Beyond Basic DAGMan Workflows

Monday PM, Lecture 2

Lauren Michael

Questions so far?

Open Science Grid

Goals for this Session

- Node-level options in a DAG
- Modular organization of DAG components
- DAG-level control
- Additional DAGMan Features

BEYOND THE BASIC DAG: NODE-LEVEL MODIFIERS

Default File Organization

my.dag

```
JOB A A.sub
JOB B1 B1.sub
JOB B2 B2.sub
JOB B3 B3.sub
JOB C C.sub
PARENT A CHILD B1 B2 B3
PARENT B1 B2 B3 CHILD C
```

(dag_dir)/

```
A.sub          B1.sub
B2.sub         B3.sub
C.sub          my.dag
(other job files)
```

- What if you want to organize files into other directories?

HTCondor Manual: DAGMan Applications > DAG Input File

Node-specific File Organization with *DIR*

- **DIR** sets the submission directory of the node

my.dag

```
JOB A A.sub DIR A
JOB B1 B1.sub DIR B
JOB B2 B2.sub DIR B
JOB B3 B3.sub DIR B
JOB C C.sub DIR C
PARENT A CHILD B1 B2 B3
PARENT B1 B2 B3 CHILD C
```

(dag_dir)/

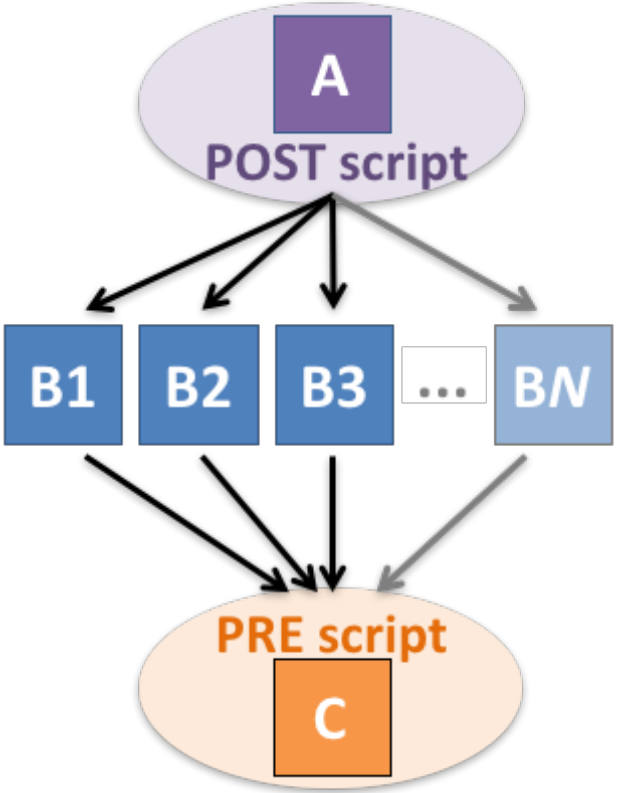
```
my.dag
A/      A.sub    (A job files)
B/      B1.sub   B2.sub
        B3.sub   (B job files)
C/      C.sub    (C job files)
```


HTCondor Manual: DAGMan Applications > DAG Input File

PRE and *POST* scripts run on the submit server, as part of the node

my.dag

```
JOB A A.sub
SCRIPT POST A sort.sh
JOB B1 B1.sub
JOB B2 B2.sub
JOB B3 B3.sub
JOB C C.sub
SCRIPT PRE C tar_it.sh
PARENT A CHILD B1 B2 B3
PARENT B1 B2 B3 CHILD C
```



- **Use sparingly for lightweight work; otherwise include work in node jobs**

HTCondor Manual: DAGMan Applications > DAG Input File

RETRY failed nodes to overcome transient errors

- Retry a node up to *N* times if the exit code is non-zero:

`RETRY node_name N`

Example:

```
JOB A A.sub
RETRY A 5
JOB B B.sub
PARENT A CHILD B
```

- **Note:** Unnecessary for nodes (jobs) that can use `max_retries` in the submit file
- See also: retry except for a particular exit code (`UNLESS-EXIT`), or retry scripts (`DEFER`)

DAGMan Applications > Advanced Features > Retrying
DAGMan Applications > DAG Input File > SCRIPT

RETRY applies to whole node, including *PRE/POST* scripts

- PRE and POST scripts are included in retries
- RETRY of a node with a POST script uses the exit code from the POST script (not from the job)
  - POST script can do more to determine node success, perhaps by examining JOB output

Example:

```
SCRIPT PRE A download.sh
JOB A A.sub
SCRIPT POST A checkA.sh
RETRY A 5
```

DAGMan Applications > Advanced Features > Retrying
DAGMan Applications > DAG Input File > SCRIPT

SCRIPT Arguments and Argument Variables

```
JOB A A.sub
SCRIPT POST A checkA.sh my.out $RETURN
RETRY A 5
```

$JOB: node name

$JOBID: *cluster.proc*

$RETURN: exit code of the node

$PRE_SCRIPT_RETURN: exit code of PRE script

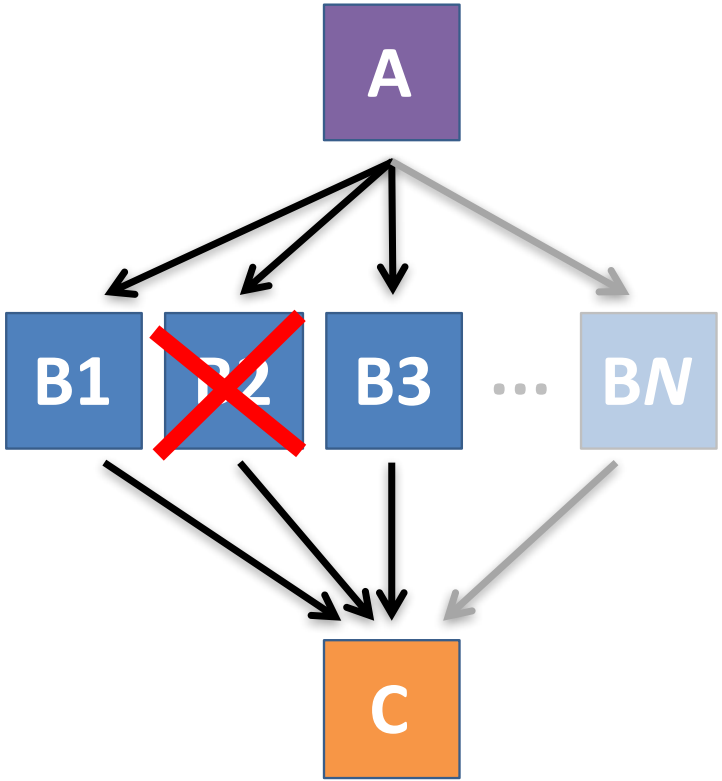
$RETRY: current retry count

(more variables described in the manual)

DAGMan Applications > DAG Input File > SCRIPT
DAGMan Applications > Advanced Features > Retrying

Best Control Achieved with One Process per JOB Node

- While submit files can ‘`queue`’ many processes, a *single process per submit* file is best for DAG JOBs
  - Failure of any process in a JOB node results in failure of the *entire node* and immediate removal of other processes in the node.
  - RETRY of a JOB node retries the entire submit file.



HTCondor Manual: DAGMan Applications > DAG Input File

MODULAR ORGANIZATION OF DAG COMPONENTS

Submit File Templates via *VARS*

- **VARS** line defines node-specific values that are passed into submit file variables

  VARS ***node_name var1="value" [var2="value"]***

- Allows a single submit file shared by all B jobs, rather than one submit file for each JOB.

my.dag

```
JOB B1 B.sub
VARS B1 data="B1" opt="10"
JOB B2 B.sub
VARS B2 data="B2" opt="12"
JOB B3 B.sub
VARS B3 data="B3" opt="14"
```

B.sub

```
…
InitialDir = $(data)
arguments = $(data).csv
$(opt)
…
queue
```

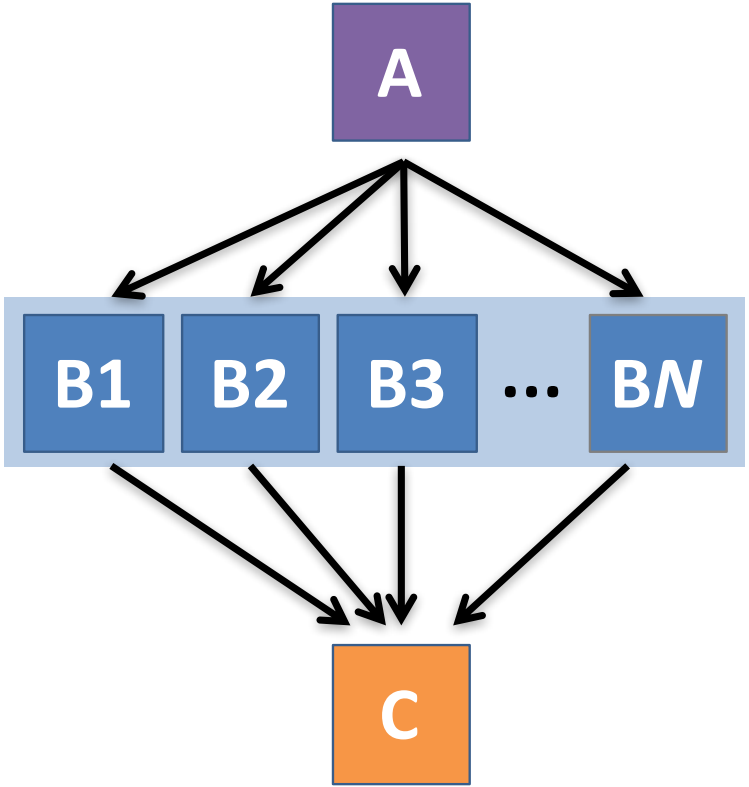
SPLICE groups of nodes to simplify lengthy DAG files

my.dag

```
JOB A A.sub
SPLICE B B.spl
JOB C C.sub
PARENT A CHILD B
PARENT B CHILD C
```

B.spl

```
JOB B1 B1.sub
JOB B2 B2.sub
…
JOB BN BN.sub
```



Use nested *SPLICE*s with DIR for repeating workflow components

my.dag

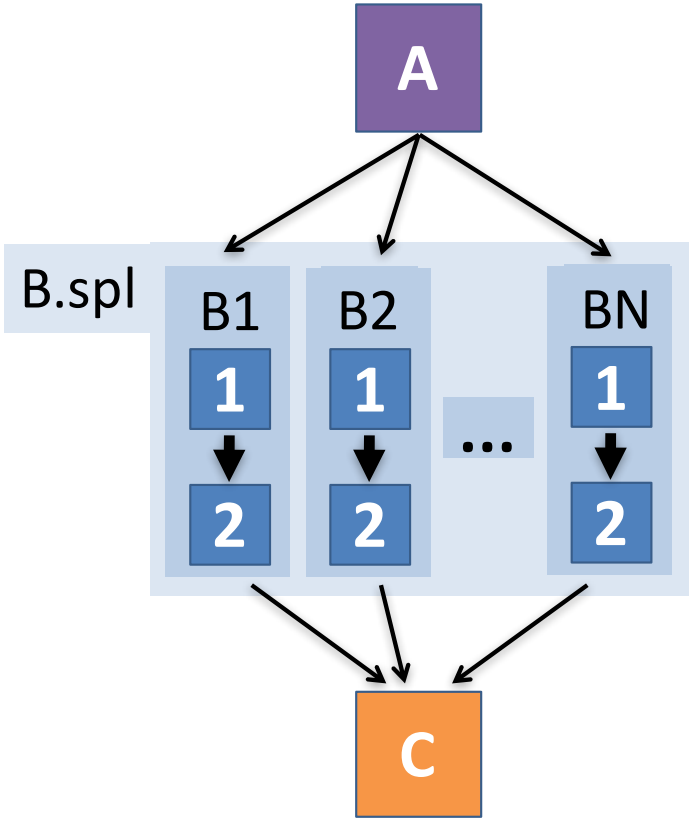
```
JOB A A.sub DIR A
SPLICE B B.spl DIR B
JOB C C.sub DIR C
PARENT A CHILD B
PARENT B CHILD C
```

B.spl

```
SPLICE B1 ../inner.spl DIR B1
SPLICE B2 ../inner.spl DIR B2
…
SPLICE BN ../inner.spl DIR BN
```

inner.spl

```
JOB 1 ../1.sub
JOB 2 ../2.sub
PARENT 1 CHILD 2
```



Use nested *SPLICE*s with DIR for repeating workflow components

my.dag

```
JOB A A.sub DIR A
SPLICE B B.spl DIR B
JOB C C.sub DIR C
PARENT A CHILD B
PARENT B CHILD C
```

B.spl

```
SPLICE B1 ../inner.spl DIR B1
SPLICE B2 ../inner.spl DIR B2
…
SPLICE BN ../inner.spl DIR BN
```

inner.spl

```
JOB 1 ../1.sub
JOB 2 ../2.sub
PARENT 1 CHILD 2
```

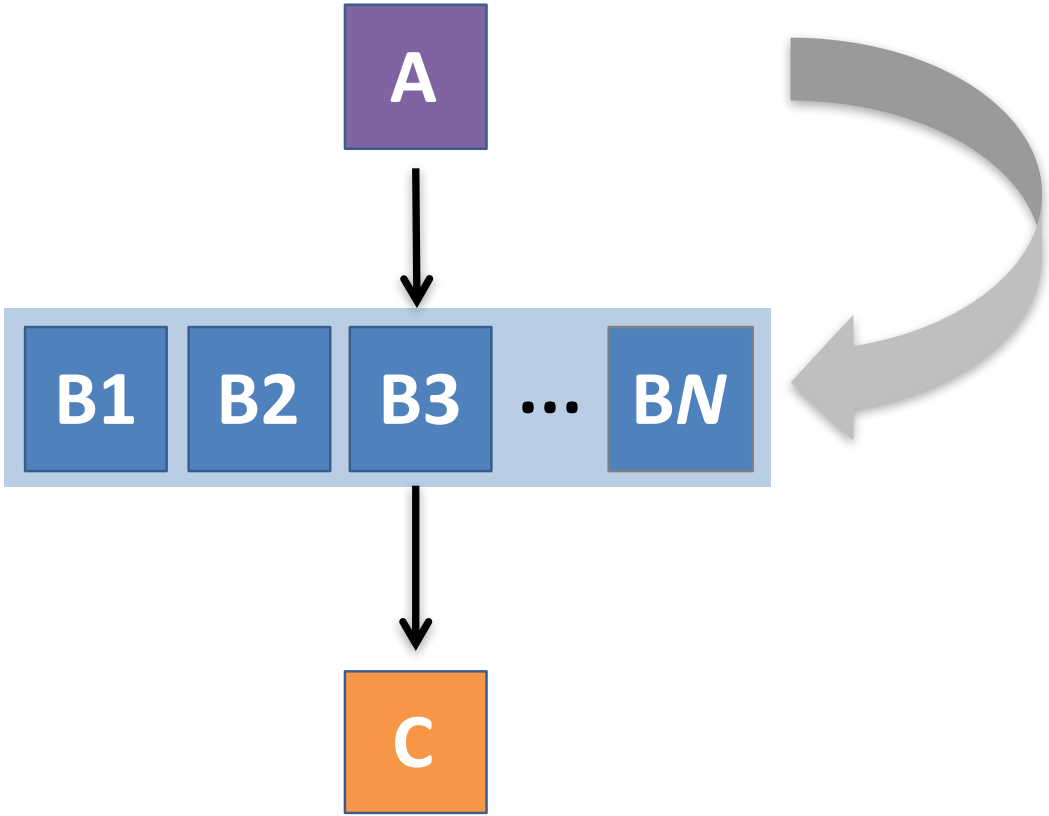
(dag_dir)/

```
my.dag
A/      A.sub (A job files)
B/      B.spl  inner.spl
        1.sub  2.sub
        B1/    (1-2 job files)
        B2/    (1-2 job files)
        …
        BN/    (1-2 job files)
C/      C.sub (C job files)
```

More on *SPLICE* Behavior

- Upon submission of the outer DAG, nodes in the SPLICE(s) are added by DAGMan into the overall DAG structure.
  - A single DAGMan job is queued with single set of status files.
- Great for gradually testing and building up a large DAG (since a SPLICE file can be submitted by itself, as a complete DAG).
- SPLICE lines are not treated like nodes.
  - no PRE/POST scripts or RETRIES (though this may change)

What if some DAG components can't be known at submit time?



If *N* can only be determined as part of the work of **A** …

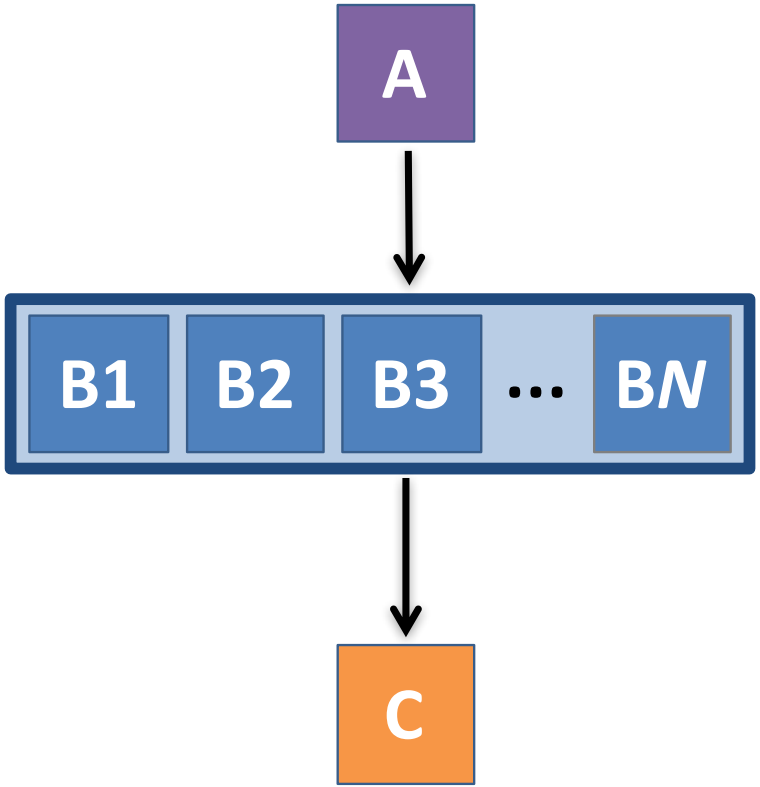
A *SUBDAG* within a DAG

my.dag

```
JOB A A.sub
SUBDAG EXTERNAL B B.dag
JOB C C.sub
PARENT A CHILD B
PARENT B CHILD C
```

B.dag (written by **A**)

```
JOB B1 B1.sub
JOB B2 B2.sub
…
JOB BN BN.sub
```



More on *SUBDAG* Behavior

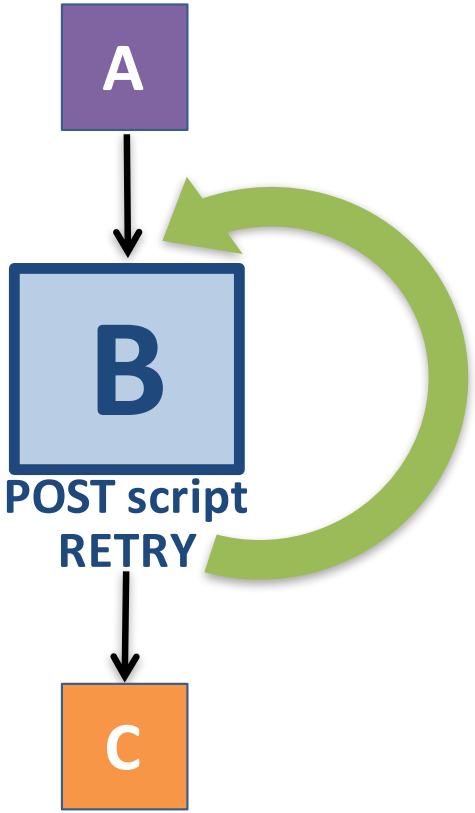
- **WARNING:** SUBDAGs should only be used (over SPLICES) when absolutely necessary!
  - *Each SUBDAG EXTERNAL has it's own DAGMan job running in the queue.*
- SUBDAGs *are nodes* in the outer DAG (can have PRE/POST scripts, retries, etc.)
- A SUBDAG is not submitted until prior nodes in the outer DAG have completed.

Use a *SUBDAG* to achieve a Cyclic Component within a DAG

- POST script determines whether another iteration is necessary; if so, exits non-zero
- RETRY applies to entire SUBDAG, which may include multiple, sequential nodes

my.dag

```
JOB A A.sub
SUBDAG EXTERNAL B B.dag
SCRIPT POST B iterateB.sh
RETRY B 1000
JOB C C.sub
PARENT A CHILD B
PARENT B CHILD C
```



DAG-LEVEL CONTROL

Pause a running DAG with hold/release

- Hold the DAGMan job process:
  `condor_hold` *`dagman_jobID`*

- Pauses the DAG
  - No new node jobs submitted
  - Queued node jobs continue to run (including SUBDAGs), but no PRE/POST scripts
  - DAGMan jobs remains in the queue until released (`condor_release`) or removed

DAGMan > Suspending a Running DAG

Pause a DAG with a halt file

- Create a file named ***`DAG_file.halt`*** in the same directory as the submitted DAG file
- Pauses the DAG
  - No new node jobs submitted
  - Queued node jobs, SUBDAGs, and POST scripts continue to run, but not PRE scripts
- DAGMan resumes after the file is deleted
  - If not deleted, the DAG creates a rescue DAG file and exits after all queued jobs have completed

DAGMan > Suspending a Running DAG
DAGMan > The Rescue DAG

Throttle job nodes of large DAGs via DAG-level configuration

- If a DAG has *many* (thousands or more) jobs, performance of the submit server and queue can be assured by limiting:
  - Number of jobs in the queue
  - Number of jobs idle (waiting to run)
  - Number of PRE or POST scripts running

- Limits can be specified in a DAG-specific **CONFIG** file (recommended) or as arguments to `condor_submit_dag`

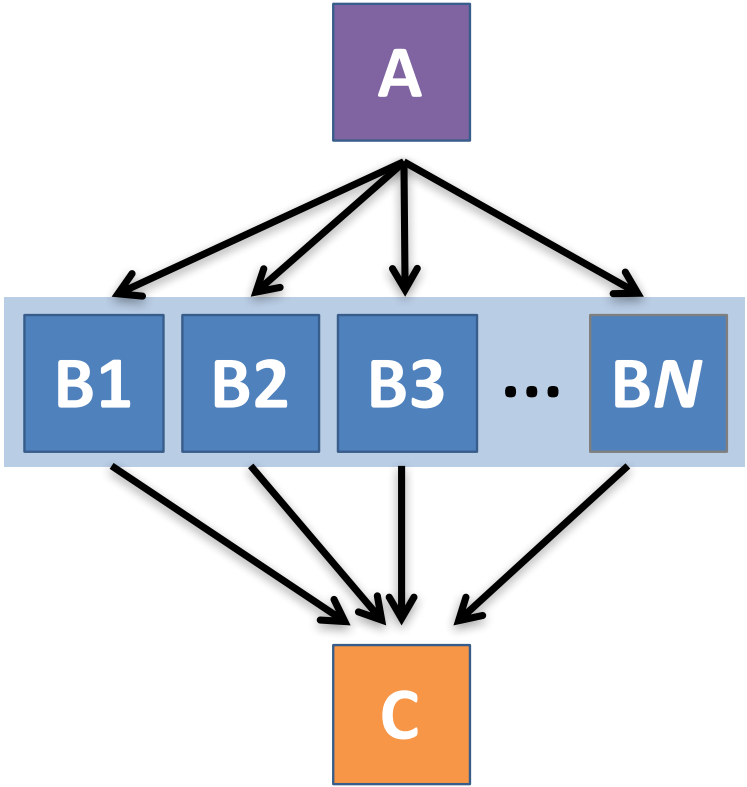
DAG-specific throttling via a CONFIG file

my.dag

```
JOB A A.sub
SPLICE B B.dag
JOB C C.sub
PARENT A CHILD B
PARENT B CHILD C
CONFIG my.dag.config
```

my.dag.config

```
DAGMAN_MAX_JOBS_SUBMITTED = 1000
DAGMAN_MAX_JOBS_IDLE = 100
DAGMAN_MAX_PRE_SCRIPTS = 4
DAGMAN_MAX_POST_SCRIPTS = 4
```



DAGMan > Advanced Features > Configuration Specific to

Other DAGMan Features

Other DAGMan Features: Node-Level Controls

- Set the **PRIORITY** of JOB nodes with:

```
PRIORITY node_name priority_value
```

- Use a **PRE_SKIP** to skip a node and mark it as successful, if the PRE script exits with a specific exit code:

```
PRE_SKIP node_name exit_code
```

DAGMan Applications > Advanced Features > Setting Priorities
DAGMan Applications > The DAG Input File > PRE_SKIP

Other DAGMan Features: Modular Control

- Append **NOOP** to a JOB definition so that its JOB process isn't run by DAGMan
  - Test DAG structure without running jobs (node-level)
  - Simplify combinatorial PARENT-CHILD statements (modular)
- Communicate DAG features separately with **INCLUDE**
  - e.g. separate file for JOB nodes and for VARS definitions, as part of the same DAG
- Define a **CATEGORY** to throttle only a specific subset of jobs

DAGMan Applications > The DAG Input File > JOB
DAGMan Applications > Advanced Features > INCLUDE
DAGMan Applications > Advanced > Throttling by Category

Other DAGMan Features: DAG-Level Controls

- Replace the ***node_name*** with **ALL_NODES** to apply a DAG feature to all nodes of the DAG

- Abort the entire DAG if a specific node exits with a specific exit code:

```
ABORT-DAG-ON node_name exit_code
```

- Define a **FINAL** node that will always run, even in the event of DAG failure (to clean up, perhaps).

```
FINAL node_name submit_file
```

DAGMan Applications > Advanced > ALL_NODES
DAGMan Applications > Advanced > Stopping the Entire DAG
DAGMan Applications > Advanced > FINAL Node

Much More in the HTCondor Manual!!!

https://research.cs.wisc.edu/htcondor/manual/current/2_Users_Manual.html

YOUR TURN!

Exercises!

- Ask questions!
- Lots of instructors around

- Coming up:
  - 4:00–5:00pm Hands-On Exercises
  - 5:00pm On Your Own