1. Ruby Versions
2. Install and Manage Ruby Gems (aka packages)
3. Example Script
4. Contributing New Examples

Ruby is a dynamic, fully object oriented, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.

1 Ruby Versions

ACCRE supports several version of ruby on the cluster. The oldest version 1.8.7 will be retired in 2015 because it is not no longer supported with security updates or development. To see the versions you need only do:

```
pkginfo | grep "^\s*ruby"
                           ruby   Ruby 2.1.1
                         ruby1.8   Ruby 1.8.7
                       ruby1.9.3   Ruby 1.9.3
                       ruby2.0.0   Ruby 2.0.0
                       ruby2.2.0   Ruby 2.2.0
                  ruby2.2.0_intel   Ruby 2.2.0
```

To select the version you want, say ruby version 2.0.0, you need only do:

```
setpkgs -a ruby2.0.0
```

You will note that there is a version of ruby compiled with the new Intel compiler. This is the version we encourage people to use because the binary takes advantage of the features of the Intel chip set.

2 Install and Manage Ruby Gems (aka packages)

Ruby gems are easy to add to a local repository in your home directory, just remember to use the `--user-install` option when you download the gem. For example:

```
gem install text-table --no-rdoc --no-ri --user-install
```

will download and install the gem into your home directory in a a directory named .gem, and if that directory does not exist, gem will create it for you. This example shows how to avoid installing the documentation for the gem.

To see the list of gems currently installed both system-wide and in your local repository do:

```
gem list
```

```
*** LOCAL GEMS ***

bigdecimal (1.2.6)
io-console (0.4.3)
json (1.8.1)
minitest (5.4.3)
power_assert (0.2.2)
psych (2.0.8)
rake (10.4.2)
rdoc (4.2.0)
test-unit (3.0.8)
text-table (1.2.4)
```

It is important to remember that gems are sensitive to the version of ruby you are using. If you are using 2.0.0, for example, and download and install a gem, that gem will **not** be available if you switch to a different version.

3 Example Script

Here is a sample script, which I wrote to make the output of sacct more managble. To see the output from `sacct` do this with a jobid (note that the command is a single long line):

```
sacct --format=user,jobid,account,jobname,
state,exitcode,timelimit,reqmem,maxrss,reqcpus,ncpus,nnodes,nodelist,
submit,start,end --jobs 2376732
User         JobID    Account    JobName      State ExitCode  Tim
elimit     ReqMem     MaxRSS  ReqCPUS      NCPUS   NNodes       Node
List               Submit               Start                  End
--------- ------------ ---------- ---------- ---------- -------- ----
------ ---------- ---------- -------- ---------- -------- -----------
---- ------------------- ------------------- -------------------
msrc 2376732       mass_spec PROD15982+  COMPLETED      0:0   02
:00:00     1000Mn                   1          1        1          vm
p515 2015-06-08T14:52:26 2015-06-08T14:52:27 2015-06-08T14:53:48
           2376732.bat+  mass_spec      batch  COMPLETED      0:0
           1000Mn     13532K        1          1        1          vm
p515 2015-06-08T14:52:27 2015-06-08T14:52:27 2015-06-08T14:53:48
```

Now, login to the cluster and do this instead:

```
rtracejob 2376732
+------------------+--------------------------+
|    User: msrc    |      JobID: 2376732      |
+------------------+--------------------------+
| Account          | mass_spec                |
```

```
| Job Name          | PROD1598283              |
| State             | Completed                |
| Exit Code         | 0:0                      |
| Wall Time         | 02:00:00                 |
| Requested Memory  | 1000Mn                   |
| Memory Used       | 13532K                   |
| CPUs Requested    | 1                        |
| CPUs Used         | 1                        |
| Nodes             | 1                        |
| Node List         | vmp515                   |
| Wait Time         | 0.0 minutes              |
| Run Time          | 1.4                      |
| Submit Time       | Mon Jun  8 14:52:26 2015 |
| Start Time        | Mon Jun  8 14:52:27 2015 |
| End Time          | Mon Jun  8 14:53:48 2015 |
+-------------------+--------------------------+
| Today's Date      | Wed Jun 10 12:41:48 2015 |
+-------------------+--------------------------+
```

Nice difference! You can use ruby to massage the output from many of the slurm commands that are used to query information about jobs. Here is the script (note the "\" symbols denote line continuation) to which you are more than welcome (if you copy this for your own use, you will need to do a **setpkgs -a ruby2.2.0_intel** first):

```
#!/usr/local/ruby/2.2.0/x86_64/intel/nonet/bin/ruby

#########################################################
#     _   ___ ___ ___ ___                               #
#   /_\ / __/ __| _ \ __|                               #
#  / _ \ (_| (__|   / _|                                #
# /_/ \_\___\___|_|_\___|                               #
#                                                       #
# - Charles Johnson <john276@accre.vanderbilt.edu>      #
# - June, 2015                                          #
#                                                       #
# Do a setpkgs -a ruby2.2.0_intel to use this           #
#########################################################

require 'text-table'
require 'time'

# Do some minimal error checking
if ARGV.size != 1
  puts "Usage: rtracejob "
  exit(1)
end
```

```
job = ARGV[0].to_i

if job < 1
  puts "Need a valid jobid."
  puts "Usage: rtracejob "
  exit(1)
end

# Query slurm for data about the job.
sacct_output  = `sacct --format=user,jobid,account,jobname,state,exitcode, \
timelimit,reqmem,maxrss,reqcpus,ncpus,nnodes,nodelist,submit,start,end \
--jobs "#{job}" --parsable`.chomp()

# if sacct returns a ".batch" in its output, we can get the maxrss
if sacct_output =~ /\.batch/
  sacct_output = \
     sacct_output.sub("\n|","").sub("\n","").sub("|End|","|End|\n")
  sacct_output = sacct_output.split("\n")
  keys   = sacct_output[0].split("|")
  values = sacct_output[1].split("|")
  values[8] = values[23]
else
  #split the return string to build a hash
  sacct_output = sacct_output.split("\n")
  keys   = sacct_output[0].split("|")
  values = sacct_output[1].split("|")
end

table_data = Hash.new()
(0..15).each { |i| table_data[keys[i]] = values[i] }

#we always have a submit time
submit_time =  Time.parse(table_data["Submit"])

# create a start time if there is one
start_time = (table_data["Start"] == "" || table_data["Start"] == \
               "Unknown") ? "" : Time.parse(table_data["Start"])

# create an end time if there is one
end_time = (table_data["End"] == "" || table_data["End"] == \
               "Unknown") ? "" : Time.parse(table_data["End"])

# create a run time if we have both start and end
run_time = (start_time != "" && end_time != "") == true ? \
           ((end_time - start_time) / 60.0).round(1) : ""
```

```
#create the wait time
wait_time = (start_time != "") ? ((start_time - \
             submit_time) / 60.0).round(1) : ((Time.now - \
             submit_time) / 60.0).round(1)

# reformat slurm data for human consumption
table_data["ExitCode"] = "" if table_data["State"] == "RUNNING"
table_data["MaxRSS"]   = "Unknown" if table_data["MaxRSS"] == ""
table_data["Submit"]  = Time.parse(table_data["Submit"]).asctime()
table_data["Start"]    = start_time == "" ? "Not yet known" : \
           Time.parse(table_data["Start"]).asctime()
table_data["End"]      = end_time == "" ? "Not yet known" : \
           Time.parse(table_data["End"]).asctime()

# create the table and add our data
table = Text::Table.new()

table.head = ["User: " + table_data["User"], "JobID: " + \
                 table_data["JobID"]]

table.rows << [["Account ",         table_data["Account"]]]
table.rows << [["Job Name",         table_data["JobName"]]]
table.rows << [["State",        table_data["State"].capitalize()]]
table.rows << [["Exit Code",        table_data["ExitCode"]]]
table.rows << [["Wall Time",        table_data["Timelimit"]]]
table.rows << [["Requested Memory", table_data["ReqMem"]]]
table.rows << [["Memory Used",      table_data["MaxRSS"]]]
table.rows << [["CPUs Requested",   table_data["ReqCPUS"]]]
table.rows << [["CPUs Used",        table_data["NCPUS"]]]
table.rows << [["Nodes",            table_data["NNodes"]]]
table.rows << [["Node List",        table_data["NodeList"]]]
table.rows << [["Wait Time",        wait_time.to_s + " minutes"]]
table.rows << [["Run Time",         run_time]]
table.rows << [["Submit Time",      table_data["Submit"]]]
table.rows << [["Start Time",       table_data["Start"]]]
table.rows << [["End Time",         table_data["End"]]]

table.foot = ["Today's Date", Time.now.asctime()]

# output the table
puts table
```

4 Contributing New Examples

In order to foster collaboration and develop local Ruby expertise at Vanderbilt, we encourage users to submit examples of their own to ACCRE's Ruby Github repository . Instructions for doing this can be found on this page .