Skip to main content.

Navigation: Home | News | Download | Publications | Contact Us
Search

Examples for Submit Description Files

Here are many examples of submit description files, to introduce submitting a job to HTCondor. It is assumed that HTCondor is both installed and configured. The examples progress from simple to sophisticated, introducing useful features.

Each job has an associated submit description file. This plain text file contains a set of line-oriented commands that describe the environment of the job as it runs.

Comments are permitted in a submit description file. They are lines that begin with the pound sign (#).

A First Job -- The Vanilla Universe

A first example submits a run of the program `mathematica` as a vanilla universe job. The vanilla universe does not require a re-compilation of the program using `condor_compile`. Where the source and/or object code to a program is not available, as well as for job execution on a Windows machine (where the standard universe is not available), execution under the vanilla universe is the simplest choice.

A very simple submit description file goes with this example. It queues up program `mathematica` to run one time under HTCondor.

```
########################
# Submit description file for mathematica
########################
executable     = mathematica
universe       = vanilla
input          = test.data
output         = test.out
error          = test.error
log            = test.log

queue
```

The contents of the submit description file identify the executable, the universe, and they specify files for input, output, error, and logging. This submit description file is placed in the same directory as the executable (`mathematica`). The input file, `test.data`, is utilized as standard input for job execution. The output file, `test.out`, is similar in function, but is used for standard output. Standard error goes to `test.error`. A log file, `test.log`, will be produced that contains events the job had during its lifetime inside of HTCondor. When the job finishes, its exit conditions will be noted in the log file. Use of a log file is always recommended, as it permits the job submitter to keep track of what happened to the job. No platform

(architecture and operating system) is specified, so HTCondor will use its default assumption: that the executable is meant to run on a machine with the same platform as the machine from which it was submitted. The queue command tells HTCondor to submit the job for execution one time. Note that the capitalization within the submit description file is unimportant for the commands, but it is preserved for file names.

A Second Job -- The Standard Universe

This second example job uses the standard universe. Therefore, the program must be linked with the HTCondor libraries to provide the remote system call technology. The standard universe is supported on a subset of Unix platforms. Preparing this simple job starts with writing, compiling, and submitting the "hello, world" program. The C language source code is written:

```
#include <stdio.h>

int main(void)
{
    printf("hello, HTCondor\n");
    return 0;
}
```

This code is compiled (using `gcc`) to produce object files:

```
gcc -c hello.c -o hello.o
```

Use of the standard universe requires that the program be linked with the HTCondor libraries to provide the remote system call technology. This I/O support will be needed to send the program's output to a file. Link in the HTCondor libraries (again using `gcc`):

```
condor_compile gcc hello.o -o hello
```

A very simple submit description file goes with this example. It queues up program `hello` to run one time under HTCondor. The submit description file is called `submit.hello` for this example.

```
########################
# Submit description file for hello program
########################
Executable     = hello
Universe       = standard
Output         = hello.out
Log            = hello.log
Queue
```

This submit description file is placed in the same directory as the executable, `hello`. The executable is specified, as it must be for every job submission. A file where output it to be sent is specified for this program. No `input` or `error` commands are given in the submit description file, so there will be no input, and any error messages will be thrown away. (Files `stdin`, and `stderr` refer to `/dev/null` on Unix by default). A log file, `hello.log`, will be produced that contains events the job had during its lifetime inside of HTCondor. When the job finishes, its exit conditions will be noted in the log file. It is recommended to always have a log file to keep track of what happened to the jobs. No platform (architecture and operating system) is specified, so HTCondor will use its default, which is to run the job on a machine with the same platform as the machine from which it was submitted. The `Queue` command tells HTCondor to submit the job for execution one time.

The program is submitted for execution under HTCondor using the program `condor_submit`.

```
condor_submit submit.hello
```

Example 3 - command line arguments

Command line arguments for the executable can be given in the submit description file. This third example shows the use of arguments.

```
####################
# Show use of command line arguments.
####################

executable     = fib
universe       = vanilla

arguments      = 40
output         = fib.out
error          = fib.error
log            = fib.log

queue
```

This submit description file submits program `fib` one time for execution under HTCondor. The vanilla universe is explicitly specified. In those cases where the universe is not explicitly specified, a configuration variable may specify a universe to use. Where the configuration variable is not present, the vanilla universe is assumed. When program `fib` is executed, it is given as command line arguments the string from the `arguments` command. So, the command line of the submitted job would appear as

```
fib 40
```

More examples of command line arguments are given in the `condor_submit` manual page, listed in the section on basic commands.

Example 4 - multiple submission

A fourth example submits the program `mathematica` twice. Separate directories are used to hold the input and output files from the two separate executions of the program. The first submission uses files in directory `run_1` and the second submission uses files in directory `run_2`. The same file names are used within the directories, and the separate directories serve to identify the two separate submissions of the program. For both submissions, `stdin` will be `test.data`, `stdout` will be `test.out`. and `stderr` will be `test.error`. This is a convenient way to organize data when multiple submissions of the same program are to be run with different input data sets.

```
####################
# Two submissions of mathematica
####################

executable  = mathematica
universe = vanilla
input   = test.data
output  = test.out
```

```
error   = test.error
log     = test.log

initialdir  = run_1
queue

initialdir  = run_2
queue
```

Example 5 - more on multiple submission

The submit description file for this fifth example queues 150 runs of program `foo` which has been compiled and linked using `condor_compile`, and therefore uses the standard universe. It shows the use of a macro to separate the input and output files for 150 executions of the program.

```
####################
# Show use of pre-defined macros.
####################

Executable     = foo
Universe       = standard

Error   = err.$(Process)
Input   = in.$(Process)
Output  = out.$(Process)
Log     = foo.log

Queue 150
```

Placing the integer after the `Queue` command tells HTCondor to run the program, not the default of one time, but the number of times specified by the integer, 150 times for this example. Using `condor_submit` with this submit description file causes a unique number, called the cluster number, to be assigned for all jobs queued with the single `Queue` command. Further, each queued execution of program `foo` is given a own unique number, called a process number. Together, these numbers uniquely identify each execution of a program under HTCondor. They are an ID number and listed as `Cluster.Process` when given, for example, by `condor_q`. The numbers are available for use within the submit description file as macros. This example uses the `Process` macro to uniquely name the input, output, and error files of the 150 executions of program `foo`. Process numbers start at 0 for each job submission. Therefore, the first execution of the program within the 150 will use `in.0` for input, `out.0` for output, and `err.0` for errors. `in.1`, `out.1`, and `err.1` will be used for the second execution of the program. A single log file containing entries about when and where HTCondor runs, checkpoints, and migrates processes for the 150 queued programs will be written into file `foo.log`.

Example 6 - one more on multiple submission

A common situation has one executable that is executed many times, each time with a different input set. If the program wants the input in a file with a fixed name, then the solution of choice runs each queued job in its own directory. This example submit description file is similar to Example 4, but it uses a macro (like Example 5) to describe the directory for each job to be queued.

```
####################
# Multiple jobs queued, each in its own directory
```

```
####################

universe = standard
executable = a_fortran_job
output = job_output
error = job_error
log = job_log
initialdir = job.$(Process)
queue 10
```

Assume there is one input file for each of the 10 jobs to be queued. The program uses a fixed name for this input file, so `input` is not specified in the submit description file. Each of the 10 input files is pre-staged within the appropriate directory before submission. The directories must be named `job.0`, `job.1`, `job.2`, . . ., `job.9` for the ten directories needed. In addition to the input file within its own directory, each directory will receive its own output in a file called `job_output`, its own error messages will go into `job_error`, and HTCondor will log each job's progress within its directory in the file called `job_log`.

Example 7 - on requirements and rank

The submit description file for this seventh example expands upon the fifth example. It queues 150 runs of program `foo` which has been compiled specifically for an X86 processor running RHEL 5.

```
####################
# Use both requirements and rank
####################

Executable     = foo
Universe       = standard
Request_memory = 32
Requirements   = OpSys == "LINUX" && Arch =="INTEL"
Rank           = Memory >= 64

Error   = err.$(Process)
Input   = in.$(Process)
Output  = out.$(Process)
Log     = foo.log

Queue 150
```

Each instance of the program will be matched with and run on a machine which has greater than 32 Megabytes of physical memory, as set by `Request_memory`. The platform is identified by the `Requirements` expression. The `Rank` expression expresses a preference to run each instance of the program on a machine with more than 64 Megabytes of physical memory, if such machines are available.

Example 8 - Executables for more than one platform

A program is compiled to execute on a specific platform (architecture and operating system combination). Therefore, HTCondor must run an executable program on a specific platform. This limits the choice of machines that can run the program, where more than one platform exists in the pool of machines. If executables can be produced for more than one platform, then the submit description file can specify the platform-specific executable by using a substitution macro.

```
####################
# executables exist for more than 1 platform
####################

Universe       =  vanilla
Executable     =  inhouse_solver.$$(OpSys).$$(Arch)
Requirements   = (OpSys == "LINUX" && Arch =="X86_64") ||
                 (OpSys == "WINDOWS" && Arch =="INTEL")

Error   = mathematica.err
Input   = mathematica.in
Output  = mathematica.out
Log     = mathematica.log

Queue
```

In this example, program `inhouse_solver` has executables for two platforms. The executables or links to the executables are in the directory with the submit description file. One of the executables is for a 64-bit Intel processor running Linux, and the other is for a Intel x86 machine running Windows 7. The `requirements` command requires that program be run on one of these two platforms. This executable is chosen after HTCondor allocates a machine for running the program. The substitution macro used in the `executable` command specifies the name of the executable. After macro substitution, the name of the executable will be one of

```
inhouse_solver.LINUX.X86_64
inhouse_solver.WINDOWS.INTEL
```

For comments or questions about this website, please email htcondor-admin@cs.wisc.edu
This work was supported in part by NSF grants MCS-8105904, OCI-0437810, OCI-0850745, and/or ACI-1321762.
Updated » January 15, 2014