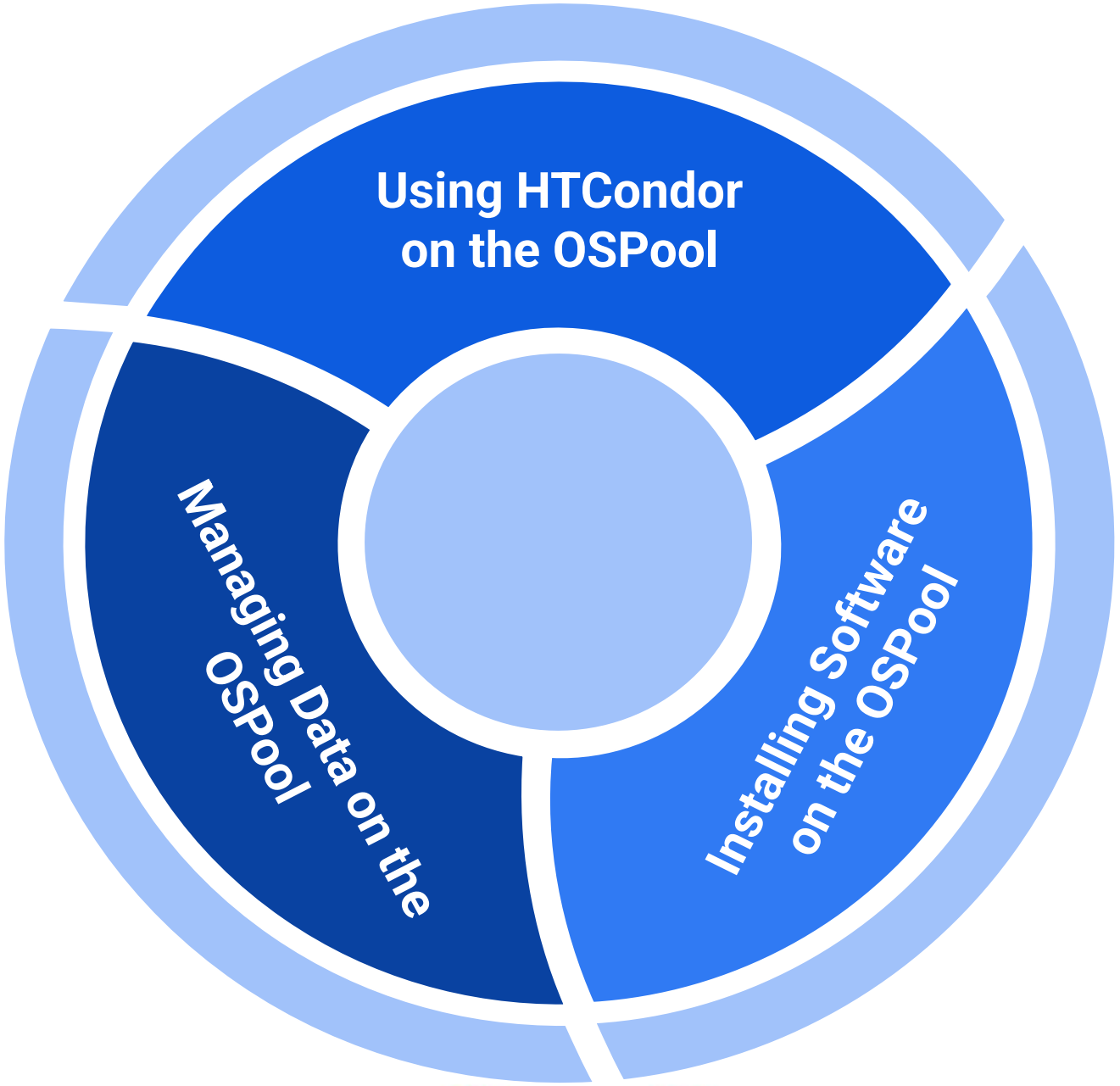
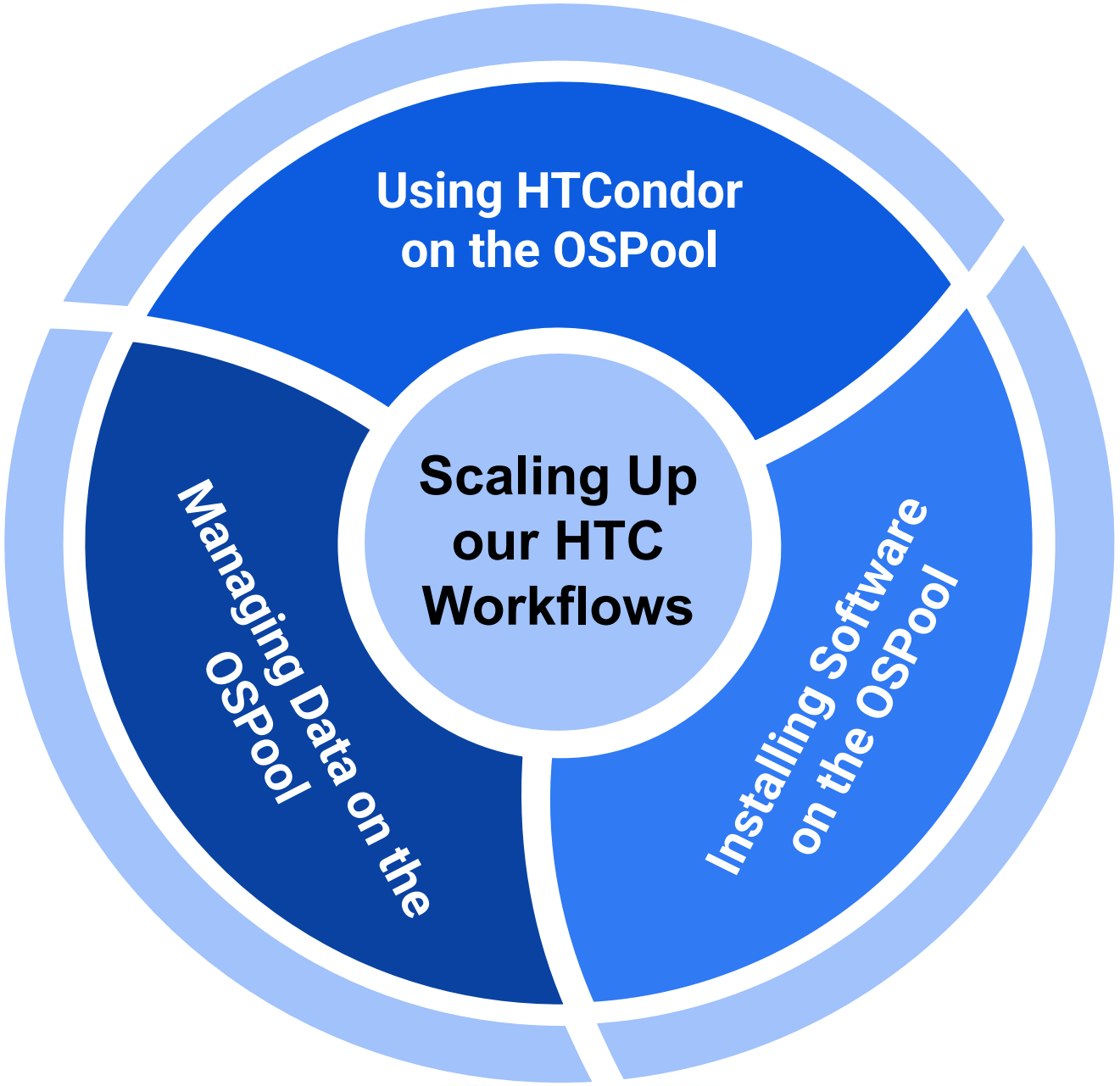
Scaling Up and Building Independence in Research Computing

Daniel Morales and Christina Koch

OSG School 2025

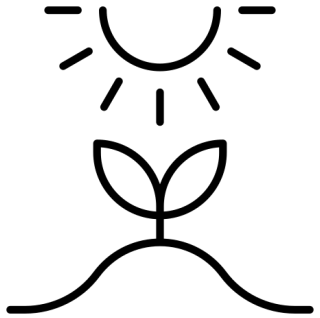
June 26, 2025


Using HTCondor on the OSPool
Installing Software on the OSPool
Managing Data on the OSPool


Using HTCondor on the OSPool
Installing Software on the OSPool
Managing Data on the OSPool
Scaling Up our HTC Workflows

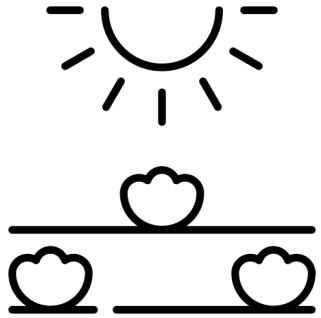
Scaling: Job Submissions

Optimize Software Only after It Works Correctly



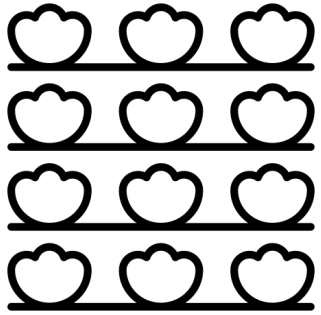
Created by Made from the Noun Project

1. Run a single job.



Created by Made from the Noun Project

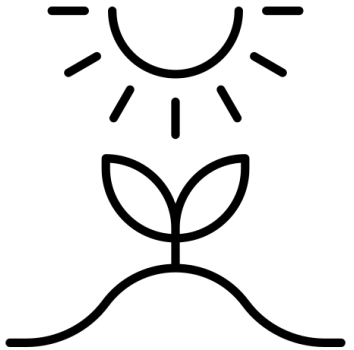
2. Test a workload of 5-10 jobs.



Created by Made from the Noun Project

3. Scale up to 100s-1000s of jobs

Stage 1: Run a Single Job



Created by Made from the Noun Project

For each job type, get a test job working reliably & tune resource needs

1. Assemble job components: executable, inputs, arguments, etc.*
2. Estimate initial resource needs*
3. Write a submit file
4. Submit!
5. Review all outputs, including log, output, and error files, Check actual resource usage and update resource needs*
6. Repeat until (fairly) accurate and reliable

* More details on next slide and in Appendix B

Components of a Job: The Game Plan

Submitting 100s-1000s of jobs, requires a little bit of planning.

As you scale up, ask yourself the following:

- File Management:
  - Which files change or are shared between jobs?
  - Which files do you expect to generate per job?
- Scripting:
  - What parts of your executable vary or stay the same between jobs?
  - What variables does your executable use between jobs?

Think about the best practices from the following slides.

Stop & Plan!

Components of a Job: Executable

A *generalized* executable & submit file is preferred!

When generalizing, think about:

1. What portions of your executable are shared across your jobs?
   - These sections can be “hard-coded” into the executable
2. What portions of your executable vary across your jobs?
   - Use arguments and variables to express these

* More details on next slide and in Appendix B

Components of a Job: Executable
Ex. Counting variations of a word in **a** book

A *generalized* executable & submit file is preferred!

words.sub

```
executable = words.sh

transfer_input_files = Alice_in_Wonderland.txt
queue 1
```

Dodo

Bird

Cat

Glass

words.sh

```
#!/bin/bash

./word-variations.py Alice_in_Wonderland.txt Dodo > variations.Dodo.txt
```

Components of a Job: Executable
Ex. Counting variations of a word in **a** book

A *generalized* executable & submit file is preferred!

words.sub

```
executable = words.sh
arguments = $(word)

transfer_input_files = Alice_in_Wonderland.txt
queue word from list_of_words.txt
```

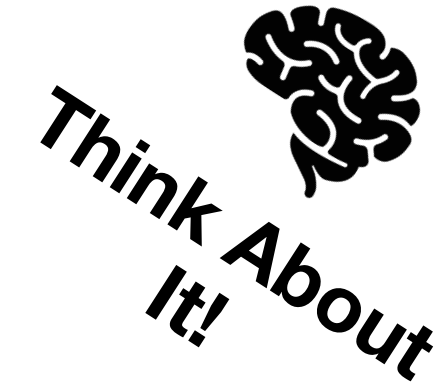
words.sh

```
#!/bin/bash


./word-variations.py Alice_in_Wonderland.txt "$1" > "variations.$1.txt"
```

Components of a Job: Executable

Ex. Counting variations of a word in **a** book



A *generalized* executable & submit file is preferred!

words.sub

```
executable = words.sh
arguments = $(word)

transfer_input_files = Alice_in_Wonderland.txt
queue word from list_of_words.txt
```

list_of_words.txt

```
Dodo
Bird
Cat
Glass
```

words.sh

```
#!/bin/bash


./word-variations.py Alice_in_Wonderland.txt "$1" > "variations.$1.txt"
```

Components of a Job: Executable

Ex. Counting variations of a word in **ANY** book

A *generalized* executable & submit file is preferred!

words.sub

```
executable = words.sh
arguments = $(word)

transfer_input_files = Alice_in_Wonderland.txt
queue word from list_of_words.txt
```

list_of_words.txt

```
Alice_in_Wonderland.txt,Dodo
Alice_in_Wonderland.txt,Bird
Alice_in_Wonderland.txt,Cat
Jurrasic_Park.txt,Dodo
Jurrasic_Park.txt,Bird
Jurrasic_Park.txt,Cat
```

words.sh

```
#!/bin/bash

./word-variations.py Alice_in_Wonderland.txt “$1” > “variations.$1.txt”
```

Components of a Job: Executable

Ex. Counting variations of a word in **ANY** book

A *generalized* executable & submit file is preferred!

words.sub

```
executable = words.sh
arguments = $(word) $(book)

transfer_input_files = $(book)
queue book, word from list_of_words.txt
```

list_of_words.txt

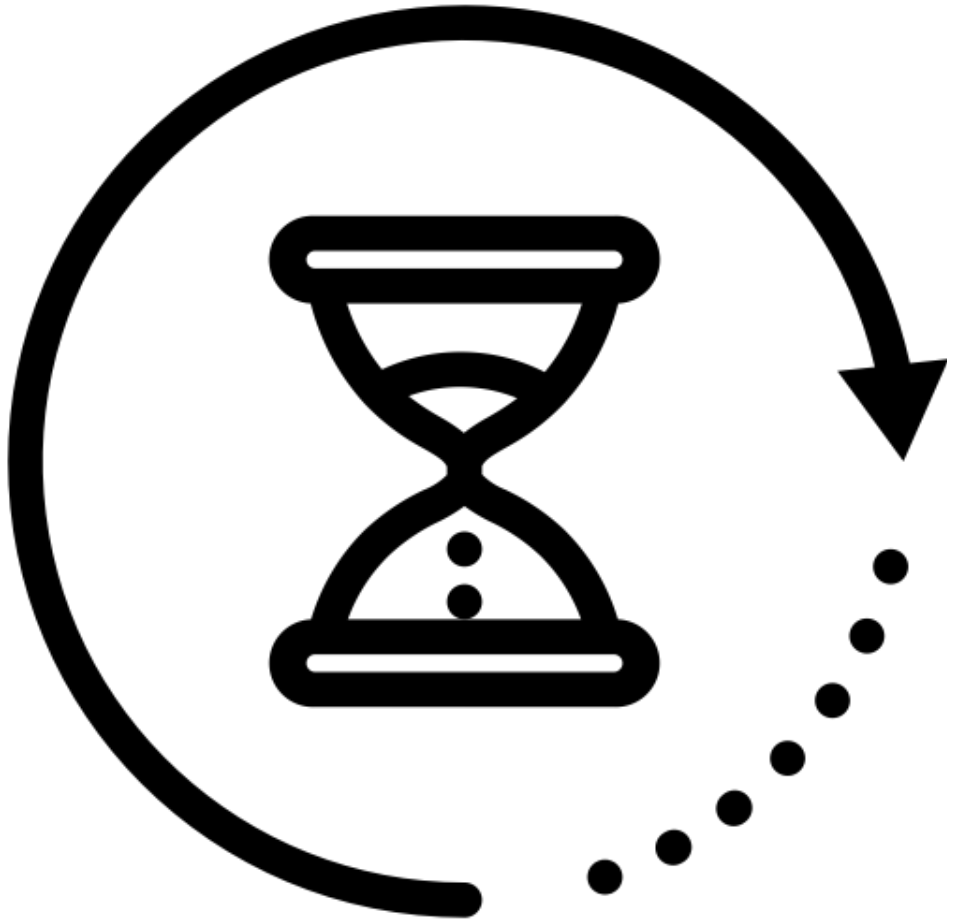
```
Alice_in_Wonderland.txt,Dodo
Alice_in_Wonderland.txt,Bird
Alice_in_Wonderland.txt,Cat
Jurrasic_Park.txt,Dodo
Jurrasic_Park.txt,Bird
Jurrasic_Park.txt,Cat
```

words.sh

```
#!/bin/bash


./word-variations.py “$2” “$1” > “variations.$1.$2.txt”
```

Your jobs run…you return 6 months later to run a new dataset



Created by LUTFI GANI AL ACHMAD
from Noun Project

Components of a Job: Executable

Ex. Counting variations of a word in **ANY** book

A *generalized* executable & submit file is preferred!

words.sub

```
executable = words.sh
arguments = $(word) $(book)

transfer_input_files = $(book)
queue book, word from list_of_words.txt
```

list_of_words.txt

```
Alice_in_Wonderland.txt,Dodo
Alice_in_Wonderland.txt,Bird
Alice_in_Wonderland.txt,Cat
Jurrasic_Park.txt,Dodo
Jurrasic_Park.txt,Bird
Jurrasic_Park.txt,Cat
```

words.sh

```
#!/bin/bash


./word-variations.py “$2” “$1” > “variations.$1.$2.txt”
```

Components of a Job: Executable

Ex. Counting variations of a word in **ANY** book

A *generalized* executable & submit file is preferred!

words.sub

```
executable = words.sh
arguments = $(word) $(book)

transfer_input_files = $(book)
queue book, word from list_of_words.txt
```

list_of_words.txt

```
Alice_in_Wonderland.txt,Dodo
Alice_in_Wonderland.txt,Bird
Alice_in_Wonderland.txt,Cat
Jurrasic_Park.txt,Dodo
Jurrasic_Park.txt,Bird
Jurrasic_Park.txt,Cat
```

words.sh

```
#!/bin/bash


./word-variations.py “$2” “$1” > “variations.$1.$2.txt”
#./word-variations.py Alice_in_Wonderland.txt Dodo > variations.Dodo.txt
```

Components of a Job: Executable

Ex. Counting variations of a word in **ANY** book

A *generalized* executable & submit file is preferred!

words.sub

```
executable = words.sh
arguments = $(word) $(book)

transfer_input_files = $(book)
queue book, word from list_of_words.txt
```

list_of_words.txt

```
Alice_in_Wonderland.txt,Dodo
Alice_in_Wonderland.txt,Bird
Alice_in_Wonderland.txt,Cat
Jurrasic_Park.txt,Dodo
Jurrasic_Park.txt,Bird
Jurrasic_Park.txt,Cat
```

words.sh

```
#!/bin/bash
word="$1"
book="$2"
./word-variations.py "$book" "$word" > "variations.$word.$book.txt"
```

Don't Repeat Yourself (or Others)

- Christina's rule of thumb: If I have copy + pasted something more than twice, it should be a variable.

```
#!/bin/bash

condor_status -const 'Gpus > 0' >> 2024-08-08-gpus.txt
diff ref.txt 2024-08-08-gpus.txt >> 2024-08-08-gpus.txt.diff
```

```
#!/bin/bash

DATE=2024-08-08
RESULT=${DATE}-gpus.txt
condor_status -const 'Gpus > 0' >> $RESULT
diff ref.txt $RESULT >> $RESULT.diff
```

Components of a Job: Managing Files

```
[daniel.morales.1@ap40 home]$ ls
alignment.sh     job_1224366.out  job_1224382.err  job_1224397.log  job_1224412.out  job_1224428.err  job_1224443.log  output_22  output_64
alignment.sub    job_1224367.err  job_1224382.log  job_1224397.out  job_1224413.err  job_1224428.log  job_1224443.out  output_23  output_65
job_1224352.err  job_1224367.log  job_1224382.out  job_1224398.err  job_1224413.log  job_1224428.out  job_1224444.err  output_24  output_66
job_1224352.log  job_1224367.out  job_1224383.err  job_1224398.log  job_1224413.out  job_1224429.err  job_1224444.log  output_25  output_67
job_1224352.out  job_1224368.err  job_1224383.log  job_1224398.out  job_1224414.err  job_1224429.log  job_1224444.out  output_26  output_68
job_1224353.err  job_1224368.log  job_1224383.out  job_1224399.err  job_1224414.log  job_1224429.out  job_1224445.err  output_27  output_69
job_1224353.log  job_1224368.out  job_1224384.err  job_1224399.log  job_1224414.out  job_1224430.err  job_1224445.log  output_28  output_7
job_1224353.out  job_1224369.err  job_1224384.log  job_1224399.out  job_1224415.err  job_1224430.log  job_1224445.out  output_29  output_70
job_1224354.err  job_1224369.log  job_1224384.out  job_1224400.err  job_1224415.log  job_1224430.out  job_1224446.err  output_3   output_71
job_1224354.log  job_1224369.out  job_1224385.err  job_1224400.log  job_1224415.out  job_1224431.err  job_1224446.log  output_30  output_72
job_1224354.out  job_1224370.err  job_1224385.log  job_1224400.out  job_1224416.err  job_1224431.log  job_1224446.out  output_31  output_73
job_1224355.err  job_1224370.log  job_1224385.out  job_1224401.err  job_1224416.log  job_1224431.out  job_1224447.err  output_32  output_74
job_1224355.log  job_1224370.out  job_1224386.err  job_1224401.log  job_1224416.out  job_1224432.err  job_1224447.log  output_33  output_75
job_1224355.out  job_1224371.err  job_1224386.log  job_1224401.out  job_1224417.err  job_1224432.log  job_1224447.out  output_34  output_76
job_1224356.err  job_1224371.log  job_1224386.out  job_1224402.err  job_1224417.log  job_1224432.out  job_1224448.err  output_35  output_77
job_1224356.log  job_1224371.out  job_1224387.err  job_1224402.log  job_1224417.out  job_1224433.err  job_1224448.log  output_36  output_78
job_1224356.out  job_1224372.err  job_1224387.log  job_1224402.out  job_1224418.err  job_1224433.log  job_1224448.out  output_37  output_79
job_1224357.err  job_1224372.log  job_1224387.out  job_1224403.err  job_1224418.log  job_1224433.out  job_1224449.err  output_38  output_8
job_1224357.log  job_1224372.out  job_1224388.err  job_1224403.log  job_1224418.out  job_1224434.err  job_1224449.log  output_39  output_80
job_1224357.out  job_1224373.err  job_1224388.log  job_1224403.out  job_1224419.err  job_1224434.log  job_1224449.out  output_4   output_81
job_1224358.err  job_1224373.log  job_1224388.out  job_1224404.err  job_1224419.log  job_1224434.out  job_1224450.err  output_40  output_82
job_1224358.log  job_1224373.out  job_1224389.err  job_1224404.log  job_1224419.out  job_1224435.err  job_1224450.log  output_41  output_83
job_1224358.out  job_1224374.err  job_1224389.log  job_1224404.out  job_1224420.err  job_1224435.log  job_1224450.out  output_42  output_84
job_1224359.err  job_1224374.log  job_1224389.out  job_1224405.err  job_1224420.log  job_1224435.out  job_1224451.err  output_43  output_85
job_1224359.log  job_1224374.out  job_1224390.err  job_1224405.log  job_1224420.out  job_1224436.err  job_1224451.log  output_44  output_86
job_1224359.out  job_1224375.err  job_1224390.log  job_1224405.out  job_1224421.err  job_1224436.log  job_1224451.out  output_45  output_87
job_1224360.err  job_1224375.log  job_1224390.out  job_1224406.err  job_1224421.log  job_1224436.out  job_1224452.err  output_46  output_88
job_1224360.log  job_1224375.out  job_1224391.err  job_1224406.log  job_1224421.out  job_1224437.err  job_1224452.log  output_47  output_89
job_1224360.out  job_1224376.err  job_1224391.log  job_1224406.out  job_1224422.err  job_1224437.log  job_1224452.out  output_48  output_9
job_1224361.err  job_1224376.log  job_1224391.out  job_1224407.err  job_1224422.log  job_1224437.out  mapping.sh       output_49  output_90
job_1224361.log  job_1224376.out  job_1224392.err  job_1224407.log  job_1224422.out  job_1224438.err  mapping.sub      output_5   output_91
job_1224361.out  job_1224377.err  job_1224392.log  job_1224407.out  job_1224423.err  job_1224438.log  output_1         output_50  output_92
job_1224362.err  job_1224377.log  job_1224392.out  job_1224408.err  job_1224423.log  job_1224438.out  output_10        output_51  output_93
job_1224362.log  job_1224377.out  job_1224393.err  job_1224408.log  job_1224423.out  job_1224439.err  output_100       output_52  output_94
job_1224362.out  job_1224378.err  job_1224393.log  job_1224408.out  job_1224424.err  job_1224439.log  output_11        output_53  output_95
job_1224363.err  job_1224378.log  job_1224393.out  job_1224409.err  job_1224424.log  job_1224439.out  output_12        output_54  output_96
job_1224363.log  job_1224378.out  job_1224394.err  job_1224409.log  job_1224424.out  job_1224440.err  output_13        output_55  output_97
job_1224363.out  job_1224379.err  job_1224394.log  job_1224409.out  job_1224425.err  job_1224440.log  output_14        output_56  output_98
job_1224364.err  job_1224379.log  job_1224394.out  job_1224410.err  job_1224425.log  job_1224440.out  output_15        output_57  output_99
job_1224364.log  job_1224379.out  job_1224395.err  job_1224410.log  job_1224425.out  job_1224441.err  output_16        output_58  proteins.sh
job_1224364.out  job_1224380.err  job_1224395.log  job_1224410.out  job_1224426.err  job_1224441.log  output_17        output_59  proteins.sub
job_1224365.err  job_1224380.log  job_1224395.out  job_1224411.err  job_1224426.log  job_1224441.out  output_18        output_6
job_1224365.log  job_1224380.out  job_1224396.err  job_1224411.log  job_1224426.out  job_1224442.err  output_19        output_60
job_1224365.out  job_1224381.err  job_1224396.log  job_1224411.out  job_1224427.err  job_1224442.log  output_2         output_61
job_1224366.err  job_1224381.log  job_1224396.out  job_1224412.err  job_1224427.log  job_1224442.out  output_20        output_62
job_1224366.log  job_1224381.out  job_1224397.err  job_1224412.log  job_1224427.out  job_1224443.err  output_21        output_63
```

Components of a Job: Managing Files

Organization is critical to scaling up jobs on the OSPool!

Before beginning any computational project, consider:

1. Directory Structure
   - Create separate folders for `inputs`, `outputs`, `logs`, and `scripts`.
2. File Naming Conventions
   - Use consistent, descriptive names (`sample_A1_reads.fastq`).
3. Separation of Shared vs. Job-Specific Files
4. Reproducibility and Reusability
   - Use version-controlled scripts and log job parameters with outputs.

Components of a Job: Managing Files
Directory Structure - Organizing Files and Directories

Directory structure and an organizational plan is step 0 of any scaled HTC workflow.

- Use separate folders for inputs, outputs, and logs.
- Use flat and shallow structures
- Keep job-specific files local /home/
- Place shared files in /ospool/ap##/data/.
- Group logs by type: log/, error/, and output/ for easy troubleshooting.

Example Directory Structure:

```
├── /home/<user.name>/
│   ├── <project_name>/
│   │   ├── inputs/          # Job-specific inputs
│   │   ├── outputs/         # Job-generated outputs
│   │   ├── logs/            # Log files
│   │   │   ├── log/         # condor log files
│   │   │   ├── error/       # stderr
│   │   │   └── output/      # stdout
├── /ospool/ap40/data/<user.name>/
│   ├── <project_name>/
│   │   ├── inputs/          # Shared input files
│   │   ├── software/        # Containers
```

Components of a Job: Managing Files
File Naming Conventions

Always use a consistent file naming convention!

- Use lowercase letters, numbers, and underscores
- Avoid spaces and special characters
- Be consistent and descriptive
- Include processing step or tool in the name if applicable
- Use unique names for file versions
  - ⚠️**This is especially important for files on the OSDF!**⚠️

Example Directory Structure:

```
├── /home/<user.name>/
│   ├── <project_name>/
│   │   ├── inputs/          # Job-specific inputs
│   │   ├── outputs/         # Job-generated outputs
│   │   ├── logs/            # Log files
│   │   │   ├── log/         # condor log files
│   │   │   ├── error/       # stderr
│   │   │   └── output/      # stdout
├── /ospool/ap40/data/<user.name>/
│   ├── <project_name>/
│   │   ├── inputs/          # Shared input files
│   │   │   ├── v4_Celegans_N2_ref_genome.fa
│   │   ├── software/        # Containers
```

Jenny Bryan How to Name Files Tutorial:
https://github.com/jennybc/how-to-name-files

Components of a Job: Managing Files
Separating Job-Specific and Shared Files

Job-Specific Files

These files are unique to each job and typically small in size.

Examples:

- Read subsets (e.g., reads_001.fastq)
- Job argument files (e.g., input_ids.txt)
- Condor log, output, and error files
- Temporary/intermediate files
- Per-job config or metadata (e.g., params_42.json)

Why here?

- These files are often created or modified during the job.
- They're not reused across jobs, so storing them locally avoids unnecessary staging overhead.

Job Shared Files

These files are read-only and shared across many jobs.

Examples:

- Software containers (e.g., minimap2.sif)
- Reference genomes
- Pre-built indexes (e.g., ref.mmi, genome.fai)
- Models, Databases, or fixed datasets

Why here?

- These files are large, used across many jobs, and do not change.
- The OSDF protocol caches copies of these files near EPs

Components of a Job: Managing Files
Reproducibility and Reusability with Version Control

Why Version Control Matters in HTC Projects

- Tracks every change to your code and config
- Helps debug errors by rolling back to known-good states
- Makes your workflow portable and shareable

Best Practices for Version Control

- Use Git for scripts, config files, and README documentation
- Commit early and often, especially before large-scale job submissions
- Use meaningful commit messages
- Push your code to a remote repository (GitHub, GitLab, etc.) to back it up

Software Carpentries Git Tutorials:
https://swcarpentry.github.io/git-novice/

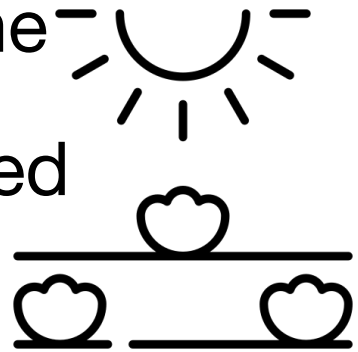
Stage 1: Tips for Initial Test Jobs

- Select smaller data sets or subsets of data for your first test jobs
- Request 1 CPU, 4-16GB of RAM, estimate disk
- Pick test jobs that will reproduce results, if possible
- **Name files** carefully (more on this later)
- Make sure you understand and can run your software
  - Software — executable, dependencies, maybe a wrapper script to prepare environment
  - Command-line arguments
  - Input files

Stage 2: Run a Small Workload

Scale up to ~10 jobs, checking reliability & Access Point resource demand

- Try a representative variety of arguments and input files
- Start developing methods for checking results of *all* jobs
  - Checking 1 job is easy; checking 10 is tedious; checking 1000s by hand?
- Estimate total resource needs (quotas) for the Access Point itself
- Repeat tests at this scale until issues are fixed & resources are accurate

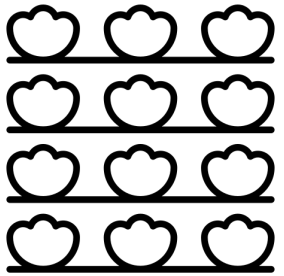


Stage 3: Scale Up!

Continue scaling up in 10–100× increments, checking for & fixing issues

As you scale up, a challenge is to distinguish among:

- Real issues with your jobs
- Real issues with SOME of your jobs
- Temporary issues with the OSPool/Access Point
- Bugs and other longer-lasting issues with the infrastructure
- (We can help! Email us with support requests if you get stuck.)



Created by Made from the Noun Project

Think about the best practices from the following slides.

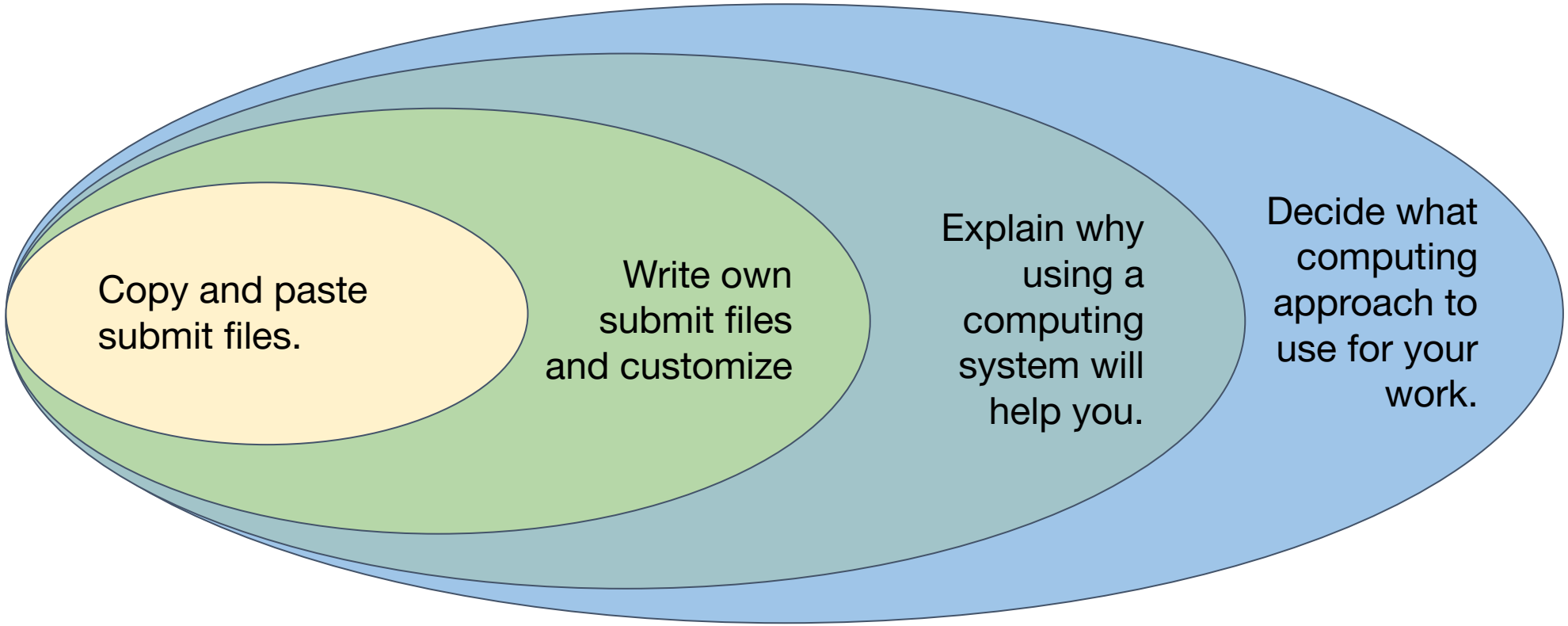
Scaling: Computing Strategy

OSG School Secret Agenda

Learn about HTC methods (and mechanics) and apply them to your work

Make intentional choices about how you approach computing.

Expanding Our Understanding



Computing Problem Flavors

- List of tasks
  - data processing
  - probabilistic style simulations
  - and many more!
- Approach: lots of cores
  - Multiple cores on a server
  - HTC jobs
  - Job array

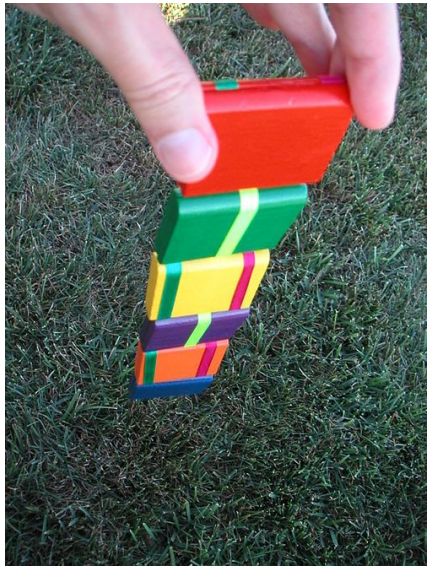
Computing Problem Flavors

- Coordinated sub-tasks
  - optimization problems (including machine learning training!)
  - physical system simulations
- Approach: lots of cores working together
  - HPC cluster
  - Node with multiple cores
  - GPUs

Computing Problem Flavors

- Reading in lots of data
  - image combination, genome assembly
- Approach: lots of memory

- Long running processes
  - MCMC chains.
- Approach: …



Computing Solution Considerations

- **Scale**
  - Local server → Local cluster → National system
- **"Style"**
  - Optimized for multi-core, multi-node work (traditional HPC cluster)
  - Optimized for multiple jobs (HTC system like OSPool)
  - Focus on special hardware: GPUs, memory
- **Service model**
  - Allocations? Monetary cost? Buy-in?
- **Security**

Trade-Offs: Resources

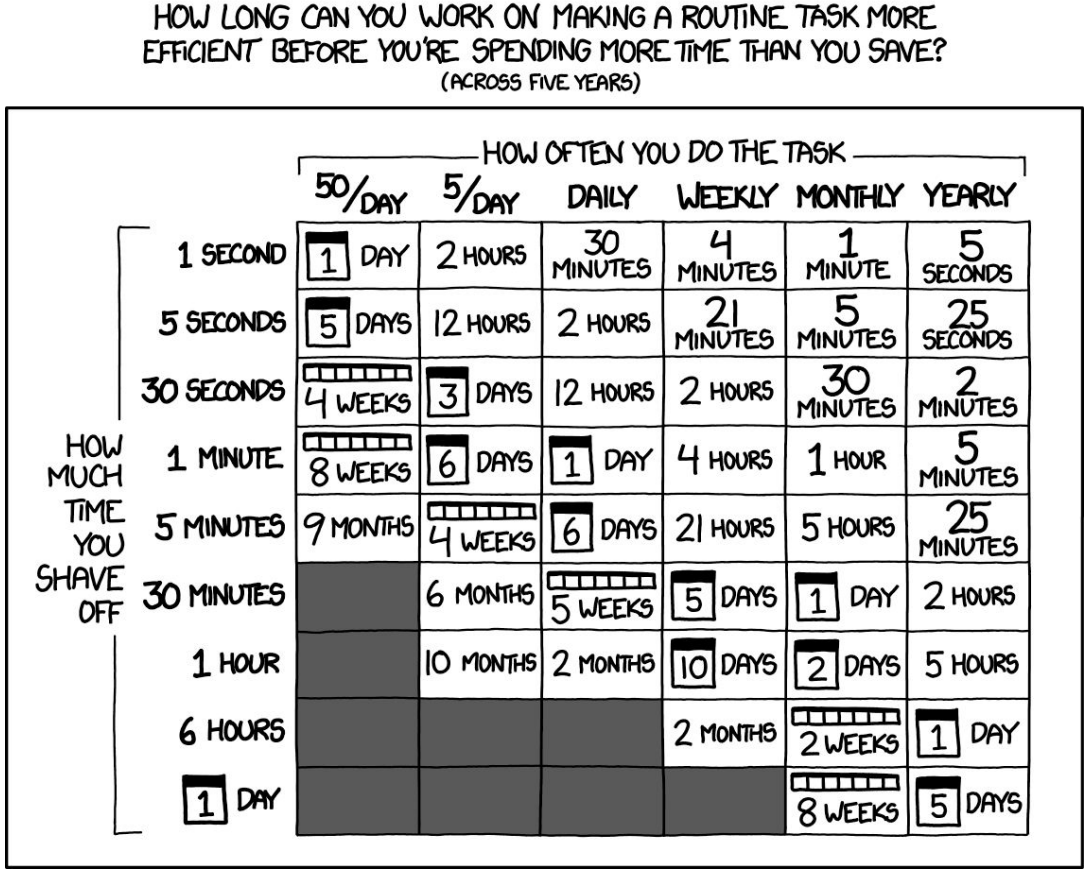
You will not always have a perfect resource for your problem!

Some questions to consider:

- What does your computing problem/approach need?
- What computing resources do you have?
- What resources *could* you have? With how much effort?
- Is that effort worth it?
- Would your outcome change if changed your approach?
- How much effort would that take?

Trade-Offs: Automation

- Once you have an approach and resource, consider automating your work.
- Possibilities:
  - Uploading and downloading files
  - Checking for errors
  - Reviewing job information (run times, memory usage)



https://xkcd.com/1205/ (the chart above)

https://xkcd.com/1319/ (less optimistic)

Trade-Offs: Motivation

You will not always care about all this!
Let's just get s*** done!

- Start small.
- Know what's important to you.
- Connect with others.

Revisiting HTC

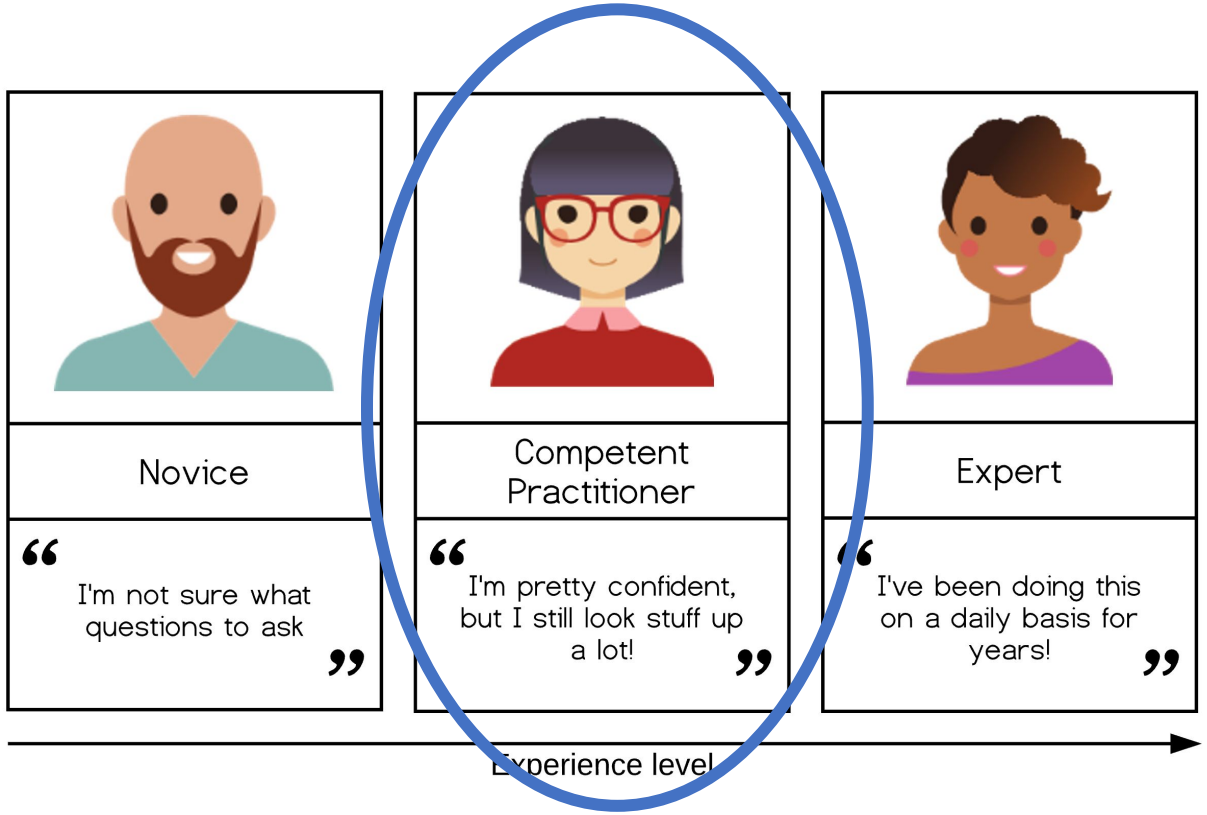
- What components do you need for your HTC workload?
- Scaling Up
  - Where are you in the scaling up process?
  - What are three things to consider in your current stage?
- What organizational strategy makes sense for the next steps in your analysis?
- Consider the balance between human effort (yours!) and computer time; will the use of HTC actually save you time in the long run and improve your research?

Scaling: Your Skills

What does success look like?

- Achieving independence in research computing doesn't mean you know everything!
- Becoming an HTC **Practitioner**
  - Understand core concepts
  - Know what questions to ask (how to Google!)
  - Can use documentation to solve problems

See more in Appendix A



https://carpentries.github.io/instructor-training/02-practice-learning.html

Becoming an (HTC) Practitioner

In general:

- Building mental models of the topic
- Lots of practice, learning from mistakes
- Looking at examples
- Mentorship from other practitioners

At the OSG School:

- Lectures on big picture concepts (with analogies, examples)
- Hands on exercises
- Consultations and support from staff

Learning HTC

- What does “independence in research computing” mean to you?
- When learning something new what resources out in the world have been transformative for you in becoming an independent practitioner? What made you feel part of a community?
- Did you know about our guides? How could we make them more “findable?” Which is most useful (in your opinion)? What is missing?

Resources We Have Now

- Guides/Documentation
  - OSPool Guides: https://portal.osg-htc.org/documentation/
  - HTCondor Manual: https://htcondor.readthedocs.io/en/latest/index.html
- Tutorials
  - Tutorials on our main docs page
    - https://portal.osg-htc.org/documentation/htc_workloads/submitting_workloads/tutorial-command/
  - Or on GitHub: https://github.com/OSGConnect/
  - Or some default notebooks: https://notebook.ospool.osg-htc.org/
- OSG School materials: https://osg-htc.org/school-2025/materials/
- Videos and Slides
  - YouTube: https://www.youtube.com/channel/UCVxyV0Lr1KiTeq7bTw3gwLw
  - Training materials: https://portal.osg-htc.org/documentation/support_and_training/training/osgusertraining/

Community Support

- We host:
  - Weekly user-focused office hours
  - Monthly training
  - OSG School (weeklong summer school)
  - Throughput Computing (once a year)
- We don’t yet have a way for researchers to connect and share with each other…what could that look like?

https://portal.osg-htc.org/documentation/support_and_training/support/getting-help-from-RCFs/

Planning Today and Tomorrow

- Revisit your goals from Monday! How would you change them? What would help you achieve your goals in the next two days?
  - Fully implement a single job?
  - Scaffold a large submission
  - Run through multiple examples?
  - Ask questions of staff?
- Lots of work time this morning and afternoon
- Talk to staff!!

Questions

Appendix A: Practitioner

There are many people who have thought about how to gain skills and become a practitioner.

Various Ways to Think about Competency

- Great seven step summary: https://www.dotkam.com/2007/05/07/the-seven-stages-of-expertise-in-software-engineering/
- Carpentries Instructor Training Material: https://carpentries.github.io/instructor-training/02-practice-learning.html
- https://en.wikipedia.org/wiki/Four_stages_of_competence
- https://en.wikipedia.org/wiki/Dreyfus_model_of_skill_acquisition

Appendix B: Scaling Up

Additional tips for each stage of the scaling up process

Stage 1: Tips for Initial Test Jobs

- Test one of each kind of job you will run (e.g., prep, simulation, analysis)
- Select smaller data sets or subsets of data for your first test jobs
- Pick test jobs that will reproduce results, if possible
- **Name files** carefully
- Make sure you understand and can run your software
  - Software — executable, dependencies, maybe a wrapper script to prepare environment
  - Command-line arguments
  - Input files

Stage 1: Estimating Initial Resource Needs

CPU

- By default, start with 1
- Unless you know for sure that you executable uses a certain number > 1

Memory

- Start with the total memory available on laptop or where it ran before
- It’s ok if this is a lot the first time, you will fine-tune later

Disk

- Estimate (as best you can) and sum sizes of: executable (+ environment), input files, output files, temporary files, standard output/error

Stage 1: Run, Refine, Repeat

After running a test job:

- Check logs and output for errors, warning, holds, etc.
- Check HTCondor job log for actual resource usage
- Fix issues, update resource needs, run 1 job again!

```
005 (1234.000.000) 2022-07-28 09:12:34 Job terminated.
    (1) Normal termination (return value 0)
[...]
    Partitionable Resources :    Usage  Request Allocated
       Cpus                 :                1         1
       Disk (KB)            :       40       30   4203309
       Memory (MB)          :        1        1         1
```

Stage 2: Try Various Inputs

For Stage 1, the suggestion was to keep things short and simple

- For Stage 2, it is time to explore the entire range of inputs to your jobs
  - Different command-line arguments; e.g., start, middle, and end of parameter sweep
  - Different input files; e.g., small, medium, and large — whatever makes sense for you
- As you explore, you may find that per-job resource needs vary
- Set your resource requests a bit higher than maximum observed usage
  - For example, if 10 test jobs used between 938 MB – 1.23 GB of memory, update your submit file to request 1.5 GB memory
- After any changes, run the same test again and re-evaluate

Stage 2: Checking Results of Multiple Jobs

Start developing methods for checking the results of multiple jobs

- Output from your executable (i.e., your research results)
- Debugging output: standard output and error files, executable logs, etc.
- HTCondor job log file (`log = job.log` in your submit file)

This may be one of the most overlooked aspects of scaling up!

- Checking 1 job is easy; checking 10 is tedious; checking 1000s by hand? 😭
- Techniques include:
  - Sampling
  - Developing tools to automate
- Sounds a bit like research, right? You know how to do that…

Stage 2: Estimate Access Point Needs

Do not forget about your Access Point – it is a shared resource, too!

- Storage space for files
  - Based on a run of 10 jobs, estimate total number and size of all files for full production
  - Do you have enough storage space on the Access Point? If not, what options exist?
- Number of running jobs
  - In theory, how many jobs could you have running at once?
  - Each running job uses some CPU and memory on the Access Point itself
  - If submitting over 10,000 jobs consider limiting (*throttling*) running and idle jobs on Access Point

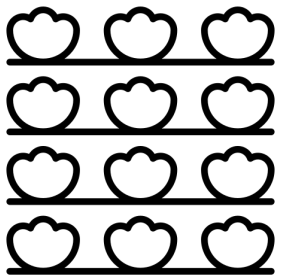
Stage 3: Iterate in Steps of 10–100×

By now, you have tested tens of jobs, maybe in a workflow; what next?

- Continue scaling up in increments of 10–100 times the number of jobs
- All the considerations from Stage 2 apply at each increment
- Be sure to find, understand, and hopefully fix issues before moving on

As you scale up, a challenge is to distinguish among:

- Real issues with your jobs, workflow, resource requests, etc.
- Real issues with certain subsets of your jobs
- Temporary issues with the HTC infrastructure itself
- Bugs and other longer-lasting issues with the infrastructure
- We can help! Email us with support requests if you get stuck.



Created by Made from the Noun Project

Appendix C: File Organization

A worked example (similar to the exercises) on how you might organize the files in a job submission

Example: Text Analysis

Book text to analyze

Python script that counts the frequency of different words

Output counts of different words in book

```
$ ./wordcount.py Dracula.txt
```

open book by Soremba from the Noun Project
PY File by Arthur Shlain from the Noun Project
Number by Travis Avery from the Noun Project

Organizational Plan For Our Files

```
wordcount.sub
wordcount.py
input/
        Dracula.txt
        ...
output/
        count.Dracula.txt
        ...
```

We will assume that we want to put our input files (books) in one folder, and our output files (word counts) in another folder.

Organizational Plan For Our Files

```
wordcount.sub
wordcount.py
input/
        Dracula.txt
        ...
output/
        count.Dracula.txt
        ...
log/
  job.0.log
  ...
errout/
  job.0.out
  job.0.err
  ...
```

There are ***additional*** files that will be produced by the job as well that we should consider – the HTCondor log, stdout and stderr. We'll put these into two folders.

Coordinate HTCondor and File Structure

```
wordcount.sub
wordcount.py
input/
        Dracula.txt
        ...
output/
        count.Dracula.txt
        ...
log/
        job.0.log
        ...
errout/
        job.0.out
        job.0.err
        ...
```

```
# submit file name: wordcount.submit
executable = wordcount.py
arguments = Dracula.txt

transfer_input_files = inputs/Dracula.txt
transfer_output_remaps = \
 “count.Dracula.txt = outputs/count.Dracula.txt”

log = logs/$(ProcId).log
error = errout/$(ProcId).err
output = errout/$(ProcId).out

queue 1
```

HTCondor Options for Organizing Files

Syntax	Purpose	Features

`Transfer_output_remaps =`
`“file1.out=path/to/file1.out;`
`file2.out=path/to/renamedFile2.out”`
	Used to save output files in a specific path and using a certain name	- Used to save output files to a **specific folder**
		- Used to **rename** output files to avoid writing over existing files

`Initialdir =` `path/to/initialDirectory`	Sets the submission directory for each job. When set, this is becomes the base path where output files will be saved.	- Used to submit multiple jobs from **different directories** - Used to avoid having to write some paths in other submit file values

More Information: https://htcondor.readthedocs.io/en/latest/users-manual/file-transfer.html

Return Output to Specified Directory with InitialDir

```
submission_dir/
        job.sub
        exec.py
        shared_vars.txt
        results/
                input.txt
                output.txt
                job.err
                job.log
                job.out
```

```
# File name: job.sub
executable = exec.py

initialdir = results
transfer_input_files = input.txt,
  ../shared_vars.txt

log = job.log
out = job.out
error = job.err

queue 1
```

Separate Jobs with InitialDir

```
submission_dir/
        job.submit
        analyze.exe
        job0/
                file.in job.log job.err
                file.out job.out
        job1/
                file.in job.log job.err
                file.out job.out
        job2/
                file.in job.log job.err
                file.out job.out
```

```
# File name = job.submit

executable = analyze.exe
initialdir = job$(ProcId)

arguments = file.in file.out
transfer_input_files = file.in

log = job.log
error = job.err
output = job.out

queue 3
```

Executable should be in the directory with the submit file, *not* in the individual job directories

Organizing Data Files

Some HTC systems will have you place small files in one directory and larger files in a different directory.

For example, on ap40 (an OSPool Access Point), we place files less than 1GB in /home and larger files in an OSDF origin.

Once inputs and outputs are placed in the right location, use the appropriate HTCondor file transfer options to move the data to jobs.

Appendix D: Adding Arguments

How to use arguments in R, Python and bash

https://gist.github.com/ChristinaLK/a672cdd5dc0c0664557befb4df69d98e

Arguments in Python

```
$ python3 args.py 1 hello True

[“args.py”, “1”, “hello”, “True”]
”1”
“results1.csv”
```

```
## args.py

import sys
values = sys.argv

print(values)

jobnumber = values[1]
print(jobnumber)

outfile = 'results'+ jobnumber + '.csv’
print(outfile)
```

Arguments in R

```
$ Rscript args.R 1 hello True

[1] “1”, “hello”, “True”
[1] ”1”
[1] “results1.csv”
```

```
## args.R

values <- commandArgs(trailingOnly=TRUE)

print(values)

jobnumber <- values[1]
print(jobnumber)

outfile <-
   paste0('results',jobnumber,'.csv')
print(outfile)
```

Arguments in Bash (shell)

```
$ ./args.sh 1 hello True

1 hello True
1
results1.csv
```

```
## args.sh

import sys
values=$@

echo values

jobnumber=$1
echo jobnumber

outfile=results${jobnumber}.csv
echo outfile
```

Appendix E: Automation

General tools that are useful for automating things

Unix Tips and Tricks

- Aliases and shell configuration: https://carpentries-incubator.github.io/shell-extras/07-aliases/index.html
- SSH configuration (for example: https://chtc.cs.wisc.edu/uw-research-computing/configure-ssh)
- Super useful shell commands: cut, grep, sort, uniq, head, tail (plus pipes: https://swcarpentry.github.io/shell-novice/04-pipefilter.html)
  - Just for fun, see your most commonly used commands with:
  - `cat ~/.bash_history | cut -d " " -f 1 | sort | uniq -c | sort -nr | head -n 20`
- Scripting! https://swcarpentry.github.io/shell-novice/06-script.html

Appendix F: Job Information

How to get more information about your jobs using HTCondor job attributes and searching through the log file

Job Attributes with condor_q

HTCondor stores a list of information about each job.

This information is stored in this format:

- `AttributeName = value`

You can find a list of attributes for a single job by running:

- `condor_q -l JobID`

You can print out specific attributes by using the “format” or “auto-format” flags with an HTCondor command:

- `condor_q -af Attribute1 Attribute2`
- adds job number: `condor_q -af:j Attribute1 Attribute2`

Interesting Job Attributes

Job identifying information

- ClusterID
- ProcID
- Cmd
- Arguments
- UserLog

Where it ran

- LastRemoteHost
- MATCH_EXP_JOBGLIDEIN_ResourceName

Resource Request and Usage

- RequestCpus (Memory, Disk)
- MemoryProvisioned (Disk)
- CPUsUsage (MemoryDisk)

Timing

- EnteredCurrentStatus
- QDate

Interesting Job Attributes

Codes

- JobStatus
- ExitCode
- HoldReasonCode
- HoldReasonSubCode,
- NumHoldsByReason

Counts

- NumJobStarts
- NumShadowStarts
- NumSystemHolds,

Checking Completed Jobs

condor_history

- Contains finalized job attributes for completed jobs
  - some have different names (HoldReason --> LastHoldReason)
- Easy to use constrain and to display values (like condor_q)
- Can be slow to search (latest first) and may drop old records quickly

HTCondor job log files (log = *job.log* in submit)

- Contain a lot of information
- That is both a blessing and a curse
- Somewhat easy to parse — or use HTCondor Python bindings/other tools to help

HTCondor Job Log Files

- One big, combined file, or one per job? Your preference, really
- With tens or hundreds of jobs (& more), not practical to review manually
  - Can try to use the grep command-line tool to find specific lines
  - Use a tool to summarize log results (logs2csv)

HTCondor Job Log Files: Terminations

To find when every job ended:

```
$ grep '^005' LOGS
```

(*LOGS* can be one file, a list of files, or a glob (using *) of files)

To find termination codes (exit codes) for every job:

```
$ grep termination LOGS
```

(will not show job IDs, though)

To get counts by termination code:

```
$ grep termination LOGS | sort | uniq -c
```

HTCondor Job Log Files: Resource Lines

To get memory resource lines:

```
$ grep -h 'Memory (MB) *:' LOGS > memory_resources.txt
```

To get disk resource lines:

```
$ grep -h 'Disk (KB) *:' LOGS > disk_resources.txt
```

Import the resulting files into Excel (with some attention to import options)

For file transfers:

```
$ grep -h 'Total Bytes Sent By Job' LOGS
$ grep -h 'Total Bytes Received By Job' LOGS
```

HTCondor Job Log Files: Checking on Holds

To view all job holds, their reasons, and related codes:

```
$ grep -h -A 2 '^012' LOGS
```

(Omit the -h option to see log filenames for each hit.)

Note: The OSPool may automatically release (rerun) some held jobs; if you don't look for them explicitly, you may never know those holds occurred

HTCSS Job Event Log to CSV Summarizer

A simple script that reads HTCondor job event logs and outputs a CSV summary of job statistics.

```
[alice@ap40 logs]$ logs2csv *.log > summary.csv
```

	A	B	C	D	E	F	G	H	I	J	K	L
1	JobId	Submitted	ExecCount	EvictCount	TermCount	TermNorm	TermAbnorm	ShdwExcpt	Abrts	Suspnds	Unsus	Holds
2	15505	9/28/22 9:51	1	0	0	0	0	1	1	0	0	1
3	15507	9/28/22 10:05	1	0	0	0	0	1	1	0	0	1
4	15508	9/28/22 11:56	1	0	0	0	0	1	1	0	0	1
5	15509	9/28/22 15:22	1	0	0	0	0	1	1	0	0	1
6	15510	9/28/22 15:32	0	0	0	0	0	1	1	0	0	1
7	15511	9/28/22 16:46	1	0	0	0	0	1	1	0	0	1
8	15514	9/29/22 10:07	0	1	0	0	0	0	1	0	0	0
9	15515	9/29/22 10:09	1	0	0	0	0	1	1	0	0	1
10	15520	9/30/22 9:44	1	0	1	1	0	0	0	0	0	0
11	15651	10/27/22 13:12	1	0	1	1	0	0	0	0	0	0
12	15653	10/27/22 13:15	1	0	1	1	0	0	0	0	0	0
13	15654	10/27/22 13:16	1	0	1	1	0	0	0	0	0	0
14	282	6/28/22 12:20	1	0	1	1	0	0	0	0	0	0
15	283	6/29/22 9:22	1	0	0	0	0	1	1	0	0	1
16	284	6/29/22 9:30	1	0	0	0	0	1	1	0	0	1

https://github.com/osg-htc/job-event-log-to-csv

Appendix G: Further Reading

Christina's favorite references and lessons for general research computing skills and best practices

Reading

- So You Want to Be a Wizard: https://wizardzines.com/zines/wizard/
  - (All of Julia's zines are amazing!!!!!)
- Best Practices for Scientific Computing: https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745
- Good Enough Practices for Scientific Computing: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510#sec027
- 10 Simple Rules for Making Research Software More Robust: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005412
- Naming Files: http://www2.stat.duke.edu/~rcs46/lectures_2015/01-markdown-git/slides/naming-slides/naming-slides.pdf
  - (Version 2, from 2022: https://github.com/jennybc/how-to-name-files)

Lessons and Books

- Software Carpentry/Incubator Lessons:
  - Unix Shell: https://swcarpentry.github.io/shell-novice/
  - Shell Extras (not super great, but some useful info): https://carpentries-incubator.github.io/shell-extras/
  - Version Control with Git: https://swcarpentry.github.io/git-novice/
  - Docker Intro: https://carpentries-incubator.github.io/docker-introduction/
- Research Software Engineering with Python
  - https://third-bit.com/py-rse/

Acknowledgements

- This work was supported by the National Science Foundation under Cooperative Agreement OAC-2030508 – Partnership to Advance Throughput Computing (PATh)