Practical Coding Series: Elzar Batch Analysis

John Hover
hover@cshl.edu
Practical Coding January 6 2021

Schedule

Jan 6	John Hover	Elzar Batch System Usage
Feb 3	Hannah Meyer	Snakemake
Mar 3	Ammar	Python packages
April 7	Batuhan	Optimizing for performance/speed (Matlab, Python)
May 5	Rohit	Github workflow
June 2	Shaina	Personal Web dev
July 7	Amber	TensorFlow

Note: This session is being recorded.

Logistics

- All materials will be available afterwards on Github.
- Please feel free to interrupt--intended to be interactive!
- We are catering to many levels, so if you have further questions afterward feel free to contact us

https://github.com/juliawang22/CSHLPracticalCoding

Outline

- General Batch Principles
  - Why use Elzar? When not to. Pros/Cons
  - Principles, Process, Layout
  - Software Environment and Data Management
- Elzar/UGE Basics
  - Work process overview
  - Job assessment
  - Submission, resource specification
  - Testing, monitoring, debugging
- Elzar Usage Examples
  - Array Jobs, GPU requests
  - Snakemake Intro

Batch Principles/ Best Practices

Those are my principles, and if you don't like them... well, I have others.

Groucho Marx

Caveats & Omissions

No MPI discussion.

All of these hints/approaches are things that have worked for me.

But not gospel! Feel free to accept or reject.

No substitute for figuring out what works for your own work...

Question 0: Why? Can I even benefit from using batch for an analysis?

- Is the work iterative with frequent tweaks? Ad-hoc analysis at each step? -> No
- Is it very small-scale (size/number files, resources, runtime, space) -> No
- Will you need to (re)do the same analysis with new/additional input? -> Maybe
- Does the analysis need to be refined enough for public consumption/publication? -> Probably.
  - If it needs to be polished and validated anyway, why not batch?

Question 0 (cont)

- Do you need to do a multi-dimensional parameter sweep (e.g. all against all?) -> Probably.
- Does the analysis involve a large number of files, where some step exceeds your server/desktop memory/ cores/ local storage capacity? -> Yes
- Does the analysis require GPUs, high performance? -> Yes

Batch Pros/Cons

Pros:

- Safe, long-running computing.
- Resources! E.g. on Elzar...
  - **34 nodes**
  - **768**GB/ 3TB memory
  - **GPUs** 32GB
  - ***Fast*** distributed filesystem, with lots of storage.
    - GPFS: TBs depending on your group's quota.
    - vs. **non-distributed FS**

Cons:

- Requires a more structured approach to effectively run on HPC.
- Requires more up-front effort to automate.
- ... at least if the unavoidable extra work is not to be lost.
- Requires learning SGE/UGE basics and probably 1 additional utility.

Principles

- Rigorous common sense.
- Take stuff you (maybe) already do and formalize it.
- Pay attention to portability.
- 99% automatic = manual.

Standardization **limits complexity** to let you deal with the underlying task...

Cost effort up front. Reduces mental effort after. Eases later **re-entry**...

- Work Process
  - Develop
  - Test
  - Run, Iterate
  - Archive
- Portability
  - Consistent Layout
  - Symlinks
  - Modularity/DRY
  - Software versioning
- Automation
  - Nomenclature
  - Consistency

Standardization -> *Reproducibility*

Project Work/Code/Data Layout

Directory	Usage	Comment	Tape?
~/data/genomes /GTEx /CoCoCoNet	External data (re-downloadable) Long term local data (usable by others)	Symlink to global /data area.	No
~/git/cshlwork/<proj1> /<proj2>	Project-specific code/scripts. Snakemake file		Yes (github)
~/project/<proj1>	Working dir. ad-hoc one-off scripts? Snakemake -> ~/git/<proj>/Snakemake. Jupyter notebooks.	On home partition.	Yes
~/work/<proj1>	Global project-specific working area. Possibly cleared after each run.	On data partition. Symlink to ~/data/work	No
~/data/<proj1>/	Intermediate data worth keeping. Final output data location.		Yes

Software Environments

- Won't talk modules...
  - Fine on HPC, but not available on desktops (or laptops). Use Conda
- Hierarchy:
  - OS-level Local Installs
  - OS-level RPMs
    - User-level local installs
    - Module system (optional)
    - Conda
      - Python
      - R
        - CRAN installations.
      - Java tools.
      - 3rd party tarball software
      - Your custom software.

Modularize software at this level

- Don't install/use modules (except maybe compilers) at OS level!
- Create script to re-create environment, e.g...

Script serves as record of dependencies.

Hard to fully automate on multiple platforms, but close is OK.

Note use of `$CONDA_PREFIX` with tarball/make.

`pip install` also good here.

```
#!/bin/bash -l
#  For werner1 project.   labelled gatk.
#
#  samtools-1.11
#  bedtools-2.29.2
#  gatk-4.1.9.0
#  igvtools-2.8.9
#  STAR 2.7.2a  (tied to genome dir version)
#
set -x

# Conda
conda create -y -n gatk python=3.8
conda activate gatk

# Conda available packages
conda install -y pandas numpy scipy seaborn matplotlib plotly h5py
conda install -c conda-forge -c bioconda snakemake

# Tarball installs
#  Samtools 1.11
wget https://github.com/samtools/samtools/releases/download/1.11/samtools-1.11.ta
tar -xvjf samtools-1.11.tar.bz2
cd samtools-1.11
./configure --prefix=$CONDA_PREFIX
make
make install
cd ..
rm -rf samtools-1.11.tar.bz2 samtools-1.11

#  bedtools
```

Nomenclature & Style

Suggestions:

Directory names should be short, lowercase, no spaces.

For files, pick underscore or hyphen and stick with it.

Script names should be specific, descriptive, versioned. E.g.

`final_filter_script.sh` vs. **`remove_bam_snp_dupes_v13.sh`**

... (see Style and Organization session from the Practical Coding Series.)

Data Management

Disk usage on batch systems is measured. You/ your group have a limited quota. Always calculate disk usage for analyses:

 # jobs * (input + temp + output)

If temp >> input + output, and you run 1000 jobs at once...

Think explicitly about data lifecycle...

- What to copy in (don't pull anything directly from the internet)
- What to keep during processing (until correctness is confirmed).
- What to keep for potential followup investigations.
- What to copy back to external storage.

Benefits of Portability

Even if you don't develop on a desktop/laptop, and don't intend to move work between platforms, it is still worth working ***as if you might*.**

If you have a complete snapshot of software and versions, paths, input data, etc. and can easily establish the same environment on another host (and especially another OS), then your confidence in the correctness of the answers it produces should be very high.

Portability -> *Reproducibility*

Reproducibility -> *Confidence*

Work Process: Develop locally

- Create software environment, preferably with script, versions.
- Define cut-down input data. < 5 minute run.
- Get simple end-to-end bash script working. Initial sanity-checking.
- Assess..
  - core/memory requirements,
  - step runtime
  - temp,intermediate and final storage usage.
- Decide batch strategy. Create git repo for project handling...
  - ad-hoc: just submit individual jobs from the command line.
  - unix/bash: use an array job to launch a large set of jobs
  - Snakemake: create a rule, or longer pipeline.

Work Process: Test/Assess on Batch

To run on batch, you must know exact core/memory usage for each tool.

- Test small/medium input batch (~5 jobs), but with full-size data, on cluster.
- Core usage control requires you to tell the tool. Many will take all...
- May use UGE to test memory usage. Run with high request, then check actual usage, e.g.

```
qacct -j <jobid> | grep maxvmem
```

- Confirm/check success and output. Check file validity.
- Don't test intensive or long-running processes on bamdev1|2.
    a. You'll get warning. Use interactive mode...

Aside: Batch vs. Interactive

Elzar is primarily a batch resource, oriented toward multiple, automated jobs submitted from a head node. But it is also a very large, standard resource, usable interactively.

```
qrsh -l m_mem_free=1024G -pe threads 48 -l gpu=1
```

Gives out local login with more resources than a rugen. **But**, you may get

```
Your "qrsh" request could not be scheduled, try again
later.
```

So, copy data from your local server, login to bamdev, qrsh, run, copy back.

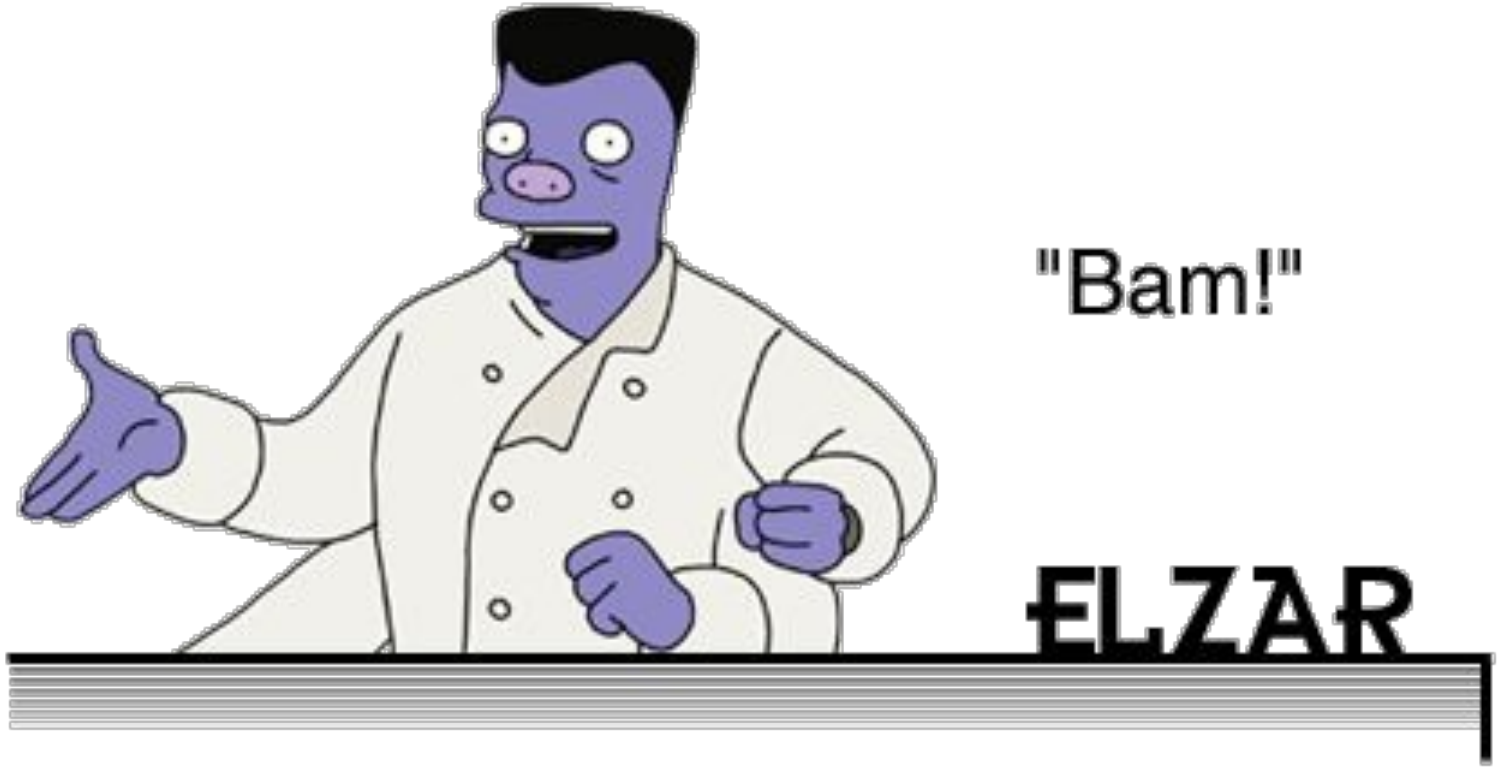
Work Process: Run at scale and monitor

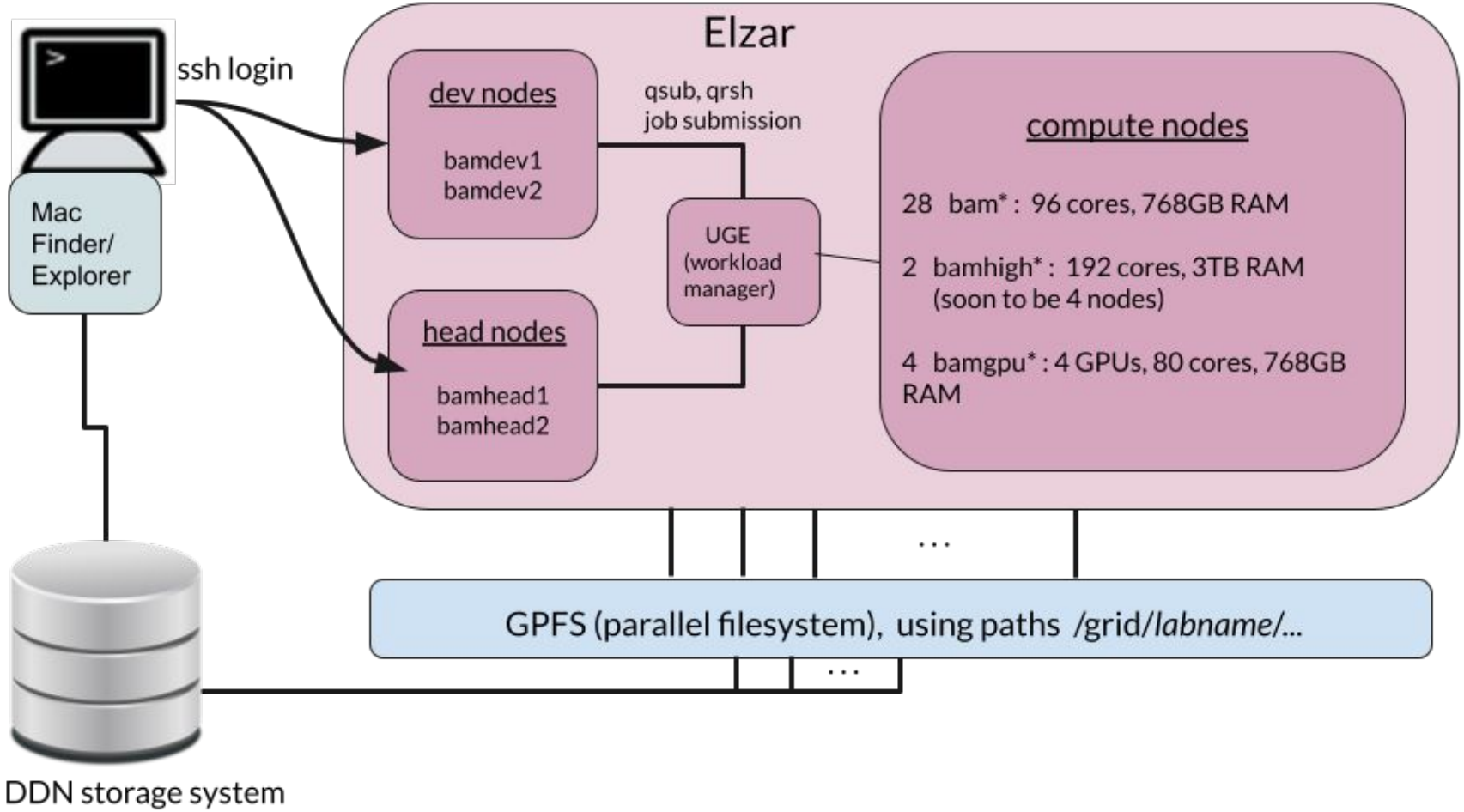
- Very rarely will large numbers of jobs all run perfectly.
- Check job exit status, file output, etc.
- If something goes wrong, consider aborting run.
- Debug the inevitable failures. Discard bad cases? Re-run.
- Sanity check/validation of intermediate and final results.

After confirmed successful run:

- Copy/move important data to final location.
- Cleanup unneeded intermediate/temp data!

Elzar/SGE/UGE Basics




ssh login
Mac
Finder/
Explorer
Elzar
dev nodes
bamdev1
bamdev2
qsub, qrsh
job submission
UGE
(workload
manager)
head nodes
bamhead1
bamhead2
compute nodes
28 bam* : 96 cores, 768GB RAM
2 bamhigh* : 192 cores, 3TB RAM
(soon to be 4 nodes)
4 bamgpu* : 4 GPUs, 80 cores, 768GB
RAM
...
GPFS (parallel filesystem), using paths /grid/labname/...
...
DDN storage system

SGE/UGE commands

`qhost, status` : information about current cluster state (nodes, queued jobs..)

`qsub <script>`: run a script non-interactively on a worker node

`qrsh`: get an interactive shell on a worker node

`qstat`: information about running jobs.

`qdel`: remove running/queued jobs

`qacct`: retrospective info about finished jobs.

Will touch on most of these as we go along...

GPFS Storage Space

Appears to be 4.3 Petabytes.

Currently $29/TB/year. First 30TB free.

You can see filesets (which map to directories) with

```
findmnt
```

Then you can you can check your quota on those directories, e.g.

```
/usr/lpp/mmfs/bin/mmlsquota -j gillis_hpc_data grid
/usr/lpp/mmfs/bin/mmlsquota -j gillis_hpc_home grid
```

Submission, resource specification

```
qsub <script>
    -N  <job name>            # a label, will appear in qstat
    -pe threads <numthreads> # pe="parallel environment" core request
    -wd <path/to/wd>          # otherwise ~/
    -l m_mem_free=2G          # mem request (or 2048M)
    -l gpu=1                  # or GPU=0 for none.
    -o  <stdout logfile>      # takes pseudo vars in script.
    -e  <stderr logfile>      # takes pseudo vars, e.g.
```

Pseudo variables usable within job script.

```
#$ -o ~/project/$JOB_NAME/logs/$JOB_NAME.o$JOB_ID.$TASK_ID
#$ -e ~/project/$JOB_NAME/logs/$JOB_NAME.e$JOB_ID.$TASK_ID
```

Interpolated at job execution time, not at submit.

Submission, resource specification (cont)

`qsub`bed jobs run on the remote worker...

- Under your user account
- From whatever working directory you submit from (unless altered).
- Under the same shell environment. (E.g conda env).

Default resources are 1 core, 2048M memory.

Logging/debugging

- Logs placed wherever you specified from -o / -e locations in submit script.
- I always direct mine in submit script, to:

  ~/project/<name>/logs/<job_name>.[e|o]<job_id>.<task_id>

  - Or, they get dropped into your home directory with default name: **<job_name>.[e|o]<job_id>.<task_id>**
- .o is standard out of running job. .e is standard error. Contents depend on executables. Different tools write to different places.
- Always useful to emit full command lines, and exit statuses to stdout.

Testing/Debugging

Do not run analysis commands on bamdev1|2! Go interactive...

Get interactive session on worker node. Specify resources as in submission. e.g.,

```
qrsh -N werner2 -pe threads 8 -l m_mem_free=5760M
```

Within the session you can run tools to confirm they work. Or to manually test a previously failed job.

This is where printing *interpolated command lines* in your code is crucial, so you can cut-and-paste specific examples upon failure.

Monitoring/ Post-run Checking

During run check job progress with `qstat ...`

Once some jobs have finished, this

```
qacct -j -d  1 -o {user}
```

will give you key-value pair lists of all jobs you have completed in the last day, which can be used to check exit statuses, e.g.

```
qacct -j -d  1 -o hover | grep exit_status
```

If things look bad and you need to abort:

```
qdel -f -u <user>
```

Both qstat and qacct have -j <jobid> args, which outputs key-value pairs for a particular job.

Miscellaneous Gotchas/Hints

- Memory usage can depend on input size. For testing I usually run 3 jobs, 1 with largest input, 1 with smallest, 1 from middle.
- Java JVM tricky to limit CPU usage. E.g,

  --java-options ' -XX:ActiveProcessorCount=8 ' Java >JDK 8u191

- Screen messes up scrolling from Mac terminal...

```
# ~.screenrc
# Enable mouse scrolling and scroll bar history scrolling
termcapinfo xterm* ti@:te@
```

Miscellaneous Gotchas/Hints (cont)

- `qstat` and `qacct` produce **key-value** pair output one per line, with **jobs** separated by bar.
- Not very useful if you're interested in lots of jobs. I wrote a quick script that formats it one job per line:

  https://raw.githubusercontent.com/jhover/elzar-example/master/bin/qhist

  ```
  qhist | head
  ```

  Gives one job per line output: jobid, name, taskid, start, end, wallclock, cores, memory, and exit status.

Array jobs: GoFocus

Array jobs

You provide an argument to qsub e.g.

```
qsub -t 1-478 [...] jobscript.sh
```

and UGE will set an environment variable within each job, e.g.:

```
SGE_TASK_ID=46
```

Your runscript must check it, and index into some list to make each job do something different, like handle a different file (or use a different set of parameters).

Array Job Example: Gofocus (Pytorch ML model)

This will be an array job, where we want to run against each file in an input directory. For this we'll just use Bash scripts...

- Simple one-step processing.
- Consumes species-specific FASTA (multi-) protein sequence file.
- Makes many GOTerm predictions, w/ probabilities, for each, to one file.
- Needs GPU.

Array Job Example (cont)

Prerequisites..

- File with base labels for all files to be processed: ~/project/jones/`baselist.txt`
- Gofocus script: `~/git/gofocus/pytorch_goterm_pred.py`
- Setup source script: ~/project/jones/setup.sh
- Common shell functions file: ~/git/elzar-example/lib/common.sh
- Execution script to be submitted: ~/git/cshlwork/jones/rungofocus.sh

Gofocus Python program:

This part of a software package provided to us by David Jones at UCL.

```
(pytorch) [hover@bamdev2 ~]$ ~/git/gofocus/gofocus/pytorch_goterm_pred.py
usage: pytorch_goterm_pred.py [-h] [-d] [-v] [-c CONFFILE] fastafile outfile
pytorch_goterm_pred.py: error: the following arguments are required: fastafile, outfile
```

So we just need the fasta input file and prediction out file.

Create a base list, check length...

```
(pytorch) [hover@bamdev2 jones]$ ls ~/data/cococonet/sequences/ | grep  'dog\|tomato' | awk -F _ '{print $1}'
 > testlist.txt
(pytorch) [hover@bamdev2 jones]$ cat testlist.txt | wc -l
2
```

This length will be used to create the job array. We will need to explicitly specify it on the command line.

`setup.sh` source file

Optional as long as you're not using modules.

Triggered from within the run script.

```
. ~/.bash_profile
# module load Java/11.0.2
conda activate pytorch
```

As long as you have established the required Conda environment the job will run with the same shell environment.

Common shell functions

Collecting info about the host.

Getting the Task id, with a default.

```
prelude() {
    echo "*********START************************"
    date
}

nodeinfo() {
    echo "*********NODE*************************"
    hostname -f
    cat /etc/redhat-release
    NPROC=`cat /proc/cpuinfo  | grep processor | wc -l`
    echo "Processors: $NPROC "
    KMEM=`cat /proc/meminfo  | grep MemTotal | awk '{print $2}'`
    MBMEM=`expr $KMEM / 1000`
    echo "Memory MB: $MBMEM"
    NPROC=`cat /proc/cpuinfo  | grep processor | wc -l`
    echo "Processors: $NPROC "
    KMEM=`cat /proc/meminfo  | grep MemTotal | awk '{print $2}'`
    MBMEM=`expr $KMEM / 1000`
    echo "Memory MB: $MBMEM"
}

taskid(){
    if [ -z ${SGE_TASK_ID} ]; then
        SGE_TASK_ID=1
    fi
    echo $SGE_TASK_ID
}

postlude() {
    date
    echo "*********END**************************"
}
```

Run script (1)

Args embedded, with usage of pseudo-vars.

Constants up front.

Ready for manual usage.

Name positional args for clarity.

```bash
#!/bin/bash
#   Usage: rungofocus.sh <setup> <basefile> <inputdir> <outputdir>
#$ -N jones
#$ -wd $HOME/project/$JOB_NAME
#$ -pe threads 8
#$ -l m_mem_free=5G
#$ -l gpu=1
#$ -o  $HOME/project/$JOB_NAME/logs/$JOB_NAME.o$JOB_ID.$TASK_ID
#$ -e  $HOME/project/$JOB_NAME/logs/$JOB_NAME.e$JOB_ID.$TASK_ID

COMMON=~/git/elzar-example/lib/common.sh
CMD=~/git/gofocus/gofocus/pytorch_goterm_pred.py
NUMARGS=4

# Check for arg count...
if [ $# -ne $NUMARGS ]; then
    echo "Incorrect number of arguments."
    echo "Usage: rungofocus.sh  <setup> <basefile> <indir> <outdir>"
    exit 1
fi

# Source common functions.
. $COMMON

# Normalize args.
setup=$1
basefile=$2
inputdir=$3
outputdir=$4
```

Run script (cont.)

Sources env setup.

Gets taskid to choose entry from list of base labels.

Prints info.

Runs the command.

Ensures that the underlying command exit code is propagated.

```
echo "Running setup from $1"
. $setup

# $SGE_TASK_ID
taskid=$(gettaskid)
echo "taskid is $taskid"
filebase=`head -$taskid test $basefile | tail -1 `
echo "filebase is $filebase"

infile="$inputdir/${filebase}_hiprio.tfa "
outfile="$outputdir/${filebase}_hiprio.predout"
echo "$infile -> $outfile"

nodeinfo
prelude

echo $CMD -v $infile $outfile
time $CMD -v $infile $outfile
RET=$?
if [ $RET -ne 0 ] ; then
    exit $RET
fi
echo "Job command Return code was $RET"

postlude
exit $RET
```

Pseudo environment vars for runscript header.

Get interpolated from script directives on a per-job basis:

Pseudo env variable	Description
$USER	User name of the submitting user
$HOME	Home directory of the submitting user
$JOB_ID	ID of the job
$JOB_NAME	Name of the job
$HOSTNAME	Hostname of the execution host
$SGE_TASK_ID	ID of the array task

Launch w/ indices. Check GPU usage once running

```
(pytorch) [hover@bamdev2 jones]$ qsub -t 1-2 ~/git/cshlwork/project/jones/rungofocus.sh
 ~/project/jones/setup.sh ~/project/jones/testlist.txt ~/data/cococonet/sequences ~/wor
k/jones/
Your job-array 763646.1-2:1 ("jones") has been submitted
(pytorch) [hover@bamdev2 jones]$ qstat
job-ID     prior   name       user         state submit/start at     queue
             jclass                          slots ja-task-ID
-----------------------------------------------------------------------------------
-----------------------------------------------------
    763646 0.50866 jones      hover        r     12/05/2020 13:58:18 gpu.q@bamgpu03
                                                8 1
    763646 0.50866 jones      hover        r     12/05/2020 13:58:18 gpu.q@bamgpu04
                                                8 2
(pytorch) [hover@bamdev2 jones]$ gpustats | grep hover
(pytorch) [hover@bamdev2 jones]$ gpustats | grep hover
[1] Tesla V100-SXM2-32GB | 36'C,  40 % |  4192 / 32510 MB | hover:python(4189M)
[0] Tesla V100-SXM2-32GB | 49'C,  56 % |  2876 / 32510 MB | hover:python(2873M)
```

Record jobids! Not easy to get after completion.

```
#!/bin/bash
#  gpustats per Heywood
tail -n 5 /grid/it/data/elzar/gpustat/bamgpu0*.log
```

Check for completion. Stats...

```
(pytorch) [hover@bamdev2 ~]$ qstat
(pytorch) [hover@bamdev2 ~]$ qacct -j 763646 | grep ^wallclock
wallclock    60.936
wallclock    108.350
(pytorch) [hover@bamdev2 ~]$ qacct -j 763646 | grep maxvmem
maxvmem      12.564G
maxvmem      11.184G
(pytorch) [hover@bamdev2 ~]$ qacct -j 763646 | grep slots
slots        8
slots        8
(pytorch) [hover@bamdev2 ~]$ qacct -j 763646 | grep failed
failed       0
failed       0
```

Snakemake:
(Very) Brief Introduction

Snakemake

A framework for reproducible data analysis

https://snakemake.github.io/

Basic concept is simple: A set of rules, each with defined:

1. input(s)
2. output(s)
3. shell/script/wrapper to create outputs from inputs.

Rules go in a Snakefile.

User executes snakemake and it processes the Snakefile, resolving rules.

Snakemake (cont.)

Basic tool runs locally--not particuarly tied to clusters/batch analyis. Focussed on running pipelines and/or complex workflow DAGs.

Has a `-cluster "qsub ....."` arg to submit to various batch systems.

Snakefiles are slightly tweaked Python syntax, so arbitrary Python code will execute within them.

Complete implementation of previous array job.

Added in post-prediction sorting step.

Way easier.

```
import os
homedir = os.path.expanduser("~/")

(SAMPLES,) = glob_wildcards(homedir + "data/cococonet/sequences/{sample}_hiprio.tfa")

wildcard_constraints:
   sample = '\w+'

rule all:
    input:
        expand(homedir + "work/jones/{sample}_hiprio.pred", sample=SAMPLES)

rule gofocus:
    input:
        homedir + "data/cococonet/sequences/{sample}_hiprio.tfa"
    output:
        homedir + "work/jones/{sample}_hiprio.predout"
    resources:
        gpu=1, mem_mb=5120
    threads: 4
    shell:
        homedir + "git/gofocus/gofocus/pytorch_goterm_pred.py -v {input} {output} "

rule sorted:
    input:
        homedir + "work/jones/{sample}_hiprio.predout"
    output:
        homedir + "work/jones/{sample}_hiprio.pred"
    resources:
        gpu=0 , mem_mb=2048
    threads: 1
    shell:
        "cat {input} | sort -k 1,1 -k 3,3rn > {output} "
```

1. list of labels.

2. list of final output.
Where's input?

4. gofocus output = sorted input.
Input exists?
Yes (first input) -> run shell.

3. 'sorted' rule **output** matches rule 'all' **input**.
Input exists?
yes -> run shell
no -> find input.

Desktop/Laptop submit command

Snakemake useful outside of a batch context as well.

Although gofocus uses GPUs, it can run (slowly) on a desktop/server.

```
cd <project directory>
conda activate pytorch
snakemake --cores 16 -l gpu=0 --resources mem_mb=81920
```

The `--cores and --resources` args tell Snakemake how much system resources to allow for **all** processes, based on rule `threads` and `resources` directive values.

"Mini"- batch system...

So, e.g. no more than 2 simultaneous jobs will run, since each uses 8 cores.

Snakemake will fill in unused resources, with various rule jobs, as long as dependencies are satisfied.

Snakemake Submit

Submit command on Elzar is a bit longer, since we need to pass args through to qsub (since they can't be embedded in a script).

```
cd <project directory>
conda activate pytorch
snakemake --jobs 50 --cluster "qsub -N jones -cwd -pe
threads {threads} -l m_mem_free={resources.mem_mb}M -l
gpu={resources.gpu} "
```

The specified variables will be filled in for each job by the values in the rule that generated it.

Snakemake Summary

- Excellent for automating file-driven analysis.
- Very natural for Python programmers.
  - But can execute arbitrary programs.
- Well supported. Large community.
- Nicely moves between desktop/laptop/server and batch cluster.
- Integrated Conda software management (not covered).
- Nice reporting/visualization tools (not covered).

Just barely scratched surface of functionality...

- Conda environment specification-> automatic install within job
- Cluster profiles
- Tool wrappers.

Overall Summary

- Batch Principles
  - Automation and encapsulation required for reliable batch usage.
  - But extra effort for standardization minimizes error, allows re-use, and eases picking up old work -> reproducible, high-confidence results

Good for science.

- Elzar/UGE
  - Large, complex HPC installation, but useable with a small command set.
  - Don't worry about using modules. Just use Conda.
  - Interactive use to get access to very large cores/memory, GPUs.

Don't *have* to use batch.

- Snakemake. My new favorite thing...
  - Use it locally for anything even slightly "pipeline-like", very easy to get going.
  - Can be a "mini" batch system, throttling by resource availability.

Good stepping stone to batch.

Questions/Discussion?

Resources/ References

Software Carpentry Analysis pipelines with Python (draft).

Covers basic local use of Snakemake, cluster usage with Slurm:

https://hpc-carpentry.github.io/hpc-python/

CSHL Practical Coding Series:

https://github.com/juliawang22/CSHLPracticalCoding

Heywood Elzar Tutorial/ Docs:

https://docs.google.com/presentation/d/1MHGlb9kF7SsBnmVL98e8OodBCPPHtjXqcFPWuRkjd_U/

https://www.youtube.com/watch?v=D3wfhM_cQPY

Wiki: http://intranet.cshl.edu/administration/information-technology/hpcc/elzar

Slack Workspace: https://elzarusers.slack.com/