Oscar: Sage

Loading and Launching Sage

1. Open the Terminal and use the following commands at the command line.

2. `module load sage/9.0` to load Sage. This command will return confirmation “*module: loading 'sage/9.0'*”.

3. Sage on Oscar usually has dependent modules. These will be output to your terminal when you load the Sage module similar to the example below.

```
[username@node ~]$ module load sage/9.0
module: loading 'sage/9.0'
module: sage: To use: module load gcc/8.3
```

4. Load any other dependent modules for Sage that were output in the above steps. These modules must be loaded in the specific order specified by the system when the Sage module is loaded. Loading them in a different order may cause Sage to throw errors.

For Sage 9.0: `module load gcc/8.3`

For Sage 8.7: `module load gcc/8.3 sqlite/3.25.2 perl/5.24.1 libgd/2.2.5 mpfr/3.1.5 ffmpeg/4.0.1 imagemagick/7.0.7 texlive/2018`

5. `sage` to launch the Sage console within your Terminal window.

Sage on VNC

The Sage install may not always work properly on Oscar's VNC nodes due to the VNC nodes running older chipsets. The easiest way to remedy this issue is to run sage in an interactive job via the terminal in your VNC session.

Use the interact command with parameters for your specific job to start the interactive session, then load your modules and run the sage binary (steps 2-4 above).

```
interact -n 2 -m 32g -t 04:00:00 -f 'haswell|broadwell|skylake'
```

Installing Sage Packages Locally with --user

It is possible to install most Sage packages locally to your home folder by passing the `–user` parameter at the end of your install command. See below for example steps to install the Sage packages “surface_dynamics” and “sage-flatsurf”. In this example, we are loading both packages from git repositories, so we need to load the git module.

1. `module load git`
2. `sage -pip install git+https://gitlab.com/videlec/surface_dynamics –user`
3. `sage -pip install git+https://github.com/videlec/sage-flatsurf –user`

Using Sage with Batch Scripts

Thanks to Trevor Hyde from Summer@ICERM 2019 for these instructions.

One method for running computations with Sage on Oscar is to write a script and use the slurm batch scheduler to have Oscar run your script. This requires two pieces:

1. A shell script to configure and submit your batch job to the cluster.
2. Your Sage code/program you'd like to run.

Example Batch Script

sage-batch.sh

```
#!/bin/bash


#SBATCH -J test_program
#SBATCH --array=0-9
#SBATCH -t 1:00:00
#SBATCH --mem=8G
```

```
#SBATCH -e data/<oscar-username>/test_output/test%a.err

#SBATCH -o data/<oscar-username>/test_output/test%a.out


module load sage/8.7

module load gcc/8.3 sqlite/3.25.2 perl/5.24.1 libgd/2.2.5 mpfr/3.1.5 ffmpeg/4.0.1 imagemagick/7.0.7
texlive/2018


sage test_program.sage $SLURM_ARRAY_TASK_ID
```

- `#!/bin/bash` tells the system this is a bash (shell) script.
- `#SBATCH -J test_program` sets the name of the job which appears when you check the status of your jobs.
- `#SBATCH –array=0-9` is an easy way of doing parallel computations. In this case it says our job will run on 10 different nodes, each node will be passed a parameter and we have specified that the parameters will take the values 0 through 9. You can specify several ranges or even list individual parameters if you prefer.
- `#SBATCH -t 1:00:00` specifies a time limit in `HH:MM:SS` for each node. Once this time runs out your program will stop running on that node. Be careful setting the time limit too high as doing so may make it take a long time for your job to get scheduled to run. Before starting a big computation try to do some smaller tests to see how long you expect to need.
- `#SBATCH –mem=8G` specifies how much memory each node gets. Standard exploratory accounts get 123GB total to use at any one time. So if you allocate too much per job, fewer jobs will run at once. On the other hand, if you allocate too little and a computation needs more than it has, then it will terminate. If this happens an "out of memory" error will show up in the `.err` file for that node.
- `#SBATCH -e data/<ccv-username>/test_output/test%a.err` and `#SBATCH -o data/<ccv-username>/test_output/test%a.out` specify where the error messages and output for each computation should be sent. You should store these files in your user folder, not on the submit node. We each have a folder inside the `data` directory which you can see from the submit node. In this example I have created a folder titled `test_output` where I'm putting both of these files. **You need to make these folders before you run the computation otherwise the output will be dumped into the void!** The `%a` will get replaced with the array parameter. So for example, since we set our array parameters to be `0-9` there will be 10 nodes running and each of them gets a number between 0 and 9; this node corresponding to the parameter 7 will create two files `test7.err` and `test7.out`.
- `module load sage/8.7` loads the sage module into the node.
- `module load gcc/8.3 sqlite/3.25.2 perl/5.24.1 libgd/2.2.5 mpfr/3.1.5 ffmpeg/4.0.1 imagemagick/7.0.7 texlive/2018` loads the dependency packages for the sage module.

Everything after this in the script happens as if you typed it yourself onto the command line.

- In our example, we want to run sage code, so the line `sage test_program.sage $SLURM_ARRAY_TASK_ID` runs our example sage program `test_program.sage`.
- The file needs to have the `.sage` extension.
- You should write this file in a text editor, not in a Jupyter notebook (although you can first write and test your program in a Jupyter notebook and then copy and paste it into a new file when it's ready).
- This program is written to accept one input and I have passed it `$SLURM_ARRAY_TASK_ID` which is the array parameter passed to each node. You can use this parameter to select which input parameters to run your program on.

Example Sage Program

test_program.sage

```
import sys


def fun_math(message):
    print message
    sys.stdout.flush()


job_id = int(sys.argv[1])
fun_math('hi this is a test')
fun_math('my job id is' + str(job_id))
```

- In the Sage program, you first define all of your functions and then you include the code you want to run.
- Import `sys` so you can access the array parameter passed to your function from the node. This is accessed in this case by `sys.argv[1]`. Make sure you explicitly coerce to be an integer if you want to use it as an integer; it's a string by default.
- The output of the `print` command is appended to the `.out` file for this node as a new line.
- Notice the line `sys.stdout.flush()` included in the function. This makes the program immediately send whatever output it has to the output file when called. Otherwise the program won't output **anything** until it has completely finished running. If each node is running 100 potentially long computations and it finishes the first 99 but then times out on the 100th computation, and you don't include any `sys.stdout.flush()` commands, everything will be lost when time runs out.

Submitting the Batch Job

- To run this batch program go back to the submit node and type `sbatch <NAME_OF_BATCH_FILE>`. In our example here, our batch file is called `sage-batch.sh`, so we simply type `sbatch sage-batch.sh` Slurm will return a line that tells you your job has been submitted together with a job id number.
- To check the progress of your jobs type `myq` from anywhere on Oscar. This will show you what jobs you have running, how much time they have left, and which jobs are still waiting to run. Be patient, sometimes it takes a minute for things to get started.
- If you realize your code is never going to finish or that you’ve made some terrible mistake, you can cancel a batch job by typing `scancel <JOB_ID>`. You can specify a single node or just put the general job id for the whole run and cancel everything.

Revision #2
Created 10 November 2021 19:50:36
Updated 18 June 2024 13:17:47