README - R on Sherlock

Dain Brademan

2025-01-07

A quick guide for running the Huttenhain lab's proteomic pipelines on Sherlock, Stanford's high-throughput computing resource. More info can be found using Sherlock's online resources.

Sherlock Overview

Sherlock is a High-Performance Computing (HPC) cluster, operated by the Stanford Research Computing Center (SRCC) to provide computing resources to the Stanford community at large. HPC clusters like Sherlock give researchers access to powerful computational resources that are typically inaccessible at a lab- or even departmental-level ($$$$). In fact, Sherlock is conveniently free for Stanford-sponsored researchers to use!

HPC clusters such as Sherlock function as a team of extremely-fast, interconnected computers which work together to handle the computational tasks given by the collective group of users in a fair and efficient manner. Users specify tasks (*e.g.* run a program to process a dataset and return results), and Sherlock distributes these tasks fairly to the "workers" using a scheduling software named SLURM.

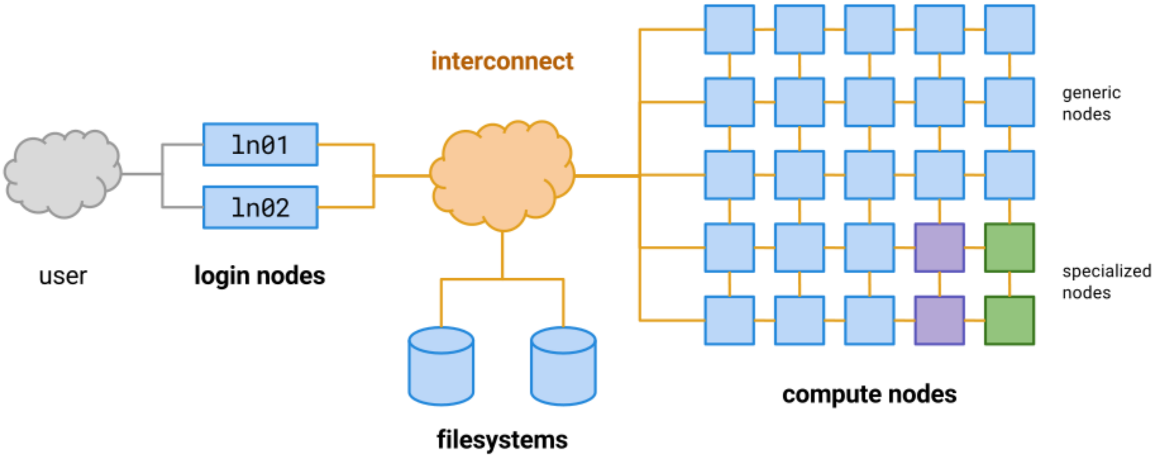

Figure 1: General structure of a high-performance computing cluster like Sherlock

Login Nodes

Users access Sherlock's infrastructure via dedicated login nodes which provide a convenient entry points to organize data, configure software, do simple tests, and submit new computational tasks to the job scheduler. Login nodes are not meant for heavy-duty data processing, so be sure not to run your full data processing jobs here.

Filesystems

Sherlock has a number of notable filesystems that you have access to (more info):

- `$HOME` - Your home directory
  - Keep code templates and important data here
  - 15 GB storage limit
- `$SCRATCH` - Your personal workspace
  note - data is automatically deleted after 90 days
  - Store temporary files and run jobs from here
  - Virtually unlimited storage (100 TB)
  - Inactive files purged after 90 days
- `$GROUP_HOME` - The lab's home directory
  - Keep group code templates and important data here
  - Total 1 TB storage limit for all lab members.
- `$GROUP_SCRATCH` - The lab's personal workspace
  - Store temporary files and run jobs from here
  - Virtually unlimited storage (100 TB)
  - Inactive files purged after 90 days
- `$OAK` - The lab's Oak Storage
  note - do not directly stage tasks from Oak
  - Direct access to/from Oak Storage into Sherlock infrastructure
  - Copy data from `\$OAK` to `\$SCRATCH` when staging tasks

Compute Nodes

Compute nodes are where your data processing tasks actually run. There are a variety of compute node types that you can request. We generally recommend using the `normal` compute node partition. You will also have access to `bigmem`, `dev`, and `gpu` for more specialized tasks.

When submitting tasks to the schedule, you will need to specify the amount of computational resources you need to run your program(s). Primarily, you will specify the number of processors (CPUs), memory (RAM), and maximum amount of time it will take for your task to run. Requesting resources is a balancing act. If you do not request enough resources your task may run inefficiently, or it may even fail or be canceled to prevent over-utilization of resources. If you request too many resources, it may take a longer time for your job to be allocated resources depending on current demand and your past utilization efficiency.

Connect to Sherlock

Using your favorite SSH client, run the following command.

If you don't have a Sherlock account set up, follow the directions here to get started. Ruth will need to email the Sherlock team to get you approved.

```
ssh <USERNAME>@login.sherlock.stanford.edu
```

You will then be prompted to enter your password & dual factor authentication method.

Load required modules on Sherlock

When you log into Sherlock for the first time, you'll be presented with a default, bare bone environment with minimal software available. To get access to specific software such as R, you need to load the specific modules. These modules along with additional documentation on what is available is listed here.

Load in the default R (v 4.2.0 as of Jan. 7th, 2025) module using the below command. `ml` is short for `module load`. Both are valid ways of loading modules. Theoretically, once you load a module it stays active. To be safe you can always stick the module load code in your Slurm batch file, but I leave that decision to each person.

```
# Load R module
ml R
```

The installation of some R packages such as `devtools` requires other modules (e.g. **CMake)** to be loaded as well. You only need to load these additional modules if you are installing certain packages that require them.

- `cmake` is required to install MSstats.

```
# handles build process
ml cmake
```

- All additional modules below are required for `devtools` as described by the Sherlock documentation

```
ml R                           # loads R
ml system harfbuzz fribidi     # More modules
ml cmake libgit2               # More modules
ml openssl                     # More modules
```

Installing R Packages

Sherlock is only configured with a select set of standard/common packages. As such, we need to install all the missing packages ourselves before. This only needs to be done once, and it's easiest to do this in the shell terminal once the R module is loaded.

First, start your R session. After running this you will be in an R environment. Type `q()` to quit out to the Unix terminal as needed.

```
# launch R
R
```

For convenience, we compiled a shortlist of packages for the LFQ pipeline that need to be installed and from what sources. This can take a while (~30+ minutes when I did it). Lots of text will flash across the terminal. You may be prompted to update certain packages.

Note: If you've never installed packages on Sherlock before, you will be asked if you want to create a personal library in your $HOME directory. Type 'yes' for both prompts, or if you're feeling fancy configure your own custom path to store your packages.

```
# Base R packages
install.packages(c("devtools", "data.table", "dplyr", "tidyr", "R.utils", "BiocManager","ggplot2","ggrep

# Bioconductor packages
BiocManager::install(c("MSstats", "UniProt.ws", "ComplexHeatmap"))
# MSstatsPTM if you are doing PTM work
BiocManager::install("MSstatsPTM")
```

Once all the packages are installed correctly, we can exit out of R (`q()`) and finally begin thinking about how to run our analysis on Sherlock.

Analysis - Required Files

We have modified the lab's proteomic pipelines to run on Sherlock through the most computationally-intensive stages for particularly large datasets. This version of the pipeline stops after protein summarization and does not do any testing for significantly-changing proteins as the selection & fine-tuning of the pairwise comparisons or time-series analysis often requires user interaction. These pipeline modifications lead to data processing completing on Sherlock in ~50% the time and utilized memory compared to running the pipeline on a personal computer with RStudio.

Just remember, you no longer will be able to interact with the data dynamically as it is processing on Sherlock. Afterwards, if you find some samples need to be tossed due to outlier effects or other issues, you may need to remove these samples from the initial Spectronaut export and re-run the pipeline without the bad samples.

Files & File Context

Put these three files into the same folder somewhere in your $SCRATCH space.

1. **sherlock_lfq_pipeline.R**
   - A version of the lab's proteomics LFQ pipeline as an R script instead of an interactive notebook.
     - This summarizes Spectronaut peptide measurements to protein abundances.
     - This script requires additional arguments specified in the sbatch file below.
2. **spectronaut_report.tsv** (*example name*)
   - Spectronaut report for your samples
3. **sherlock_pipeline.sbatch** (*example name*)
   - Text-based `sbatch` file containing the Sherlock resource requests, arguments, and necessary configurations specific to your dataset.

Configuring the Sherlock SBATCH file

SBATCH files are batch scripts (*i.e.* a set of commands run sequentially) used to submit jobs to the compute nodes for execution. These file contains both directives (commands prefixed with `#SBATCH`) that define the job's configuration and the shell commands to be executed. The Slurm scheduler uses this information to allocate the necessary resources and schedule the job.

Here is an example SBATCH file that was used to process a ~80-sample dataset on Sherlock. This input dataset was 53 GB and took the script ~15 hours to complete. As we were initially unsure how much memory was required to run the pipeline we allocated 224 GB of RAM, but the job only used ~140 GB. Ideally the resources requested for your submitted jobs are closer to what is actually utilized.

```
#!/usr/bin/bash
#SBATCH --job-name=lfq_pipeline
#SBATCH --output=%j.lfq_pipeline.out
#SBATCH --error=%j.lfq_pipeline.err
#SBATCH --time=2-00:00:00
#SBATCH --partition=normal
#SBATCH --qos=long
#SBATCH --ntasks=1
#SBATCH --nodes=1
#SBATCH --mem=224G

# Make sure R is loaded
ml R

# Define R arguments, file paths, and data processing parameters

R_SCRIPT="sherlock_lfq_pipeline.R"                # Name of R script
PROJECT_NAME="MOR04"                             # Output folder names
INPUT_FILE="./20250106_093125_MOR04_Report.tsv"   # Path to Spec. Report
```

```
IS_CASE_CONTROL="TRUE"                            # Make replicate #s unique
                                                    # "TRUE" or "FALSE" only

NORMALIZATION_ENABLED="TRUE"                      # Normalize dataset
                                                    # "TRUE" - median normalization
                                                    # "FALSE" - no normalization

REMOVE_THESE_SAMPLES="ctrl"                       # Removes samples containing

# Run the R script with all specified arguments
Rscript $R_SCRIPT $PROJECT_NAME $INPUT_FILE $IS_CASE_CONTROL $NORMALIZATION_ENABLED $REMOVE_THESE_SAMPL
```

The first set of commands in the file are `#SBATCH` commands. These commands tell the Slurm controller what computational resources your processing task requires. These commands must be in first lines in your SBATCH file as all SBATCH commands are ignored after the first non-SBATCH line.

The lines after the #SBATCH commands define arguments that are used to run the LFQ pipeline script. When using Read the comments in the example batch command above to determine what each of these arguments do. These all need to be specified, or the default LFQ R script will crash.

Rough Estimate for Requesting Resources

Our general rule of thumb is to request memory *around 3X* the file size of your input dataset. If you do not request enough memory, your processing job will crash. You will also want to request 8 GB memory per CPU, the maximum memory/CPU ratio, to reduce your processor allocation. MSstats is not parallelized and cannot fully take advantage of these extra processors SLURM allocates. We don't have a good sense yet of the maximum run time a dataset might take to analyze, but a safe overestimate is 1 day for every 50 GB of input data. Just note that Sherlock has a maximum runtime of 7 days for a job, and any job running over 2 days needs the additional SBATCH argument `#SBATCH --qos=long` added. The number of samples will also impact these requirements, but this metric is harder to estimate.

To give some examples of resource allocation requests:

Spectronaut	–time			
Export Size	(d-hh:mm:ss)	–mem (G = GB)	–ntasks	–nodes
10 GB	0-6:00:00	30G	1	1
25 GB	0-12:00:00	75G	1	1
50 GB	1-00:00:00	150G	1	1
100 GB	2-00:00:00	300G	1	1
150 GB	*3-00:00:00	‡384G	1	1

* Run times longer than 2 days requires the flag `#SBATCH --qos=long` to be set. Using this flag limits you to a maximum of 32 CPUs and 256 GB RAM due to fair use policy.

‡ The maximum job size you can run is 384 GB. R has an internal limitation in that it must run on a single node. Anything more and your processing task will be split across several nodes.

Submitting a Job

Once you have your R script, Spectronaut report, and .sbatch file configured and in the same directory, you can finally submit the processing job to the cluster using the `sbatch` command.

```
# Format: sbatch [NameOfSBatchFile]
sbatch sherlock_pipeline.sbatch
```

When you run the above command, Sherlock will read your .sbatch file, register the resources that you are requesting, and will enter your processing task into the job queue. When your job is assigned resources, it will then run the R script with the additional arguments you specified in the batch file. Queuing your job does not mean it will start immediately. You will often need to wait for computational resources to come available.

Monitoring a job's progress

Once your job has been submitted, you can monitor its status using the following command

```
squeue -u $USER
```

This command will show all processing jobs, their status if waiting, or their runtime if actively running. If you submitted a job and you no longer see it in the queue, this means it finished or crashed. If your job ran successfully, you should see `$PROJECT_NAME_data` and `$PROJECT_NAME_figures` folders in your working directory. If you don't see these, check the error and output logs (`lfq_pipeline.err` and `lfq_pipeline.out`) for troubleshooting information.