NAME

init − Upstart init daemon job configuration

SYNOPSIS

/etc/init/

DESCRIPTION

On startup, the Upstart **init**(8) daemon reads its job configuration from the */etc/init* directory, and watches for future changes using **inotify**(7).

To be considered by Upstart, files in this directory must have a recognized suffix and may also be present in sub-directories. There are two recognized suffixes:

- Files ending in *.conf* are called configuration files, or simply "conf files" for short. These are the primary vehicle for specifying a job.
- Files ending in *.override* are called override files. If an override file is present, the stanzas it contains take precedence over those equivalently named stanzas in the corresponding configuration file contents for a particular job. The main use for override files is to modify how a job will run without having to modify its configuration file directly. See the section **Override File Handling** below for further details.

A job can thus be defined by either:

- A single configuration file.
- A single configuration file **and** a single override file.

Unless explicitly stated otherwise, any reference to a jobs configuration can refer both to a configuration file or an override file.

Each configuration file defines the template for a single *service* (long-running process or daemon) or *task* (short-lived process).

Note that a configuration file is not itself a job: it is a description of an environmenta job could be run in. A job is the runtime embodiment of a configuration file.

The configuration file name as displayed by Upstart and associated tooling is taken from its relative path within the directory without the extension. For example a configuration file */etc/init/rc-sysinit.conf* is named *rc-sysinit*, while a configuration file */etc/init/net/apache.conf* is named *net/apache*. Since override files only modify the way a configuration file is interpreted, they are not named.

Configuration files are plain text and should not be executable.

Configuration File Format

Each line begins with a configuration stanza and continues until either the end of the line or a line containing a closing stanza. Line breaks within a stanza are permitted within single or double quotes, or if preceeded by a blackslash.

If a stanza is duplicated, the last occurence will be used. Unrecognized stanzas will generate parse errors, which will stop a job from running.

Stanzas and their arguments are delimited by whitespace, which consists of one or more space or tab characters which are otherwise ignored unless placed within single or double quotes.

Comments begin with a ‘#’ and continue until the end of the line. Blank lines and lines consisting only of whitespace or comments are ignored.

Process definition

The primary use of jobs is to define services or tasks to be run by the **init**(8) daemon. Each job may have one or more different processes run as part of its lifecycle, with the most common known as the main process.

The main process is defined using either the **exec** or **script** stanzas, only one of which is permitted. These specify the executable or shell script that will be run when the job is considered to be running. Once this process terminates, the job stop.

All processes are run with the full job environment available as environment variables in their process.

exec *COMMAND* [*ARG*]...

This stanza defines the process to be run as the name of an executable on the filesystem, and zero or more arguments to be passed to it. Any special characters, e.g. quotes or '$' specified will result in the entire command being passed to a shell for expansion.

```
exec /usr/sbin/acpid -c $EVENTSDIR -s $SOCKET
```

script ... **end script**

This stanza defines the process to be run as a shell script that will be executed using **sh(1).** The *-e* shell option is always used, so any command that fails will terminate the script.

The **script** stanza appears on its own on a line, the script is everything up until the first **end script** stanza appearing on its own on a line.

```
script
  dd bs=1 if=/proc/kmsg of=$KMSGSINK
  exec /sbin/klogd -P $KMSGSINK
end script
```

There an additional four processes that may be run as part of the job's lifecycle. These are specified as the process name, followed by an **exec** or **script** stanza.

pre-start exec|**script**...

This process will be run after the job's **starting**(7) event has finished, but before the main process is run. It is typically used to prepare the environment, such as making necessary directories.

post-start exec|**script**...

This process will be run before the job's **started**(7) event is emitted, but after the main process has been spawned. It is typically used to send necessary commands to the main process, or to delay the **started**(7) event until the main process is ready to receive clients.

pre-stop exec|**script**...

This process is run if the job is stopped by an event listed in its **stop on** stanza or by the **stop**(8) command. It will be run before the job's **stopping**(7) event is emitted and before the main process is killed. It is typically used to send any necessary shutdown commands to the main process, and it may also call the **start**(8) command without arguments to cancel the stop.

post-stop exec|**script**...

This process is run after the main process has been killed and before the job's **stopped**(7) event is emitted. It is typically used to clean up the environment, such as removing temporary directories.

All of these process, including the main process, are optional. Services without a main process will appear to be running until they are stopped, this is commonly used to define states such as runlevels. It's quite permissable to have no main process, but to have **pre-start** and **post-stop** processes for the state.

pre-start exec ifup -a
post-stop exec ifdown -a

Event definition

Jobs can be manually started and stopped at any time by a system adminstrator using the. **start**(8) and **stop**(8) tools, however it is far more useful for jobs to be started and stopped automatically by the **init**(8) daemon when necessary.

This is done by specifying which events should cause your job to be started, and which cause your process to be stopped again.

The set of possible events is limitless, however there are a number of standard events defined by the **init**(8) daemon and **telinit**(8) tools that you will want to use.

When first started, the **init**(8) daemon will emit the **startup**(7) event. This will activate jobs that implement System V compatibility and the **runlevel**(7) event. As jobs are started and stopped, the **init**(8) daemon will emit the **starting**(7), **started**(7), **stopping**(7) and **stopped**(7) events on their behalf.

start on *EVENT* [[*KEY*=]*VALUE*]... [**and**|**or**...]

The **start on** stanza defines the set of events that will cause the job to be automatically started. Each *EVENT* is given by its name. Multiple events are permitted using the **and** & **or** operators, and complex expressions may be performed with parentheses (within which line breaks are permitted).

You may also match on the environment variables contained within the event by specifying the *KEY* and expected *VALUE*. If you know the order in which the variables are given to the event you may omit the *KEY*.

VALUE may contain wildcard matches and globs as permitted by **fnmatch**(3) and may expand the value of any variable defined with the **env** stanza.

Negation is permitted by using *!=* between the *KEY* and *VALUE*.

start on started gdm or started kdm

start on device-added SUBSYSTEM=tty DEVPATH=ttyS*

start on net-device-added INTERFACE!=lo

stop on *EVENT* [[*KEY*=]*VALUE*]... [**and**|**or**...]

The **stop on** stanza defines the set of events that will cause the job to be automatically stopped. It has the same syntax as **start on**.

VALUE may additionally expand the value of any variable that came from the job's start environment (either the event or the command that started it).

stop on stopping gdm or stopping kdm

stop on device-removed DEVPATH=$DEVPATH

Job environment

Each job is run with the environment from the events or commands that started it. In addition, you may define defaults in the job which may be overridden later and specify which environment variables are exported into the events generated for the job.

The special **UPSTART_EVENTS** environment variable contains the list of events that started the job, it

will not be present if the job was started manually.

In addition, the **pre-stop** and **post-stop** scripts are run with the environment of the events or commands that stopped the job. THe **UPSTART_STOP_EVENTS** environment variable contains the list of events that stopped the job, it will not be present if the job was stopped manually.

All jobs also contain the **UPSTART_JOB** and **UPSTART_INSTANCE** environment variables, containing the name of the job and instance. These are mostly used by the **initctl**(8) utility to default to acting on the job the commands are called from.

env *KEY=VALUE*
Defines a default environment variable, the value of which may be overriden by the event or command that starts the job.

export *KEY*
Exports the value of an environment variable into the **starting**(7), **started**(7), **stopping**(7) and **stopped**(7) events for this job.

Services, tasks and respawning

Jobs are *services* by default. This means that the act of starting the job is considered to be finished when the job is running, and that even exiting with a zero exit status means the service will be respawned.

task This stanza may be used to specify that the job is a *task* instead. This means that the act of starting the job is not considered to be finished until the job itself has been run and stopped again, but that existing with a zero exit status means the task has completed successfully and will not be respawned.

The **start**(8) command, and any **starting**(7) or **stopping**(7) events will block only until a service is running or until a task has finished.

respawn
A service or task with this stanza will be automatically started if it should stop abnormally. All reasons for a service stopping, except the **stop**(8) command itself, are considered abnormal. Tasks may exit with a zero exit status to prevent being respawned.

respawn limit *COUNT INTERVAL*
Respawning is subject to a limit, if the job is respawned more than *COUNT* times in *INTERVAL* seconds, it will be considered to be having deeper problems and will be stopped. Default COUNT is 10. Default INTERVAL is 5 seconds.

This only applies to automatic respawns and not the **restart**(8) command.

normal exit *STATUS|SIGNAL...*
Additional exit statuses or even signals may be added, if the job process terminates with any of these it will not be considered to have failed and will not be respawned.

normal exit 0 1 TERM HUP

Instances

By default, only one instance of any job is permitted to exist at one time. Attempting to start a job when it's already starting or running results in an error. Note that a job is considered to be running if its pre-start process is running.

Multiple instances may be permitted by defining the names of those instances. If an instance with the same name is not already starting or running, a new instance will be started instead of returning an error.

instance *NAME*

This stanza defines the names of instances, on its own its not particularly useful since it would just define the name of the single permitted instance, however *NAME* expands any variable defined in the job's environment.

These will often be variables that you need to pass to the process anyway, so are an excellent way to limit the instances.

```
instance $CONFFILE
exec /sbin/httpd -c $CONFFILE
```

```
instance $TTY
exec /sbin/getty -8 38300 $TTY
```

These jobs appear in the **initctl**(8) output with the instance name in parentheses, and have the **INSTANCE** environment variable set in their events.

Documentation

Upstart provides several stanzas useful for documentation and external tools.

description *DESCRIPTION*

This stanza may contain a description of the job.

```
description "This does neat stuff"
```

author *AUTHOR*

This stanza may contain the author of the job, often used as a contact for bug reports.

```
author "Scott James Remnant <scott@netsplit.com>"
```

version *VERSION*

This stanza may contain version information about the job, such as revision control or package version number. It is not used or interpreted by **init**(8) in any way.

```
version "$Id$"
```

emits *EVENT...*

All processes on the system are free to emit their own events by using the **initctl**(8) tool, or by communicating directly with the **init**(8) daemon.

This stanza allows a job to document in its job configuration what events it emits itself, and may be useful for graphing possible transitions.

usage *USAGE*

This stanza may contain the text used by **initctl**(8) **usage** command. This text may be also shown when commands **start**(8), **stop**(8) or **status**(8) fail.

```
usage "tty DEV=ttyX - where X is console id"
```

Process environment

Many common adjustments to the process environment, such as resource limits, may be configured directly in the job rather than having to handle them yourself.

console output|**owner**

By default the standard input, output and error file descriptors of jobs are connected to */dev/null*

If this stanza is specified, they are connected to */dev/console* instead.

console owner is special, it not only connects the job to the system console but sets the job to be the owner of the system console, which means it will receive certain signals from the kernel when special key combinations such as Control-C are pressed.

umask *UMASK*
A common configuration is to set the file mode creation mask for the process. *UMASK* should be an octal value for the mask, see **umask**(2) for more details.

nice *NICE*
Another common configuration is to adjust the process's nice value, see **nice**(1) for more details.

oom *ADJUSTMENT*|**never**
Normally the OOM killer regards all processes equally, this stanza advises the kernel to treat this job differently.

ADJUSTMENT may be an integer value from *-16* (very unlikely to be killed by the OOM killer) up to *14* (very likely to be killed by the OOM killer). It may also be the special value **never** to have the job ignored by the OOM killer entirely.

chroot *DIR*
Runs the job's processes in a **chroot(8)** environment underneath *DIR*

Note that *DIR* must have all the necessary system libraries for the process to be run, often including */bin/sh*

chdir *DIR*
Runs the job's processes with a working directory of *DIR* instead of the root of the filesystem.

limit *LIMIT SOFT*|**unlimited** *HARD*|**unlimited**
Sets initial system resource limits for the job's processes. *LIMIT* may be one of *core*, *cpu*, *data*, *fsize*, *memlock*, *msgqueue*, *nice*, *nofile*, *nproc*, *rss*, *rtprio*, *sigpending* or *stack*.

Limits are specified as both a *SOFT* value and a *HARD* value, both of which are integers. The special value **unlimited** may be specified for either.

Override File Handling
Override files allow a jobs environment to be changed without modifying the jobs configuration file. Rules governing override files:

- If a job is embodied with only a configuration file, the contents of this file define the job.
- If an override files exists where there is no existing cofiguration file, the override file is ignored.
- If both a configuration file **and** an override file exist for a job and both files are syntactically correct:
  - stanzas in the override file will take precedence over stanzas present in the corresponding configuration file.
  - stanzas in the override file which are not present in the corresponding configuration file will be honoured when the job runs.
- If both a configuration file and an override file exist for a job and subsequently the override file is deleted, the configuration file is automatically reloaded with the effect that any changes introduced by the override file are undone and the configuration file alone now defines the job.
- If both a configuration file and an override file exist for a job and subsequently the configuration file is deleted, a new instance of the job can no longer be started (since without a corresponding configuration file an override file is ignored).
- If both a configuration file and an override file exist for a job and any of the contents of the override file are invalid, the override file is ignored and only the contents of the configuration file are considered.

Miscellaneous

kill timeout *INTERVAL*

Specifies the interval between sending the job's main process the *SIGTERM* and *SIGKILL* signals when stopping the running job. Default is 5 seconds.

expect stop

Specifies that the job's main process will raise the *SIGSTOP* signal to indicate that it is ready. **init**(8) will wait for this signal before running the job's post-start script, or considering the job to be running.

init(8) will send the process the *SIGCONT* signal to allow it to continue.

expect daemon

Specifies that the job's main process is a daemon, and will fork twice after being run. **init**(8) will follow this daemonisation, and will wait for this to occur before running the job's post-start script or considering the job to be running.

Without this stanza **init**(8) is unable to supervise daemon processes and will believe them to have stopped as soon as they daemonise on startup.

expect fork

Specifies that the job's main process will fork once after being run. **init**(8) will follow this fork, and will wait for this to occur before running the job's post-start script or considering the job to be running.

Without this stanza **init**(8) is unable to supervise forking processes and will believe them to have stopped as soon as they fork on startup.

BUGS

The **and** and **or** operators allowed with **start on** and **stop on** do not work intuitively: operands to the right of either operator are only evaluated once and state information is then discarded. This can lead to jobs with complex **start on** or **stop on** conditions not behaving as expected *when restarted*. For example, if a job encodes the following condition:

```
start on A and (B or C)
```

When 'A' and 'B' become true, the condition is satisfied so the job will be run. However, if the job ends and subsequently 'A' and 'C' become true, the job will *not* be re-run even though the condtion is satisfied. Avoid using complex conditions with jobs which need to be restarted.

AUTHOR

Manual page written by Scott James Remnant <**scott@netsplit.com**> and James Hunt <**james.hunt@canonical.com**>**.**

REPORTING BUGS

Report bugs at <**https://launchpad.net/upstart/+bugs**>

COPYRIGHT

SEE ALSO

init(8) **sh**(1)