HTCondor Configuration (and Submit) Language

John (TJ) Knoeller

HTCondor Week UK 2018

Overview

- HTCondor uses a common "language" to parse and query config files and submit files.
- It is a complex and quirky language that does a lot of different things
- Prepare to be amazed (and appalled...)

Submit/Config “language”

- Submit files and config files consist of
  - *Key* = *Value*
    - *Key* is case-insensitive
    - *Value* has no type (its all just text to config/submit)
    - Values can refer to other values using $(key)
  - Statements (if, include, queue)
- Keys are loaded into a key:value store
- Statements are executed

Submit/Config key:value store

- Submit files and config files are loaded into a key:value store that can be queried.
  - Last definition of a key wins
  - All keys are stored but...
  - Only some keys have meaning to HTCondor
    - Config and Submit have different schemas/defaults
- Statements are executed as the file is read
  - Python bindings see only the key:value store

Behind the key:value store

- Compiled into HTCondor is **another** key:value store containing default values
  - For config, we call this the **param table**
  - config lookups use the param table when there is no entry in the config key:value store
  - submit has a similar (but much smaller) table of default values for Arch, Opsys, etc

Config/Submit File Syntax

```
# Keys (aka knobs, macros, params)
# are case insensitive
log=/var/log/condor
LOG = /var/log/condor
# use \ for line continuation
collector_host=condor.cs.wisc.edu,\
    secondary.cs.wisc.edu \
#    tertiary.cs.wisc.edu
```

Line continuation after comment

```
# We want to frob the bobulator \
FROB_BOBULATOR = true
```

- In 8.2+ : \ at the end of a comment line is ignored, so every comment line needs its own #
- Before 8.2 : **** at the end of a comment line ‘eats’ the next line, so `FROB_BOBULATOR` is not set

Comment after line continuation

```
ALLOW = A \
   B \
   # C \
   D
```

- In 8.2+ : The ALLOW list is A B D
- Before 8.2 : The ALLOW list is A B # C D
  - # is a member of the list!

Macro substitution

- Values can reference the value of other Keys using $(key)

```
A = $(B)

SCHEDD = $(SBIN)/condor_schedd
```

- Reference is a **text** substitution of the last value assigned to the key
- Whitespace around the = and at the end of line are removed before key assignment

Substitution times

- Last definition of a key wins, so if

```
A=1
B=$(A)
A=2
```

  A and B will both evaluate to 2

- Substitutions happen at time of lookup/use. Which is **after** all files have been read
  - **Except** self references and statements - they substitute as the file is read

Self References

- Self references are substituted as the file is read. For example:

```
A = $(B)
A = $(A) stuff
```

  Is the same as configuring: `A=$(B) stuff`

- Circular references are not allowed

```
A = $(B)
B = $(A)
```

  Daemon or tool will (eventually) abort

Substitution with a default

`$(key:default)`

- Substitutes as the value of ***`key`*** if it is defined, otherwise it is ***`default`***

  example:

  `NUM_SLOTS = $(NUM_CPUS:2)/2`

  Number of slots will be either the final value of NUM_CPUS divided by 2 or it will be 2/2

Expressions

- Many values can be classad expressions
  - Depends on who does the lookup
    - Values like Requirements **must** be expressions
  - Most numeric value lookups are evaluated

```
# this works, evaluates to 4
NUM_SLOTS = size(split("a,b,c,d"))
```

  - Most string value lookups are not evaluated

```
# this does NOT work as intended
Executable = strcat("sleep",".exe")
```

Multiline values

```
key @=tag
value
...
@tag
```

- The value of key will be the lines between @=*tag* and @*tag.* for example

```
CLASSAD_USER_MAPDATA_Groups @=end
   * Bob   Physics,Music
   * Alice Physics,Math
@end
```

Only a few uses for this at present

Substitution functions

- In addition to $() substitution, there are substitution functions

```
$FUNCTION_NAME(key [,arg1,...])
```

  - Function names are all upper case
  - Some functions have arguments
  - Some arguments are optional

(see section 3.3.10 in the manual)

Environment substitutions

`$ENV(`*`name`*`[:`*`default`*`])`

- Substitute with the value of environment variable *name*
- If *name* does not exist, substitute with UNDEFINED or *default* (if specified)
- To substitute with nothing if *name* does not exist use **`$ENV(`*`name`*`:)`**

(Some) Substitution functions

$INT(*key*[,*format*])

$REAL(*key*[,*format*])

- Evaluate value of *key* and printf
- optional *format* is **everything** after the comma

$CHOICE(*key*,*list*)

$CHOICE(*key*,*item*,*item*,*item*)

- Evaluate *key* as index into item list

$F[pdnxwuqa](*key_or_value*)

- Extract filename parts and strip/add quotes

No $ inside a $FUNC()

```
A = 1
# This will not work (parse error)
tot = $INT($(A) + 1)
# But this will work
A_PLUS = $(A) + 1
tot = $INT(A_PLUS)
```

Statements

- In statements $ substitutions happen as the file is read - the **current** value is used
- Statements are
  - Include
  - Use (but not metaknob arguments)
  - Conditionals
  - Queue
  - Error/Warning

Include statements

- Include : <file>
  - read <file>, abort if it cannot be read
- Include ifexist : <file>
  - read <file> if it exists
- Include command : <script> <args>
  - run <script> and include its stdout as part of config/submit

(Remember: $() substitutions will use current value here)

Example of Include

```
# assume HOSTNAME is cheese
LOCAL_DIR = /home/bob
FILE = config.$(HOSTNAME)
Include : $(LOCAL_DIR)/$(FILE)
FILE = $(LOCAL_DIR)/script.cmd
Include command : $(FILE) $(HOSTNAME)
```

- HTCondor 8.2+ will do this
  - Include **`/home/bob/config.cheese`**
  - run **`/home/bob/script.cmd cheese`**

Digression - Backward Include

```
FILE = config.$(HOSTNAME)
Include : $(LOCAL_DIR)/$(FILE)
FILE = $(LOCAL_DIR)/script.cmd
Include command : $(FILE) $(HOSTNAME)
Foo = bar
```

- HTCondor 8.0 and earlier sees this as just key = value statements, as if it were

```
FILE = config.$(HOSTNAME)
Include = $(LOCAL_DIR)/$(FILE)
FILE = $(LOCAL_DIR)/script.cmd
Include = $(FILE) $(HOSTNAME)
Foo = bar
```

Use Include Carefully!

- Every daemon and every tool will
  - Read every config file
  - Run every config script (if any)
- Sometimes several tools at the same time!
  - Scripts should have NO side effects
- Config is read as root on startup but as condor on reconfig
  - All config files should be owned by root
  - World readable, root (only) writable

Include with cache (8.6+)

`Include command into <file> : <script> <args>`

- read <file> if it exists

  otherwise

- run <script> and write output into <file>

› For config <file> should be absolute path

› <file> must be deleted by hand

- Useful mostly for submit and configurations that get thrown away after one use (glide-in, annex)

conditionals

- **`If, Elif`** support only basic conditionals
  - **`[!] <boolean-or-number>`**
  - **`[!] defined <key>`**
  - **`[!] version [><=]= x.y[.z]`**
- No comparison or complex conditionals
  - **`If version`** is a special case
  - $INT() is a workaround
- Empty conditional is false, not an error

Special macros for If

- For config files only, these “knobs” are set based on who is reading config

```
$(IsMaster)
$(IsNegotiator)
$(IsSchedd)
$(IsShadow)
$(IsStartd)
$(IsStarter)
$(IsTool)
$(IsWindows)
```

Examples of If / Else

```
# useful in temporary HTCondor config
If $(IsMaster)
  include command into $(cache) : $(script)
else
  include ifexist $(cache)
endif

# useful in submit
if version >= 8.7.10
  materialize_max_idle = 100
endif
```

If tricks (8.4 or later)

```
HAVE_SCHEDD_DAEMON = \
   stringListMember("SCHEDD","$(DAEMON_LIST)")

If $INT(HAVE_SCHEDD_DAEMON)
   MASTER_NEW_BINARY_RESTART = FAST
else
   MASTER_NEW_BINARY_RESTART = GRACEFUL
endif
```

Gotcha

`If` and **`include`** will use the current value for $() expansion.

So the previous example only works if it is after the last DAEMON_LIST assignment in your config

Prefixes (a.k.a. daemon overrides)

- Config/submit key can have prefixes

```
SCHEDD.COLLECTOR_HOST = 192.168.100.2
FRED.COLLECTOR_HOST = 192.168.100.3
MY.CUSTOM_ATTRIBUTE = "custom value"
```

- Prefixed knobs are stored but ignored except
  - Daemons will use prefixed knobs if the prefix is their localname or their subsystem name
  - Submit will treat the MY. prefix as a direct assignment into the job classad

Prefix example for config

```
# this is how HTCondor implements the IsMaster macro
#
IsMaster = false
MASTER.IsMaster = true
if $(IsMaster)
  # config statements only the condor_master
  # will parse
endif
```

Gotcha

- condor_config_val output can differ from what the daemon sees if you use the $(IsXXX) macros. You would need to use

```
condor_config_val -daemon
   or
condor_config_val -subsys daemon
```

To see the effective config for a daemon

Prefix example for submit

```
# submit file
Executable = process_data
transfer_input_files = $(DATA)
arguments = $Fqanx(DATA)
# store the input filename into the job classad
MY.DataFile = $Fqnx(DATA)
Queue DATA matching *.dat
```

- To see which job is processing which datafile use:

```
condor_q -af:jh DataFile
```

Use statements (aka meta-knobs)

- Use *category* : *Name* [*Name2*]
  - Like a pre-defined include

```
use ROLE : Personal
```

- Use [POLICY | FEATURE] : *Name*(*args*)
  - Some POLICY and FEATURE meta-knobs accept arguments

```
use FEATURE : PartitionableSlot(1,80%)
```

Currently no use in submit files

Explore the config meta-knobs

- Categories are currently

  `ROLE, FEATURE, POLICY, SECURITY`

- Find out what meta-knobs exist with

  `condor_config_val use` ***`category`***

- Examine contents of a meta-knob with

  `condor_config_val use` ***`category`*****`:`**********`name`**

condor_config_val tricks

`condor_config_val -schedd -verbose`

- Ask the Schedd about it's config

`condor_config_val -subsys schedd -verbose`

- Parse the config as the schedd would

`condor_config_val -writeconfig:upgrade -`

- Write an 'upgrade' file containing *only* the knobs that you've changed

Any Questions?