










-  [Overview](#)
-  **Getting Started**
-  [FAQ](#)
-  [Download](#)
-  [Development](#)
-  [Wiki](#)
-  [Launchpad](#)

Please see the Upstart Cookbook for more up-to-date information:

- <http://upstart.ubuntu.com/cookbook/>
- http://upstart.ubuntu.com/cookbook/upstart_cookbook.pdf

Getting Started

Once you've [downloaded](#) and unpacked upstart, you will need to configure the source tree, build and install it. The main question here is deciding whether or not you want to take the plunge and replace `sysvinit` immediately, or whether you want to test first.

The brave will want to configure the source such that the executable parts are placed on the root filesystem and the data parts (man pages, etc.) are in the usual places.

```
./configure --prefix=/usr --exec-prefix= --sysconfdir=/etc
```

Everyone else will prefer to install it under an alternate prefix like `/opt/upstart`. You will need to boot with an alternate kernel command-line such as `init=/opt/upstart/sbin/init`.

```
./configure --prefix=/opt/upstart --sysconfdir=/etc
```

Job Definitions

Don't reboot just yet, you haven't configured anything to be started so your machine will just sit there. You need to write some job definitions that instruct upstart what to do, and when.

Upstart comes with a set of default jobs which it installs into `/etc/init`. These are based on the `sysvinit` configuration of Debian-based systems, including running the `/etc/init.d/rc` script.

This is recommended, as it allows you to boot your machine normally, as well as support existing applications, while you convert things to using upstart jobs.

The defaults will probably need modification to work in your distribution, as paths and maybe even arguments will need to be changed to match. Your existing `/etc/inittab` should be a useful guide.

Once happy, place the files in `/etc/init` and now you're ready to reboot and use upstart.

Writing Jobs

Once you're up and running, you'll want to start writing your own jobs. Note that the job file format is not stable yet, so if you upgrade upstart later, you may need to fix existing files.

Jobs are defined in files placed in `/etc/init`, the name of the job is the filename under this directory without the `.conf` extension. They are plain text files and should not be executable.

The format treats one or more space or tabs as whitespace, which is skipped unless placed in single or double quotes. Line breaks are permitted within quotes, or if preceeded by a backslash. Comments begin with a `#` and continue until the end of the line.

exec and script

All job files must have either an `exec` or `script` stanza. This specifies what will be run for the job.

`exec` gives the path to a binary on the filesystem and optional arguments to pass to it; any special characters (e.g. quotes or `'$'`) will result in the command being passed to a shell for interpretation instead.

```
exec /bin/foo --opt -xyz foo bar
```

`script` instead gives shell script code that will be executed using `/bin/sh`. The `-e` shell option is used, so any command that fails will terminate the script. The stanza is terminated by a line containing just `end script`.

```
script
    # do some stuff
    if [ ... ]; then
        ...
```

```
fi
end script
```

pre-start script and post-stop script

Additional shell code can be given to be run *before* or *after* the binary or script specified with `exec` or `script`. These are **not** expected to start the process, in fact, they can't. They are intended for preparing the environment and cleaning up afterwards.

pre-start script specifies the shell code to be run before the main process, as with `script` any command that fails will terminate the script and it is terminated with "end script"

```
pre-start script
# prepare environment
mkdir -p /var/run/foo
end script
```

post-stop script specifies the shell code to be run after the main process terminates or is killed, as with `script` and post-start script any command that fails will terminate the script and it is terminated with "end script"

```
post-stop script
# clean up
rm -rf /var/run/foo
end script
```

start on and stop on

Your job is now able to be started and stopped manually by a system administrator, however you also probably want it to be started and stopped automatically when events are emitted.

The primary event emitted by upstart is `startup` which is when the machine is first started (without writable filesystems or networking).

If you're using the example jobs, you will also have `runlevel x` events, where *x* is one of 0-6 or s. Jobs will be run alongside the init scripts for that runlevel.

Finally other jobs generate events as they are run; you can have yours run when another job stops by using `stopped job`. The other useful job event is `started job`.

You list the events you want to start your job with `start on`, and the events that stop your job with `stop on`.

```
start on startup

start on runlevel [23]

start on stopped rcS
```

```
start on started tty1
```

console

You can change the settings for where a job's output goes, and where its input comes from, with the `console` stanza. This should be one of `output` (input and output from `/dev/console`), `owner` (as output with the addition that certain signals (such as Control-C) are also sent to the process) or `none` (the default; input and output to `/dev/null`).

```
exec echo example
console output
```

Job Control

start and stop

Jobs may be started and stopped manually by using the `start` and `stop` commands, usually installed into `/sbin`. Each takes a job name, and outputs the final status (see below).

```
# start tty1
tty1 start/running, process 7490

# stop tty1
tty1 stop/running, process 7490
```

status

The status of any job may be queried by using the `status` command, again usually installed into `/sbin`. It takes a job name and outputs the current status.

```
# status tty1
tty1 stop/waiting

# start tty1
tty1 start/running, process 4418

# status tty1
tty1 start/running, process 4418
```

The output can be read as follows; the job name is followed by whether the job was last started (`start`) or last stopped (`stop`); the next word is the current state of the job and finally the process id (if any) is given.

initctl list

A list of all jobs and their states can be obtained by using `initctl list`.

```
# initctl list
control-alt-delete stop/waiting
rc stop/waiting
rc-sysinit stop/waiting
rcS stop/waiting
tty1 start/running, process 4418
tty2 start/running, process 7367
tty3 start/running, process 7368
tty4 start/running, process 7369
tty5 start/running, process 7370
tty6 start/running, process 7371
```

initctl emit

A custom event may be emitted by using `initctl emit`, any jobs started or stopped by this event will be affected. Assuming the following job:

```
start on bounce
exec echo --Bounced--
console output
```

The following will run it:

```
# initctl emit bounce
# --Bounced--
```

Events can take arguments (passed on the `emit` command-line) in the form of environment variables.

[Back to top](#)

Copyright © 2009 [Canonical Ltd.](#) Upstart is a trademark of Canonical Ltd.