SCP, SFTP, RSYNC

SCP usage:

```
$ scp user@host1:filepath2/filename1 user@host2:filepath2/filename2
```

Example (from your computer to a directory workdir on Knot):

```
$ scp file.txt user@knot.cnsi.ucsb.edu:~/workdir/file_copy.txt
```

You may need a client for transferring files from Windows, such as WinSCP

Globus Online is another option to use, see:

<http://csc.cnsi.ucsb.edu/docs/globus-online>



# Running Jobs

Running jobs that require large resources and time, requires submission to the queue

Job submission to the queue is done by the PBS scheduler

E.g. Suppose, you want to run an executable (`prog.x`) which is compiled in parallel, you will need a job submission script (`submit.job`):

```
submit.job
```

```
#!/bin/bash
#PBS -l nodes=1:ppn=4
#PBS -l walltime=2:00:00

export NCORES=4
cd $PBS_O_WORKDIR

mpirun -np $NCORES -machinefile -$PBS_NODEFILE prog.x
```

`$ qsub submit.job` → Job submission to queue

## submit.job

```
#!/bin/bash
#PBS -l nodes=1:ppn=4
#PBS -l walltime=2:00:00

export NCORES=4
cd $PBS_O_WORKDIR

mpirun -np $NCORES -machinefile -$PBS_NODEFILE prog.x
```

- `#!/bin/bash` : the shell you are using
- `#PBS -l nodes=1:ppn=4` : Asking one node, 4 processors per node
- `#PBS -l walltime=2:00:00` : Asking two hours of walltime
- `export NCORES=4` : set NCORES to value 4
- `cd $PBS_O_WORKDIR` : change directory to the one where job is submitted from
- `mpirun -np $NCORES ...` : Run the executable `prog.x` with `mpirun`

There are various PBS options available. Simply add them in your script if you need. Here is a list of some of them:

[https://www.olcf.ornl.gov/kb\\_articles/common-batch-options-to-pbs/](https://www.olcf.ornl.gov/kb_articles/common-batch-options-to-pbs/)

For a serial job to be submitted in queue, use

```
#!/bin/bash
#PBS -l nodes=1:ppn=1
```

To check the queue for the status of your job:

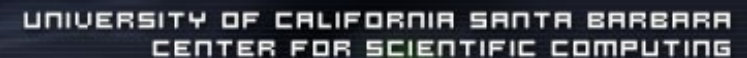
```
qstat -> detailed report, all jobs running
showq -> less detailed, all jobs running
qstat -u username / showq -u username -> report for the
user "username"
```

To cancel a submitted job, use the job id (a number) printed from showq or qstat:

```
$ qdel job_id
```

Some more information can be found on:

<http://csc.cnsi.ucsb.edu/docs/running-jobs-torque>

The logo for the Center for Scientific Computing (CSC) at the University of California Santa Barbara. It features the letters 'CSC' in a stylized, white, italicized font with a slight shadow effect, set against a dark background.The logo for the University of California Santa Barbara Center for Scientific Computing. It consists of the text 'UNIVERSITY OF CALIFORNIA SANTA BARBARA' and 'CENTER FOR SCIENTIFIC COMPUTING' in a white, sans-serif font, arranged in two lines. The background of the logo area is dark with some blurred light effects.

# Available Queues on Knot

- **Short queue:** `$ qsub -q short submit.job`
  - For jobs that run less than 1 hour
  - Shorter wait time in the queue
  - Ideal for short jobs and testing
- **Large memory:** `$ qsub -q largemem submit.job`  
`$ qsub -q xlargemem submit.job`
  - For jobs that need to run on fat (large memory) nodes
  - largemem: 256GB/node, xlargemem: 512Gb/node
- **Submitting jobs to GPUs:**  
`$ qsub -q gpuq submit.job`

# Compiling & linking libraries

- For compiling, first one needs to locate the compiler and set the environment variables:

Common environment variables:

**\$PATH** : It is used to define where to search for executable files. Type "echo \$PATH" to see which directories are listed.

**\$LD\_LIBRARY\_PATH** : Same as \$PATH, for searching libraries for linking.

- Check the environment variables using `env` command.
- Environment variables can be set in the `.bashrc` or `.cshrc` (depending on the shell you are using). This allows them to be specified every time you login, instead of setting them manually.

Below is an example for linking Intel compilers, MKL libraries and setting up openmpi as the MPI on Knot cluster:

.bashrc

```
# OPENMPI

export PATH=/opt/openmpi-1.6.4/bin/:/usr/local/bin/:/sw/bin:$PATH

# Use Intel's scripts to set environment variables in 64bit
architecture

. /opt/intel/composer_xe_2015/bin/compilervars.sh intel64
. /opt/intel/composer_xe_2015/mkl/bin/mklvars.sh intel64
```

This example uses Intel's scripts to automatically set environment variables for Intel compilers and MKL libraries.

We just need to specify the MPI implementation

Check executable locations using `which` command:

```
$ which mpirun
/opt/openmpi-1.6.4/bin/mpirun
```

```
$which ifort
/opt/intel/composer_xe_2015.2.164/bin/intel64/ifort
```

For using other compilers, libraries, search for them using `locate` command and find where they are located. Then, you can change your environment variables accordingly.

```
$ locate -b mpirun
/opt/intel/composer_xe_2013_sp1.0.080/mpirt.bac/bin/ia32/mpirun
/opt/intel/composer_xe_2013_sp1.0.080/mpirt.bac/bin/intel64/mpirun
/opt/intel/composer_xe_2013_sp1.0.080/mpirt.bac/bin/mic/mpirun
/opt/intel/composer_xe_2015.2.164/mpirt/bin/ia32/mpirun
/opt/intel/composer_xe_2015.2.164/mpirt/bin/intel64/mpirun
...
...
```