HTCondor IAC User's Manual

Ángel de Vicente*

February 26, 2023

Contents

*angel.de.vicente@iac.es

1 Introduction

If you have no experience with HTCondor, we recommend that you contact us before running any job so we can give you a quick introduction (bear in mind that you will be using other users' computers and there are some basic guidelines that you must follow to avoid disturbing them).

The HTCondor infrastructure at the IAC has been recently expanded and improved, with about 100 new Linux desktop PCs financed by the Ministry of Economy and Competitiveness through FEDER funds, code IACA13-3E-2493.

1.1 What is HTCondor?

At the IAC we have several Supercomputing resources that allow you to obtain your computational results in much less time and/or work with much more complex problems. One of them is HTCondor, a High Throughput Computing (HTC) system. The underlying idea is quite simple (and powerful): let's use idle machines to perform computations while their owners are away. So, in a nutshell, HTCondor is an application that is installed in our PCs to make it possible to run a large number of yours and others' computations at a time in different machines when they are not being used, achieving a better utilization of our resources. A more detailed overview of HTCondor is available at the official documentation.

1.2 How can HTCondor help us?

HTCondor is very useful when you have an application that has to run a large number of times over different input data. For instance, suppose you have a program that carry out some calculations taking an image file as input. Let's say that the processing time is about one hour per image and you want to process 250 images. Then you can use your own machine and process all images one by one, and wait more than 10 days to get all results, or you can use HTCondor to process each image in different computers and hopefully get all results in one hour, or maybe two or four, but for sure less than 10 days. And HTCondor will do all the work for you: it will copy the input files to the remote machines, execute your program there with different inputs and bring back the results to your machine when they are complete.

HTCondor has also other powerful features. For instance, you can also use it to create specific or periodic checkpoints of your @@C@@, @@C++@@ or @@fortran@@ programs. Then you can normally run your application (without using HTCondor) and if something happens (your program crashes, PC is unexpectedly halted, etc.), you will be able to restart the execution from any of the checkpoints, saving a lot of time. Please, read this introduction and the other sections to discover how HTCondor can help you.

1.3 How "powerful" is HTCondor?

HTCondor calls """'slot"'"' the unit that executes a job, typically a CPU or core if the CPU has several of them. Right now we have around 900 slots (Feb 2017) that might execute applications submitted via HTCondor. It means that everyday more than 21000 hours could be available to run HTCondor jobs, more than 2 years of computation in just 24 hours! OK, this is the theoretical limit if no one were using their computers and all slots were idle... ;) The number of idle slots is always changing, but based on our experience a more realistic number of idle slots could be around 400 slots in office hours and 650 at nights and weekends, so still we have more than one year of HTCondor computation time per day.

You can see the real-time HTCondor statistics here: nectarino ("Pool Resource Stats" show the number of slots being used by their owners, by HTCondor and the idle ones; while "Pool User Stats" show the number of HTCondor jobs and consumed hours per user). If you want more detailed info about which and when jobs have been executing on specific machines, check stats at `http://carlota:81/condor_stats`. Also you can visit the Hall of Fame of HTCondor.

1.4 Which machines are running HTCondor?

HTCondor is already installed in most of the Desktop PCs running Linux that we have at IAC Headquarters in La Laguna, with a total number of more than 230 machines.

If you are concerned about hardware specifications, you may know that those machines are rather heterogeneous and its availability and specifications change from time to time. At the present status (Sept, 2014), most CPUs are Intel (and also some AMD) from 2.40 to 3.20 GHz. Each CPU has typically 2, 4 or 8 cores, although there are also more powerful machines with up to 32 cores per CPU. As for memory, the most common is 2GB per slot, while some of them have from 3 to 8GB per slot and a few just 1GB per slot.

On the other hand, software specifications are quite homogeneous and all machines are running the same OS: Fedora Linux. Almost all machines run Fedora21 (as of June 2017, there are still a few machines running older versions, and a handful with Fedora 25). Installed software should be also more or less the same in every machine (see the software supported by the SIE), which makes it easy to run almost every application in any machine (although the available software could be different in some machines that belong to the Instrumentation area).

If your application has special requirements about memory per slot, OS version, etc., you can rank and/or limit these parameters and also a quite large set of other ones. "Please, visit FAQs page for more information and examples."

1.5 Who can use HTCondor? How does it work? Do I need to change my application?

If you have a computer account at the IAC and you can log in on a Linux PC Desktop connected to the internal network, then you should be able to use HTCondor with no problems (try @@condor$_{\text{version}}$@@ command to check whether HTCondor is installed. Please contact us if it is not or you experience any issue).

HTCondor is a batch-processing system, so you only need to submit your jobs to the HTCondor queue and it will do all the work. The submission is done using a HTCondor script where you specify your executable, its arguments, inputs and outputs, etc. (visit HTCondor submit files page to see some examples and recommendations). You do not need to prepare or compile your programs in any special way to run them, and almost all programming languages that are commonly used at IAC should be suitable to be run with HTCondor (shell scripts, Python, Perl, C, Fortran, IDL, etc.). Sometimes a few minor modifications may be needed in order to specify arguments and the locations of inputs or outputs, so that HTCondor can find them, but that should be all.

Once the submitted jobs are in HTCondor queue, it uses its allocation algorithm to send and execute your jobs on those idle slots that satisfy your requirements. Idle slots are those located in machines where there has been no keyboard/mouse activity for a long while and the computer load is low enough to ensure that there is no interference with the owner's processes. While HTCondor is running its jobs, it also keeps checking that the owner is not using the machine. If HTCondor detects any activity in the computer (for instance, a key is pressed), then it will suspend all its jobs and wait a little while to see whether the machine gets idle again so as to resume the jobs. If the owner keeps working, HTCondor will interrupt all jobs and send them to other available slots in any other idle machine. HTCondor will repeat this process till all jobs are done, sending notifications via email when they are finished or if any errors show up.

1.6 I am using HTCondor, should I add an acknowledgement text in my publications?

Yes, you should mention it in the acknowledgments of your papers or any other publications where you have used HTCondor. Although there is no standard format, we suggest the following:

»frame« ""This paper made use of the IAC Supercomputing facility HTCondor (`http://research.cs.wisc.edu/htcondor/`), partly financed by the Ministry of Economy and Competitiveness with FEDER funds, code IACA13-3E-2493."" »«

If you have used any other IAC Supercomputing facilities (LaPalma, TeideHPC, etc.), please, add them in the acknowledgments too:

"'LaPalma"': ""The author thankfully acknowledges the technical expertise and assistance provided by the Spanish Supercomputing Network (Red Española de Supercomputación), as well as the computer resources used: the LaPalma Supercomputer, located at the Instituto de Astrofísica de Canarias.""

"'TeideHPC"': ""The author(s) wish to acknowledge the contribution of Teide High-Performance Computing facilities to the results of this research. TeideHPC facilities are provided by the Instituto Tecnológico y de Energías Renovables (ITER, SA). URL: `http://teidehpc.iter.es/`""

1.7 I need more information or have some problems, who can help me...?

If you need further information, please check the other pages about HTCondor at the SIEpedia: Useful Commands, Submit Files (description and examples), Submit Files (HowTo), FAQs, etc. HTCondor at SIEpedia is continuously updated, but we also have more documentation about older versions of HTCondor at the the HTCondor section at IAC (most of that information is still valid, but some may be obsolete, including broken links). For detailed and complete information, check the official documentation about HTCondor.

If you need help or you are having any kind of issues related to HTCondor, "'the SIE gives direct support"' to IAC's users who want to use HTCondor: we will not code your whole application, but we help and advise you about how to get the most out of HTCondor: use its commands, create submit files, modify your application to run it with HTCondor (in case it is needed), fix common mistakes, etc. We also organize workshops about HTCondor for IAC's users (the last one was on February, 25th 2014 - slides, and we can organize a new workshop on demand if you and your colleges need it: if the group is large enough -10 or 12 people-, just contact us!).

2 Useful Commands

HTCondor has several dozens of commands, but in this section we will present just the most common ones (if you want to check the complete list, try the Command Reference page). Also remember that you can get further information running @@man condor_<cmd>@@ in your shell or visiting the official Users' Manual. The main command will be shown together with some useful options that may help work with HTCondor:

2.1 Checking pool status

- `condor_status`: list slots in HTCondor pool and their status: `Owner` (used by owner), `Claimed` (used by HTCondor), `Unclaimed` (available to be used by HTCondor), etc. Useful options:
  - `-avail`: List those slots that are not busy and could run HTCondor jobs at this moment

- `-submitters`: Show information about the current general status, like number of running, idle and held jobs (and submitters)
- `-run`: List slots that are currently running jobs and show related information (owner of each job, machine where it was submitted from, etc.)
- `-compact`: Compact list, with one line per machine instead of per slot
- `-state -total`: List a summary according to the state of each slot
- `-master`: List machines, but just their names (status and slots are not shown)
- `-server`: List attributes of slots, such as memory, disk, load, flops, etc.
- `-sort Memory`: Sort slots by Memory, you can try also with other attributes
- `-af <attr1> <attr2> <...>`: List specific attributes of slots, using autoformat (new version, very powerful)
- `-format <fmt> <attr>`: List attributes using the specified format (old version). For instance, next command will show the name of each slot and the disk space: `condor_status -format "%s\t " Name -format "%d KB\n" Disk`
- `<machine>`: Show the status of a specific machine
- `<machine> -long`: Show the complete "ClassAd" of a machine (its specifications). We can use these specifications to add restrictions in the submit file so we can control which machines we want to use.
- `-constraint <constraint>`: Only Show slots that satisfy the constraint. I.e: `condor_status -constraint 'Memory > 1536'` will only show slots with more than 1.5GB of RAM per slot.

2.2 Submitting jobs

- `condor_submit <submit_file>`: Submit jobs to the HTCondor queue according to the information specified in `submit_file`. Visit the **submit file page** to see some examples of these files. There are also some FAQs related to the submit file. Useful options:
  - =-dry-run $<\text{dest}_{\text{file}}>$ =: this option parses the submit file and saves all the

  related info (name and locations of input and output files after expanding all variables, value of requirements, etc.) to `<dest_file>`, but jobs are "'not"' submitted. Using this option is highly recommended when debugging or before the actual submission if you have made some modifications in your submit file and you are not sure whether they will work.
  - `'var=value'`: add or modify variable(s) at submission time, without changing

  the submit file. For instance, if you are using `queue $(N)` in your submit file, then `condor_submit <submit_file> 'N = 10'` will submit 10 jobs. You can specify several pairs of `var=value`.
  - `-append <command>`: add submit commands at submission time, without changing

  the submit file. You can add more than one command using several times `-append`.

When submitted, each job is identified by a pair of numbers "'X.Y"', like 345.32. The first number (X) is the "'cluster id"': every submission gets a different cluster id, that is shared by all jobs belonging to the same submission. The second number (Y) is the "'process id"': if you submitted N jobs, then this id will go from 0 for the first job to N-1 for the last one. For instance, if you submit a file specifying 4 jobs and HTCondor assign id 523 to that cluster, then the ids of your jobs will be 523.0, 523.1, 523.2 and 523.3 (you can get these ids and more info using `condor_q` command).

Caution!: Before submitting your jobs, always do some simple tests in order to make sure that both your submit file and program work in a proper way: if you are going to submit hundreds of jobs and each job takes several hours to finish, before doing that try with just a few jobs and change the input data in order to let them finish in minutes. Then check the results to see if everything went fine before submitting the real jobs. Bear in mind that submitting untested files and/or jobs may cause a waste of time and resources if they fail, and also your priority will be lower in following submissions.

2.3 Checking and managing submitted jobs

Note: Each machine manages its own HTCondor queue, so it has information only about those jobs that were submitted on it (and no information about any other jobs you may have submitted on other machines). Most of the commands explained in this section get information asking only the local queue, which means that you will only see those jobs that you have submitted on that specific machine. If you submit jobs from different machines, and later you want to check, hold, release, remove, etc. those jobs, you may need to connect to each one of those machines where you have submitted jobs from, or, when possible, use the commands with extra options to communicate with other machines.

- `condor_q`: Show my jobs that have been submitted in this machine. By default

you will see the ID of the job(`clusterID.processID`), the owner, submitting time, run time, status, priority, size and command. [*STATUS*: **I**:idle (waiting for a machine to execute on); **R**: running; **H**: on hold (there was an error, waiting for user's action); **S**: suspended; **C**: completed; **X**: removed; <: transferring input; and >: transferring output]. Useful options:

- `-global`: Show my jobs submitted in any machine, not only the current one
- `-nobatch`: Starting in version HTCondor 8.6.0 installed in January 2017, data is displayed in a compact mode (one line per cluster). With this option output will be displayed in the old format (one line per process)
- `-wide`: Do not truncate long lines. You can also use `-wide:<n>` to truncate lines to fit `n` columns
- `-analyze <job_id>`: Analyse a specific job and show the reason why it is in its current state (useful for those jobs in Idle status: Condor will show us how many slots match our restrictions and may give us suggestion)
- `-better-analyze <job_id>`: Analyse a specific job and show the reason why it is in its current state, giving extended info
- `-long <job_id>`: Show all information related to that job
- `-run`: Show your running jobs and related info, like how much time they have been running, in which machine, etc.
- `-currentrun`: Show the consumed time on the current run, the cumulative time from last executions will not be used (you can combine also with `-run` flag to see only the running processes at the moment)
- `-hold`: Show only jobs in the "on hold" state and the reason for that. Held jobs are those that got an error so they could not finish. An action from the user is expected to solve the problem, and then he should use the `condor_release` command in order to check the job again
- `-af <attr1> <attr2> <...>`: List specific attributes of jobs, using autoformat
- [-{-''-`global -submitter <user>`-}: Show all jobs from user `<user>` in all machines. **Note**: starting in HTCondor version 8.6.0 installed at IAC in January 2017, HTCondor will NOT show other users' jobs" by default, but you can use some flags like `-allusers` to change this behaviour -]
- `*condor_tail* <job_id>`: Display on screen the last lines of the `stdout` (screen) of a running job on a remote machine. You can use this command to check whether your job is working fine, you can also visualize errors (`stderr`) or output files created by your program (see also CondorFAQs#ssh). Useful options:
  - `-f`: Do not stop displaying the content, it will be displayed until interrupted with `Ctrl+C`
  - `-no-stdout -stderr`: Show the content of `stderr` instead of `stdout`
  - `-no-stdout <output_file>`: Show the content of an output file (`output_file` has to be listed in the `transfer_output_files` command in the submit file).
- `*condor_release* <job_id>`: Release a specific held job in the queue. Useful options:
  - `<cluster_id>`: Instead of giving a `<job_id>`, you can specify just the `<cluster_id>` in order to release all held jobs of a specific submission
  - `-constraint <constraint>`: Release all my held jobs that satisfy the constraint
  - `-all`: Release all my held jobs
  - **Note:** Jobs with "on hold" state are those that HTCondor was not able

  to properly execute, usually due to problems with executable, paths, etc. If you can solve the problems changing the input files and/or the executable, then you can use `condor_release` command to run again your program since it will send again all files to the remote machines. If you need to change the submit file to solve the problems, then `condor_release` will NOT work because it will not evaluate again the submit file. In that case you can use `condor_qedit` (see the HOWTOs.CondorFAQs#ch_{submit}) or cancel all held jobs and re-submit them again-]
- `*condor_hold* <job_id>`: Put jobs into the hold state. It could be useful when you detect that there are some problems with your input data (see CondorFAQs#bad_{inputs} for more info), you are running out of disk space for outputs, etc. With this command you can delay the execution of your jobs holding them, and, after solving the problems, assign them the idle status using `condor_release`, so they will be executed again. Useful options:
  - `<cluster_id>`: Instead of giving a `<job_id>`, you can specify just the `<cluster_id>` in order to hold all jobs of a specific submission
  - `-constraint <constraint>`: Hold all jobs that satisfy the constraint
  - `-all`: Hold all my jobs from the queue
- `*condor_rm* <job_id>`: Remove a specific job from the queue (it will be removed even if it is running). Jobs are only removed from the current machine, so if you submitted jobs from different machines, you need to remove your jobs from each of them. Useful options:
  - `<cluster_id>`: Instead of giving a `<job_id>`, you can specify just the `<cluster_id>` in order to remove all jobs of a specific submission
  - `-constraint <constraint>`: Remove all jobs that satisfy the constraint

  - `-all`: Remove all my jobs from the queue
  - `-forcex <job_id>`: It could happen that after removing jobs, they don't disappear from the queue as expected, but they just change status to **X**. That's normal since HTCondor may need to do some extra operations. If jobs stay with 'X' status a very long time, you can force their elimination adding `-forcex` option. For instance: `condor_rm -forcex -all`.

- `*condor_prio*`: Set the priority of my jobs. A user can only change the priority of her own jobs, to specify which ones she would like to run first (the higher the number, the bigger the priority). Priority could be absolute or relative, use `man condor_prio` for further information
- `*condor_ssh_to_job <job_id>*`: Create an ssh session to a running job in a remote machine. You can use this command to check whether the execution is going fine, download/upload inputs or outputs, etc. More information about this command is available in CondorFAQs#ssh.

2.4 Getting info from logs

- `*condor_userlog* <file.log>`: Show and summarize job statistics from the job log files (those created when using `log` command in the submit file)
- `*condor_history*`: Show all completed jobs to date (it has to be run in the {~~same machine~~} where the submission was done). Useful options:
  - `-userlog <file.log>`: list basic information registered in the log files (use `condor_logview <file.log>` to see information in graphic mode)
  - `-long XXX.YYY -af LastRemoteHost`: show machine where job XXX.YYY was executed
  - `-constraint <constraint>`: Only show jobs that satisfy the constraint. I.e: `condor_history -constraint 'RemoveReason`!=UNDEFINED'=: show your jobs that were removed before completion
- `condor_logview <file.log>`: This is not an original HTCondor command, we have created this link to the script that allows you to display graphical information contained in the log of your executions.
- There is also an online tool to analyze your log files and get more information: `HTCondor Log Analyzer` (`http://condorlog.cse.nd.edu/`).

2.5 Other commands

- `condor_userprio`: Show active HTCondor users' priority. Lower values means higher priority where 0.5 is the highest. Use `condor_userprio -allusers` to see all users' priority, you can also add flags `-priority` and/or `-usage` to get detailed information
- `condor_qedit`: use this command to modify the attributes of a job placed on the queue. This may be useful when you need to change some of the parameters specified in the submit file without re-submitting jobs (see HOWTOs.CondorFAQs#ch_{submit}).
- `condor_compile`: Relink a program with HTCondor libraries so it can be used in the `standard` universe where checkpoints are enable (check CondorFAQs#checkpoints for more info). Relinked programs can be also executed as an standalone checkpointing executable, what means that you can run it directly in your shell (no HTCondor submission is needed) and create specific or periodic checkpoints that allow you to recover the execution in case of problems. See CondorFAQs#run_{ckpt} for more information and examples.
- `condor_submit_dag <dag_file>`: Submit a DAG file, used to describe jobs with dependencies. Visit the CondorHowTo#$howto_{dag}$ section for more info and examples.
- `condor_version`: Print the version of HTCondor.
- If you want some general information about HTCondor queue, the pool of machines, where jobs have been executed on, etc., you can try our online stats about HTCondor: `http://carlota:81/condor_stats/` and `http://nectarino/`.

3 Submit files (desc. & examples)

3.1 Introduction

To execute your application with HTCondor, you have to specify some parameters like the name of your executable, its arguments, inputs and outputs, requirements, etc. This information is written in a plain text using **submit commands** in a file called "HTCondor Submit Description File" or simply **submit file**. Once that file is filled with all needed info, you have to submit it to HTCondor using `*condor_submit*` in your terminal, and then it will be processed and your jobs will be added to the queue in order to be executed.

Submit files have considerably changed after the release of versions 8.4.X (first version 8.4.0 released in Sept 2015, since Feb 2017 we are using versions 8.6.X). Some operations were not possible or highly painful in previous versions (like dealing with an undetermined number of files with arbitrary names, declaring variables and macros and performing operations with them, including submission commands from other files, adding conditional statements, etc.). To solve that, many researchers developed external scripts (perl, python, bash, etc.) to dynamically create description

files and submit them, what in most cases resulted in complex submissions and less efficient executions, not to mention that usually it was needed a hard work to adapt those scripts when the application, arguments and/or IO files changed.

With the addition of new, powerful and flexible commands most of those problems have been solved, so there should be no need of using external scripts and *we highly recommend you always use a HTCondor submit description file instead of developing scripts in other languages*. If you did that in the past, please, consider migrating your old scripts, we will give you support if you find any problems.

In this section you will find templates and examples of HTCondor Submit Description Files. Use them as reference to create your own submit files and contact us if you have any doubt or issue. Topics:

- Creating a submit file (description and structure of submit files: comments, variables, commands, etc.)
- Templates and examples of submit files
- OLD examples
- Some more useful commands and info

Caution!: Before submitting your real jobs, perform always some simple tests in order to make sure that both your submit file and program will work in a proper way: if you are going to submit hundreds of jobs and each job takes several hours to finish, before doing that try with just a few jobs and change the input data in order to let them finish in minutes. Then check the results to see if everything went fine before submitting the real jobs. Also we recommend you use `condor_submit *-dry-run*` to debug your jobs and make sure they will work as expected, see **useful commands** page). Bear in mind that submitting untested files and/or jobs may cause a waste of time and resources if they fail, and also your priority will be lower in following submissions.

3.2 Creating a Submit File

As many other languages, HTCondor submit files allow the use of comments, variable, macros, commands, etc. Here we will describe the most common ones, you can check the official documentation for a complete and detailed information about submit files and submitting process.

3.2.1 Comments

HTCondor uses symbol `*#*` for comments. Everything found after that symbol will be ignored. Please, do not mix commands and comments in the same line, since it may produce errors. We recommend you always write commands and comments in different lines.

```
# This is a valid comment
A = 4  # This may produce errors when expanding =A=, do not use comments and
       #  anything else in the same line!
```

3.2.2 Variables and macros

There are many predefined variables and macros in HTCondor that you can use, and you can define your own ones.

- To **define a variable**, just chose a valid name (names are case-insensitive) and assign a value to it, like `N = 4`, `Name = "example"`
- To **get the value** of a variable, use next syntax: `$(varName)`, both `$` symbol and parentheses `()` are mandatory.
- You can do **basic operations** with variables, like `B = $(A) + 1`, etc. (since version 8.4.0 is not needed to use the old and complex syntax [@$$[(...)]@] for the operations). To get the expression evaluated, you may need to use function macros like `$INT(B)`, `$REAL(B)`, etc.
- There are several special **automatic variables** defined by HTCondor that will help you when creating your submit file. The most useful one is `*$(Process)*` or `$(ProcId)`, that will contain the Process ID of each job (if you submit `N` jobs, the value of `$(Process)` will be `0` for the first job and `N-1` in the last job). This variable is like an **iteration counter** and you can use it to specify different inputs, outputs, arguments, ... for each job. There are some **automatic variables**, like `$(Cluster)` or `$(ClusterId)` that stores the ID of each submission, `$(Item)`, `$(ItemIndex)`, `$(Step)`, `$(Row)`, etc. (see **Example1** for further information).
- There are several **pre-defined Function Macros**. Their syntax is `*$FunctName(varName)*` and they can perform some operations on variable `varName` like evaluating expressions and type conversions, selecting a value from a list according an index, getting random numbers, string operations, filenames processing, setting environment variables, etc. Before creating your own macros, check if HTCondor has already a **pre-defined Function Macro** with the same purpose.

3.2.3 Submit commands

You will need to add several HTCondor submit commands in your script file in order to specify which executable you want to run and where it is located, its arguments if any, input files, which result files will be generated, etc. There is a wide set of HTCondor with almost 200 different **submit description file commands** to cover many different scenarios. But in most situations you will only need to specify a few of them (usually about 10-15). Here we will present the most common ones (commands are case-insensitive):

- **Mandatory commands:**
  - `*executable*`: specify where your executable is located (you can use an absolute path, a relative one to the directory where you do the submission or to another directory specified with `initialdir`). You should specify **only the executable** and not other things like arguments, etc., there are specific commands for that. HTCondor will automatically copy the executable file from your machine to any machine where your job will be executed, so you do not need to worry about that.
  - `*queue*`: this command will send your job(s) to the queue, so it should be the last command in your submit file. In previous versions of HTCondor it was quite limited, only allowing the number of jobs as argument. But since version 8.4.0, this command is very powerful and flexible, and you can use it to specify variables, iterations over other commands, files to be processed, list of arguments, etc. **see complete syntax and examples**.
- **Highly recommended commands:**
  - `*output*`: it will copy the standard output printed on the screen (`stdout`) of the remote machines when executing your program to the local file you specify here. Since all the jobs will use the same name, the filename should include some variable parts that change depending on the job to avoid overwritten the same file, like `$(Process)` (and also `$(Cluster)` if you do not want that different submissions ruin your output files). Even if your program does not print any useful results on screen, it is very recommended you save the screen output to check if there were errors, debug them if any, etc.
  - `*error*`: the same as previous command, but for standard error output (`stderr`).
  - `*log*`: it will save a log of your submission that later can be analysed with HTCondor tools. This is very useful when there is any problem with your job(s) to find the problem and fix it. The log should be the same for all jobs submitted in the same cluster, so you should **not** use `$(Process)` in the filename (but including `$(Cluster)` is recommended).
  - `universe`: there are several **runtime environments** in HTCondor called "universes", we will mostly use the one named `vanilla` since it is the easiest one. This is the universe by default, so if you miss this command, your jobs will also go to `vanilla` universe.
- **Useful commands when working with inputs and outputs (arguments, files, keyboard, etc.)**:
  - `*arguments*`: it is used to specify options and flags for your executable file, like when using it in command line.
  - `*should_transfer_files*`: assign `YES` to it in order to activate HTCondor file transfer system (needed when working with files).
  - `*when_to_transfer_output*`: it will usually have a value of `ON_EXIT` to only copy output files when your job is finished, avoiding the copy of temporary or incomplete files if your job fails or it is moved to another machine.
  - `*transfer_input_files*`: it is used to specify where the needed input files are located. We can use a comma-separated list of files (with absolute or relative paths, as mentioned in `executable` command). Local path will be ignored, and HTCondor will copy all files to the root directory of a virtual location on the remote machine (your executable will be also copy to the same place, so input files will be in the same directory). If you specify a directory in this command, you can choose if you want to copy only the content of the directory (add a slash `"*/*"` at the end, for instance `myInputDir*/*`) or the directory itself and its content (do not add a slash).
  - `*transfer_output_files*`: a comma-separated list of result files to be copied back to our machine. If this command is omitted, HTCondor will automatically copy all files that have been created or modified on the remote machine. Sometimes omitting this command is useful, but other times our program creates many temporary or useless files and we only want to get the ones we specify with this command.
  - More commands for input/output files:
    - `transfer_output_remaps`: it changes the name of the output files when copying them to your machine. That is useful when your executable generates result file(s) with the same name, so changing the filename to include a variable part (like `$(Process)` and maybe also `$(Cluster)`) will avoid overwritten them.
    - `initialdir`: this command is used to specify the base directory for input and output files, instead of the directory where the submission was performed from. If this command include a variable part (like `$(Process)`), you can use this command to specify a different base directory for each job.
    - `input`: if your program needs some data from keyboard, you can specify a file or a comma-separated list of files containing it (each end of line in the file will have the same behaviour as pressing `Intro` key in the keyboard, like when using `stdin` redirection in command line with `*<*`). As other similar commands, you can use absolute or relative paths.
    - `transfer_executable`: by default its value is `True`, but if it is set to `False`, HTCondor will not copy the executable file to the remote machine(s). This is useful when the executable is a system command or a program that is installed in all machines, so it is not needed to copy it.
- **Other useful commands:**

- `request_memory`, `request_disk`: if your program needs a certain amount of total RAM memory or free disk space, you can use these commands to force that your jobs will be only executed on machines with at least the requested memory/free disk space **HowTo**
- `requirements`: this is a very useful command if your program has any special needs. With it you can specify that your job can be only executed on some machines (or some machines cannot run your program) according to a wide set of parameters (machine name, operative system and version and a large etc.) **HowTo**
- `rank`: you can specify some values or combination of them (total memory, free disk space, MIPS, etc.) and HTCondor will choose the best machines for your jobs according to your specifications, where the higher the value, the better (this command is used to specify preferences, not requirements) **HowTo**
- `getenv`: if it is set to `True`, all your environment variables will be copied at submission time and they will be available when your program is executed on remote machines (if you do not use this command or it is set to `False`, then your jobs will have no environment variables). This is useful when running some programs that need a special environment, like python, etc. **HowTo**
- `nice_user`: if it is set to `True`, your jobs will be executed with a fake user with very low priority, what could be very useful when the queue is (almost) empty, so you can run your jobs without wasting your real user priority (you can activate and deactivate this feature when your jobs are being executed, so you can begin running your jobs as nice user if the queue is empty and change to normal user when the queue has many other jobs, or vice versa) **HowTo**
- `concurrency_limits`: you can limit the maximum number of your jobs that could be executed at the same time. You should use this command if your program needs licences and there are a few of them (like `IDL`, see also **CondorAndidlvirtualmachine**) or if for any reason you cannot use the HTCondor file transfer system and all your jobs access to the same shared resource (`/scratch`, `/net/nas`, etc.), in order to avoid that too many concurrent access can stress the network **CondorHowTo#howto$_{\text{limit}}$**
- `include`: since HTCondor v8.4.0, it is possible to **include externally defined submit commands** using syntax: `*include :* ''<myfile>''`. You can even include the output of external scripts that will be executed at submission time, adding a pipe symbol after the file: `*include :* ''<myscript.sh>'' *|*`
- More useful commands:
  - `environment`: this command will allow you to set/unset/change any environment variable(s) **CondorHowTo#howto$_{\text{e}}$**
  - `priority`: if some of your jobs/clusters are more important than others and you want to execute them first, you can use `priority` command to assign them a priority (the higher the value, the higher priority). This command only have an effect on your own jobs, and it is not related to users priority **CondorHowTo#howto$_{\text{priority}}$**.
  - `job_machine_attrs`, `job_machine_attrs_history_length`: use these commands to reduce the effects of "black holes" in HTCondor, what causes that many of your jobs could fail in a short time **CondorHowTo#howto$_{\text{failing}}$**
  - `noop_job`: you specify a condition and those jobs that evaluate it to true will not be executed. This is useful when some of your jobs failed and you want to repeat only the failing jobs, not all of them **CondorHowTo#howto$_{\text{failing}}$**
  - `+PreCmd`, `+PreArguments`, `+PostCmd`, `+PostArguments`: These commands allow you to run some scripts before and/or after your executable. That is useful to prepare, convert, decompress, etc. your inputs and outputs if needed, or debug your executions **CondorHowTo#howto$_{\text{prepostcmd}}$**
  - `notify_user`, `notification`: use these commands if you want to receive a notification (an email) when your jobs begin, fail and/or finish **CondorHowTo#howto$_{\text{notify}}$**
  - `if ... elif ... else ... endif`: since HTCondor version 8.4.0, a **limited conditional semantic** is available. You can use it to specify different commands or options depending on the defined/undefined variables, HTCondor version, etc.
  - `on_exit_hold`, `on_exit_remove`, `periodic_hold`, `periodic_remove`, `periodic_release`, etc.: you can modify the default behaviour of your jobs and the associated status. These commands can be used in a wide set of circumstances. For instance, you can force that jobs that are running for more than X minutes or hours will be deleted or get a "on hold" status (with this you can prevent that failing jobs will be running forever, since they will be stopped or deleted if they run for a much longer while than expected) or the opposite, hold those jobs that finish in an abnormal short time to check later what happened. Or you can also periodically release your held jobs, to run them on other machines if for any reason your jobs work fine on some machines, but fail on others **CondorHowTo#howto$_{\text{failing}}$**
  - `deferrall_time`, `deferral_window`, `deferral_prep_time`: you can force your jobs begin at a given date and time. That is useful when the input data is not ready when submitting and your jobs have to wait till a certain time **CondorHowTo#howto$_{\text{runintime}}$**

3.3 Templates and examples

Here you can find basic templates of submit files, you can use them as starting point and then do the customizations needed for your executions. Check the examples in following sections for details and explanations.

3.3.1 Common Template

```
#####################################################
# HTCondor Submit Description File. COMMON TEMPLATE
# Next commands should be added to all your submit files
#####################################################
if !defined FNAME
  FNAME = condor_exec
endif
ID      = $(Cluster).$(Process)

output  = $(FNAME).$(ID).out
error   = $(FNAME).$(ID).err
log     = $(FNAME).$(Cluster).log

universe                = vanilla
should_transfer_files   = YES
when_to_transfer_output = ON_EXIT
```

- **Explanation:**
  Let's analyse the common template:
  - First block:
    - Here we will define some variables that will be used later. The first of them is `FNAME` and first we ask with the `if defined` condition whether that variable is not already defined (if so, we will use the previous value). This variable will contain the base name for the files where HTCondor will save the information displayed on the screen (`stdout` and `stderr`) and the log file. It is interesting to give a common name to those files generated by HTCondor so later we can identify and manage them together. Since all jobs will use the name specified there, we have to include a variable part that has to be different in each job, in order to avoid overwriting the files. We recommend you use a combination of `$(Process)` (it contains the process ID that is different for each job) and `$(Cluster)` (it contains the cluster ID that is different for each submission), as we have done when defining `$(ID)`. In this way, different jobs and different submission will use different filenames and none of them will be overwritten.
  - Second block:
    - With `output` command we force HTCondor to write in the specified file all the screen output (`stdout`) generated by each job. We have used the variables `$(FNAME)` and `$(ID)` defined above.
    - With `error` command we manage `stderr` in the same way we did with `output`.
    - Then we have also specified a HTCondor log file with `log` command. You should not use `$(Process)` in the filename of the log since all jobs should share the same log.
  - Third block:
    - `universe`: there are **runtime environments** in HTCondor called "universes", we will mostly use the one named `vanilla` since it is the easiest one. This is the universe by default, so if you miss this command, your jobs will go also to `vanilla` universe.
    - `should_transfer_files=YES` and `when_to_transfer_output=ON_EXIT` commands are used to specify that input files have to be copied to the remote machines and output files must be copied back to your machine only when our program is finished. Although these commands are only needed when working with files, we recommend you always use them unless you are totally sure you can omit them.

3.3.2 Examples when working with input/output files and arguments

Most times you will want to run applications that deal with input and/or output files. Commonly, the input files will be located on your local machine, but since your application will be executed on other machine(s), it will be needed to copy your input files there, and then copy the result files back to your computer once your program is done. HTCondor have some commands to automatically do both operations in an easy way, so you do not need to worry about the file transfers: you just need to specify where your files are and HTCondor will copy them.

Note: All these examples will begin defining a specific variable `FNAME` that contains the base name of the files that HTCondor will generate to save the `stdout`, `stderr` and log. Next, the common template explained above with be included using command `include` (we assume that the common template filename is `condor_common.tmpl`).

3.3.2.1 Example A (arbitrary filenames)

Process all input files with extension `.in` in a given directory with next program: `./myprogram -i inputFile -o outputFile`

```
# Including Common Template
FNAME = exampleA
include : /path/to/condor_common.tmpl
```

```
transfer_input_files    = $(mydata)
transfer_output_files   = $Fn(mydata).out

executable    = myprogram
arguments     = "-i $Fnx(mydata) -o $Fn(mydata).out"

queue *mydata* matching files /path/to/inputs/*.in
```

- **Explanation:** We use `transfer_input_files` to specify where the needed input files are located. We can use a comma-separated list of files, but since we do not know the name of the files, we will use the variable `mydata` to specify them. That variable is defined in the last line, with the `queue` command: there, we choose to process all files in `/path/to/inputs` with extension `.in`. When submitting, HTCondor will check that directory and it will automatically create a job for each `.in` file found there, assigning the complete filename to `mydata` (in this way, each job will work with a different file). We have used the `matching files` to specify that we only want files matching the condition, but we can also select only directories (`matching dirs`) or both of them (just `matching`).

  With `transfer_output_files` we set the name of the output files, that is the same as the input file with `.out` extension. To remove the old extension we use the `$Fn` macro, that is one of the **new Function Macros** available since version 8.4.0, used to operate the filename and extract the path, name without extension, extension, etc.

  Then we use `executable` to specify the name of the executable (it can be a system command, your own application, a script, etc). We can use a absolute path or a relative one to the directory where we will perform the submission. This executable will be copied to all remote machines automatically. Finally, `arguments` is used to specify the options for the program. We have to employ again `Fpdnxq` macros, first `Fnx` to remove the original path (file we be copied to the root of a virtual location where HTCondor will run the executable on the remote machine) and then `Fn` to remove path and change extension of the output file.

3.3.2.2 Example B (based on ProcessID, old system before HTCondor v8.4.0)

Process 50 input files with consecutive names (from data0.in to data49.out) using the same program as previous example

```
# Including Common Template
FNAME = example2
include : /path/to/condor_common.tmpl

transfer_input_files    = /path/to/inputs/data$(Process).in
transfer_output_files   = data$(Process).out

N             = 50
executable    = myprogram
arguments     = "-i data$(Process).in -o data$(Process).out"

queue $(N)
```

- **Explanation:** `transfer_input_files` command allows a comma-separated list of files or directories that will be copied to the remote machine. Local path will be ignored, and HTCondor will copy all files to the root directory of a virtual location on the remote machine (your executable will be also copy to the same place, so input files will be in the same directory). If you specify a directory in this command, you can choose if you want to copy only the content of the directory (add a slash `"*/*"` at the end, for instance `myInputDir*/*`) or the directory itself and its content (do not add a slash). In this case, each job will process a different input file, and since they have a consecutive name beginning from `0`, we will use HTCondor macro `$(Process)` to build the proper name, since the process ID will be `0` from the first job to `N-1` for the last job.

  With `transfer_output_files` we specify a comma-separated list of result files to be copied back to our machine. In this case, we specify just one file, with the same name as the input file, but with `.out` extension.

  Then we define the variable `N` to specify the number of jobs to be executed. Our program is set using `executable` command and with `arguments` command we specify all the needed options (here the name of the input and output file with the corresponding flags).

  At the end, we send all jobs to the queue with `queue` command, specifying how many jobs we want (we have used the variable `N`).

3.3.2.3 Example C (lists of files and arguments written in submit file)

Process all arbitrary files and arguments of a given list. Executable is `myprogram` and it needs an input file with extension `.dat` and some arguments. Results will be printed on screen (`stdout`).

```
# Including Common Template
FNAME = exampleC
include : /path/to/condor_common.tmpl

executable    = myprogram
```

```
queue transfer_input_files,arguments *from* (
  xray434.dat, -d 345 -p f034
  sunf37.dat,  -d 2   -p f302
  light67.dat, -d 62  -p f473
)
```

- **Explanation:**

We will use the flexibility of `queue` command to assign values of a list to several commands. We must specify which files must be transferred and which arguments are needed by each file. We specify then `transfer_input_files` and `arguments` commands using the `from` option, and then we add a list of pairs "file,argument".

At submission time, HTCondor will iterate over the list and expand the assignations. For instance, our jobs will have next values:

- `transfer_input_files = xray434.dat, arguments = -d 345 -p f034-`
- `transfer_input_files = sunf37.dat, arguments = -d 2 -p f302-`
- `transfer_input_files = light67.dat, arguments = -d 62 -p f473-`

When using this format you can specify as many commands separated by commas as needed between `queue` and `from`, but check that each line in the list has the right number of elements also separated by commas.

Writing the list of items in the submit file can be a little bit tedious, but it may be easily done in an external file using scripts. Then you can directly specify the file. For instance, suppose you have all items in a file named `data.lst`, then you can use next `queue` command: `queue transfer_input_files,arguments from /path/to/data.lst`

3.3.2.4 Example D (lists of files and arguments in external file)

Process arbitrary files and arguments stored in file `data.lst` (process only lines from 28 to 43, both inclusive, with step 5). Executable is `myprogram` as in previous example, but this time it saves the result in a file named `output.out`.

```
# Including Common Template
FNAME = exampleD
include : /path/to/condor_common.tmpl

transfer_output_files  = output.out
line                   = $(Row)+1
transfer_output_remaps = "output.out=output$INT(line).out"

executable    = myprogram

queue transfer_input_files,arguments *from* [27:43:5] data.lst
```

- **Explanation:**

This example is similar to the previous one, but this time the list of input files and arguments is written in a file with the following format:

```
[input_file1,args1]
[input_file2,args2]
[input_file3,args3]
...
```

To illustrate the **slice** feature, we have been asked to process only items (lines) from 28 to 43 with step 5 (28, 33, 38 and 43), this could be useful when we want to run only certain experiments. The syntax for the slices is very easy, the same as Python: `[init:end:step]`. Since the first index is 0, but we do not use line 0 but line 1, the `init` should be 27. Then the `end` is 43 (it should be 42, but we need to add 1 because the upper limit is included according to our example). So we specify the slice using `[27:43:5]` in the `queue` command, between the `from` clause and the file.

We have to be careful with the results. Our program writes them in a file named `output.out`. We cannot get all files with the same name because they will be overwritten, so we need to use `transfer_output_remaps` to change names when copying from remote machines to our. We can add the `$(Process)` variable to the new name, so all of them will be different, but then it could be a little bit complicated to identify each result. Instead, we will use another of the **automatic variables**, called `$(Row)`. It stores the number of the row in the list that is being processed, that is, almost the number of the line: since `$(Row)` begins in 0, we need to add 1 to get the line number. We do that in variable `$(line)`. Then, HTCondor will process rows 27, 32, 37 and 42, and our output files will be `output28.out`, `output33.out`, `output38.out` and `output43.out`.

3.3.2.5 Example E (`stdin`, `initialdir` external scripts and lists)

Our program `myprogram` works with `stdin` (keyboard is used to specify input data). We have written that input data in 4 files (`dataFeH.in`, `dataOFe.in`, `dataOH.in` and `dataHe.in`) and there is a set of 3 different experiments in directories (`dir000`, `dir001` and `dir002`). Output files will be generated with the same name as inputs and extension `.out` (use `-o` argument) and they must be located in the same directory where the respective input file is. Program also needs all `*.tbl` files located in `/path/to/tables`.

```
# Including Common Template
FNAME = exampleE
include : /path/to/condor_common.tmpl

N            = 3
input        = data$(met).in
initialdir   = /path/to/dir$INT(Step,%03d)
include      : input_tables.sh |
transfer_output_files = data$(met).out

executable   = myprogram
arguments    = "-o data$(met).out"

queue $(N) *met* *in* FeH, OFe, OH, He
```

- **Explanation:**

The key of this example is the `queue` command in last line. We are using the clause `*in*` to specify a list of values. HTCondor will create a job for each element in the list and the current value will be assigned to the variable `met` that we have declared (this variable is optional, you can omit it and use the automatic variable `Item`). We have 3 set of experiments, so we need to go over the list 3 times, that is why we have defined `N = 3` and we are using `$(N)` in the `queue` command. So, at the end, HTCondor will execute 12 jobs (3 runs * 4 elements in the list): we will use automatic variable `$(Step)` to get the number of the present run (0, 1 or 2) and `$(met)` (or `$(Item)` if we omit the variable) to get the value of the current element in the list.

`input` command is used to specify a file that will be used as `stdin`, using variable `$(met)` to get the proper filename. That variable will be also used when building the name of the output files (`transfer_output_files` command) and the arguments (`arguments` command).

We use `initialdir` to specify a base directory that changes according to the current job, using the automatic variable `$(Step)`. HTCondor will use this directory as base for the relative paths, so it will affect the input and output files, including the `stdout`, `stderr` and log files created by HTCondor (see common template). We use `$INT(Step,%03d)` to get a 3-digit number (000, 001 and 002) to build the proper path for each experiment, then HTCondor will go to the right directory to get the input files and to place later the respective output files there.

Last thing we have to solve is the problem with the required input files (all `*.tbl` files located in `/path/to/tables`). HTCondor does not allow globbing in `transfer_input_files`, but instead we can use the new feature of **including external files** with `include` command. This command not only include other files, but also invoke them if the command finish with a **bar** `*|*`. Then we can easily make a external script to get the list of needed files with linux command `ls` and options `-m` (commas are used to separate elements) and `-w` (used to specify the wide of the screen before adding a new line. Since we need all elements in the same line, we should specify a number big enough). In this case, our external script `input_tables.sh` is the following one:

```
#!/bin/bash
echo "transfer_input_files = `ls -w 400 -m /path/to/tables/*.tbl`"
```

3.3.2.6 Example F (loops)

Execute each iteration of a 3-level nested loop using: `myprogram -dim1 i -dim2 j -dim3 k = with the following ranges:  =i:[0,20)`, `j:[0,15)` and `k:[0,35)`. Output will be written on screen, no input files are needed.

```
# Including Common Template
FNAME = exampleF
include : /path/to/condor_common.tmpl

MAX_I = 20
MAX_J = 15
MAX_K = 35

N = $(MAX_I) * $(MAX_J) * $(MAX_K)

I = ( $(Process) / ($(MAX_K)  * $(MAX_J)))
J = (($(Process) /  $(MAX_K)) % $(MAX_J))
K = ( $(Process) %  $(MAX_K))

executable = myprogram
arguments  = "-dim1 $INT(I) -dim2 $INT(J) -dim3 $INT(K)"

queue $(N)
```

- **Explanation:**

In this example we only need to "simulate" a 3 nested loops from a 1-level loop (we will use `$(Process)` as main loop counter). The 3-level loop will be the next ones, and HTCondor will create a job for each iteration:

```
for (i = 0; i < MAX_I; i++)
  for (j = 0; j < MAX_J; j++)
    for (k = 0; k < MAX_K; k++)
      ./myprogram  -dim1 i -dim2 j -dim3 k
```

Then we only need to set the limits (`MAX_I`, `MAX_J`, `MAX_K`), the number of total iterations (`N = $(MAX_I) * $(MAX_J) * $(MAX_K)`) and use some maths to get the values of `I`, `J` and `K` according the value of `$(Process)`, as we have done above (just a few multiplications, integer divisions and remeinders are needed).

For a 2-level loop, you can use next code:

```
I = ($(Process) / $(MAX_J))
J = ($(Process) % $(MAX_J))
```

3.3.2.7 Example G This example shows the use of several useful commands for specific conditions. It is also a summary of the **CondorHowTo**, you can find further details and explanation about the submit commands there.

- Execute `myprogram` with argument "=-run =" from 0 to 99 by default.
- **BLOCK A**: Execute only on machines with at least 4GB RAM and 2GB of free disk space. The higher memory and the faster calculations, the better (we can use KFLOPS to choose the faster machines doing floating point operations, but since memory and kflops have different units, we need to weight them, for instance, multiplying memory by 200).
- **BLOCK B**: Execute only on machines with Linux Fedora21 or upper and avoid executing on `cata`, `miel` and those with hostname beginning with letter `m` or `d`.
- **BLOCK C**: It is needed to run script `processData.sh` before (argument: `-decompress`) and after (argument: `-compress`) to prepare our data.
- **BLOCK D**: Our executable needs the environment variables and variable `OUT` has to be set with the argument.
- **BLOCK E**: Avoid **black holes** (when your jobs do not execute correctly on a machine, and since they finish quickly, that machine is getting most of the jobs).
- **BLOCK F**: Get a notification via email when errors in the job. If the job finishes before 5 minutes or takes more than 2 hours to be done, there was a problem: hold it to check later what happened.
- **BLOCK G**: Our program needs licenses, so we cannot run more than 20 jobs at the same time. Execute jobs as **nice user** to save priority since there are no other jobs running at this moment.

```
# Including Common Template
FNAME = exampleG
include : /path/to/condor_common.tmpl

if !defined N
  N = 100
endif

#BLOCK A
requested_memory = 4 GB
requested_disk   = 2 GB
rank             = (200 * Memory) + KFLOPS

#BLOCK B
letter           = substr(toLower(Target.Machine),0,1)
requirements     = (UtsnameSysname == "Linux")
        && (OpSysName == "Fedora") && (OpSysMajorVer >= 21)
        && !stringListMember(UtsnameNodename, "cata,miel")
        && !stringListMember($(letter), "m,d")


#BLOCK C
transfer_input_data = processData.sh
+PreCmd             = "processData.sh"
+PreArguments       = "-decompress"
+PostCmd            = "processData.sh"
+PostArguments      = "-compress"

#BLOCK D
getenv              = True
environment         = "OUT$(Process)"

#BLOCK E
```

```
job_machine_attrs = Machine
job_machine_attrs_history_length = 5
requirements = $(requirements)
      && (target.machine =!= MachineAttrMachine1)
      && (target.machine =!= MachineAttrMachine2)

#BLOCK F
notify_user       = myuser@iac.es
notification      = Error

on_exit_hold = ((CurrentTime - JobStartDate) < (5 * 60)
periodic_hold = ((JobStatus == 2)
         && (time() - EnteredCurrentStatus) >  (2  $(HOUR)))

#BLOCK G
concurrency_limits = myuser$(Cluster):50
nice_user = True

executable = myprogram
arguments  = "-run $(Process)"

queue $(N)
```

IMPORTANT: Although your program could use shared locations (`/net/XXXX/scratch`, `/net/nasX`, etc.) to read/write files from any machine so there is no need to copy files, we highly recommend **you always use the HTCondor file transfer system** to avoid network congestion since files will be accessed locally on the remote machines. Bear in mind that HTCondor can execute hundreds of your jobs at the same time, and if all of them concurrently access to the same shared location, network could experience a huge stress and fail. If for any reason you cannot copy files and you have to use shared locations -you are using huge files of several GB, etc.-, then contact us before submitting to adapt your jobs in order to avoid network congestion.

4 Submit files (HowTo)

HTCondor has a huge set of commands that should cover most possible scenarios. It is impossible to describe here all of them, but you can read the official documentation to get further information (there are many of these commands at the **condor$_{\text{submit}}$ page**. Just to complement **some of the previous examples**, we will mention here a few useful commands that can be added to the submit file when needed. You can also find more details about these and other commands at the **FAQs page**.

4.1 How to ... add requirements on the target machines where my jobs will be run?

If your program has some limitations (memory, disk, libraries, etc.) and cannot run in all machines, you can use `requirements` command to tell HTCondor what those limitations are so it can execute your jobs only on machines that satisfy those requirements. If you try next command `condor_status -long <your_machine>` in your shell, it will list all the parameters that HTCondor has about each slot of your machine, and most of those parameters can be used to add requirements. Conditions and expressions could be as complex as needed, there is a number of **operators, predefined functions, etc.** that can be used. For memory, disk and CPUs you can also use `request_memory`, `request_disk` and `request_cpus`, respectively.

For example, if your program needs at least 1.5 GB of RAM and 5 GB of free space in disk, and due to library dependencies it can only run on machines with Linux Fedora17 or above, add next commands in your submit file:

`request_disk` = 5 GB `request_memory` = 1.5 GB `requirements` = (UtsnameSysname `= "Linux") && (OpSysName` = "Fedora") && (OpSysMajorVer >= 17)

Be careful when specifying the values since the default unit for `request_disk` is KB and MB for `request_memory`. It is much better to always specify the unit (`KB` or `K`, `MB` or `M`, `GB` or `G`, `TB` or `T`).

Caution!: Be careful when choosing your restrictions, using them will reduce the amount of available slots for your jobs so it will be more difficult to execute them. Also check that you are asking for restrictions that can be satisfied by our current machines, or your jobs will stay always in idle status (you can check the reasons why a job is idle using `condor_q -analyze <job.id>`). Before adding a requirement, always check if there are enough slots that satisfy it. For instance, to see which slots satisfy the requirements of this example, use next command (you can add flag `-avail` to see only the slots that could execute your job at this moment):

[...]$ `condor_status` -constraint '(UtsnameSysname `= "Linux") && (OpSysName` = "Fedora") && (OpSysMajorVer >= 17) && (Memory > 1536) && (Disk >= 5120000)'

If you already know which machines are not able to run your application, you can force HTCondor to avoid them... or the opposite: run your application only on some machines (see **HOWTOs.#howto$_{\text{failing}}$** where this is explained).

4.2 How to ... add preferences on the target machines where my jobs will be run?

Preferences are similar to requirements, but they do not limit the machines (you can use both preferences and requirements in the same submit file). HTCondor will try to satisfy your preferences when possible, assigning a rank to the available machines and choosing those with higher value. For example, we would like to use slots with at least 4GB of RAM if they are available and the more available disk space, the better. Then commands to add should be the following ones:

`Rank` = Disk && (Memory >= 4096)

Rank is evaluated as a float point expression, and always higher values are the better ones. Then, you can do arithmetic operations to emphasize some parameters. For example, the first expression will consider the floating point speed but will give more importance to run on machines with my same Operating System, while the second expression will choose machines with higher values of RAM, and those with also more than 100GB of disk will have 200 extra "points":

`Rank` = kflops + (1000000 * (TARGET.OpSysAndVer `= MY.OpSysAndVer)) =Rank` = Memory + (200 * (Disk >= 102400))

4.3 How to ... get/set environment variables?

If you application needs them, use the `getenv` command and HTCondor will create a copy of your environment variables {~~at the submitting time~~} so they will be available for your program on the target machine. Also you can create/modify environment variables if needed with the `environment` command. The environment variables can be also used in the submit file using the `ENV` command.

For example, add next commands if you want that your executable can access your environment at the submitting time, then set variables called `working_dir` and `data_dir` pointing to some directories, and finally create a macro called `home_dir` that contains your home directory to be used in your submit file:

`getenv` = True `environment` = "working$_{\text{dir}}$=/path/to/some/place data$_{\text{dir}}$=/path/to/data" home$_{\text{dir}}$ = $ENV(HOME)

If you want to run your python program using HTCondor, you might need to define some environment variables, please, read this FAQ **HOWTOs.CondorFAQs#python-fedora**.

4.4 How to ... control HTCondor notifications?

If you are submitting a large set of jobs, receiving notifications from all of them can be annoying. You can set the email address and the type of the notifications that you want to receive. For example, to send notifications to `someaddress@iac.es` only in case of errors, use following commands:

`notify_user` = someaddress@iac.es `notification` = Error

`notify_user` changes the email address used to send notifications, if you need to add more addresses, you can use `email_attributes` command. With `notification` command we tell HTCondor when it should send those notifications, it could be set to `Never`, `Complete`, `Error` or `Always`; we recommend you use `Error`.

4.5 How to ... run some shell commands/scripts/programs before/after our application?

If your application needs some pre- or post-processing, you can use `+PreCmd` and `+PostCmd` commands to run it before and after your main executable, respectively. For example, these commands may be useful if you need to rename or move input or output files before or after the execution. You can also use them for debugging purpose, for instance, use `tree` command to check where the input/output files are located:

`+PreCmd` = "tree" `+PreArguments` = "-o tree$_{\text{before}}$.$(Cluster).$(Process).txt" `+PostCmd` = "my$_{\text{postscript.sh}}$" `+PostArguments` = "-g"

`should_transfer_files` = YES `transfer_input_files` = my$_{\text{postscript.sh}}$, /usr/bin/tree `transfer_output_files` = tree$_{\text{before}}$.$(Cluster).$(Process).txt

Remember that you have to add those scripts/programs to the list of files to be copied with `transfer_input_files` command, and also check that your submit file contains the following command: `should_transfer_files = YES`. When using a shell command (like `tree`), you can get its location using the command `which`. For instance, `which tree` will show `/usr/bin/tree`, this is the path you should add to `transfer_input_files` command.

4.6 How to ... specify the priority of your jobs?

HTCondor uses two different types of priorities: **job priority** (which of your jobs will run first) and **users priority** (which users will run their jobs and how many of them).

[+*Job priority*+]

If some of your jobs/clusters are more important than others and you want to execute them first, you can use the `priority` command to assign them a priority (the higher the value, the higher priority). For instance, if you want to execute first the last jobs of a cluster (reverse order), you can use next

command: `priority` = $(Process)

Remember that after submitting your jobs, you can set or change their priority using the shell command `condor_prio`.

[+*Users priority*+]

Whenever your jobs are being executed, your user priority is decreased (the more jobs are executed, the faster you lose priority). Users with best priority will run more jobs and they will begin sooner, so if your jobs are not so important

or the queue is empty, you can use `nice_user` command to run them without wasting your priority. If you set this command to `True`, your jobs will be executed by a "fake user" with very low priority, so you will save your real priority (but it is likely your jobs will not be executed unless the queue is almost empty).

`nice_user` = True

Those jobs will be run with user `nice-user.<your_user>` and they will not change your user's priority (you can use shell command `condor_userprio -allusers` to see your and other users' priority).

Remember that using `condor_qedit` command you can change the attributes of your jobs after submitting them (see this FAQ **HOWTOs.CondorFAQs#ch$_{\text{submit}}$**). We can use this command to change the status of `NiceUser` attribute depending on how many slots are free (if there are many free slots, then we can set our jobs as "nice" to run them without affecting our priority, or the opposite, setting `NiceUser` to `false` when there are no free slots). For instance, use next commands to set all jobs belonging to Cluster ID `1234`:

`[...]$ condor_q 1234 -af ProcId NiceUser` "#Check current status" 0 false 1 false ...

`[...]$ condor_qedit 1234 NiceUser True` Set attribute "NiceUser".

`[...]$ condor_q 1234 -af ProcId NiceUser` "#Check current status" 0 true 1 true ...

If user priority is a critical factor to you, you may want to periodically check the queue to change the `NiceUser` attribute according to the current status, setting it to `True` when you are the only active user or there is a large number of available slots, and set it to `False` when there are more active users or a few available slots. In order to simplify this process, we have developed a script that automatically performs those operations, you only need to specify your username or a particular `clusterID` (and optionally a minimum number of available slots) and it will change the `NiceUser` attribute to save your real priority as much as possible. You can copy the script from `/net/vial/scratch/adorta/htcondor_files/htcondor_niceuser.sh` and use it (or modify it) whenever you want. You can even periodically execute it using `crontab` (but please, do NOT run it too often to avoid overloading the system, every 30 or 60 minutes is fine). Run it with no arguments to get the description and syntax.

Notes:

- You can use `condor_qedit *-constraint* ...` to change the attributes of only some of your jobs.
- Condor can evaluate the attributes only when jobs begin to run, so new values may not affect the currently running jobs at the time of using `condor_qedit`, but they will be valid in jobs that begin to run after using the command.

4.7 How to ... deal with jobs that fail?

Sometimes jobs fail because there are problems when executing your program. It could happen that the problem is not in your program, but in the machine that executed it (a missing or misconfigured application, a library with a different version from the one you need, etc.). Then you should identify those problematic machines and use `requirements` commands in your submit file in order to block them, as is explained in this FAQ **HOWTOs.CondorFAQs#blackholes**. For example, to block machines with names `piston` and `loro` use only one of the next commands (both are equivalent):

`requirements` = ((UtsnameNodename `!` "piston") && (UtsnameNodename `!` "loro")) `requirements` = *!*=stringListMember=(UtsnameNodename, "piston,loro")

You can also block all machines that satisfy a pattern. For instance, to avoid executing your jobs on those machines with names beginning with "k", "c" and "l", add next lines (you can specify more complex patterns using the **predefined functions and macros**:

letter = `substr=(toLower(Target.Machine),0,1)` `=requirements` = *!*=stringListMember=($(letter), "k,c,l")

Sometimes it is better to specify a list of machines where your application can run (and avoid any other that is not in that list). For that purpose, just use previous expressions after negating them with an "exclamation mark" "`!`" (or remove it if they were already negated).

After avoiding machines that are not able to run your program, you should submit again your jobs. But, please, execute {~~only~~} those jobs that failed (check this FAQ **CondorFAQs#repeat** to see how), do not execute again jobs that were already correctly executed to avoid wasting time and resources. For instance, add next command to only execute jobs with Process ID `0`, `13`, `25` and those from `37` to `44`:

`noop_job` = *!*=(stringListMember=("(*Process*)","0, 13, 25")||(((Process) >= 37) && ($(Process) <= 44)))

Note: `noop_job` will "not" execute those jobs where the condition is `True`. Therefore, if you want to specify a list of jobs "to be executed", you need to "negate" your expression adding an exclamation mark at the beginning: `noop_job = *!*(...)`. On the other hand, if you want to specify a list of jobs that should "not" be executed, then use the expression without negating it.

Jobs that are not executed may stay in the queue with `Complete` status (when using `condor_q` you will see that `ST` column is `C`). To remove all `C` jobs from the queue, try next command in your shell (use the second one to remove {~~only~~} `Complete` jobs that belongs to cluster `XXX`):

`condor_rm` -constraint 'JobStatus `= 4' =condor_rm` -constraint 'JobStatus `= 4 && clusterID =` XXX'

Also, it could be interesting to avoid the **black holes**: suppose that each of your jobs needs hours to finish, but they fail in an specific machine after a few minutes of execution time. That means that machine will be idle every few minutes, ready to accept another of your jobs, that will also fail, and this process may repeat again and again... sometimes a failing machine could even execute almost all your jobs... That is known as "black hole". To avoid it, we can force HTCondor to change machines when sending jobs. For that purpose add these lines to your submit file:

"#Avoid black holes: send to different machines" `job_machine_attrs` = Machine `job_machine_attrs_history_length` = 5 `requirements` = $(requirements) && (target.machine `!` MachineAttrMachine1) && (target.machine `!` MachineAttrMachine2)

When there are problems with your jobs, you should receive an email with an error and some related information (if it was not disabled using `notification` command as explained above) and the job will leave the queue. You can change

this behavior with `on_exit_hold` and/or `on_exit_remove` commands, forcing HTCondor to keep that job in the queue with status "on hold" or even as "idle" so it will be executed again:

(:table border=1 cellpadding=5 cellspacing=0 width=70% align=center:) (:cell align=center valign=middle :) **Command** (:cell align=center valign=middle :) * True * (:cell align=center valign=middle :) * False * (:cellnr align=center valign=middle :) `on_exit_hold` (:cell align=center valign=middle :) Stay in the queue with "on hold" status (:cell align=center valign=middle :) Leave the queue (:cellnr align=center valign=middle :) `on_exit_remove` (:cell align=center valign=middle :) Leave the queue (:cell align=center valign=middle :) Stay in the queue with "idle" status (it can be executed again) (:tableend:) \$\

Last commands will be evaluated when jobs are ready to exit the queue, but you can force a periodic evaluation (using a configurable time) with commands like `periodic_hold`, `periodic_remove`, `periodic_release`, etc., and then decide if you want to hold/remove/release them according to your conditions. There are also some other commands to add a "reason" and/or a "subcode" when holding/removing/releasing these jobs. On the other hand, you can force your jobs to exit the queue when they satisfy a given condition using `noop_job`, or they stay in the queue even after their completion using `leave_in_queue` command (those jobs will stay in the queue with `Complete` status till you remove them using shell command `condor_rm`).

In the `http://research.cs.wisc.edu/htcondor/manual/v8.6/condor_submit.html#condor-submit-on-exit-hold` official HTCondor documentation there are some examples about how to use these commands (all valid `JobStatus` could be displayed using shell command: `condor_q -help status`):

- With the next command, if the job exits after less than an hour (3600 seconds), it will be placed on hold and an e-mail notification sent, instead of being allowed to leave the queue:

  `on_exit_hold` = ((CurrentTime - JobStartDate) $<$ 3600)

- Next expression lets the job leave the queue if the job was not killed by a signal or if it was killed by a signal other than 11, representing segmentation fault in this example. So, if it exited due to signal 11, it will stay in the job queue. In any other case of the job exiting, the job will leave the queue as it normally would have done.

  `on_exit_remove` = ((ExitBySignal `= False) || (ExitSignal !` 11))

- With next command, if the job was killed by a signal or exited with a non-zero exit status, HTCondor would leave the job in the queue to run again:

  `on_exit_remove` = ((ExitBySignal `= False) && (ExitCode =` 0))

- Use the following command to hold jobs that have been executing (`JobStatus =` 2=) for more than 2 hours (by default, all periodic checks are performed every 5 minutes. Please, contact us if you want a shorter period):

  `periodic_hold` = ((JobStatus == 2) && (time() - EnteredCurrentStatus) $>$ 7200)

- The following command is used to remove all "completed" (`JobStatus =` 4=) jobs 15 minutes after their completion:

  `periodic_remove` = ((JobStatus == 4) && (time() - EnteredCurrentStatus) $>$ 900)

- Next command will assign again the "idle" status to "on hold" (`JobStatus =` 5=) jobs 30 min. after they were held:

  `periodic_release` = ((JobStatus == 5) && (time() - EnteredCurrentStatus) $>$ 1800)

[-**IMPORTANT**: `periodic_release` command is useful when your program is correct, but it fails in specific machines and gets the "on hold" status. If that happens, this command will allow HTCondor to periodically release those jobs so they can be executed on other machines. But {~~use this command with caution~~}: if there are problems in your program and/or data, then your application could be indefinitely held and released, what means a big waste of resources (CPU time, network used in file transferring, etc.) and inconveniences for other users, be careful! (you can always remove your jobs using `condor_rm` command in your shell).-]

When using `periodic_remove` or `periodic_hold` HTCondor submit commands, running jobs that satisfy the condition(s) will be killed and all files on remote machines will be deleted. Sometimes you want to get some of the output files that have been created on the remote machine, maybe your program is a simulation that does not converge for some sets of inputs so it never ends, but it still produces valid data and you want to get the output files. In those cases, do not use the mentioned submit commands because you will lose the output files, and use instead utilities like `timeout` in order to limit the time that your application can be running. When using this linux command, you specify the maximum time your program can run, and once it reaches that limit, it will be automatically killed. Then HTCondor will detect your program has finished and it will copy back the output files to your machine as you specified. Next example will show how to limit the execution of your program up to 30 minutes:

"# Some common commands above..." ...

"# Max running time (in seconds)" MAX_{TIME} = 30 * 60

"# Your executable and arguments" MY_{EXEC} = $\text{your}_{\text{exec}}$ MY_{ARGS} = "$\text{your}_{\text{arg1}}$ $\text{your}_{\text{arg2}}$"

"# If your executable is not a system command, do not forget to transfer it!" `transfer_input_files` = $\text{your}_{\text{inputs}}$,$(MY$_{EXEC}$)
"# By default all new and modified files will be copied. Uncomment next line to indicate only specific output files"
"#=$\text{transfer}_{\text{outputfiles}}$= = $\text{your}_{\text{outputs}}$"

`executable` = /bin/timeout "# Since timeout is a system command, we do not need to copy it to remote machines"
`transfer_executable` = False `arguments` = "$INT($\text{MAX}_{\text{TIME}}$) $(MY$_{EXEC}$) $(MY$_{ARGS}$)"

`queue` ...

4.8 How to ... limit the number of concurrent running jobs?

There are some situations where it could be interesting to limit the number of jobs that can concurrently run. For instance, when your application needs licenses to run and few of them are available, or when your jobs access a shared resource (like directly reading/writing files located at `scratch`, too many concurrent access could produce locks and a considerable slowdown in your and others' computer performance).

To deal with these situations, HTCondor is able to manage limits and apply them to running job. Different kinds of limits can be defined in the "negotiator" (the machine that decides which job will run on which slot), but, unfortunately, you cannot change its configuration (for obvious security reasons, only administrators can do that). If you want to use a limit, you can contact us so we will configure it, but there is an easier way to use this feature without changing the configuration: we have set a high default value (1000 units) for any undefined limit, so you only need to use a limit not defined yet and adjust the number of consumed units per job. For example, suppose that you would like to limit your concurrent running jobs to 20: then you only need to specify that every job consumes 50 units of that limit (1000 / 20 = 50). In this way no more than 20 jobs could concurrently run.

The command used to specify limits is `concurrency_limits = XXX:YYY`, where `XXX` is the name of the limit and `YYY` is the number of units that each job uses. You can use any name for the limit, but it should be unique, so we recommend you include your username in it.

- For instance, if your username is `jsmith` and you want to specify a limit of 12 running job (1000 / 12 ~= 83 units/job), just add next line to your submit file:

  `concurrency_limits` = jsmith:83
- Previous command will affect all your jobs that use that limit, even in different submissions. If you want to set limits that are only applied to each submission, you can use a combination of your username and the cluster ID in the name of the limit:

  `concurrency_limits` = jsmith$(Cluster):83
- If you need it, you can use several limits and specify them in the same command line, using "commas" to build the list of limits and consumed units per job. For instance, next line will limit to 12 the number of running jobs in this submission and to 25 (1000 / 25 = 40) the number of your total running jobs where the common limit `jsmith_total` has been used:

  `concurrency_limits` = jsmith$(Cluster):83,$\text{jsmith}_{\text{total}}$:40
- If you are executing jobs with **IDL without the IDL Virtual Machine**, then each job will be using one license. Since the total amount of licenses is limited, you must add next line in your submit file:

  `concurrency_limits` = idl:40

Limits can be changed after jobs are submitted using `condor_qedit` command. For instance, we want to change the limit that we have previously set to `jsmith:83` (12 concurrent jobs) to `jsmith:50` (20 concurrent jobs) in all jobs belonging to Cluster with ID `1234`. Then use next commands:

```
[...]$ condor_q 1234 -af ProcId ConcurrencyLimits "#Check current limit" 0 "jsmith:83" 1 "jsmith:83" ...
[...]$ condor_qedit 1234 ConcurrencyLimits '"jsmith:50"' Set attribute "ConcurrencyLimits".
[...]$ condor_q 1234 -af ProcId ConcurrencyLimits "#Check current limit" 0 "jsmith:50" 1 "jsmith:50" ...
```

Values may have to be specified using quotes; be careful if your value is a string since you will be need to combine simple and double quotes, like `'"..."'` (see example above).

Note: HTCondor may evaluate the attributes only when jobs begin to run, so new values may not affect the currently running jobs at the time of using `condor_qedit`, but they will be valid in jobs that begin to run after using the command.

4.9 How to ... do some complex operations in my submit file?

If you need to do some special operations in your submit file like evaluating expressions, manipulating strings or lists, etc. you can use the **predefined functions** and some **special macros** that are available in HTCondor. They are specially useful when defining conditions used in commands like `requirements`, `rank`, `on_exit_hold`, `noop_job`, etc. since they will allow you to modify the attributes received from the remote machines and adapt them to your needs. We have used some of these predefined functions in our examples, but there are many others that could be used:

- evaluate expressions: `eval()`, ...
- flow control: = ifThenElse()=, ...
- manipulate strings : `size()`, `strcat()`, `substr()`, `strcmp()`, ...
- manipulate lists: `stringListSize()`, `stringListSum()`, `stringListMember()`, ...
- manipulate numbers: `round()`, `floor()`, `ceiling()`, `pow()`, ...
- check and modify types: `isReal()`, `isError()`, `int()`, `real()`...
- work with times: `time()`, `formatTime()`, `interval()`, ...
- random: `random()`, `$RANDOM_CHOICE()`, `$RANDOM_INTEGER()`, ...
- etc.

Check the documentation to see the complete list of **predefined functions**, and also the **special macros**.

4.10 How to ... work with nested loops?

You can use `$(Process)` macro to simulate simple loops in the submit file and use the iterator to specify your arguments, input files, etc. However, sometimes simple loops are not enough and nested loops are needed. For example, assume you need to run your program with the arguments expressed in the next pseudocode:

$MAX_I = 8$ $MAX_J = 5$

`for` (i = 0; i < MAX_I; i++) `for` (j = 0; j < MAX_J; j++) ./myprogram -var1==i= -var2==j=

To simulate these 2 nested loops, you will need to use next macros in your HTCondor submit file:

$MAX_I = 8$ $MAX_J = 5$ N = MAX_I * MAX_J ... I = ($(Process) / $(MAX_J$))J = (($Process) % $($MAX_J$)) ... `executable` = myprogram `arguments` = "-var1=$=INT=(I) -var2=$=INT=(J)" `queue` $(N)

Last code will produce a nested loop where macro `$(I)` will work like the external iterator with values from `0` to `7`; and `$(J)` will be the internal iterator with values from `0` to `4`.

If you need to simulate 3 nested loops like the next ones: `for` (i = 0; i < MAX_I; i++) `for` (j = 0; j < MAX_J; j++) `for` (k = 0; k < MAX_K; k++) ...

then you can use the following expressions: N = $($MAX_I$) * $($MAX_J$) * $($MAX_K$)

I = (($Process)/(($MAX_K$) * ($MAX_J$)))$J$ = ((($Process) / $($MAX_K$)) % $($MAX_J$)) K = ($($Process) % $($MAX_K$))

`executable` = myprogram `arguments` = "-var1 $=INT=(I) -var2 $=INT=(J) -var3 $=INT=(K)" ... `queue` $(N)

4.11 How to ... program my jobs to begin at a predefined time?

Sometimes you may want to submit your jobs, but those jobs should not begin at that moment (maybe because they depend on some input data that is automatically generated at any other time). You can use `deferral_time` command in your submit file to specify when your jobs should be executed. Time has to be specified in "Unix epoch time" (the number of seconds elapsed since 00:00:00 on January 1, 1970, Coordinated Universal Time), but, do not worry, there is a linux command to get this value:

`date` –date "MM/DD/YYYY HH:MM:SS" +%s

For instance, we want to run a job on April 23rd, 2016 at 19:25. Then, the first step is to get the epoch time:

[...]$ `date` –date "04/23/2016 19:25:00" +%s

Our value is `1461435900`, so we only need to add next command to the submit file:

`deferral_time` = 1461435900

Bear in mind that HTCondor will run jobs at that time according to remote machines, not yours. If there are wrong dates or times in remote machines, then your jobs could begin at other dates and/or times.

Also you can add expressions, like the next one to run your jobs one hour after the submission:

`deferral_time` = (CurrentTime + 3600)

It may happen that your job could not begin exactly at that time (maybe it needs that some files are transferred and they are not ready yet), and in that case HTCondor may kill your job because the programmed time has expired and your job is not already running. To avoid that, you can specify a "time window" to begin the execution, a few minutes should be enough. For instance, add next command to tell HTCondor that your job could begin up to 3 minutes (180 seconds) after the programmed time:

`deferral_window` = 180

Important: When you submit your programmed jobs, HTCondor will check which machines are able to run them and once the match is done, those machines will wait for the programmed time and will not accept any other jobs (actually, it will show "Running" status while waiting for the programmed time). That means a considerable loss of resources that should be always avoided. Using `deferral_prep_time` command we can specify that HTCondor could use those matched machines till some time before really running your jobs.

Then, add next lines to begin your jobs on April 23rd, 2016 at 19:25, specifying that they can begin up to 3 minutes after that date and that HTCondor could run other jobs on the matched machines till one minute before the programmed time:

`deferral_time` = 1461435900 `deferral_window` = 180 `deferral_prep_time` = 60

HTCondor also allows you to use more powerful features, like specifying jobs that will be periodically executed at given times using the "CronTab Scheduling" functionality. Please, read the **Time Scheduling for a Job Execution** section in the official documentation to get more information.

4.12 How to ... run jobs that have dependencies among them?

If your jobs have dependencies related to inputs, outputs, execution order, etc., you can specify these dependencies using a "directed acyclic graph (DAG)". HTCondor has a manager (called `http://research.cs.wisc.edu/htcondor/manual/v8.6/2_10DAGMan_Applications.html` DAGMan) to deal with these jobs.

First, you have to create a DAG input file, where you specify the jobs (including the respective HTCondor submit file for each one) and the dependencies. Then, you submit this DAG input file using `condor_submit_dag <dag_file>`. Next code describes a basic example of DAG file where job A depends on B and C, that depend on D (diamond shape).

JOB A $condor_{A.submit}$ JOB B $condor_{B.submit}$ JOB C $condor_{C.submit}$ JOB D $condor_{D.submit}$ PARENT A CHILD B C PARENT B C CHILD D

[...]$ `*condor_submit_dag*` diamond.dag

Examples about working with HTCondor DAGMan can be found in the `http://research.cs.wisc.edu/htcondor/manual/v8.6/2_10DAGMan_Applications.html#SECTION003102000000000000000` Official documentation mentioned above. You can also try the easy example located at the end of this page `http://research.iac.es/sieinvens/SINFIN/`

`Condor/curso/course/node7.php` used in a course about HTCondor imparted by SIE some years ago (solution here `http://research.iac.es/sieinvens/SINFIN/Condor/curso/course/node9.php#SECTION00095000000000000000`).

4.13 How to ... know the attributes of the machines where our jobs are run?

There is a special macro to get the string attributes of target machines that we can use in our submit file. In this way, some of the parameters of each machine where HTCondor executes jobs can be accessed with `$$(parameter)`. Also there are other special macros, like the one used to print the symbol `$` since it is reserved by HTCondor: `$(DOLLAR)`.

For example, we want to know the name of each slot and machine where our jobs were executed, adding `.$.name.$.` to the results of `stdout` and `stderr`. Then we should use next commands:

`output` = myprogram.$(ID).$(DOLLAR).

$$(Name).$$

(DOLLAR).out =error= = myprogram.(ID).(DOLLAR).

$$(Name).\$(DOLLAR).err$$

Ading those commands in your submit file will create output and error files with names similar to `$.slot3@xilofon.ll.iac.es.$`.