# Effectively Using Failover Events

**Contents**



## Introduction

When an event fails to be launched, hcron can trigger an event if it is listed in the `failover_event` field. Failover events are typically not intended to be run in general, although they can.

In the examples below, we will see how such events can be organized and used.

## Simple Case

We first need an event that will be guaranteed to fail:

**simple**

```
as_user=
host=thishostdoesnotexist
command=/bin/true
notify_email=
notify_message=
when_month=*
when_day=*
when_hour=*
when_minute=*
when_dow=*
template_name=
failover_event=failover_simple
```

Notes:

- this event should always fail because connecting to `thishostdoesnotexist` should never succeed
- hcron will attempt to run this event every minute
- if it fails, hcron will try to run `/failover_simple`

Next, we provide the failover event:

**failover_simple**

```
as_user=
host=localhost
command=/bin/true
notify_email=<email address>
notify_message=Failed event ($HCRON_EVENT_CHAIN[0])
when_month=*
when_day=*
when_hour=*
when_minute=*
when_dow=*
template_name=failover_simple
failover_event=
```

Notes:

- run the event on the localhost (change the `host` or make sure that localhost is allowed to run hcron events)
- sends email notification to address at `notify_email`
- email notification message indicates failure and identifies the failed event (`$HCRON_EVENT_CHAIN[0]`)
- `template_name` is set so that hcron ignores/rejects the event and does not try to run it

Reload the events and wait:

```
hcron-reload
```

Check the status of the events:

```
hcron-info -es
```

The events should be marked as:

```
accepted::/simple
rejected:template:/failover_simple
```

The notification email body will look like:

```
Failed event (/simple)
```

# A Common Failover Event

Instead of defining multiple failover events, a common failover event could be defined for all other events to reference. It could be defined at `/failover/common`:

**/failover/common**

```
as_user=
host=localhost
command=/bin/true
notify_email=<email address>
notify_message=Failed event ($HCRON_EVENT_CHAIN[0])
when_month=*
when_day=*
when_hour=*
when_minute=*
when_dow=*
template_name=common
failover_event=
```

Notes:

- the `template_name` is updated to `common` to make sure it is treated as a template

And all regular events would contain:

```
failover_event=/failover/common
```

# Targetted Failover Events

When a lot of events are defined, it may not be appropriate to use a single, common failover event. E.g., a common failover can only notify one email address via the `notify_email` field. Perhaps failures should be reported to different individuals/groups depending on the failed event. To support this, we can create multiple failover events. But, this would mean managing multiple event files. So, instead, we will create a template and symlinks to identify the various email recipients.

The template is very familiar:

**/failover/email_target_template**

```
as_user=
host=localhost
command=/bin/true
notify_email=$HCRON_EVENT_NAME[-1]
notify_message=Failed event ($HCRON_EVENT_CHAIN[0])
when_month=*
when_day=*
when_hour=*
when_minute=*
when_dow=*
template_name=email_target_template
failover_event=
```

Notes:

- the event name is `email_target_template`
- the `template_name` field is set to `email_target_template` so that it is treated as a template
- the `notify_email` field is set to `$HCRON_EVENT_NAME[-1]` which is the name of the event

So, for each target we want to support, we create symlinks:

```
$ ln -s email_target_template bob@abc.xyz
$ ln -s email_target_template pager@abc.xyz
$ ln -s email_target_template webmaster@abc.xyz
```

Then, in the various regular event files, we use:

```
failover_event=/failover/bob@abc.xyz
```

or:

```
failover_event=/failover/pager@abc.xyz
```

or:

```
failover_event=/failover/webmaster@abc.xyz
```

and so on.

## Conclusion

There are many different ways to set up failover events. The level of complexity depends on the need. But being able to set up failover events with such flexibility and then have them be triggered as needed makes for a documentable and traceable way to manage a large number of relationships easily.