Submitting Multiple Jobs With HTCondor

Amber Lim

OSG User School 25

Objectives

After this lecture, you should be able to:

- Submit lists of jobs using a single submit file
- Define and use variables in your submit file
- Strategize how to structure files related to your job

Why multiple jobs?

Mei Monte Carlo

Needs to run many random simulations to model particles in a detector

Image credit: [The Carpentries Instructor Training]

Why multiple jobs?

Mei Monte Carlo	**Tamara Trials**
	
Needs to run many random simulations to model particles in a detector.	Testing different design parameters for designing clinical trials.

Image credit: [The Carpentries Instructor Training]

Why multiple jobs?

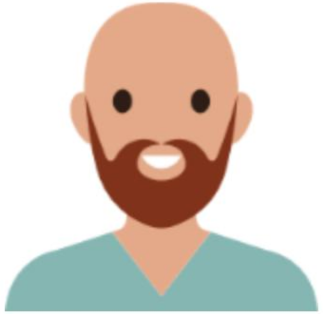
Mei Monte Carlo	Tamara Trials	Ben Bioinformatics
		
Needs to run many random simulations to model particles in a detector.	Testing different design parameters for designing clinical trials.	Applying a quality control / processing pipeline to 20 RNA samples.

Image credit: [The Carpentries Instructor Training]

Why multiple jobs?

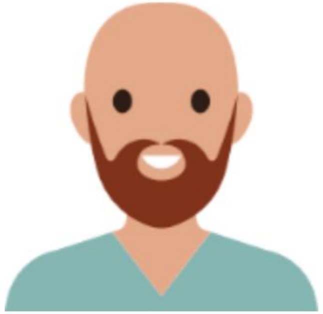
Mei Monte Carlo	**Tamara Trials**	**Ben Bioinformatics**
		
Needs to run many random **simulations** to model particles in a detector.	Testing different design **parameters** for designing clinical trials.	Applying a quality control / processing pipeline to 20 RNA **samples**.

Image credit: [The Carpentries Instructor Training]

Why multiple jobs?

Mei Monte Carlo

Needs to run many random simulations to model particles in a detector

Tamara Trials

Testing different design parameters for designing clinical trials.

Ben Bioinformatics

Applying a quality control / processing pipeline to 20 RNA samples.

When running many jobs we want to avoid:

- starting each job manually
- creating separate submit files for each job

Image credit: The Carpentries Instructor Training

Many jobs, one submit file

HTCondor

to the rescue

HTCondor has several built-in ways to submit many independent jobs from one submit file

Let’s review: one job

```
executable = analyze.sh
arguments  = file.in file.out
transfer_input_files = file.in

log    = job.log
output = job.stdout
error  = job.stderr

queue
```

This is the command we want HTCondor to run.

Let's review: one job

```
executable = analyze.sh
arguments  = file.in file.out
transfer_input_files = file.in

log     = job.log
output = job.stdout
error  = job.stderr

queue
```

These are the files we need for the job to run.

Let’s review: one job

```
executable = analyze.sh
arguments  = file.in file.out
transfer_input_files = file.in

log    = job.log
output = job.stdout
error  = job.stderr

queue
```

These files track information about the job.

Let’s review: one job

```
executable = analyze.sh
arguments  = file.in file.out
transfer_input_files = file.in

log    = job.log
output = job.stdout
error  = job.stderr

queue
```

The queue term tells HTCondor how many jobs to run.

Submitting Multiple Jobs

When submitting multiple jobs using one submit file, it is helpful to start by thinking about:

1. What is ***constant*** across all jobs?
2. What is ***changing*** from job to job?

Submitting Multiple Jobs

When submitting multiple jobs using one submit file, it is helpful to start by thinking about:

1. What is ***constant*** across all jobs?
2. What is ***changing*** from job to job?

When editing the submit file,
it is helpful to start by editing the **queue** statement.

Variable and `queue` options

Syntax	List of Values	Variable Name
queue ***N***	Integers: 0 through N-1	$(ProcId)
queue *Var* **matching** *pattern**	List of values that match the wildcard pattern.	$(*Var*) If no variable name is provided, default is $(Item)
queue *Var* **in** (*item1 item2 …*)	List of values within parentheses.	
queue *Var* **from** *list*	List of values from *list*, where each value is on its own line.	

Variable and `queue` options

Syntax	List of Values	Variable Name
queue ***N***	Integers: 0 through N-1	$(ProcId)
queue *Var* **matching** *pattern**	List of values that match the wildcard pattern.	$(*Var*) If no variable name is provided, default is $(Item)
queue *Var* **in** (*item1 item2* …)	List of values within parentheses.	
queue *Var* **from** *list*	List of values from *list*, where each value is on its own line.	

Example 1:

`Queue *variable* from *list*`

Example 1

Scenario: Use an executable to analyze Wisconsin population data

```
$ ./compare_states state.wi.dat out.state.wi.dat
```

```
executable = compare_states
arguments  = state.wi.dat out.state.wi.dat

transfer_input_files = state.wi.dat

queue
```

Example 1

Scenario: Use an executable to analyze Wisconsin population data

Suppose we have data for all 50 states: `state.wi.dat`, `state.mn.dat`, `state.il.dat`, …

Let’s use HTCondor to automatically queue a job to analyze each state’s data file!

```
e
arguments   = state.wi.dat out.state.wi.dat

transfer_input_files = state.wi.dat

queue
```

Provide a list of values with `queue … from`

One option is to create another file with the list of input files and use the **`queue`** ***`variable`*** **`from`** ***`list`*** syntax.

File name: state_list.txt

```
state.wi.dat
state.mn.dat
state.il.dat
state.ia.dat
state.mi.dat
```

```
executable = compare_states
arguments  = state.wi.dat out.state.wi.dat

transfer_input_files = state.wi.dat

queue state from state_list.txt
```

Which job components vary?

- Now, what parts of our submit file vary depending on the input?
- We want to vary the job's **arguments** and one **input file**.

```
executable = compare_states
arguments  = state.wi.dat out.state.wi.dat

transfer_input_files = state.wi.dat

queue state from state_list.txt
```

Use a custom variable

Replace all our varying components in the submit file with a variable.

```
state.wi.dat
state.mn.dat
state.il.dat
state.ia.dat
state.mi.dat
```

```
executable = compare_states
arguments  = $(state) out.$(state)

transfer_input_files = $(state)

queue state from state_list.txt
```

Use multiple variables with `queue … from`

- The queue from syntax can also support multiple values per job.
- Suppose our command was like this:

```
$ ./compare_states -i [input file] -y [year]
```

File name: state_list.txt

```
state.wi.dat,2010
state.wi.dat,2015
state.mn.dat,2010
state.mn.dat,2015
```

```
executable = compare_states
arguments  = -i $(state) -y $(year)

transfer_input_files = $(state), country.us.dat

queue state,year from state_list.txt
```

Example 2:

`Queue` *`N`*

List of numerical input values

Suppose we have many input files and we want to run one job per input file.

`file.0.in`

`file.1.in`

`file.2.in`

`file.3.in`

`file.4.in`

List of numerical input values

Suppose we have many input files and we want to run one job per input file.

We can capture this set of inputs using a **list of integers**.

`file.0.in`

`file.1.in`

`file.2.in`

`file.3.in`

`file.4.in`

Provide a list of integer values with `queue N`

```
executable = analyze.sh
arguments  = file.in file.out
transfer_input_files = file.in

log    = job.log
output = job.stdout
error  = job.stderr

queue 5
```

This queue statement will generate a list of integers, 0 - 4

Provide a list of integer values with `queue N`

```
executable = analyze.sh
arguments  = file.in file.out
transfer_input_files = file.in

log    = job.log
output = job.stdout
error  = job.stderr

queue 5
```

If we *only* change our queue statement to queue N, HTCondor will queue N *identical* jobs.

This queue statement will generate a list of integers, 0 - 4

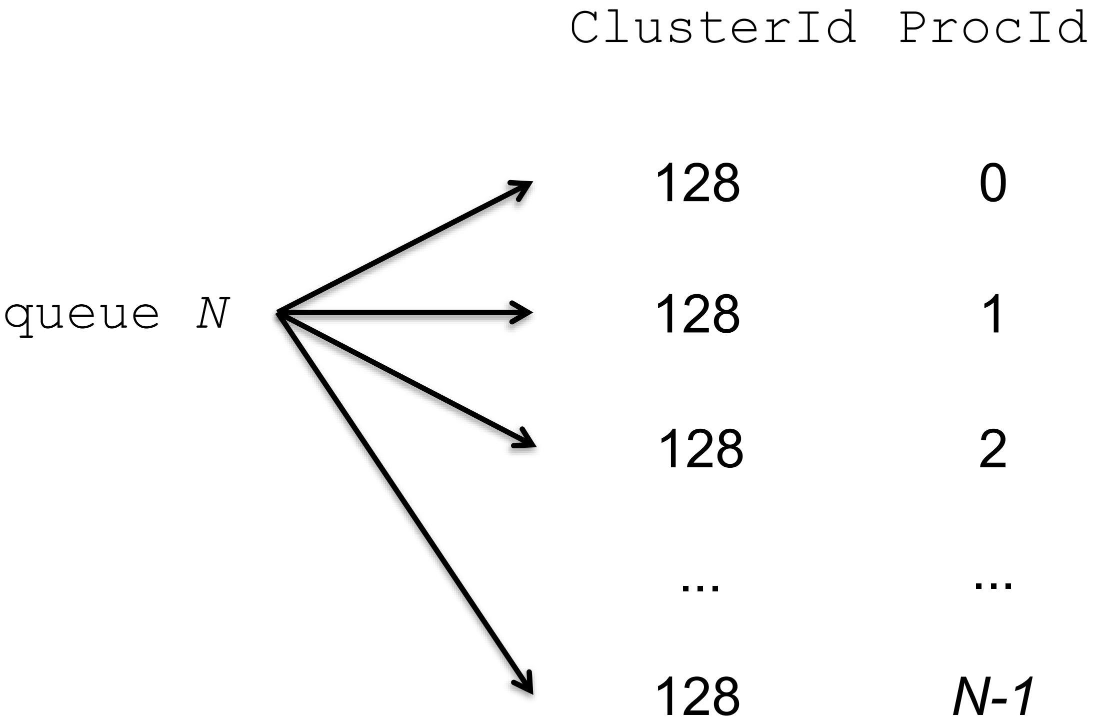
Which job components vary?

```
executable = analyze.sh
arguments  = file.in file.out
transfer_input_files = file.in

log    = job.log
output = job.stdout
error  = job.stderr

queue 5
```

The arguments for our command and the input files would be different for each job.

We might also want to differentiate these job files.

HTCondor Automatic Variables



`queue` *N*

`ClusterId`	`ProcId`
128	0
128	1
128	2
…	…
128	*N-1*

Each job's `ClusterId` and `ProcId` can be accessed inside the submit file using:

`$(ClusterId)`

`$(ProcId)`

* May also see `$(Cluster)`, `$(Process)` in documentation

Use `$(ProcID)` as the variable

```
executable = analyze.sh
arguments  = file.$(ProcID).in file.$(ProcID).out
transfer_input_files = file.$(ProcID).in

log    = job.log
output = job.$(ProcID).stdout
error  = job.$(ProcID).stderr

queue 5
```

The default variable representing the changing numbers in our list is `$(ProcID)`

Submitting Jobs

Jobs in the queue will be grouped in batches
(default: cluster number)

```
$ condor_submit job.submit
Submitting job(s).
5 job(s) submitted to cluster 128.
```

```
$ condor_q
-- Schedd: submit-1.chtc.wisc.edu : <128.104.101.92:9618?... @ 05/09/19 10:35:54
OWNER  BATCH_NAME   SUBMITTED    DONE    RUN    IDLE  TOTAL  JOB_IDS
alice  ID: 128      5/9  11:03      _      _       5      5   128.0-4

5 jobs; 0 completed, 0 removed, 5 idle, 0 running, 0 held, 0 suspended
```

To see individual jobs, use:
condor_q -nobatch

Other options: `queue N`

Can I start from 1 instead of 0?

- Yes! These two lines increment the $(ProcId) variable

```
tempProc = $(ProcId) + 1
newProc = $INT(tempProc)
```

- You would use the second variable name $(newProc) in your submit file

Can I create a certain number of digits (i.e. 000, 001 instead of 0,1)?

- Yes, this syntax will make $(ProcId) have a certain number of digits

```
$INT(ProcId,%03)
```

Other Options for Submitting Multiple Jobs

Variable and `queue` options

Syntax	List of Values	Variable Name
queue *N*	Integers: 0 through N-1	$(ProcId)
queue *Var* matching *pattern**	List of values that match the wildcard pattern.	$(*Var*) If no variable name is provided, default is $(Item)
queue *Var* in (*item1 item2* …)	List of values within parentheses.	
queue *Var* from *list.txt*	List of values from *list.txt*, where each value is on its own line.	

Other options: `queue … matching`

`Queue matching` has options to select only files or directories

```
queue infile matching files *.dat
```

```
queue indirs matching dirs job*
```

If you have questions about which queue statement would work best for *your* workflow, don’t hesitate to reach out to OSG staff this week!

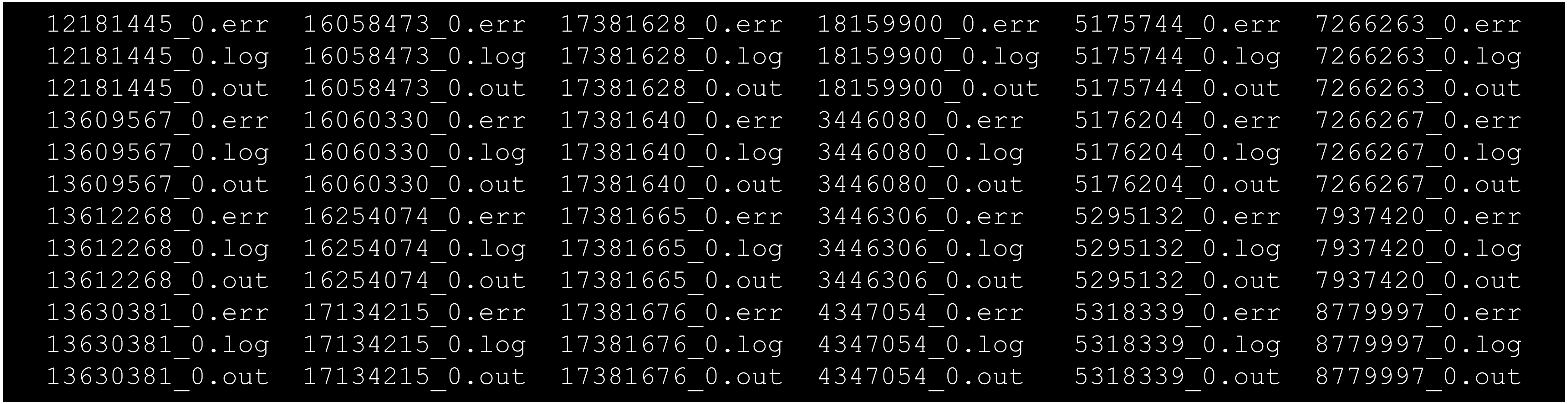
Queue options, pros and cons

`queue` *`N`*	- Simple, good for multiple jobs that only require a numerical index.
`queue matching` *`pattern*`*	- Natural nested looping, minimal programming, use optional “files” and “dirs” keywords to only match files or directories - Requires good naming conventions.
`queue in (`*`list`*`)`	- All information contained in a single file, reproducible - Harder to automate submit file creation
`queue from` *`file`*	- Supports multiple variables, highly modular (easy to use one submit file for many job batches), reproducible - Additional file needed

Additional Thoughts

Organization

(more on this later!)



```
12181445_0.err  16058473_0.err  17381628_0.err  18159900_0.err  5175744_0.err  7266263_0.err
12181445_0.log  16058473_0.log  17381628_0.log  18159900_0.log  5175744_0.log  7266263_0.log
12181445_0.out  16058473_0.out  17381628_0.out  18159900_0.out  5175744_0.out  7266263_0.out
13609567_0.err  16060330_0.err  17381640_0.err  3446080_0.err   5176204_0.err  7266267_0.err
13609567_0.log  16060330_0.log  17381640_0.log  3446080_0.log   5176204_0.log  7266267_0.log
13609567_0.out  16060330_0.out  17381640_0.out  3446080_0.out   5176204_0.out  7266267_0.out
13612268_0.err  16254074_0.err  17381665_0.err  3446306_0.err   5295132_0.err  7937420_0.err
13612268_0.log  16254074_0.log  17381665_0.log  3446306_0.log   5295132_0.log  7937420_0.log
13612268_0.out  16254074_0.out  17381665_0.out  3446306_0.out   5295132_0.out  7937420_0.out
13630381_0.err  17134215_0.err  17381676_0.err  4347054_0.err   5318339_0.err  8779997_0.err
13630381_0.log  17134215_0.log  17381676_0.log  4347054_0.log   5318339_0.log  8779997_0.log
13630381_0.out  17134215_0.out  17381676_0.out  4347054_0.out   5318339_0.out  8779997_0.out
```

Many jobs means many files.

Test and Scale Up Slowly

(more on this later!)

- Before submitting 1,000s of jobs, make sure 10 work!
- Saves you time and hassle, and helps avoid wasted computing cycles

Watching Progress of Jobs

- To get a live update of the progress of your jobs, use **`condor_watch_q`**

This command does an initial **`condor_q`** and then tracks the entries of the corresponding .log file(s)

```
$ condor_watch_q
BATCH        IDLE  RUN  DONE  TOTAL  JOB_IDS
ID: 129        3    4     3     10   129.0 ... 129.9 [######========------]

[######################=============================----------------------]

Total: 10 jobs; 3 completed, 4 idle, 3 running

Updated at 2024-08-01 10:11:52
Input ^C to exit
```

HTCondor Manual: condor_watch_q

Questions?

Additional Slides of Interest

Case Study 1

Mei Monte Carlo

Needs to run many random simulations to model particles in a detector

What varies?

- Not much – just needs an index to keep simulation results separate.

Use `queue N`

- Simple, built-in
- No need for specific input values

Case Study 2

Tamara Trials

Testing different design parameters for designing clinical trials.

What varies?

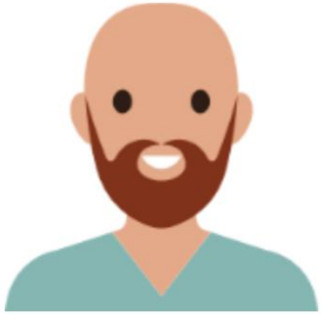
- Five parameter combinations per job
- Parameters are given as arguments to the executable

Use `queue … from`

- queue from can accommodate multiple values per job
- Easy to re-run combinations that fail by using subset of original list

Case Study 3

Ben Bioinformatics



Applying a quality control / processing pipeline to 20 RNA samples.

What varies?

- Each job analyzes one sample; each sample consists of two fastq files in a folder with a standard prefix.

Use `queue … matching`

- Folders have a standard prefix, input files have standard suffix, easy to pattern match

Good alternative: `queue … from`

- Provide list of folder names/file prefixes, construct paths in the submit file.

Tip: Organize with Directories

```
executable = analyze.sh
transfer_input_files = input/file$(ProcID).in,
                       shared/

log    = logs/job.$(ProcID).log
output = output/job.$(ProcID).stdout
error  = error/job.$(ProcID).stderr

queue 5
```

```
submit_dir/
  jobs.submit
  analyze.sh
  shared/
    script1.sh
    reference.dat
  input/
    file0.in
    ...
  logs/
    job.0.log
    ...
  output/
    job.0.stdout
    ...
  error/
    job.0.stderr
    ...
```

Tip: Organize with Directories

```
executable = analyze.sh
transfer_input_files = input/file$(ProcID).in,
                       shared/

log    = logs/job.$(ProcID).log
output = output/job.$(ProcID).stdout
error  = error/job.$(ProcID).stderr

queue 5
```

Transfer an entire directory (**shared**)
or just the contents of a directory (**shared/**)

```
submit_dir/
  jobs.submit
  analyze.sh
  shared/
    script1.sh
    reference.dat
  input/
    file0.in
    ...
  logs/
    job.0.log
    ...
  output/
    job.0.stdout
    ...
  error/
    job.0.stderr
    ...
```

Submit File Options for Organizing Files

Syntax	**Purpose**	**Features**
`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
`Transfer_output_remaps =` `“`**`file1.out`**`=path/to/`**`file1.out;`** **`file2.out`**`=path/to/`**`renamedFile2`**`.out”`	Used to save output files to 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

Job-specific directories with `initialdir`

- Use `initialdir` to set the submission directory.
- All output files will be saved back to this directory.

```
executable = analyze.sh
transfer_input_files = file.in
initialdir = job$(ProcId)

output = job.stdout
error  = job.stderr

queue 5
```

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

```
submit_dir/
  jobs.submit
  analyze.sh
  job0/
    file.in
    job.stdout
    job.stderr
  job1/
    file.in
    job.stdout
    job.stderr
  job2/
    ...
```

Send output to a specific directory

- **Reminder**: by default, HTCondor transfers all files back to the submission directory
- Use `transfer_output_remaps` to save output files to a specific path and using a certain name to avoid a cluttered workspace/ writing over other files

```
submit_dir/
  jobs.submit
  analyze.sh
  input/
    file.in
  output/
    file.out
```

```
executable             = analyze.sh
arguments              = file.in file.out

transfer_input_files   = input/file.in

output = job.out
transfer_output_remaps = “file.out=output/file.out”

queue
```