s6

s6 is a process supervision suite that also provides tools for service management (s6-rc) and system initialization/shutdown (s6-linux-init).

Contents

General overview
Installation
  Installation of services
Programs
Files
Basic usage
Source directory structure
See also

General overview

The various s6 software are all designed to be very modular and can be mixed and matched with other things. However, s6 software is also designed to work well with one another. In Artix, we take full advantage of what s6 offers and use all of the available tools to provide a complete init, process supervisor suite, and service manager for users.

`s6-linux-init` is what actually initiates your system. It mounts a tmpfs onto `/run` and copies the `/etc/s6/current/run-image` directory into `/run`. It execs into the `s6-svscan` program (provided by `s6`) which functions as your PID1 for the lifetime of the machine. Once this is done, the stage 1 part of init is finished. In addition to being PID1, `s6-svscan` is also the root of your process supervision tree. It monitors every service found and appropriately spawns an `s6-supervise` for every service.

Additionally, during bootup the `rc.init` script in the `/etc/s6/current/scripts` directory gets executed. `s6` itself does not do service management, but only process supervision. In `rc.init`, the actual service manager, `s6-rc` is started. From that point, we launch the default runlevels defined in `/etc/s6/current/scripts/runlevel`

and begin the desired services.

On shutdown, every service in s6-rc is brought down, followed by killing all `s6-supervise` processes and other processes, then unmounting everything, and finally killing `s6-svscan`.

Installation

Install the `s6` package.

Installation of services

s6 service packages are named *`package_name`*`-s6` and, when installed, will be available in `/etc/s6/sv`.

Programs

The s6 software suite comes with many different binaries, but in general you only need to directly interact with a small subset of them.

- `s6-rc` - the main program for controlling and managing services
- `s6-rc-bundle` - adding and deleting bundles without having to recompile the live database
- `s6-rc-bundle-update` - helper script for removing and adding services to existing bundles
- `s6-rc-db` - a tool for analyzing a compiled service database

Files

Most of the files associated with the s6 software packages are installed in the `/etc/s6` directory.

- `/etc/s6/current` - the base directory for s6-linux-init
- `/etc/s6/current/scripts` - Various startup/shutdown scripts executed by s6-linux-init
- `/etc/s6/rc` - where compiled databases are stored; the current live database is always symlinked to `/etc/s6/rc/compiled`
- `/etc/s6/rc.local` - file for executing arbitrary shell commands on bootup (any shell scripts in /etc/local.d suffixed with *.start are executed on bootup and those with *.stop are executed on shutdown)
- `/etc/s6/s6.conf` - a global s6 config file for setting certain values for startup s6-rc services

- `/etc/s6/skel` - Contains the default startup/shutdown scripts that come with artix linux
- `/etc/s6/sv` - the directory containing s6-rc source definitions

Basic usage

A key concept to using s6-rc is to understand the notion of "bundles." You can take their namesake literally. A bundle in s6-rc is any collection of services, oneshots, and even other bundles. These are quite similar to openrc's runlevels and are used in similar ways in Artix. The package, `s6-scripts`, which contains essential startup oneshots and daemons for an Artix system internally uses many different bundles for convenience, but for users there are 2 main bundles they should concern themselves with.

- `boot` - This is a collection of startup/shutdown boot oneshots and daemons essential for a working system. These are mostly provided by the `s6-scripts` package with some optional dependencies that install themselves into the directory.
- `default` - This is the last bundle started by s6-rc. For the vast majority of users, they will want to enable their services and scripts here. The existence of this bundle, however, is entirely optional.

There are two kinds of bundles in s6-rc: those already defined in the source directories and live bundles. `boot` is a defined bundle in the `/etc/s6/sv`. It will always be created whenever an s6-rc database is generated. However, `default` is a live bundle. In fact by default, it doesn't even exist. You can manage live bundles with the `s6-rc-bundle` and `s6-rc-bundle-update` tools. To create and add services to the default bundle, you can do the following: `# s6-rc-bundle add default service1 service2 service3 etc`.

To modify an already existing bundle, you can use the `s6-rc-bundle-update` helper script. To add something to an existing bundle you can just execute: `# s6-rc-bundle-update add default service4`. To remove something from an existing bundle, do `# s6-rc-bundle-update delete default service4`. To completely remove a bundle, you use the regular `s6-rc-bundle` command like so: `# s6-rc-bundle delete default`.

Note: the *-s6 packages in Artix that are an optional dependency of s6-scripts install themselves inside the boot bundle. They do **not** need to be enabled manually or added to any other bundle since the boot bundle is always brought up first on any Artix system.

Also note that s6-rc manages dependencies so it is not necessary to manually start all needed dependencies. When service `foo` is started, all of its dependencies (if they

are not already up) are automatically started. Here are some handy commands.

- Stop a service/bundle `# s6-rc -d change service_name`
- Start a service/bundle `# s6-rc -u change service_name`
- List all active services `# s6-rc -a list`
- List all services/bundles in the database `# s6-rc-db list all`

Because s6-rc uses a database format, all *-s6 packages in Artix execute a libalpm hook ensuring that all newly installed scripts are immediately available in a new database. This consists of 3 main steps. First, it compiles a new database with a unique name generated by the `date` command with `s6-rc-compile`. Then, it executes `s6-rc-update` to update the live database to the newly compiled database. Finally, it atomically updates the symlink to `/etc/s6/rc/compiled` so on the next boot, the system executes the newest database. Additionally, there is logic to preserve bundles created with the `s6-rc-bundle` command and recreate them in the new database (otherwise, users would have to manually recreate them).

Note that this will leave behind old database directories in `/etc/s6/rc`. Feel free to remove them if they are not needed. If you wish to add your own services to `/etc/s6/sv`, it is probably best to simply execute the libalpm hook manually `# sh /usr/share/libalpm/scripts/s6-rc-db-update-hook` so all of the above boilerplate is handled for you.

Source directory structure

s6-rc has three types of services: longrun, oneshot, and bundle. Most *-s6 packages are longrun services (AKA daemons). Oneshot services do exactly what their name implies: execute once on bootup and optionally on shutdown. Bundles are simply a collection of longruns, oneshots, and even other bundles. Every source directory will have a manadatory `type` file that contains a single line defining what type the service is. Longrun services must contain a file called `run`, oneshots must contain a file called `up`, and bundles must contain a file called `contents`.

In Artix, a typical service script comes with two different parts: the daemon itself and the logger daemon that uses the `s6-log` tool. The scripts for the actual service `foo` will be installed in the `/etc/s6/sv` directory as `foo-srv`. Additionally, a small logger daemon for that specific service will be installed in `foo-log`. `foo-log` will catch any output from `foo-srv` and save it in `/var/log/foo` directory. The `s6-log` logger daemon is run as the `s6log` user and group. To give a user permission to view logs, just add him to the `s6log` group. `s6-log` does log rotation for you and has many different configurable options.

When interacting with s6-rc, the name of `foo` doesn't change (i.e. you still do `# s6-rc -u change foo`). This starts both `foo-srv` and `foo-log` and handles any of the dependencies for you.

This is a tree of a longrun directory structure (aka `/etc/s6/sv/foo-srv`). In many scripts, only `producer-for`, `type`, and `run` exist.

```
foo-srv
├── conf (optional)
├── dependencies (optional)
├── notification-fd (optional)
├── producer-for (required for foo-log)
├── run
└── type
```

A tree for a logger daemon longrun (aka `/etc/s6/sv/foo-log`) looks like this.

```
foo-log
├── conf (s6-log directive options)
├── consumer-for
├── notification-fd
├── pipeline-line
├── run
└── type
```

See also

- https://skarnet.org/software/s6/ - Official s6 documentation
- https://skarnet.org/software/s6-rc/ - Official s6-rc documentation
- https://skarnet.org/software/s6-linux-init/ - Official s6-linux-init documentation

---