hcron Guide - v0.21 - 20180810

For hcron v0.21. See here for all releases.

Contents

Introduction

cron, the ubiquitous periodic scheduler, is one of the workhorses in the UNIX toolbox. There is no need to produce yet another cron ... unless it brings something new to the table. hcron does just that in a number of really useful and practical ways. For example:

- events are stored individually, each in their own file, rather than all in a single file
- events are organized hierarchically in the filesystem rather than as a table in a single file
- events are named and referenceable
- events are defined as multiple key=value settings rather than a ordered columns on a single line
- hcron is network oriented rather than system oriented
- support for template events to reducing and reuse settings
- support for failover events if an event cannot be spawned
- support for settings and working with variables

hcron is enterprise-ready having proven itself over many years of use in development and operational environments, by hundreds of users, with tens of thousands of events being scheduled each day.

User Configuration

hcron uses `ssh` to launch events across the network. As such, it requires that the user ssh configuration be set up to **not** require user interaction to run (e.g., no password, no passphrase, or yes/no responses).

This is done by setting the ssh key:

```
ssh-keygen
cd ~/.ssh
cat id_dsa.pub id_rsa.pub >> authorized_keys
```

and the ssh config file:

~/.ssh/config

```
Host *.<domain>
    StrictHostKeyChecking no
    BatchMode yes
```

If full hostnames (i.e., which include the domain) are not used, then it may be necessary to add the following, too:

~/.ssh/config

```
Host *
    StrictHostKeyChecking no
    BatchMode yes
```

Using `Host *` applies to all hosts and may pose a security problem.

If host keys change frequently, you may also benefit from:

~/.ssh/config

```
UserKnownHostsFile /dev/null
```

For more, see man page for `ssh_config`.

Commands

All operations are carried out with:

- `hcron-conv` - tool to convert between a `crontab` file and hcron event files
- `hcron-event` - create and modify event files
- `hcron-info` - show hcron-related information on the hcron scheduler machine; must be executed on the hcron scheduler machine
- `hcron-reload` - generate snapshot of event files and trigger hcron scheduler to reload; must be executed on the hcron scheduler machine
- `hcron-run` - Simulate scheduling of events

Events

Event File

Each event is defined in its own file. An event file contains key=value lines, empty lines, or comment lines (starting with #).

Using `hcron-event`, an empty event file will be created as:

```
as_user=
host=
command=
notify_email=
notify_message=
when_month=
when_day=
when_hour=
when_minute=
when_dow=
```

where:

Key	Description
as_user	(optional) username to use to when launching event. Must not require interaction.
host	Host on which to launch the command.
command	Command to run on the host (with minimal environment).
notify_email	Email address to send notification to on successful launch.
notify_message	Message in email when sent.
when_month	Month(s) the event should be scheduled. January is month 1.
when_day	Days of the month the event should be scheduled. The range is variable and depends on the month.
when_hour	Hour (0-23) of the day the event should be scheduled.
when_minute	Minute (0-59) of the hour the event should be scheduled.
when_dow	Day (0-6) of the week the event should be scheduled. Sunday is day 0.

The when_* fields support the standard cron wildcard (*), ranges (e.g., 1-3), steps (e.g., 0-23/2), lists (e.g., 0,2,6,23).

Additional settings:

Key	Description
failover_event	(optional) Event to execute if the event is not successfully launched.
next_event	(optional) Next event to execute if the event is successfully launched.
notify_subject	(optional) Subject in email when sent. hcron provides a default when not specified.
template_name	(optional) Event is ignored if the value matches the last component of the event name.
when_year	(optional) Year (2000-2050) the event should be scheduled. Default is *.

Examples

Email reminder of weekly meeting on Mondays:

/monday_meetings_reminder

```
as_user=
host=abc.xyz
command=/bin/true
notify_email=John.Smith@abc.xyz
notify_message=Group meeting at 9am
when_month=*
when_day=*
when_hour=0
when_minute=0
when_dow=1
```

Rotate logs daily at 00:05:

/rotatelogs

```
as_user=
host=abc.xyz
command=rotatelogs /var/log/syslog /var/log/messages
notify_email=
notify_message=
when_month=*
when_day=*
when_hour=0
when_minute=5
when_dow=*
```

Augmenting command execution environment

By default, the event command is executed in a minimal environment. To augment it, set environment variables or source profile scripts. E.g., for bash,

```
command=. ~/.profile; /bin/true
```

or, to have a full login environment loaded (highly discouraged):

```
command=bash -l -c "/bin/true"
```

Event Tree

Event files are organized in a event tree under the user home directory:

```
~/.hcron/
    <hcronfqdn>/
        events/
            ...
```

where <hcronfqdn> is the fully qualified host name of the machine running the hcron scheduler. Run "`hcron-info --fqdn`" on the machine running the hcron-scheduler to get the correct value.

Make sure that the <hcronfqdn> directory your events files are under matches the host name of the machine running the hcron scheduler. Otherwise, they will not be picked up by `hcron-reload` and the snapshot will fail.

Event files under `events/` should be given meaningful names and may be organized using directories. Directory names will be part of the full event name. If symlinks are used, they must reference only items under the same `events/` tree and use relative paths.

Examples

Rotate logs on 3 machines:

```
~/.hcron/hcron.xyz/events/
    rotatelogs/
        mach1.xyz
        mach2.xyz
        mach3.xyz
```

where the event files `rotatelogs/mach1.xyz`, `rotatelogs/mach2.xyz`, `rotatelogs/mach3.xyz` use the event definition above but with the following modifications, respectively:

/rotatelogs/mach1 (snippet)

```
host=mach1.xyz
```

/rotatelogs/mach2 (snippet)

```
host=mach2.xyz
```

/rotatelogs/mach3 (snippet)

```
host=mach3.xyz
```

Ignored Names

Depending on the configuration (see `names_to_ignore_regexp`), some filenames may be ignored.

The default setting is:

```
(\..*)|(.*~$)
```

- all names starting with "`.`" (hidden files; gvim temporary files)
- all names ending with "~" (commonly used for emacs temporary files)

Variables

hcron supports variables for use in event files. Some variables are provided by the system, others are defined by the user. Some are substituted early while others late.

Early substitution means that they are substituted when the event is loaded. Late substitution is done just before the event is executed.

All variables are of string type. They are specified as `name=value` and processed in order

When referencing variables, they must be prefixed by an operator:

- $ - value of (e.g., `$HCRON_EVENT_NAME`)
- # - count of (e.g., `#HCRON_EVENT_NAME`)

System Variables

All system variables start with the prefix `HCRON_` and is reserved.

Early substitution variables:

Name	Description
`HCRON_HOST_NAME`	The fully qualified hostname running the hcron scheduler.
`HCRON_EVENT_NAME`	The full event name relative to the `events/` directory. Always starts with `/`.

Late substitution variables:

Name	Description
`HCRON_ACTIVATE_DATETIME`	Local date and time of the event activation. Format: `YYYY:MM:DD:hh:mm:ss:WOY:DOW`.
`HCRON_ACTIVATE_DATETIME_UTC`	Same as `HCRON_ACTIVATE_DATETIME` in UTC (adjusted for local timezone offset).
`HCRON_EVENT_CHAIN`	Colon-separated list of event names executed in an event chain by `next_event` or `failover_event`. `HCRON_EVENT_CHAIN[0]` is the initial event executed.
`HCRON_SCHEDULE_DATETIME`	Local date and time of the event schedule. Format: `YYYY:MM:DD:hh:mm:ss:WOY:DOW`.
`HCRON_SCHEDULE_DATETIME_UTC`	Same as `HCRON_SCHEDULE_DATETIME` in UTC (adjust for local timezone offset).
`HCRON_SELF_CHAIN`	Colon-separated list of consecutive chained events to self by `next_event` or `failover_event`.

The activation time is when the event is activated/executed. The schedule time is when the event is scheduled based on the `when_*` settings. The two are the same except when hcron is behind schedule (for whatever reason), which is rare.

Split/Join Operator

Given a string, splitting will return a list of items. Joining will put a list of items together giving a string.

The split and join operators are:

```
<split_sep>!
<split_sep>?<join_sep>!
```

where:

- `<split_sep>` is the separator used to split the string
- `<join_sep>` is the separator used to join the items

The default split and join separators are ":" if not specified; the join separator is the same as the split separator if not specified. The separators may be zero or more characters.

The only exception to this rule is when working with the `HCRON_EVENT_NAME` variable for which "/" is the default separator.

Examples

Given:

```
H_LETTERS=a:b:c
H_TENS=10_20_30_40
H_MACHS=mach1:mach2:mach3
H_DIGITS=1234
```

we get:

Operation	Split	Join
`$H_LETTERS[:!]`	"a", "b", "c"	a:b:c
`$H_TENS[_!]`	"10", "20", "30", "40"	10_20_30_40
`$H_MACHS[:!]`	"mach1", "mach2", "mach3"	mach1:mach2:mach3
`$H_TENS[_?:!]`	"10", "20", "30", "40"	10:20:30:40
`$H_LETTERS[:?_!]`	"a", "b", "c"	a_b_c
`$H_DIGITS[?-!]`	"1", "2", "3", "4"	1-2-3-4

A split operation is **always** accompanied by a join operation. Recall that variables are of string types only and so the intermediate "list" produced by a split operation is always joined.

Indexing

Given the results of a split operation, the index operator can be used to extract items to be joined.

Indexing takes the forms:

- single index: positive or negative
- range: `<start>:<end>`
- step: `<start>:<end>:<step>`
- multiple: comma-separated single, range, and step indexes

When using indexing shorthand, if `<start>`, `<end>`, and/or `<step>` are not specified, the values default to 0, the number of elements, and 1, respectively, unless the step is negative in which case the `<start>`, `<end>` values are reversed. See Python for more on indexing/slicing which is followed here.

Examples

Given the settings above:

Operation	Split	Extraction	Join
`$H_LETTERS[1]`	"a", "b", "c"	"b"	b
`$H_LETTERS[0,2]`	"a", "b", "c"	"a", "c"	a:c
`$H_TENS[_!::2]`	"10", "20", "30", "40"	"10", "30"	10_30
`$H_TENS[_!-1]`	"10", "20", "30", "40"	"40"	40

`$H_DIGITS[?-!::-1]`	"1", "2", "3", "4"	"4", "3", "2", "1"	4-3-2-1

Counting

Rather than using the value of a variable, the count can be obtained.

Examples

Give the settings above:

Operation	Split	Extraction	Count
`#H_LETTERS[1]`	"a", "b", "c"	"b"	1
`#H_LETTERS[0,2]`	"a", "b", "c"	"a", "c"	2
`#H_TENS[_!::2]`	"10", "20", "30", "40"	"10", "30"	2
`#H_TENS[_!-1]`	"10", "20", "30", "40"	"40"	1
`#H_DIGITS[?-!::-1]`	"1", "2", "3", "4"	"4", "3", "2", "1"	4

Combined Operations

Other than what has already been described, operations cannot be combined such as:

```
$H_LETTERS[0,2][1]
```

but, instead, must be carried out with multiple steps:

```
H_X=$H_LETTERS[0,2]
H_X=$H_X[1]
```

In most cases, this is not much of an issue and will add only a few extra lines.

Advanced Events

include Directive

As the number of event files grows, it is sometimes helpful to be able to reuse the contents of an existing event file. This can be achieved using the include directive:

```
include <event_name>
```

The include directive is resolved when the including event is loaded prior to early substitution.

Included event files need not be fully specified (they will be rejected if not). In fact, included files will often contain variable settings alone.

Example

Given:

/vars

```
H_X=hello
H_Y=world
```

and:

event (before include)

```
include /vars
as_user=...
host=...
...
```

we get:

event (after include)

```
H_X=hello
H_Y=world
as_user=...
host=...
...
```

Templates

When many similar events must be defined and the differences can be handled using variables, a common event file, a template, can be used. A template is like any other event file except for the field `template_name`. If, when hcron loads an event file, the `template_name` field is set and its value matches the event file, hcron will recognize it as a template and ignore it. To use the template, symlinks are created which point to it.

Example

Give the template file:

/rotatelogs/template

```
as_user=
host=abc.xyz
command=rotatelogs /var/log/$HCRON_EVENT_NAME[-1]
notify_email=
notify_message=
when_month=*
when_day=*
when_hour=0
when_minute=5
when_dow=*
template_name=template
```

we can set up some symlinks pointing to the template file:

```
.../events/
  rotatelogs/
    template
    auth.log -> template
    messages -> template
    syslog -> template
```

where three non-template events are defined (using only symlinks) with specialized `command` settings so that `/var/log/auth.log`, `/var/log/messages`, and `/var/log/syslog` will be rotated. The `HCRON_EVENT_NAME` is used to specialize the template to specify the log file in the `command` field but could be used to set other fields, also.

Event Chaining

When an event is scheduled, it becomes the first event in an event chain. If no other events are configured to be called afterward, then the chain remains with one event. Otherwise another event is called during the same scheduling period. Event chains are configured using `next_event` (successful launch) and `failover_event` (failed event launch), each of which can specify their own `next_event` and `failover_event` settings.

The event chain is available in the `HCRON_EVENT_CHAIN` variable, and the current event in `HCRON_SELF_EVENT`. The maximum number of chained events is determined by the server configuration.

Example

Given a set of event files:

```
.../events/
  one
  two
  three
  fail
```

where event `one` should call `two` which should call `three` and `fail` only if the others failed to launch. The respective events files would be (snippets only):

/one (snippet)

```
next_event=two
failover_event=fail
```

/two (snippet)

```
next_event=three
failover_event=fail
```

/three (snippet)

```
failover_event=fail
```

and the failover event:

/fail (snippet)

```
notify_subject=fail
```

Monitoring

Log File

All actions taken by the hcron scheduler are logged (e.g., to `/var/log/hcron/hcron.log`).

Tag	Description and Format
`alarm`	Alarm triggered because an execute operation has timed out. `<datestamp>|alarm||<msg>`
`chain-events`	Indicates that a `next_event` or `failover_event` is being followed. `<datestamp>|chain-events|<username>|<eventname>|<nexteventname>|<nextEventType>|<eventchainnames>|<cyclemsg>` Where: • <cyclemsg> is "cycle" or an empty string • <nextEventType> is "failover" or "next"

`discard-events`	Discard user event information. `<datestamp>|discard-events|<username>|<count>`
`execute`	Event `command` is being executed. `status` of 0 for success, -1 for failure (usually caused by bad/insufficient ssh configuration). `<datestamp>|execute|<username>|<as_user>|<host>|<eventname>|<pid>|<spawn_elapsed>|<status>`
`exit`	Server is exiting. `<datestamp>|exit`
`load-allow`	(Re)load the `hcron.allow` file. `<datestamp>|load-allow`
`load-config`	(Re)load the `hcron.config` file. `<datestamp>|load-config`
`load-events`	Load user events. `<datestamp>|load-events|<username>|<count>|<elapsed>`
`notify-email`	Sent email for event `<datestamp>|notify-email|<username>|<address>|<eventname>`
`sleep`	Sleep time between subsequent work periods. `<datestamp>|sleep||<seconds>`
`start`	Server has started. `<datestamp>|start`
`work`	Statistics for work period. `<datestamp>|work||<count>|<elapsed>`

Event Info

When the scheduler loads an event tree snapshot, it writes the load status of each event to a file at `/var/lib/hcron/event_lists/<user_name>`:

- `accepted::<event_name>`
- `rejected:<reason>:<event_name>`

Reasons for a rejected event are:

- bad definition - something is wrong with the event defintion
- template - event name matches the value in `template_name`
- ignored - the event name matches the ignored event names regular expression (see `names_to_ignore_regexp`).

Use "`hcron-info -es`" to get this information.

Scheduler Configuration

hcron.allow

Users are **not** allowed to use hcron by default. Instead, each user must be added to the `/etc/hcron/hcron.allow`, one username per line. Comment lines must start with #.

hcron.conf

All other scheduler configuration is done in the `/etc/hcron/hcron.conf` file which takes the form of a Python dictionary. The keys of the configuration file are:

Key	Description
`allow_localhost`	Boolean indicating whether the local host may be targeted to run commands. The scheduler machine does not usually run commands in order to avoid overloading.
`allow_root_events`	Boolean indicating whether events belonging to root will be run. Default is `False`.
`command_spawn_timeout`	Time (in seconds) allowed for a command to successfully spawn.
`events_base_path`	Base path for hcron events. Default is `~<username>/.hcron/<hostname>/events`.
`log_path`	Path to the log file. A relative path is anchored at `/var/log/hcron`.
`max_chain_events`	Maximum number of chained events allowed. Prevents against infinite recursion.
`max_events_per_user`	Maximum number of events a user may have.
`max_hcron_tree_snapshot_size`	Maximum size of the hcron event tree snapshot created at `hcron-reload`. hcron working space under `/var/lib/hcron` must be protected against exhaustion.
`names_to_ignore_regexp`	Regular expression used to determine event names to ignore.
`smtp_server`	SMTP server to use for sending email notifications.
`test_net_delay`	Delay between retry for obtaining information for `test_net_username`.
`test_net_retry`	Number of retries to obtaining information for `test_net_username`.
`test_net_username`	Username used to test if credentials information is available/working.
`use_syslog`	Boolean indicating whether to use syslog or the log file.