# Lecture 16: Supercomputing @CRC

LING 1340: Data Science for Linguists

Na-Rae Han

# Objectives

- We didn't finish our grep lesson (in lecture15.pdf). We'll put it on hold for now...
- Supercomputing at CRC
  - Server access through SSH
  - Running a job on CRC

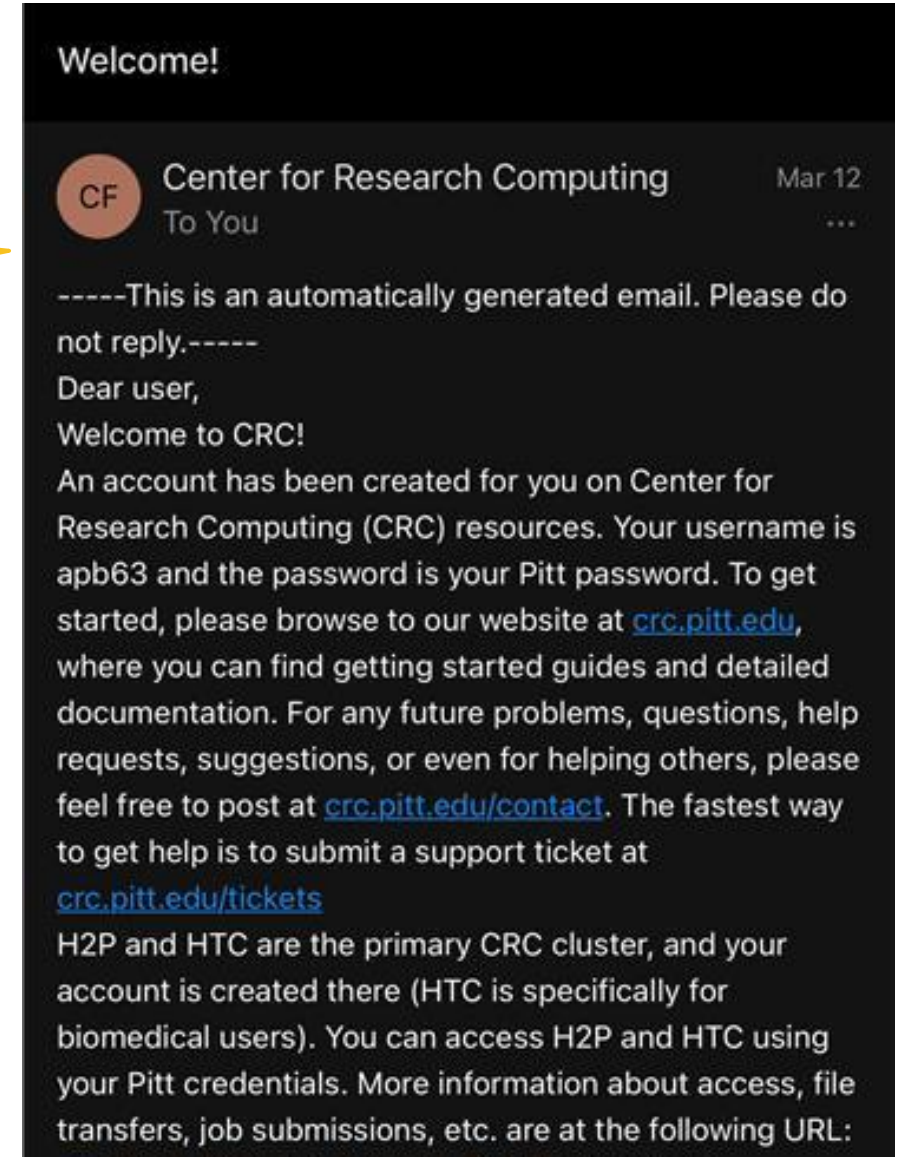
# Let us now supercompute.


By Argonne National Laboratory's Flickr page - originally posted to Flickr as Blue Gene / PFrom Argonne National Laboratory Uploaded using F2ComButton, CC BY-SA 2.0, https://commons.wikimedia.org/w/index.php?curid=6412306

# You got a supercomputing account.

- You received this mysterious email:

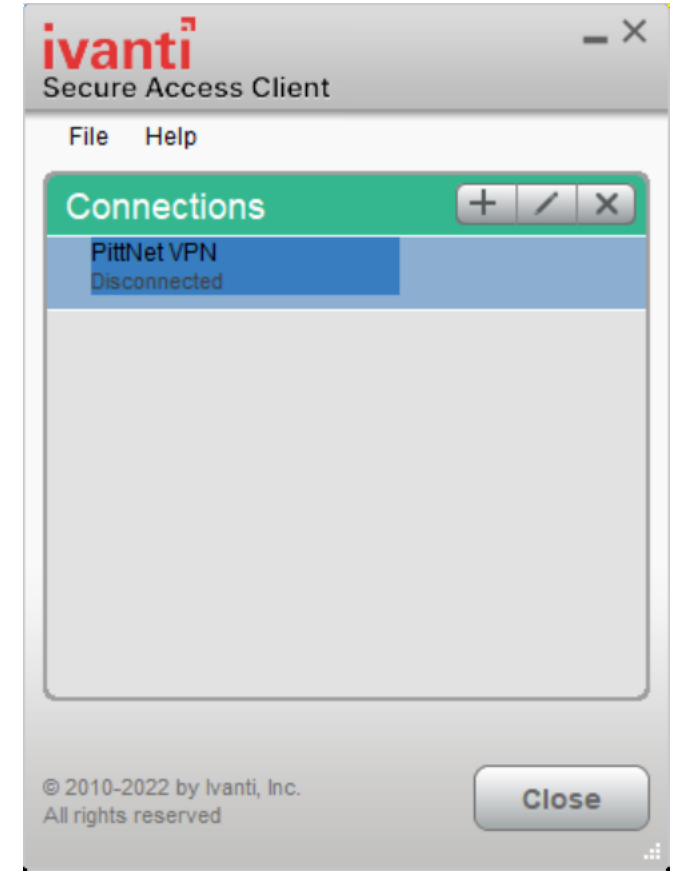
I got you all an account at Pitt's **Center for Research Computing** (CRC)




- CRC: Center for Research Computing
  - https://crc.pitt.edu
  - Handy links in "Resource" page!

# Accessing CRC's cluster

- If you're **OFF CAMPUS**, your laptop should be running a **Secure Remote Access client**.
  - Install and run GlobalProtect (Ivanti) →
  - Details in the h2p cluster user guide: https://crc.pitt.edu/getting-started/accessing-cluster
- Remote-access your account via SSH:
  - `ssh yourpittid@h2p.crc.pitt.edu`
- Getting your bearings:
  - Where are you? `pwd`
  - What is your user 'group'? `groups`
  - Is python installed on this machine? `which python`
  - What are your configuration files? `ls -a`
    - `.bash_profile`
      ← Customize with your own aliases, etc.
    - `.bash_history`
      ← Bash commands you typed in are logged here.



# Na-Rae's .bash_profile on CRC

- PATH configuration
- Prompt in pink!! Add this line:

```
export PS1="\[\e[0;35m\][\u@\h \W]\$ \[\e[m\]"
```

- Some aliases
  - grep: perl style, colored output
  - ls: colored output, folders marked with "/"

If you edit this file, changes take effect after logging back in.

For immediate effect, run:
`source .bash_profile`

naraehan@login0:~

```
[naraehan@login0 ~]$ cat .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin
export PATH

# Prompt in pink color
export PS1="\[\e[0;35m\][\u@\h \W]\$ \[\e[m\]"

# perl-style regex, color
alias grep='grep -P --color'
alias ls='ls -F --color=auto'
```

# Using the right python module

```
naraehan@login0:~
[naraehan@login0 ~]$ which python
/usr/bin/python
[naraehan@login0 ~]$ python --version
Python 2.7.5
[naraehan@login0 ~]$ module load python/ondemand-jupyter-python3.11
[naraehan@login0 ~]$ which python
/ihome/crc/install/python/ondemand-jupyter-python3.11-2023.09/bin/python
[naraehan@login0 ~]$ python --version
Python 3.11.5
[naraehan@login0 ~]$
```

Oh no, default Python is 2.7.5…

- We have to "load" the correct python module via

  `module load python/ondemand-jupyter-python3.11`

- Popular data science libraries are already installed (pandas, sklearn, nltk…):

```
naraehan@login0:~
[naraehan@login0 ~]$ python
Python 3.11.5 (main, Sep 11 2023, 13:54:46) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas
>>> import nltk
>>>
```

# Using CRC clusters

- Job submissions
  - On a computing cluster, many people are using the same resources so we have a "job queue" that accepts job submissions
  - CRC and many other clusters use Slurm for managing and scheduling these jobs.

- What this means:
  - You don't directly execute your Python script. (A big NO-NO)
  - You create a BASH SCRIPT to run a PYTHON SCRIPT (job).

# Before you get carried away

- Do NOT yet run any commands/jobs that may be resource-intensive.
- This is a powerful super-computer, shared by many research groups at Pitt.
  - Our class as a group has a limited, shared allocation. We have a reserve of **10000 Service Units** (SUs), which is 10k hours of computing time.
  - You do not want to accidentally initiate a run-away process and hog resources.
- There are PROPER ways to run jobs.
  - We will show you now.

# Slurm Jobs

- To make a slurm job script, you basically need to write a **bash script** of what you would do to run your program on the command line. This is just a text file, usually with a **.sh** ending.
- Also need some slurm configs
- Example (let's call this `hello.sh`)

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

#SBATCH --job-name=hello
#SBATCH --output=hello.out
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --partition=smp
#SBATCH --cluster=smp

echo "hello world"
```

<-- Copy this into a file and name it something like **hello.sh**

Below are some other Slurm config options (prefix with #SBATCH) as in hello.sh. EVEN MORE at https://slurm.schedmd.com/sbatch.html

| Option | Environment Variables |
|---|---|
| --output | – |
| --time | (Format: DAYS-HOURS:MINUTES:SECONDS) |
| --job-name | SLURM_JOB_NAME |
| --nodes | SLURM_NNODES |
| --ntasks | SLURM_NTASKS |
| --cpus-per-task | SLURM_CPUS_PER_TASK |
| --ntasks-per-node | SLURM_NTASKS_PER_NODE |
| --partition | SLURM_JOB_PARTITION |
| --mem | SLURM_MEM_PER_NODE |
| --account | SLURM_JOB_ACCOUNT |

# Job management commands

So from the directory with our `hello.sh` script, we can submit it with:

```
sbatch hello.sh
```

This should run pretty much instantly and we can check our `hello.out` output file.

| Command | Description |
|---|---|
| `sinfo` | Quick view of partitions |
| `sbatch <job>` | Submit your job[a] |
| `squeue` | View all jobs |
| `squeue -u <user>` | Look at **your** jobs |
| `scancel <jobid>` | Cancel your job |
| `crc-sinfo.py` | `sinfo` wrapper |
| `crc-squeue.py` | `squeue` wrapper |
| `crc-scancel.py <jobid>` | `scancel` wrapper |
| `crc-usage.pl` | View your group's usage |

# To-do #13 redux on CRC: setting up

(1) Location of yelp review data file (you all have access):

```
/ihome/nhan/naraehan/shared_data/yelp_dataset_2021/yelp_academic_dataset_review.json
```

(2) We'll sample 1 million lines (shuffled):

```
shuf yelp_academic_dataset_review.json -n 1000000 > ~/review_1mil.json
```

(3) Copy over our python script. Running it on this data will look like: (but we shouldn't be running this command directly!)

```
python process_reviews.py review_1mil.json
```

(4) But before that, we should load the appropriate python environment:

```
module load python/ondemand-jupyter-python3.11
```

(5) Now we can toss all this into a bash script. Let's call it `todo13.sh` :

- Start with `hello.sh` (make a copy using `cp file1 file2`, then edit)
- Change the bash commands at the bottom to run our script for To-do 13, and change the job name and output file to something like `todo13` and `todo13.out`

todo13.sh

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

#SBATCH --job-name=todo13
#SBATCH --output=todo13.out
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --partition=smp
#SBATCH --cluster=smp

module load python/ondemand-jupyter-python3.11
python process_reviews.py review_1mil.json
```

SLURM Job script

Python script

process_reviews.py
(from To-do #13)

```
import pandas as pd
import sys
from collections import Counter

filename = sys.argv[1]

df = pd.read_json(filename, lines=True, encoding='utf-8')
print(df.head(5))

wtoks = ' '.join(df['text']).split()
wfreq = Counter(wtoks)
print(wfreq.most_common(20))
```

# Nope, the job keeps getting killed…

- Processing 1mil reviews required too big a memory usage, which prompts slurm to kill the job:

```
[naraehan@login0 ~]$ cat todo13.out
/var/spool/slurmd/job14746448/slurm_script: line 13: 240176 Killed                python process_reviews.py revie
w_1mil.json
slurmstepd: error: Detected 1 oom-kill event(s) in StepId=14746448.batch. Some of your processes may have been kill
ed by the cgroup out-of-memory handler.
```

- We need to configure our job to request more **memory** up front:

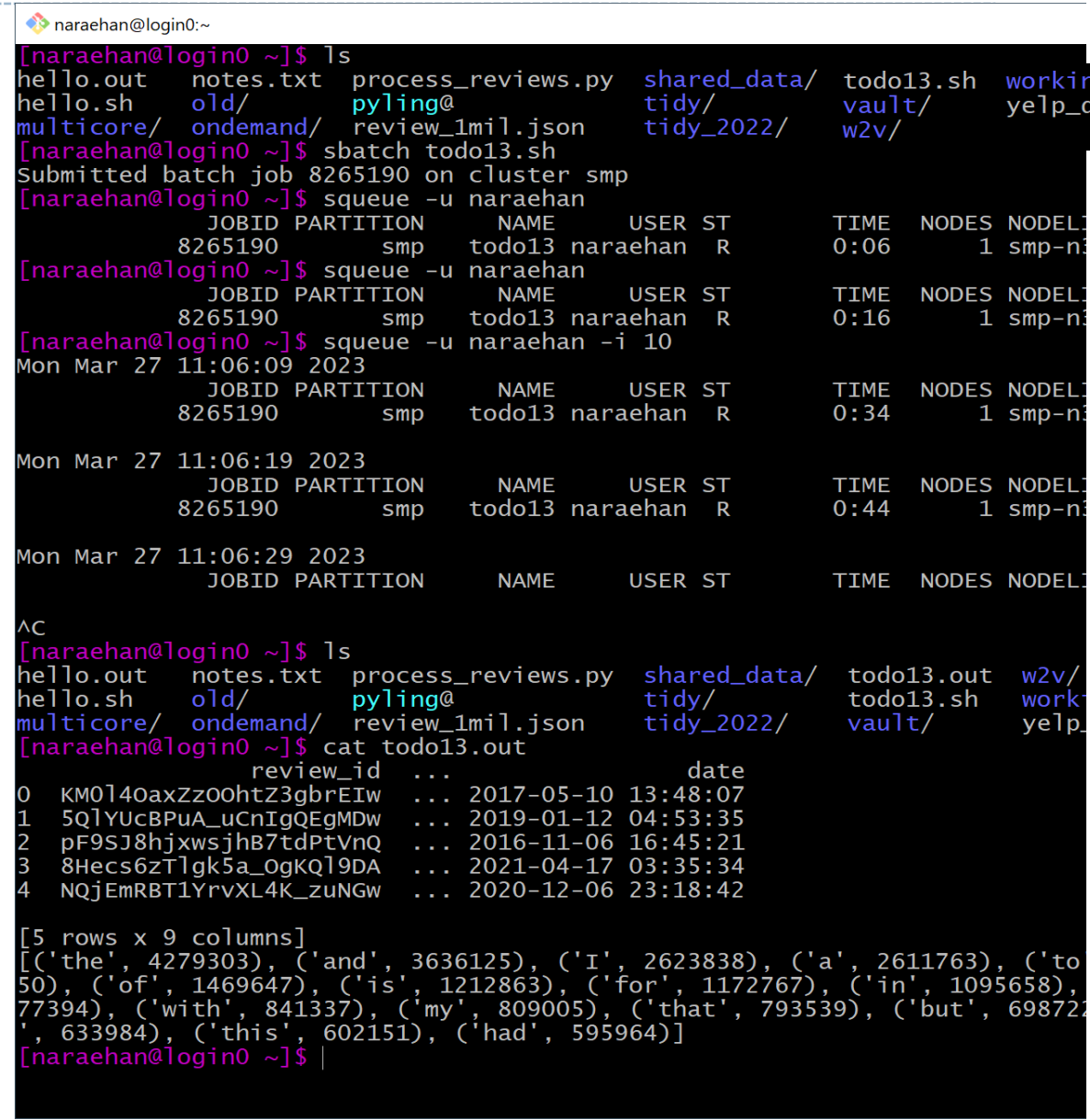
todo13.sh

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

#SBATCH --job-name=todo13
#SBATCH --output=todo13.out
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --partition=smp
#SBATCH --cluster=smp
#SBATCH --mem-per-cpu=12000

module load python/ondemand-jupyter-python3.11
python process_reviews.py review_1mil.json
```

12000 MB = about 12GB of memory per CPU
Hopefully this is enough…?

# To-do #13 redux on CRC

- Submit your job:
  - `sbatch todo13.sh`
- and check status with:
  - `squeue –u <user-id>`
  - Done when squeue no longer shows job (keep re-running with up arrow)
  - Or: add `-i 10` to auto-run every 10 seconds (Ctrl+c to get out)
- Check the output with:
  - `cat todo13.out`
- Success! 1 million reviews didn't take too long



# How did the job go?

- Job ID was shown earlier →

```
[naraehan@login0 ~]$ sbatch todo13.sh
Submitted batch job 14746453 on cluster smp
[naraehan@login0 ~]$ squeue -u naraehan -i 10
Mon Mar 25 12:08:56 2024
             JOBID PARTITION     NAME     USER ST
          14746453       smp   todo13 naraehan  R
```

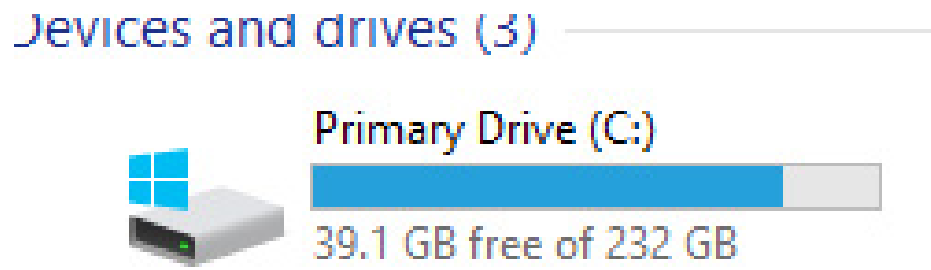
- Check finished job's stats by:
  - `seff <job-id>`
- Our Python script on 1million reviews took up:
  - 9.17GB of memory (RAM)
  - 51 seconds of CPU time

```
[naraehan@login0 ~]$ seff 14746453
Job ID: 14746453
Cluster: smp
User/Group: naraehan/nhan
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 00:00:51
CPU Efficiency: 96.23% of 00:00:53 core-walltime
Job Wall-clock time: 00:00:53
Memory Utilized: 9.17 GB
Memory Efficiency: 78.24% of 11.72 GB
[naraehan@login0 ~]$
```

# Quick aside: computing hardware

- **Memory** here refers to **Random Access Memory** (RAM)
  - You probably have 8 or 16 GB on your laptop
  - Running programs uses **RAM** to store temporary data (in our case opened file content, variables, lists, DataFrame, etc) that they use or produce
  - Stuff stored in RAM is removed when a program terminates, or if your computer shuts off.
  - Running out of RAM on your laptop could cause your computer to freeze/crash
  - Expensive per GB

- NOT disk drive -->
  - Disk space stores files long-term
  - Cheap per GB, 256+GB is pretty standard.



# To-do #14: bigger data + better code @ CRC!

- Take 1: use 4 million reviews
- Take 2: use 4 million reviews, with a new (better!) python script

  ← Compare Take 1 vs. Take 2
- Take 3 (optional): all 7 million reviews, with the new (better!) python script