**High Performance Computing**

What is high performance computing (HPC)?

HPC involves the use of computers to complete a “job” as fast as possible. This often leads to using parallel processing and supercomputers to complete the job. These supercomputers have large numbers of multi-core processors tied together into a “computer cluster” for parallel processing purposes.

Holland Computing Center (HCC)

This center controls the supercomputers for the U. of Nebraska. It has locations at the Schorr Center on city campus (just south of Memorial Stadium) and the Peter Kiewit Institute on UNO’s south campus.

Supercomputers available ([https://hcc-docs.unl.edu/  
display/HCCDOC/HCC+Documentation](https://hcc-docs.unl.edu/display/HCCDOC/HCC+Documentation)):

| **Cluster** | **Overview** | **Processors** | **RAM** | **Connection** | **Storage** |
| --- | --- | --- | --- | --- | --- |
| **Crane** | 452 node Production-mode LINUX cluster | Intel Xeon E5-2670 2.60GHz 2 CPU/16 cores per node | \*64GB RAM per node | QDR Infiniband | ~1452 TB shared Lustre storage ~1.8 TB local scratch per node |
| **Tusker** | 106 node Production-mode LINUX cluster | Opteron 6272 2.1GHz, 4 CPU/64 cores per node | \*\*256 GB RAM per node \*\*\*2 Nodes with 512GB per node | QDR Infiniband | ~500 TB shared Lustre storage  ~500GB local scratch |
| **Sandhills** | 108 Node Production-mode LINUX cluster  (condominium model) | 62 4-socket Opteron 6376 (2.3 Ghz, 64 cores/node)  44 4-socket Opteron 6128 (2.0 Ghz, 32 cores/node)  2 4-socket Opteron 6168 (1.9 Ghz, 48 cores/node) | 62 nodes @ 192GB  44 nodes @ 128GB  2 nodes @ 256GB | QDR Infiniband Gigabit Ethernet | 175 TB shared Lustre storage  ~1.5TB per node |
| **Red** | 337 node Production-mode LINUX cluster | Various Xeon and Opteron processors models  5,888 cores maximum, actual number of  job slots depends on RAM usage | 1.5-4GB RAM per job slot | Gigabit and 10Gb  Ethernet | ~4.0PB of raw storage space |

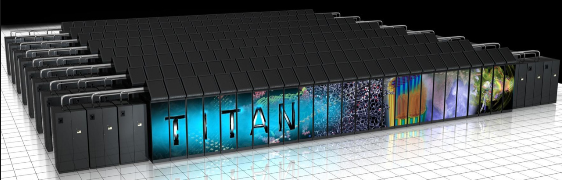
The word “cluster” is used because one of these supercomputers is a group of processors put together into one computer.

We are going to focus on HCC’s supercomputer named Tusker. Below are some specifics:

* Processors: There are 4 Opteron 6272 CPUs per node and each has 16 cores. Because there are 106 nodes, this means there are a total of 416106 = 6784 cores. Each core runs at 2.1GHZ.
* RAM: 256GB per node
* Connection: QDR Infiniband is the type of connection that nodes use to communicate
* Storage: Lustre is a portmanteau from Linux and cluster
* How fast is Tusker? 40TF – It can do 40 trillion simple calculations (like “x + y”) per second. The “TF” stands for “teraFLOPS”. “FLOPS” stands for FLoating-point Operations Per Second.

How does Tusker compare to the world’s fastest computers?

Please note that the world of supercomputers changes quickly! The world’s fastest computer in November 2012 was Titan ([http://www.olcf.ornl.gov/  
titan](http://www.olcf.ornl.gov/titan)) which is at Oak Ridge National Laboratory.



Its specifications are:

* 299,008 Opteron cores
* 18,688 NVIDIA TESLA K20 GPUs
* 18,688 nodes
* 17.6PF (petaFLOPS), which is 17,600 trillion calculations per second (17.6 quadrillion)

A NPR segment on Titan is available at <http://www.npr.org/blogs/alltechconsidered/2012/10/29/163894669/why-is-this-supercomputer-so-superfast>.

As of November 2014, Titan is the second fastest computer. First place belongs to the National Super Computer Center in Guangzhou

China (see [http://www.top500.org/featured/top-systems/tianhe-2-milkyway-2-national-university-of-defense](http://www.top500.org/featured/top-systems/tianhe-2-milkyway-2-national-university-of-defense/)). This computer has 3,120,000 cores which can do 33.9PF.

An example of increases in speed:

|  |  |
| --- | --- |
| **Date** | **Speed of 500th fastest computer** |
| November 2012 | 76.4TF |
| November 2013 | 117.8TF |
| November 2014 | 153.4TF |

New supercomputers that will be constructed with Federal funds: <http://www.reuters.com/article/2014/11/14/us-nvidia-supercomputers-idUSKCN0IY1MX20141114>. See <https://www.olcf.ornl.gov/summit/> for more on one of these computers.

Crane became fully operational in January 2014. It runs at 121.8TF. An e-mail from HCC announcing Crane said the following:

Crane is 474 on the current Top500 list at 121.8 TeraFLOPS, and supports over 1.5 PetaBytes of disk (/work). … This is roughly 3 times the machine that Tusker is in several metrics -- we anticipate it will quickly become a preferred resource for many algorithms. One aspect where Tusker will continue to be preferred is for large shared memory set (RAM) programs -- Tusker has 256 GB/node, Crane 64 GB/node. In each case this is 4 GB/core -- but Tusker has 64 AMD "cores" while Crane has 16 Intel cores per node.

How fast is a desktop/laptop computer compared to a supercomputer?

My desktop computer has an AMD Phenom II X6 1090T processor where each core runs at 3.2GHZ. There are six cores with one thread per core. From the web page, <http://www.tomshardware.com/reviews/fx-8150-zambezi-bulldozer-990fx,3043-14.html>, I found that the AMD Phenom II X6 1100T processor (3.3GHZ) runs at 57.7GF where “GF” is gigaFLOPS. Thus, it can perform 57.7 billion calculations per second.

To put this into prospective, my desktop computer is then 57.7/40,000 = 0.0014 as fast as Tusker and 57.7/17,600,000 = 0.0000033 as fast as Titan.

How do you obtain these speed advantages?

You need to use parallel processing. If you run all of your calculations on a single core, you will not see any significant advantages.

What is high throughput computing (HTC)?

HTC is for computations that can occur sequentially over time. This type of computing is more concerned with completing as many jobs as possible, not necessarily the speed in completion (<http://en.wikipedia.org/wiki/High-throughput_computing>). Computations are often distributed across independent computers to complete. These computers are not connected by a central bus and can be anywhere in the world.

UNL participates in The Open Science Grid (<http://www.opensciencegrid.org>) to do “grid computing”. This involves a collection of supercomputers from around the world that can be used. Therefore, if the computers at UNL are not enough, there are even more resources available! HCC’s Red supercomputer is a computer that participates in the grid. The Condor scheduling system is used to distribute computation jobs to these computers.

Additional information

* AMD’s HPC web page: [http://sites.amd.com/us/  
  business/it-solutions/compute-intensive-hpc/Pages/  
  compute-intensive-hpc.aspx](http://sites.amd.com/us/business/it-solutions/compute-intensive-hpc/Pages/compute-intensive-hpc.aspx)
* AMD Opteron 6200 series: [http://www.amd.com/us/  
  products/server/processors/6000-series-platform/  
  6200/Pages/6200-series-processors.aspx](http://www.amd.com/us/products/server/processors/6000-series-platform/6200/Pages/6200-series-processors.aspx)
* Video on FLOPS: [http://www.youtube.com/  
  watch?v=ZaJp6ZYP6Xc&feature=youtu.be](http://www.youtube.com/watch?v=ZaJp6ZYP6Xc&feature=youtu.be)
* Speed comparisons between an iPad2 and supercomputers: [http://www.tuaw.com/2011/05/09/  
  ipad-2-would-have-bested-1990s-era-supercomputers](http://www.tuaw.com/2011/05/09/ipad-2-would-have-bested-1990s-era-supercomputers); iPad2 is at 1.5-1.65GF, which is similar to the fastest computer in 1985.

Tusker account

In order gain access to Tusker, you need to request an account at [http://hcc.unl.edu/new-user-requests](http://hcc.unl.edu/newusers). Students need to be part of a research group, which is usually requested by a faculty member. For our class, I have set-up the stattools group that you can join.

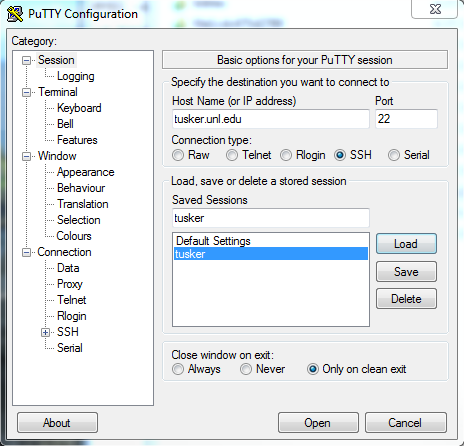
When the semester concludes, your access to stattools will be discontinued.

PuTTy

Using Tusker is like using a Windows-based computer’s command prompt interface. There is no nice windows-based environment where you can use a mouse. You must use text-based commands for everything.

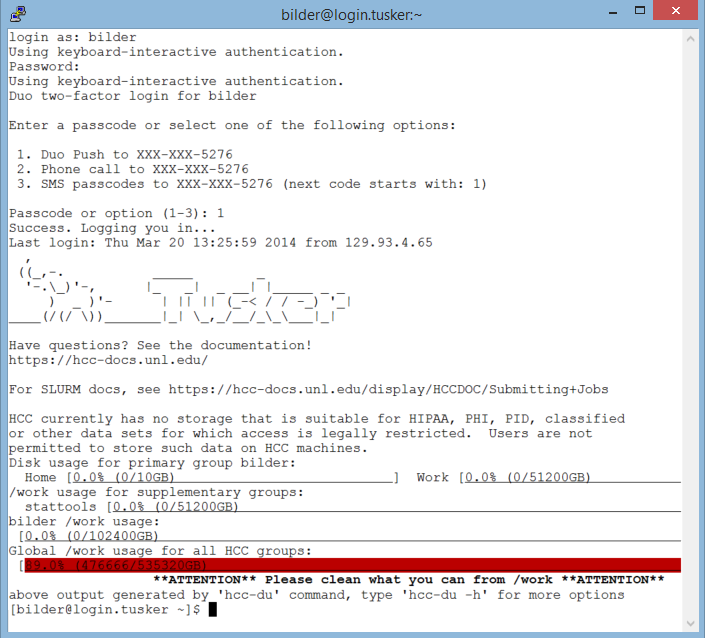
I use the program named “PuTTy” as a terminal interface to Tusker. It is a free program, and you can download it from <http://www.putty.org>. Simply run the executable file (putty.exe) whenever you want to use it (the program does not need to be “installed”).

When you start PuTTy, you will obtain a configuration window as shown below:

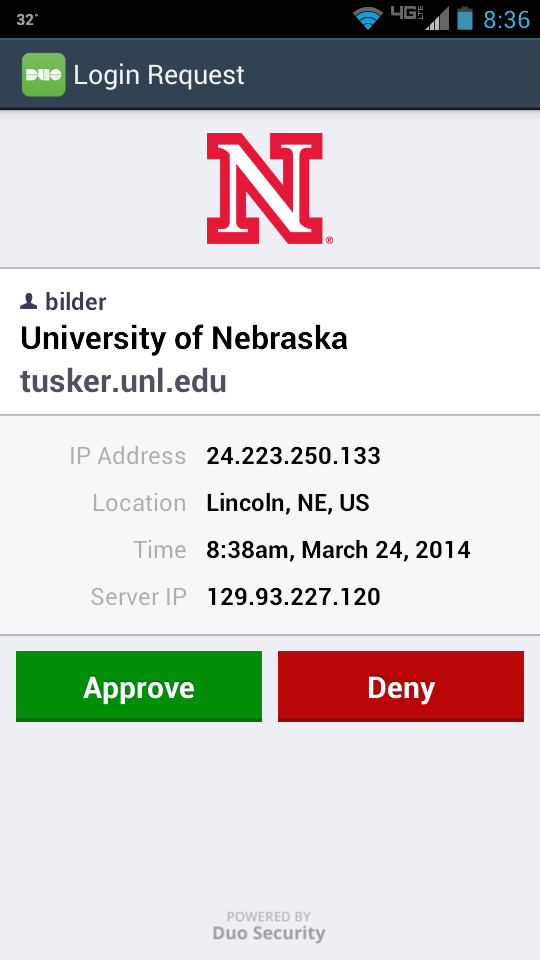


I have already filled in the tusker.unl.edu address in the Host Name box. I recommend you “Save” it so that you can more easily access it at a later time. Select Open to connect to Tusker.

There is a two-factor authentication process to log in (see [https://hcc-docs.unl.edu/display/HCCDOC/  
Setting+up+and+using+Duo](https://hcc-docs.unl.edu/display/HCCDOC/Setting+up+and+using+Duo)). First, you will be prompted for your login and password. Second, you will be asked to complete 1 of 3 options for an additional layer of security. For example, I use the Duo Push which sends a message to my Duo Mobile app on my phone to verify that it is me who is trying to log in. Below is what my Putty screen looks like:



Once I select “Accept” on my phone, I am allowed to log into Tusker.

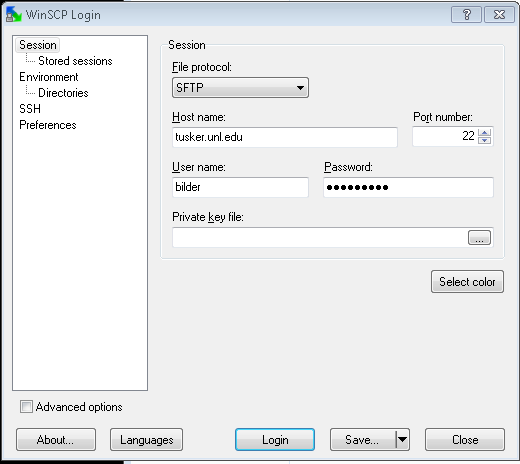


The $ command prompt is where you type commands to use Tusker. Note that I changed the default colors for the terminal window.

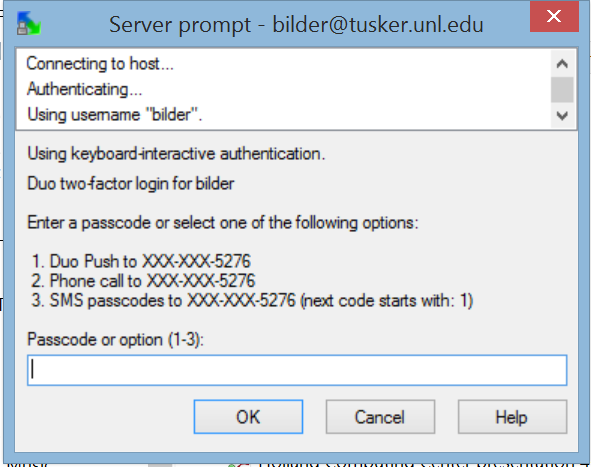
WinSCP

You will typically want to write and test programs on your own computer. Eventually, these programs will need to be transferred to Tusker. This can be done by using a FTP program.

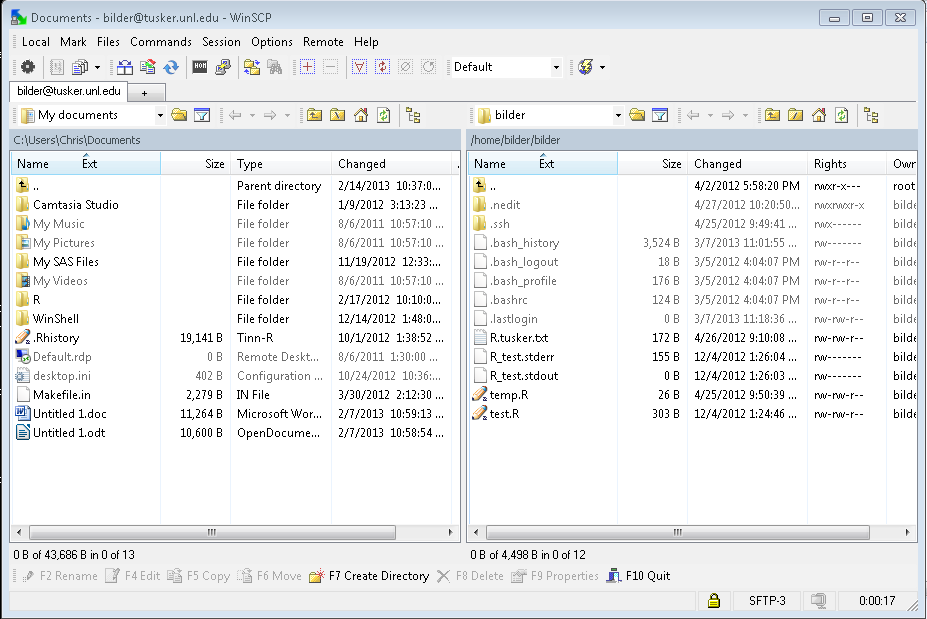
I use the WinSCP for transferring files. It is a free program, and you can download it from <http://sourceforge.net/projects/winscp>. After installing the program on my computer and opening it, I enter tusker.unl.edu as the host name and proceed with the same login and password used with PuTTy.



Below is a window showing the second stage of the authentication:



After logging in, you will obtain the following window:



The left side is the folder structure for your computer, and the right side is the folder structure for Tusker. To go up a folder level, click on the folder name. To go down a folder level, click on the .

To upload a file to Tusker, simply drag a file from anywhere on your computer (can be outside of the WinSCP interface) and drop it into the Tusker window. To download a file to Tusker, drag a file from the Tusker window to a location on your computer (can be outside of the WinSCP interface).

Tunnelier

An alternative to the PuTTy and WinSCP combination is to use a program named Tunnelier. This is also a free program for individuals to use, and it can be obtained from <http://www.bitvise.com/download-area>. Note that I have not been able to get this to work since the two factor authentication was implemented.

Basics of command line Unix/Linux

In order to communicate with the operating system, you use a shell. This essentially is like the command prompt interface in Windows. There are a number of different shells, and we will use a bash (Bourne-Again SHell) shell. A reference manual on its use is at

<http://www.gnu.org/software/bash/manual/bashref.html>.

Below are some basic commands that you can give at a command prompt. A listing and further explanation of the commands is available at [http://www.gnu.org/software/  
bash/manual/bashref.html#Shell-Builtin-Commands](http://www.gnu.org/software/bash/manual/bashref.html#Shell-Builtin-Commands). Note that many optional arguments can be given after the commands by using a dash and then an argument value.

* ls and dir – lists contents of the directory (folder) that you are in

[bilder@login.tusker ~]$ ls

R\_test.stderr R\_test.stdout R.tusker.txt temp.R test.R

[bilder@login.tusker ~]$ ls -l

total 7

-rw------- 1 bilder bilder 155 Dec 4 13:26 R\_test.stderr

-rw------- 1 bilder bilder 0 Dec 4 13:26 R\_test.stdout

-rw-rw-r-- 1 bilder bilder 172 Apr 26 2012 R.tusker.txt

-rw-rw-r-- 1 bilder bilder 26 Apr 25 2012 temp.R

-rw-rw-r-- 1 bilder bilder 303 Dec 4 13:24 test.R

[bilder@login.tusker ~]$

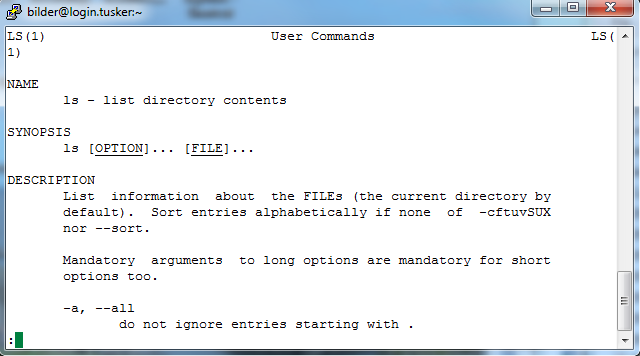
To copy the text from PuTTy, I highlighted the text with my mouse in PuTTy and then paste it into Word the usual way (do not use a CNTRL-C keyboard sequence in PuTTy to put the text on the clipboard).To paste text into PuTTy from a Windows application, highlight the text and use CNTRL-C to put the text on the clipboard. Right click the mouse in PuTTy to paste.

The tilde in [bilder@login.tusker ~]$ means that I am in my default (home) directory. This is the directory you are in after first logging into Tusker. Note that the name “home” is used a little differently shortly.

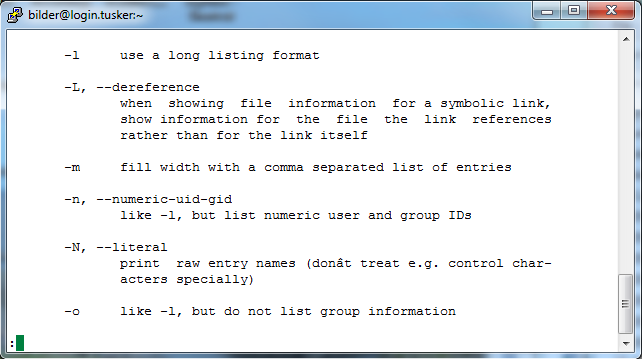
* man – Access the help manual for a command

[bilder@login.tusker ~]$ man ls

which leads to



You can page down in the manual by using the <Page down> button on your keyboard. The <Enter> button allows you to page down one line a time. Below is the entry for –l



To exit the manual, type “q” or CNTRL-Z. Note that the latter option is a common way to get back to a command prompt in other settings.

* pwd – returns your current directory

[bilder@login.tusker ~]$ pwd

/home/bilder/bilder

When you log into Tusker, you will be in a home directory with your group and user name. In the above case, I have a general group named bilder for my research group. My user name is also called bilder, so that’s why you see my name twice.

* cd – changes the directory that you are in

[bilder@login.tusker ~]$ cd .. #Moves one directory down

[bilder@login.tusker bilder]$ ls

bilder

[bilder@login.tusker bilder]$ cd ..

[bilder@login.tusker home]$ ls

baenziger bobaru jwang7 reid swanson unmc\_ngs

belashchenko choueiry ladunga rowe templin uno\_cs

benson deogun larsonm sabirianov tsymbal zeng

bilder hu lhoffman starace unmc\_cbsb

[bilder@login.tusker home]$ cd bilder #Moves up to bilder directory

[bilder@login.tusker bilder]$ ls

bilder

[bilder@login.tusker bilder]$ cd /work/stattools/bilder

#Moves to this directory

[bilder@login.tusker bilder]$ ls

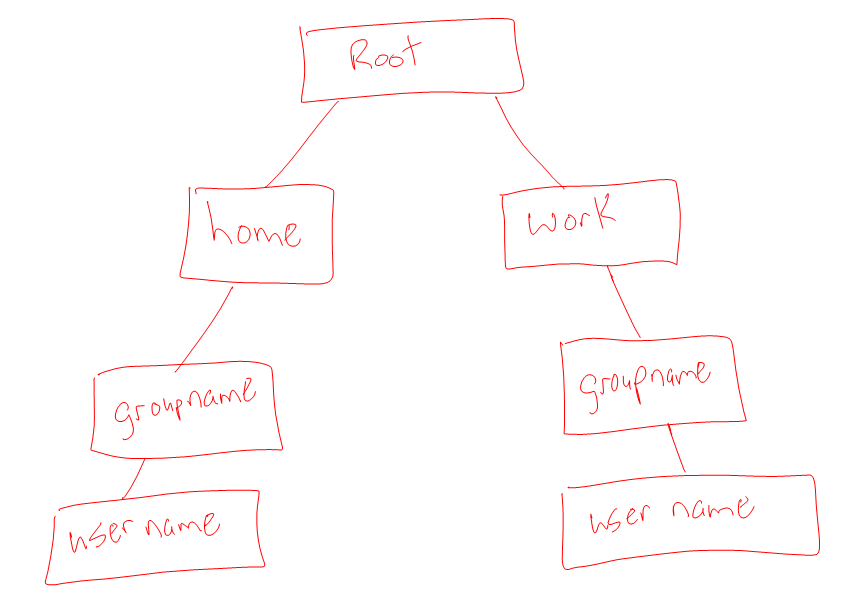
Hello\_world\_PBS.txt Hello\_world.r

[bilder@login.tusker bilder]$ pwd

/work/stattools/bilder

Notice the use of “#” to give comments within a command line.

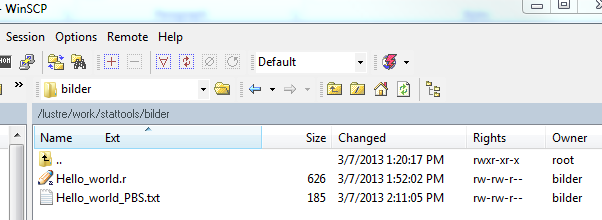
Below is a diagram of the directories that you will be using



The root directory is like “C:\” in Windows.

Notes:

* + All R programs should be run in the work/stattools/<user name> folder. Therefore, one of the first things you should do after logging in is to change your directory to this one. Note that this directory is not backed up. Rather than specifying the exact work directory location when using cd to get to this directory, you can also use cd $WORK.
  + Using the windows-based FTP programs allows you to see the folder structure better. For example,





shows my location to be root/work/stattools/  
bilder. The “lustre” part is just the name of the type of storage device (see p. HPC.2).

* mkdir – makes a directory
* rmdir – removes a directory

[bilder@login.tusker bilder]$ pwd

/work/stattools/bilder

[bilder@login.tusker bilder]$ ls

Hello\_world\_PBS.txt Hello\_world.r

[bilder@login.tusker bilder]$ mkdir temp

[bilder@login.tusker bilder]$ ls

Hello\_world\_PBS.txt Hello\_world.r temp

[bilder@login.tusker bilder]$ rmdir temp

[bilder@login.tusker bilder]$ ls

Hello\_world\_PBS.txt Hello\_world.r

* rm – removes a file
* cp – copies a file from one location to another
* mv – moves a file from location to another
* cat – prints the contents of a file
* head -6 – prints the first 6 lines of a file

[bilder@login.tusker bilder]$ head -6 Hello\_world.r

#Chris Bilder

# Introduction to using computers at HCC

cat("Hello world! \n")

#Simulate data for regression analysis

This can be convenient when you want to take a quick look at an output file (to be discussed later).

* tail -6 – prints the last 6 lines of a file
* \* can be used as a wildcard

[bilder@login.tusker bilder]$ ls H\*

Hello\_world\_PBS.txt Hello\_world.r

[bilder@login.tusker bilder]$ ls \*.r

Hello\_world.r

* passwd – change password
* module avail – shows what programs (or “modules”) are available

[bilder@login.tusker bilder]$ module avail

------- /util/opt/modulefiles/Core ----------------------

allinea/4.2 lsdyna/6.0.0

allinea/5.0 (D) lsdyna/6.1.2

ant/1.8 lsdyna/7.0.0

asreml/3.0 lsdyna/7.1.1 (D)

bam-readcount/0.6 maq/0.7

bamtools/2.3 mathematica/9.0

blast/2.2 matlab/r2012b

blast-legacy/2.2.26 matlab/r2013b (D)

blat/35x1 maven/3.2

bowtie/0.12.8 metaphlan/1.7

bowtie/0.12.9 mosaik/2.2

bowtie/1.0.0 mplus/6.12

bowtie/2.0.0-beta6 mplus/7.0 (D)

bowtie/2.1.0 mummer/3.23

bowtie/2.2 (D) muscle/3.8

breakdancer/1.4 mutect/1.1

bwa/0.7 mytaxa/1.0

clustal-omega/1.2 NCL/6.0dist

cmake/2.8 NCL/6.1dist (D)

compiler/gcc/4.6 oases/0.2.8

compiler/gcc/4.7 openbugs/3.2

compiler/gcc/4.8 (D) phobius/1.01

compiler/intel/11 phymmbl/4.0

compiler/intel/12 prodigal/2.60

compiler/intel/13 python/2.7 (D)

compiler/intel/15 (D) python/3.3

compiler/open64/4.5 qiime/1.7

compiler/pgi/11 R/2.15

compiler/pgi/12 R/3.0

compiler/pgi/13 R/3.1 (D)

compiler/pgi/14 (D) RAPSearch2/2.19

cplex/12.5 ruby/2.1

cuda/5.5 sage/6.3

cuda/6.0 samtools/0.1

cuda/6.5 (D) sas/9.3

cufflinks/2.0.2 scan-for-matches/1.0

cufflinks/2.1.1 scythe/0.991

cufflinks/2.2 (D) sga/0.10

cutadapt/1.4 shrimp/2.2

dmtcp/2.0 sickle/1.2

dmtcp/2.3 (D) signalp/4.1

espresso/5.0 snap/2013.11

exonerate/2.2 soapdenovo2/r240

express/1.5 SRAtoolkit/2.3

fastqc/0.10 tassel/3

fastx\_toolkit/0.0.14 tassel/4

gap/4.5 tassel/5.1 (D)

gaussian/09/RevD taxypro/1.0

gdl/0.9 taxytoolbox/1.0

gnuplot/4.6 tophat/2.0.5

HTSeq/0.5.4p5 tophat/2.0.8

HTSeq/0.6.1p1 (D) tophat/2.0.9

hugeseq/1.0 tophat/2.0.12 (D)

idba/1.1 transrate/0.1

intel-mkl/11 trf/4.0

intel-mkl/12 trinity/r2013-02-25

intel-mkl/13 trinity/r2013-11-10

intel-mkl/15 (D) trinity/r2014-04-13p1 (D)

interproscan/5.8 usearch/7.0

java/1.6 varscan/2.3

java/1.7 vcftools/0.1

java/1.8 (D) velvet/1.2

kracken/0.10 WRF/WRF

Where:

(D): Default Module

Use "module spider" to find all possible modules.

Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".

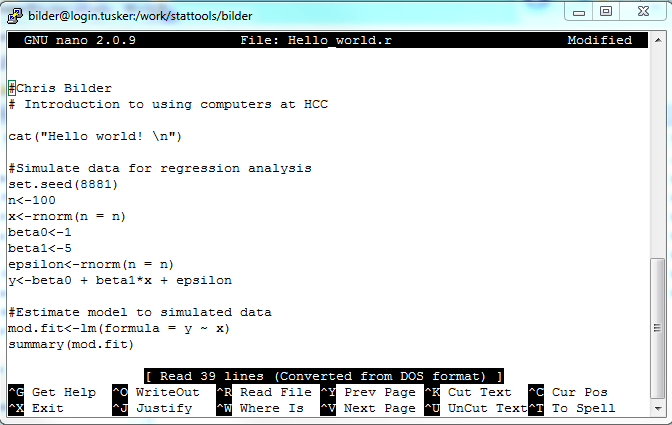
Notice that R and SAS are listed. Also, there are a number of compilers listed including “gcc” which contains a C++ compiler (<http://gcc.gnu.org>).

* Use up and down arrows to cycle through past commands issued at the command prompt
* nano – nano is a simple text based editor that can be used to view and edit files.

Suppose I have an R program named Hello\_world.r in my current directory. When I issue the command:

[bilder@login.tusker bilder]$ nano Hello\_world.r

I obtain the following window:

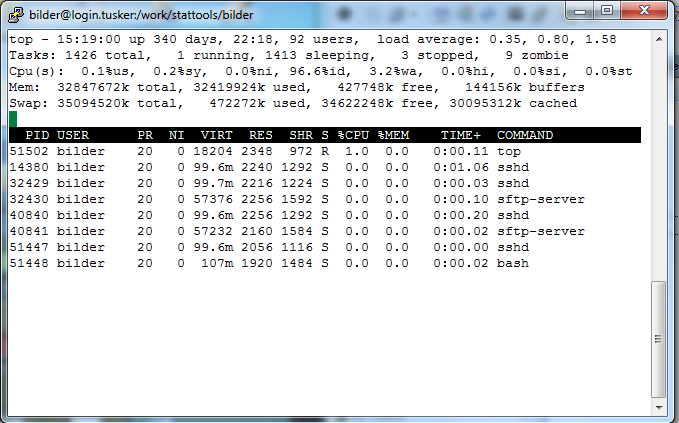


You can maneuver around in nano with your arrow keys and make changes to a file as you would expect. The CNTRL-o key sequence allows you to save the file. Note that I only use nano when I want to make a small change. Otherwise, I transfer the particular file back to my computer via FTP and edit with a better editor.

* top – lists all processes being run like the Windows Task Manager. When I issue the command:

[bilder@login.tusker test.boot]$ top -u bilder

my screen will bring up processes for the user bilder:



This updates every few seconds. Note that sshd under COMMAND corresponds to a “secure shell” program (PuTTy) and sftp-server under COMMAND corresponds to WinSCP. If R was running, you would see R under COMMAND. Under S, “R” means running and “S” means “sleeping”. Type “q” to exit from the window.

Similar commands to top include:

1. ps –u <username> shows same type of information as top but without the updating
2. jobs shows what actual programs are running

* exit or logout – logout

Note that it can be quicker to remove files, make directories, copy files, … by using WinSCP rather than typing Linux commands.

Additional information:

* NC State‘s ST 810 Advanced Computing course – see the lecture notes from 1/7; [http://www.stat.ncsu.edu/  
  people/zhou/courses/st810](http://www.stat.ncsu.edu/people/zhou/courses/st810)
* Nice introduction to Unix – [http://www.ee.surrey.ac.uk/  
  Teaching/Unix/index.html](http://www.ee.surrey.ac.uk/Teaching/Unix/index.html)
* Another introduction to Unix – [http://www.doc.ic.ac.uk/  
  ~wjk/UnixIntro/](http://www.doc.ic.ac.uk/~wjk/UnixIntro/)

Batch mode

When using Tusker, R code can not be run interactively like in an R Console window. All software programs need to be run in batch mode. This means an R program needs to be submitted to R for it to be run. R will produce an output file that contains the results when the program has completed running.

Details on running R in batch mode are available in Appendix B.1 of the “An Introduction to R” user manual.

Example: Hello world (Hello\_world.r)

Below is a simple R program:

*#Chris Bilder*

*# Introduction to using computers at HCC*

cat("Hello world! \n")

*#Simulate data for regression analysis*

set.seed(8881)

n<-100

x<-rnorm(n = n)

beta0<-1

beta1<-5

epsilon<-rnorm(n = n)

y<-beta0 + beta1\*x + epsilon

*#Estimate model to simulated data*

mod.fit<-lm(formula = y ~ x)

summary(mod.fit)

*#Create scatter plot with estimated regression model*

pdf(file = "plot.pdf")

plot(x = x, y = y)

curve(expr = mod.fit$coefficients[1] +

mod.fit$coefficients[2]\*x, col = "red", add = TRUE)

dev.off()

This program simulates 100 (x,y) observations, estimates a simple linear regression, and plots the data where the plot goes to a PDF file.

After transferring the program to work/stattools/  
bilder, I ran the following:

[bilder@login.tusker bilder]$ ls

Hello\_world.r

[bilder@login.tusker bilder]$ module load R/3.1 [bilder@login.tusker bilder]$ R CMD BATCH --vanilla

Hello\_world.r

[bilder@login.tusker bilder]$ ls

Hello\_world.r Hello\_world.r.Rout plot.pdf

Comments:

* The module command tells Tusker than you want to use R version 3.1 (actually, its 3.1.1). This command only needs to be given once per session.
* The R CMD BATCH tells Tusker that you want to run R in batch mode.
* By default, R will create a .RData file that contains ALL of the objects created while running R in batch mode. Also, if R sees a .RData file in the folder you are running the program from, R will automatically load it first into an R session. If you do not want either of these items to occur, use the --vanilla option. Other options are available, see the “An Introduction to R” user manual and [http://people.su.se/~lundh/  
  reproduce/batchmode.pdf](http://people.su.se/~lundh/reproduce/batchmode.pdf).
* Two additional files were produced from running the script:
  + Hello\_world.r.Rout – The R session output
  + plot.pdf – The requested plot by the program
* I had to wait for Tusker to complete running the program before I could access the command prompt again. This is simply due to computation time AND a submitted job is not necessarily executed immediately.
* If you would like to have access to the command prompt after submitting the program, use & after the file name:

[bilder@login.tusker bilder]$ R CMD BATCH --vanilla

Hello\_world.r &

However, there is a problem of knowing when the program has completed. You will need to check when the .Rout file is created to know when the program is done (use the FTP program and refresh the Tusker folder periodically) or use top at the command prompt.

I transferred the first two files back to my computer via FTP. I viewed my output file in Tinn-R because the formatting was not correct in Notepad:

R version 3.1.1 (2014-07-10) -- "Sock it to Me"

Copyright (C) 2014 The R Foundation for Statistical Computing

Platform: x86\_64-unknown-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.

You are welcome to redistribute it under certain conditions.

Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.

Type 'contributors()' for more information and

'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or

'help.start()' for an HTML browser interface to help.

Type 'q()' to quit R.

> #Chris Bilder

> # Introduction to using computers at HCC

>

> cat("Hello world! \n")

Hello world!

>

> #Simulate data for regression analysis

> set.seed(8881)

> n<-100

> x<-rnorm(n = n)

> beta0<-1

> beta1<-5

> epsilon<-rnorm(n = n)

> y<-beta0 + beta1\*x + epsilon

>

> #Estimate model to simulated data

> mod.fit<-lm(formula = y ~ x)

> summary(mod.fit)

Call:

lm(formula = y ~ x)

Residuals:

Min 1Q Median 3Q Max

-2.93307 -0.73927 0.00591 0.91998 2.01593

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 0.9063 0.1105 8.205 9.25e-13 \*\*\*

x 5.1988 0.1054 49.342 < 2e-16 \*\*\*

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.07 on 98 degrees of freedom

Multiple R-squared: 0.9613, Adjusted R-squared: 0.9609

F-statistic: 2435 on 1 and 98 DF, p-value: < 2.2e-16

>

> #Folder on my computer used to test program

> #setwd(dir = "C:\\chris")

> #Create scatter plot with estimated regression model

> pdf(file = "plot.pdf")

> plot(x = x, y = y)

> curve(expr = mod.fit$coefficients[1] + mod.fit$coefficients[2]\*x, col = "red", add = TRUE)

> dev.off()

null device

1

>

> #

>

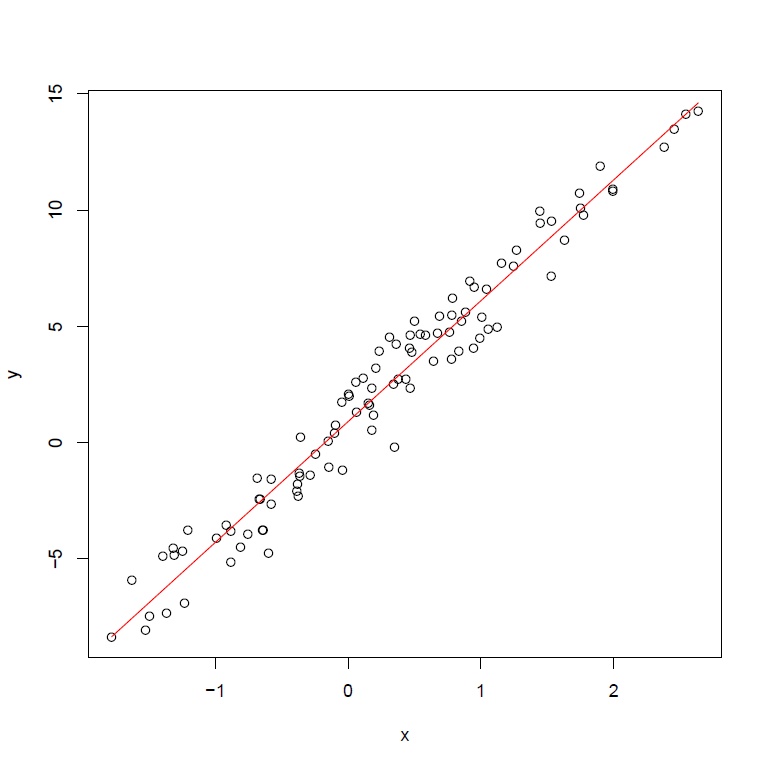
> proc.time()

user system elapsed

0.436 0.057 0.815

The proc.time() provides the amount of time that R was running.

Below is a screen capture from the PDF file:



An alternative way to run R in batch mode is to use the Rscript command at a command prompt. This will put all of the R output directly into the terminal window. Below is an example. Note that the -e argument stands for an executable expression.

[bilder@login.tusker bilder]$ Rscript -e "qnorm(p = 0.975)"

[1] 1.959964

[bilder@login.tusker bilder]$ Rscript Hello\_world.r

Hello world!

Call:

lm(formula = y ~ x)

Residuals:

Min 1Q Median 3Q Max

-2.93307 -0.73927 0.00591 0.91998 2.01593

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 0.9063 0.1105 8.205 9.25e-13 \*\*\*

x 5.1988 0.1054 49.342 < 2e-16 \*\*\*

---

Signif. codes: 0 â\*\*\*â 0.001 â\*\*â 0.01 â\*â 0.05 â.â 0.1 â â 1

Residual standard error: 1.07 on 98 degrees of freedom

Multiple R-squared: 0.9613, Adjusted R-squared: 0.9609

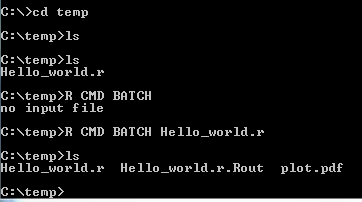
F-statistic: 2435 on 1 and 98 DF, p-value: < 2.2e-16

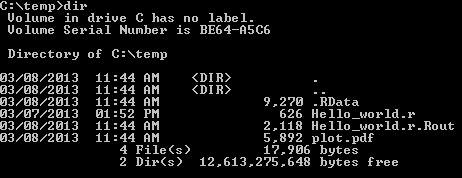
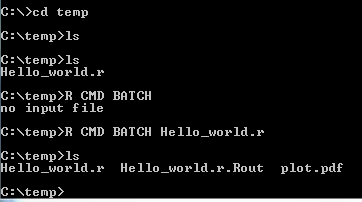
null device

1

The usefulness of Rscript may be limited. However, please note that it used with some R packages, such as knitr, that allows one to embed R code and output into LaTeX and LyX documents.

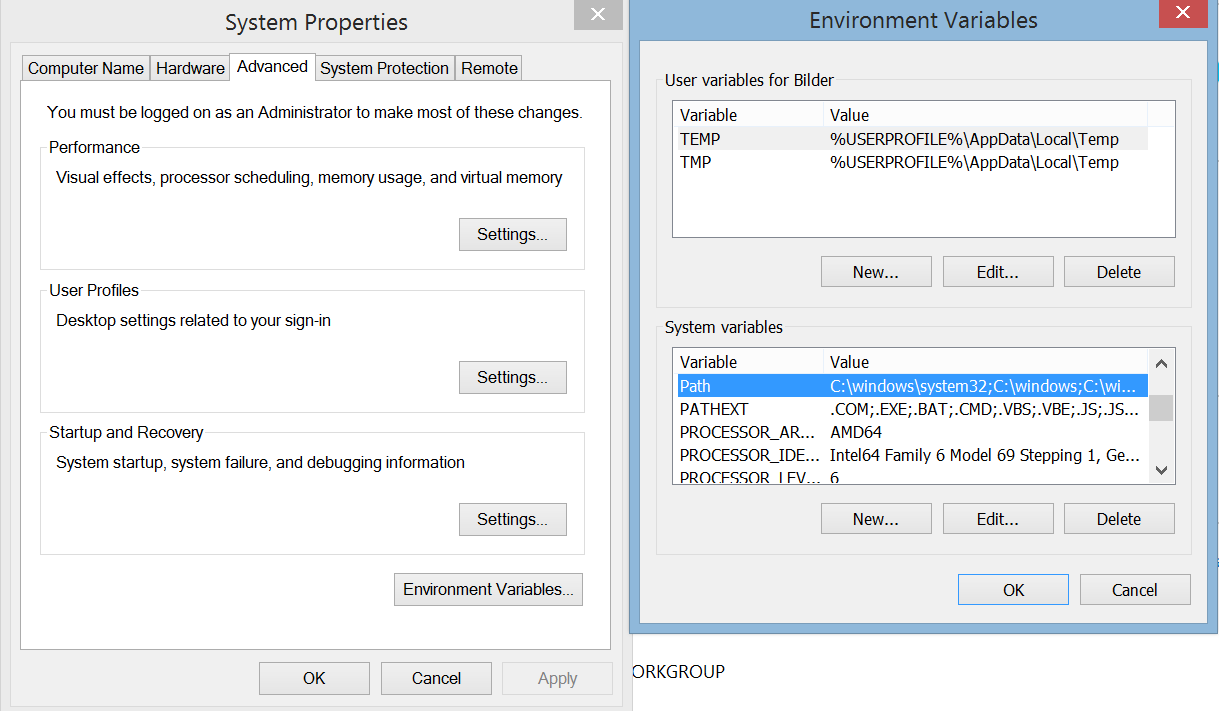
R can be run in batch mode on a Windows-based computer too! As a short example, suppose I run the same program in a folder named temp. Below is how the program can be run by using the command prompt window:

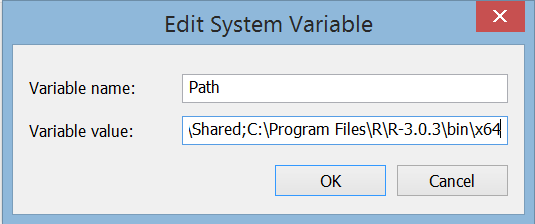




Comments:

* I ran the program this time without the --vanilla option to show you the .RData file was created.
* The above screen captures are from a Windows 7 computer. On my Windows 8.1 computer, ls no longer works! Instead, you will need to use dir.
* You may need to set a “path” so that Windows recognizes where R is located when trying to use it from a command prompt. I had to do this for my Windows 8 computer. This is done by opening the Control Panel and selecting System. Select Advanced System Settings > Advanced > Environmental Variables. Edit the Path variable under system variables:





If the command prompt window is already open, you will need to close it and then re-open it before running R programs.

* List of commands available at the command prompt: <http://pcsupport.about.com/od/commandlinereference/a/run-commands-windows-8.htm>

Installing R packages

Because R is run in batch mode, point-and-click methods for installing new packages are not available. Instead, you need to install them using code. Also, you can not install packages in the same directory where all of the default packages are installed. Instead, you should create a directory in work/stattools/  
<user name> for the new packages.

There are two important functions for installing and using packages that may be new to you:

* install.packages()downloads and installs a package into a desired directory
* .libpaths() puts on the search path a directory location for R to find installed packages

Example: Fortunes (install.r)

Below is a simple program that installs the package “fortunes”:

*#library() Shows a list of all packages installed*

*#Create new location for packages*

.libPaths(new = "/work/stattools/bilder/packages")

.libPaths() *#Check to see it is #1 in the search path*

*#Install a package*

install.packages(pkgs = "fortunes", repos =

"http://streaming.stat.iastate.edu/CRAN/",

destdir = "/work/stattools/bilder/packages")

library(package = fortunes)

fortune()

In order to keep my directories tidy, I created a directory in /work/stattools/bilder/ named packages which I can use for all packages installed. After I use FTP to upload the program into the directory, I issued the following commands:

[bilder@login.tusker packages]$ module load R/3.1 #If this

was not done already

[bilder@login.tusker packages]$ ls

install.r

[bilder@login.tusker packages]$ R CMD BATCH --vanilla

install.r

[bilder@login.tusker packages]$ ls

fortunes fortunes\_1.5-0.tar.gz install.r install.r.Rout

Below is the output from install.r.Rout:

> #Create new location for packages

> .libPaths(new = "/work/stattools/bilder/packages")

> .libPaths() #Check to see it is #1 in the search path

[1] "/lustre/work/stattools/bilder/packages"

[2] "/util/opt/R/3.1.3/gcc/4.4.7/64/lib64/R/library"

>

> #Install a package

> install.packages(pkgs = "fortunes", repos =

"http://streaming.stat.iastate.edu/CRAN/",

destdir = "/work/stattools/bilder/packages")

Installing package into ‘/lustre/work/stattools/bilder/packages’

(as ‘lib’ is unspecified)

trying URL 'http://streaming.stat.iastate.edu/CRAN/src/contrib/fortunes\_1.5-2.tar.gz'

Content type 'application/x-gzip' length 182533 bytes (178 Kb)

opened URL

==================================================

downloaded 178 Kb

\* installing \*source\* package â€˜fortunesâ€™ ...

\*\* package â€˜fortunesâ€™ successfully unpacked and MD5 sums checked

\*\* R

\*\* inst

\*\* preparing package for lazy loading

\*\* help

\*\*\* installing help indices

\*\* building package indices

\*\* installing vignettes

â€˜fortunes.Rnwâ€™

\*\* testing if installed package can be loaded

\* DONE (fortunes)

> library(package = fortunes)

> fortune()

SAS seems to be to statistical computing what Microsoft is to personal computing.

-- Bill Venables

'Exegeses on Linear Models' paper (May 2000)

> proc.time()

user system elapsed

1.532 0.256 2.530

R downloaded fortunes\_1.5-2.tar.gz, which is a form of a zip file, from the Iowa State CRAN mirror and unzipped it. Notice a directory named fortunes is now available. Whenever I want to use the fortunes package in the future, I need to use

.libPaths(new = "/work/stattools/bilder/packages")

in my program (the program can be in any directory), so that R knows to check this location for packages before using library(fortunes).

Parallel processing

We can use the same code as from past notes!

Example: Estimate true confidence level for a confidence interval using parallel package (MC\_sim\_HPC\_parallel.R, Run\_on\_each\_core.R)

The purpose of this example is to re-run simulation examples from the parallel processing section of this course.

I executed the MC\_sim\_HPC\_parallel.R program using

[bilder@login.tusker simpp]$ module load R/3.1 #If this

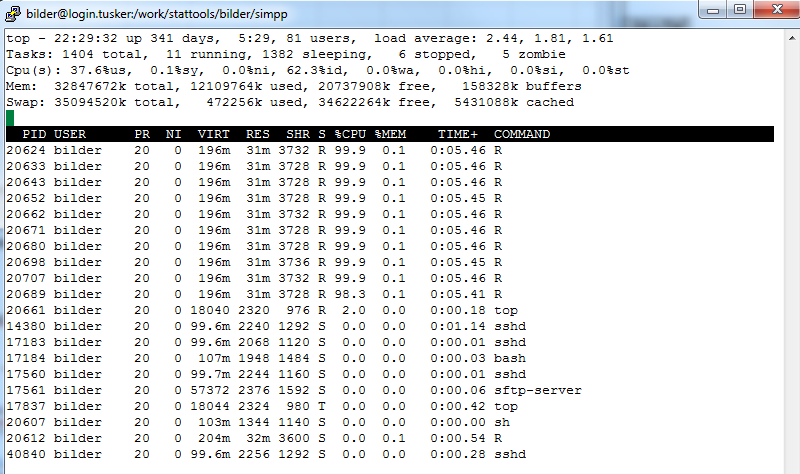
was not done already

[bilder@login.tusker simpp]$ R CMD BATCH --vanilla MC\_sim\_HPC\_parallel.R &

and watched the number of cores used with

[bilder@login.tusker simpp]$ top -u bilder

Below is what the results from top look like when I am using 10 cores:



There are 11 R processes being run. The first 10 are each computing the intervals for 500/10 = 50 data sets. The last R session is the original session where the calculations will continue after the parallel processing part is complete.

Below is a plot summarizing the time it takes for these calculations to complete.



The time to completion values level off. This is likely due to 1) communication time among cores, 2) the non-parallel processing portion of the program, and 3) only 64 cores are available on a single node (sometimes only 32 on this particular node – more on this later).

When running a program that takes a long time to complete (e.g., a long Monte Carlo simulation), there will be times when you would like to start running the program and then immediately exit Tusker. You can then come back to Tusker later to retrieve the results from the completed program. I have found no problem with doing this on Tusker; however, a number of other Linux resources say to run the program using the nohup command. I have not been able to get this command to work on Tusker though.

Example: Estimate true confidence level for a confidence interval using foreach package (MC\_sim\_HPC\_foreach.R, install\_foreach.R, Run\_on\_each\_core.R)

I had difficulty getting the foreach() function to work here. After installing the foreach and doParallel packages (make sure the iterators package is installed too), I ran my simulation code as we used with Windows. However, I always obtained the following error messages in my .Rout file:

> save.res2<-foreach(i = 1:R, .combine = rbind) %dopar% {

sim.func(y = y.sim[i,])

}

Error in unserialize(node$con) : error reading from connection

Calls: %dopar% ... FUN -> recvData -> recvData.SOCKnode -> unserialize

Execution halted

After discussing the problem with the HCC support, they said that R appears to forget about the location where foreach and doParallel are located. They suggested setting an “environment variable” to prevent this from happening. The easiest way to set this variable is in a file named “.bash\_profile” located in your home directory that you log in to. The file is hidden when using a simple ls -l to view your files, but can be seen with ls -all (it is always visible in WinSCP):

[bilder@login.tusker ~]$ ls -l

total 8

-rw-rw-r-- 1 bilder bilder 72 Mar 21 20:55 MC\_sim2014.Rout

-rw------- 1 bilder bilder 155 Dec 4 2012 R\_test.stderr

-rw------- 1 bilder bilder 0 Dec 4 2012 R\_test.stdout

-rw-rw-r-- 1 bilder bilder 172 Apr 26 2012 R.tusker.txt

-rw-rw-r-- 1 bilder bilder 26 Apr 25 2012 temp.R

-rw-rw-r-- 1 bilder bilder 303 Dec 4 2012 test.R

[bilder@login.tusker ~]$ ls -all

total 29

drwx--x--- 4 bilder bilder 16 Mar 21 20:55 .

drwxr-x--- 4 root bilder 3 Apr 2 2012 ..

-rw------- 1 bilder bilder 15698 Mar 21 20:56 .bash\_history

-rw-r--r-- 1 bilder bilder 18 Mar 5 2012 .bash\_logout

-rw-r--r-- 1 bilder bilder 229 Mar 21 20:50 .bash\_profile

-rw-r--r-- 1 bilder bilder 124 Mar 5 2012 .bashrc

-rw-rw-r-- 1 bilder bilder 0 Mar 21 20:56 .lastlogin

-rw-rw-r-- 1 bilder bilder 72 Mar 21 20:55 MC\_sim2014.Rout

drwxrwxr-x 2 bilder bilder 2 Apr 27 2012 .nedit

-rw------- 1 bilder bilder 155 Dec 4 2012 R\_test.stderr

-rw------- 1 bilder bilder 0 Dec 4 2012 R\_test.stdout

-rw-rw-r-- 1 bilder bilder 172 Apr 26 2012 R.tusker.txt

drwx------ 2 bilder bilder 5 Apr 25 2012 .ssh

-rw-rw-r-- 1 bilder bilder 26 Apr 25 2012 temp.R

-rw-rw-r-- 1 bilder bilder 303 Dec 4 2012 test.R

-rw------- 1 bilder bilder 1529 Mar 21 14:17 .viminfo

When you log on to Tusker, this file is immediately executed. Below is my file where I added the last line to indicate where my packages are located:

*# .bash\_profile*

*# Get the aliases and functions*

if [ -f ~/.bashrc ]; then

. ~/.bashrc

fi

*# User specific environment and startup programs*

PATH=$PATH:$HOME/bin

export PATH

export R\_LIBS=/lustre/work/stattools/bilder/packages

With this addition, I no longer need to use .libpaths() in any program as long as I install packages into the specified directory.

Shell script file and SLURM

Tests of programs can be run in Tusker as shown so far. These programs have been run on the Tusker node that everyone logins. Note that this core sometimes only has 32 cores available. For heavy use of Tusker (e.g., a simulation study), programs should be run via a shell script file (see p. HPC.16 for what a “shell” is). This is a plain text file where R CMD BATCH is used in the same manner as before. Also, additional commands need to be given for the Simple Linux Utility for Resource Management (SLURM). This manager finds a node to put your program to run on. Note that the program may not be run immediately if there are not nodes available.

An example script file is given at: <https://hcc-docs.unl.edu/display/HCCDOC/Submitting+R+jobs>



I will discuss a script file specifically in the next example.

The script file is submitted for execution to the job queue by using the sbatch command at a command prompt. Thus,

sbatch script\_file

submits “script\_file” to the job queue for SLURM. You can check the status of the queue (Is your job done?) by using

squeue –u <user name>

If needed, you can use

scancel <identification number>

to remove the job from the queue, where the identification number will be given in the output from squeue. Additional commands for SLURM are shown at <https://computing.llnl.gov/linux/slurm/quickstart.html>.

Example: Hello world (Hello\_world\_SLURM.txt, Hello\_world.r)

Below is my script file:

#!/bin/sh

#SBATCH --ntasks-per-node=1

#SBATCH --nodes=1

#SBATCH --time=00:01:00

#SBATCH --mem-per-cpu=1024

#SBATCH --job-name=Test

#SBATCH --error=Hello\_world.stderr

#SBATCH --output=Hello\_world.stdout

module load R/3.1

R CMD BATCH --vanilla Hello\_world.r

Comments:

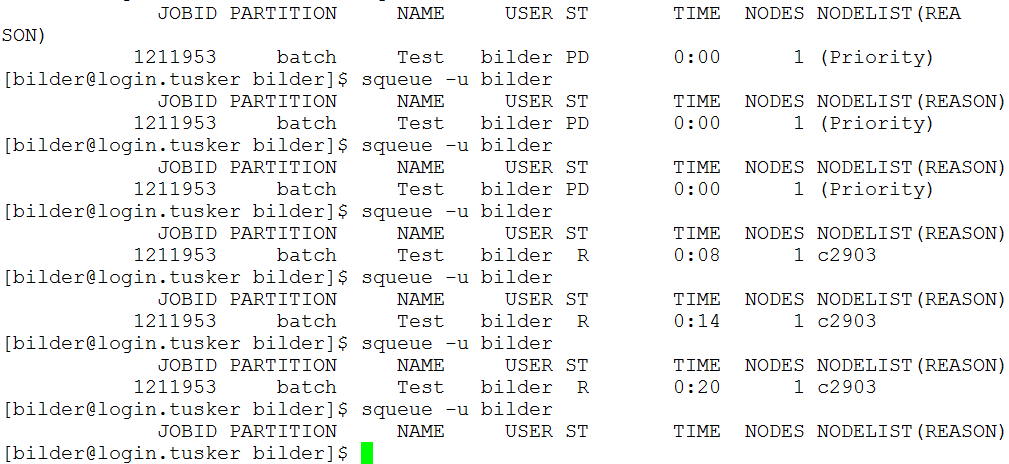
* The first line simply indicates this is a shell script file. The “#!” is referred to has a “sha-bang”.
* ntasks-per-node – Number of cores in a node that you want to use
* nodes – Number of nodes that you want to use
* time – maximum number of hours:minutes:seconds you are giving your program to finish. If it does not finish within the allotted time, SLURM will terminate the program.
* Mem-per-cpu – The amount of memory (in MB) that you want available for each core. Thus, I specified 1 GB here. If I went over this amount, SLURM will terminate the program.
* job-name – This is what will appear when using squeue to see the list of jobs running. This is just an identifier to be shown in the queue, and it does not need to be the actual R program name.
* Hello\_world.stdout and Hello\_world.stderr are files that correspond to your script file for output and errors, respectively. Unless you are using a script file differently from here or if you have a script code error, nothing will go into these files.
* module load R/3.1 allows us to use R.
* R CMD BATCH --vanilla Hello\_world.r works like we saw earlier in the batch mode section.
* Additional information is given at <https://hcc-docs.unl.edu/display/HCCDOC/Submitting+Jobs>.

While at a command prompt within the directory that contains both files, I use the following commands to submit my script to SLURM and check on its completion:

[bilder@login.tusker bilder]$ sbatch Hello\_world\_SLURM.txt

Submitted batch job 1211953

[bilder@login.tusker bilder]$ squeue -u bilder



bilder@login.tusker bilder]$ ls

Hello\_world\_PBS.txt Hello\_world.r.Rout Hello\_world.stdout

Hello\_world.r Hello\_world.stderr plot.pdf

Comments:

* Because the program would normally take a very short period of time to complete, I added Sys.sleep(20) at the end of it. This code causes the program to run for 20 seconds longer than it normally would, which helps us see the status of the program on the queue.
* The program was not immediately run. In the table generated by squeue, we have under “ST”: “PD” means the script is pending to be run and “R” means the script is running.
* Four additional files were produced from running the script:
  + Hello\_world.r.Rout – The R session output
  + plot.pdf – The requested plot by the program
  + Hello\_world.stderr – Errors generated by the script; this is an empty file
  + Hello\_world.stdout – Output generated by the script; this is an empty file

While not necessary, you can add the following lines to your script:

#SBATCH --mail-type=END

#SBATCH --mail-user=bilder@unl.edu

to have Tusker send an e-mail to the specified e-mail address when the program has ended. Of course, change the e-mail address above to your own!!! I have used this type of notification mechanism throughout my career when I have long simulations to complete.

Rather than executing squeue many times to see when a job gets completed, you can use the watch command. For example,

watch squeue -u bilder

will show you the usual output from squeue, but the output is updated every two seconds. You can use CNTRL-Z to exit from watching.

Because the Linux and Windows operating systems represent line breaks as different characters (you will not see these characters in Windows), Linux can have difficulty with Windows files. For example, when I first tried to run Hello\_world\_SLURM.txt, I had the following result:

[bilder@login.tusker bilder]$ sbatch Hello\_world\_SLURM.txt

sbatch: error: Batch script contains DOS line breaks (\r\n)

sbatch: error: instead of expected UNIX line breaks (\n).

There are two ways to handle this problem (2nd way is likely easier):

1. Use the dos2unix command on the shell script file:

[bilder@login.tusker bilder]$ dos2unix Hello\_world\_SLURM.txt

dos2unix: converting file Hello\_world\_SLURM.txt to UNIX format ...

1. Specify in your text editor to use Linux line breaks. In Tinn-R, this is done by selecting Encoding > Change line endings (delimiter) > UNIX (LF). Now, the file will be of the correct format. In the left hand corner of Tinn-R, you will see the format listed



You can do the same change in other text editors, although not Notepad. HCC recommends using Notepad++ for general editing and making the change in it.

Example: Estimate true confidence level for a confidence interval (MC\_sim\_HPC\_parallel20only.R, MC\_sim\_HPC\_parallel20only.txt)

Below is my script file. The parallel package is for parallel processing on ONE node, so the third line is changed here allowing me to use 20 cores. A few additional lines were changed as well.

#!/bin/sh

#SBATCH --time=00:15:00

#SBATCH --ntasks-per-node=20

#SBATCH --nodes=1

#SBATCH --mem-per-cpu=4096

#SBATCH --job-name=sim

#SBATCH --mail-type=END

#SBATCH --mail-user=bilder@unl.edu

#SBATCH --error=sim20.stderr

#SBATCH --output=sim20.stdout

module load R/3.1

R CMD BATCH --vanilla MC\_sim\_HPC\_parallel20only.R

I submitted the shell file the same way as before. The squeue command with the –o %C option shows that 20 cores are requested on 1 node

[bilder@login.tusker simpp]$ squeue -u bilder -o %C

CPUS

20

The R program did not start immediately because the necessary resources were not available.

Example: R code within the script file

You can also put your R code within the script file. The R part of the file is

module load R/3.1

R --vanilla <<EOF

qnorm(0.975)

EOF

and this should put output in the .stdout file. However, I have not been able to get the code to work since HCC switched to SLURM in 2013.

Multiple nodes

In order to use cores from other nodes, the Rmpi or snow packages can be used. I have not tried to do this yet. Also, there may not be anyone who has tried it using the HCC computers (beyond the examples in the help for Rmpi) according to HCC support.

I have installed the Rmpi package. The installation process was not as simple as it should be! HCC support helped me install it through the following process at a command prompt:

$ module load compiler/gcc openmpi/1.6 R/3.0

$ wget http://rweb.quant.ku.edu/cran/src/contrib/Rmpi\_0.6-3.tar.gz

$ LIBRARY\_PATH=/util/opt/openmpi/1.6/gcc/4.7/lib C\_INCLUDE\_PATH=/util/opt/openmpi/1.6/gcc/4.7/include R CMD INSTALL --configure-args=--with-Rmpi-type=OPENMPI -l /work/stattools/bilder/packages Rmpi\_0.6-3.tar.gz

This code uses a C++ compiler and the openmpi program along with R to install the package.

SAS

While HCC prefers free programs, SAS is available on Tusker. An introduction to using SAS on it is available at <http://www.lesahoffman.com/944/HCC_Statistical_Computing_Instructions_for_SAS.pdf>, which was created by Lesa Hoffman when she was in the UNL Department of Psychology.

Example: Old SAS program (sas.zip)

I created this program in 2002 when teaching a SAS Programming class at OSU ([http://statistics.unl.edu/  
faculty/bilder/stat4091](http://statistics.unl.edu/faculty/bilder/stat4091)). I made no attempts to change any parts of it.

All of my work is done in a new directory that I named “sas\_example” to help organize my work:

[bilder@login.tusker sas\_example]$ ls

general\_func.sas

[bilder@login.tusker sas\_example]$ module load sas/9.3

[bilder@login.tusker sas\_example]$ sas general\_func.sas

No device name has been given--please enter device name: pdf

[bilder@login.tusker sas\_example]$ ls

general\_func.log sasgraph.pdf

general\_func.lst general\_func.sas

The .log and .lst files contain the contents of the log and output windows, respectively. The PDF file came from a plot that the program produced. Note that there are some errors given in the .log file which are produced when reading in data with a datalines statement. These errors are not present when running the program in the windows version of SAS.

More about using SAS in batch mode is available at <http://support.sas.com/documentation/cdl/en/hostwin/63047/HTML/default/viewer.htm#p16esisc4nrd5sn1ps5l6u8f79k6.htm>.

C++

There is a module available for a GNU C++ compiler.

Example: Two C++ programs (cplusplus.zip)

These are two simple programs that give a “Hello World” and a conversion of an age in years to days. All of my work is done in a new directory that I named “cplusplus” to help organize my work:

[bilder@login.tusker bilder]$ module load

compiler/gcc/4.7

[bilder@login.tusker cplusplus]$ ls

age.cpp hello.cpp

[bilder@login.tusker cplusplus]$ g++ hello.cpp -o hello1 [bilder@login.tusker cplusplus]$ ls

age.cpp hello1 hello.cpp

[bilder@login.tusker cplusplus]$ ./hello1

Hello World!

Wasn't that easy?

[bilder@login.tusker cplusplus]$ g++ age.cpp -o ageprog

[bilder@login.tusker cplusplus]$ ./ageprog

Enter your age in years: 40

You are 14600 days old!

The -o option corresponds to the outfile produced. For example, hello1 is the outfile executable produced by the compiler.

At first, I tried to use gcc rather than g++ when compiling, but I received error messages. Also, you may not actually need to load the module! I logged out and logged back into Tusker to verify my code would work a second time, and I did not need to submit the module command to get it to work.

Amazon Elastic Compute Cloud (Amazon EC2)

This is a pay-as-you-go form of using supercomputers, but there is a free usage tier too (which is slow).

Below are some useful links:

* Amazon EC2: <http://aws.amazon.com/ec2>
* Search on the Revolutions blog for “Amazon EC2” to see a number of posts. One post is at <http://blog.revolutionanalytics.com/2009/05/running-r-in-the-cloud-with-amazon-ec2.html>.
* Step-by-step sequence to run R: <http://decisionstats.com/2010/09/25/running-r-on-amazon-ec2>
* RStudio:
  + [http://toreopsahl.com/2011/10/17/securely-using-r-and-rstudio-on-amazons-ec2](http://toreopsahl.com/2011/10/17/securely-using-r-and-rstudio-on-amazons-ec2/)
  + <http://www.louisaslett.com/RStudio_AMI>
  + [http://inundata.org/2011/03/30/r-ec2-rstudio-server](http://inundata.org/2011/03/30/r-ec2-rstudio-server/)

Additional resources

* Other university help web pages for using Linux and R:
  + Iowa State: [http://streaming.stat.iastate.edu/wiki/  
    index.php/Running\_Jobs\_in\_the\_Background](http://streaming.stat.iastate.edu/wiki/index.php/Running_Jobs_in_the_Background)
  + Missouri: [http://umbc.rnet.missouri.edu/resources/  
    How2RunR.html](http://umbc.rnet.missouri.edu/resources/How2RunR.html)
* Some current research in the area: <http://blog.revolutionanalytics.com/2015/02/the-hp-workshop-on-distributed-computing-in-r.html>