# systemd/User

< Systemd

systemd offers users the ability to manage services under the user's control with a per-user systemd instance, enabling users to start, stop, enable, and disable their own units. This is convenient for daemons and other services that are commonly run for a single user, such as **mpd**, or to perform automated tasks like fetching mail. With some caveats it is even possible to run xorg and the entire window manager from user services.

**Related articles**

**systemd**

**Automatic login to virtual console**

**Start X at login**

## Contents

## How it works

As per default configuration in `/etc/pam.d/system-login`, the `pam_systemd` module automatically launches a `systemd --user` instance when the user logs in for the first time. This process will survive as long as there is some session for that user, and will be killed as soon as the last session for the user is closed. When **#Automatic start-up of systemd user instances** is enabled, the instance is started on boot and will not be killed. The systemd user instance is responsible for managing user services, which can be used to run daemons or automated tasks, with all the benefits of systemd, such as socket activation, timers, dependency system or strict process control via cgroups.

Similarly to system units, user units are located in the following directories (ordered by ascending precedence):

- `/usr/lib/systemd/user/` where units provided by installed packages belong.
- `~/.local/share/systemd/user/` where units of packages that have been installed in the home directory belong.
- `/etc/systemd/user/` where system-wide user units are placed by the system administrator.
- `~/.config/systemd/user/` where the user puts its own units.

When systemd user instance starts, it brings up the target `default.target`. Other units can be controlled manually with `systemctl --user`.

**Note:**

- Be aware that the `systemd --user` instance is a per-user process, and not per-session. The rationale is that most resources handled by user services, like sockets or state files will be per-user (live on the user's home dir) and not per session. This means that all user services run outside of a session. As a consequence, programs that need to be run inside a session will probably break in user services. The way systemd handles user sessions is pretty much in flux. See **[1] (https://mail.gnome.org/archives/desktop-devel-list/2014-January/msg00079.html)** and **[2] (http://lists.freedesktop.org/archives/systemd-devel/2014-March/017552.html)** for some hints on where things are going.

- `systemd --user` runs as a separate process from the `systemd --system` process. User units can not reference or depend on system units.

## Basic setup

All the user services will be placed in `~/.config/systemd/user/`. If you want to run services on first login, execute `systemctl --user enable service` for any service you want to be autostarted.

### Environment variables

The user instance of systemd does not inherit any of the **environment variables** set in places like `.bashrc` etc. There are several ways to set environment variables for the systemd user instance:

1. For users with a `$HOME` directory, create a *.conf* file in the `~/.config/environment.d/` directory with lines of the form `NAME=VAL`. Affects only that user's user unit. See

environment.d(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/environment.d.5) for more information.

2. Use the `DefaultEnvironment` option in `/etc/systemd/user.conf` file. Affects all user units.
3. Add a drop-in config file in `/etc/systemd/system/user@.service.d/`. Affects all user units; see **#Service example**
4. At any time, use `systemctl --user set-environment` or `systemctl --user import-environment`. Affects all user units started after setting the environment variables, but not the units that were already running.
5. Using the `dbus-update-activation-environment --systemd --all` command provided by **dbus**. Has the same effect as `systemctl --user import-environment`, but also affects the D-Bus session. You can add this to the end of your shell initialization file.
6. For "global" environment variables for the user environment you can use the environment.d directories which are parsed by systemd generators. See environment.d(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/environment.d.5) for more information.
7. You can also write an environment generator script which can produce environment variables that vary from user to user, this is probably the best way if you need per-user environments (this is the case for XDG_RUNTIME_DIR, DBUS_SESSION_BUS_ADDRESS, etc). See systemd.environment-generator(7) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.environment-generator.7).

One variable you may want to set is `PATH`.

After configuration, the command `systemctl --user show-environment` can be used to verify that the values are correct.

### Service example

Create the **drop-in** directory `/etc/systemd/system/user@.service.d/` and inside create a file that has the extension `.conf` (e.g. `local.conf`):

```
/etc/systemd/system/user@.service.d/local.conf
```

```
[Service]
Environment="PATH=/usr/lib/ccache/bin:/usr/local/bin:/usr/bin:/bin"
Environment="EDITOR=nano -c"
Environment="BROWSER=firefox"
Environment="NO_AT_BRIDGE=1"
```

### DISPLAY and XAUTHORITY

`DISPLAY` is used by any X application to know which display to use and `XAUTHORITY` to provide a path to the user's `.Xauthority` file and thus the cookie needed to access the X server. If you plan on launching X applications from systemd units, these variables need to be set. Systemd provides a script in `/etc/X11/xinit/xinitrc.d/50-systemd-user.sh` to import those variables into the systemd user session on X launch. **[3] (https://github.com/systemd/systemd/blob/v219/NEWS#L194)** So unless you start X in a nonstandard way, user services should be aware of the `DISPLAY` and `XAUTHORITY`.

### PATH

If you customize your `PATH` and plan on launching applications that make use of it from systemd units, you should make sure the modified `PATH` is set on the systemd environment. Assuming you set your `PATH` in `.bash_profile`, the best way to make systemd aware of your modified `PATH` is by adding the following to `.bash_profile` after the `PATH` variable is set:

```
~/.bash_profile
```

```
systemctl --user import-environment PATH
```

Note that this will not affect systemd services started before `~/.bash_profile` is sourced.

### pam_environment

Environment variables can be made available through use of the `pam_env.so` module. See **Environment variables#Using pam_env** for configuration details.

## Automatic start-up of systemd user instances

The systemd user instance is started after the first login of a user and killed after the last session of the user is closed. Sometimes it may be useful to start it right after boot, and keep the systemd user instance running after the last session closes, for instance to have some user process running without any open session. Lingering is used to that effect. Use the following command to enable lingering for specific user:

```
# loginctl enable-linger username
```

**Warning:** systemd services are **not** sessions, they run outside of *logind*. Do not use lingering to enable automatic login as it will **break the session**.

## Writing user units

See **systemd#Writing unit files** for general information about writing systemd unit files.

### Example

The following is an example of a user version of the mpd service.

```
~/.config/systemd/user/mpd.service
```

```
[Unit]
Description=Music Player Daemon

[Service]
ExecStart=/usr/bin/mpd --no-daemon

[Install]
WantedBy=default.target
```

## Example with variables

The following is an example of a user version of `sickbeard.service`, which takes into account variable home directories where **SickBeard** can find certain files:

```
~/.config/systemd/user/sickbeard.service
--------------------------------------------------------------------------------
[Unit]
Description=SickBeard Daemon

[Service]
ExecStart=/usr/bin/env python2 /opt/sickbeard/SickBeard.py --config %h/.sickbeard/config.ini --datadir %h/.sickbeard

[Install]
WantedBy=default.target
```

As detailed in `systemd.unit(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.unit.5)`, the `%h` variable is replaced by the home directory of the user running the service. There are other variables that can be taken into account in the **systemd** manpages.

## Note about X applications

Most X apps need a `DISPLAY` variable to run. See **#DISPLAY and XAUTHORITY** for how to this variable is set for the entire systemd user instance.

## Reading the journal

The journal for the user can be read using the analogous command:

```
$ journalctl --user
```

To specify a unit, one can use

```
$ journalctl --user-unit myunit.service
```

Or, equivalently:

```
$ journalctl --user -u myunit.service
```

## Temporary files

*systemd-tmpfiles* allows users to manage custom volatile and temporary files and directories just like in the system-wide way (see **systemd#Temporary files**). User-specific configuration files are read from `~/.config/user-tmpfiles.d/` and `~/.local/share/user-tmpfiles.d/`, in that order. For this functionality to be used, it is needed to enable the necessary systemd user units for your user:

```
$ systemctl --user enable systemd-tmpfiles-setup.service systemd-tmpfiles-clean.timer
```

The syntax of the configuration files is the same than those used system-wide. See the **systemd-tmpfiles(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd-tmpfiles.8)** and **tmpfiles.d(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/tmpfiles.**

d.5) man pages for details.

## Xorg and systemd

There are several ways to run xorg within systemd units. Below there are two options, either by starting a new user session with an xorg process, or by launching xorg from a systemd user service.

### Automatic login into Xorg without display manager

This option will launch a system unit that will start a user session with an xorg server and then run the usual `~/.xinitrc` to launch the window manager, etc. You need to have xlogin-git (https://aur.archlinux.org/packages/xlogin-git/)AUR installed. Set up your xinitrc as specified in the Xinit#xinitrc section.

The session will use its own dbus daemon, but various systemd utilities will automatically connect to the `dbus.service` instance. Finally, enable the `xlogin@username` service for automatic login at boot. The user session lives entirely inside a systemd scope and everything in the user session should work just fine.

### Xorg as a systemd user service

Alternatively, **xorg** can be run from within a systemd user service. This is nice since other X-related units can be made to depend on xorg, etc, but on the other hand, it has some drawbacks explained below.

**`xorg-server (https://www.archlinux.org/packages/?name=xorg-server)`** provides integration with systemd in two ways:

- Can be run unprivileged, delegating device management to logind (see Hans de Goede commits around **this commit (http://cgit.freedesktop.org/xorg/xserver/commit/?id=82863656ec449644cd34a86388ba40f36cea11e9)**).
- Can be made into a socket activated service (see **this commit (http://cgit.freedesktop.org/xorg/xserver/commit/?id=b3d3ffd19937827bcbdb833a628f9b1814a6e189)**). This removes the need for **`systemd-xorg-launch-helper-git (https://aur.archlinux.org/packages/systemd-xorg-launch-helper-git/)`**AUR.

Unfortunately, to be able to run xorg in unprivileged mode, it needs to run inside a session. So, right now the handicap of running xorg as user service is that it must be run with root privileges (like before 1.16), and can't take advantage of the unprivileged mode introduced in 1.16.

**Note:** This is not a fundamental restriction imposed by logind, but the reason seems to be that xorg needs to know which session to take over, and right now it gets this information calling **logind (http://www.freedesktop.org/wiki/Software/systemd/logind)**'s `GetSessionByPID` using its own pid as argument. See **this thread (http://lists.x.org/archiv**

es/xorg-devel/2014-February/040476.html) and xorg sources (http://cgit.freedesktop.org/xorg/xserver/tree/hw/xfree86/os-support/linux/systemd-logind.c). It seems likely that xorg could be modified to get the session from the tty it is attaching to, and then it could run unprivileged from a user service outside a session.

**Warning:** On xorg 1.18 socket activation seems to be broken. The client triggering the activation deadlocks. See the upstream bug report [4] (https://bugs.freedesktop.org/show_bug.cgi?id=93072). As a temporary workaround you can start the xorg server without socket activation, making sure the clients connect after a delay, so the server is fully started. There seems to be no nice mechanism te get startup notification for the X server.

This is how to launch xorg from a user service:

1. Make xorg run with root privileges and for any user, by editing `/etc/X11/Xwrapper.config`

```
/etc/X11/Xwrapper.config
```

```
allowed_users=anybody
needs_root_rights=yes
```

2. Add the following units to `~/.config/systemd/user`

```
~/.config/systemd/user/xorg@.socket
```

```
[Unit]
Description=Socket for xorg at display %i
```

```
[Socket]
ListenStream=/tmp/.X11-unix/X%i
```

```
~/.config/systemd/user/xorg@.service
--------------------------------------------------------------------------------
[Unit]
Description=Xorg server at display %i

Requires=xorg@%i.socket
After=xorg@%i.socket

[Service]
Type=simple
SuccessExitStatus=0 1

ExecStart=/usr/bin/Xorg :%i -nolisten tcp -noreset -verbose 2 "vt${XDG_VTNR}"
```

where `${XDG_VTNR}` is the virtual terminal where xorg will be launched, either hard-coded in the service unit, or set in the systemd environment with

```
$ systemctl --user set-environment XDG_VTNR=1
```

**Note:** xorg should be launched at the same virtual terminal where the user logged in. Otherwise logind will consider the session inactive.

3. Make sure to configure the `DISPLAY` environment variable as explained **above**.

4. Then, to enable socket activation for xorg on display 0 and tty 2 one would do:

```
$ systemctl --user set-environment XDG_VTNR=2     # So that xorg@.service knows which vt use
$ systemctl --user start xorg@0.socket            # Start listening on the socket for display 0
```

Now running any X application will launch xorg on virtual terminal 2 automatically.

The environment variable `XDG_VTNR` can be set in the systemd environment from `.bash_profile`, and then one could start any X application, including a window manager, as a systemd unit that depends on `xorg@0.socket`.

**Warning:** Currently running a window manager as a user service means it runs outside of a session with the problems this may bring: **break the session**. However, it seems that systemd developers intend to make something like this possible. See **[5] (https://mail.gnome.org/archives/desktop-devel-list/2014-January/msg00079.html)** and **[6] (http://lists.freedesktop.org/archives/systemd-devel/2014-March/017552.html)**

## Some use cases

### Persistent terminal multiplexer

You may wish your user session to default to running a terminal multiplexer, such as **GNU Screen** or **Tmux**, in the background rather than logging you into a window manager session. Separating login from X login is most likely only useful for those who boot to a TTY instead of to a display manager (in which case you can simply bundle everything you start in with myStuff.target).

To create this type of user session, procede as above, but instead of creating wm.target, create multiplexer.target:

```
[Unit]
Description=Terminal multiplexer
Documentation=info:screen man:screen(1) man:tmux(1)
After=cruft.target
Wants=cruft.target

[Install]
Alias=default.target
```

`cruft.target`, like `mystuff.target` above, should start anything you think should run before tmux or screen starts (or which you want started at boot regardless of timing), such as a GnuPG daemon session.

You then need to create a service for your multiplexer session. Here is a sample service, using tmux as an example and sourcing a gpg-agent session which wrote its information to `/tmp/gpg-agent-info`. This sample session, when you start X, will also be able to run X programs, since DISPLAY is set.

```
[Unit]
Description=tmux: A terminal multiplixer
Documentation=man:tmux(1)
After=gpg-agent.service
Wants=gpg-agent.service

[Service]
Type=forking
ExecStart=/usr/bin/tmux start
ExecStop=/usr/bin/tmux kill-server
Environment=DISPLAY=:0
EnvironmentFile=/tmp/gpg-agent-info
```

```
[Install]
WantedBy=multiplexer.target
```

Once this is done, `systemctl --user enable` `tmux.service`, `multiplexer.target` and any services you created to be run by `cruft.target` and you should be set to go! Activated `user-session@.service` as described above, but be sure to remove the `Conflicts=getty@tty1.service` from `user-session@.service`, since your user session will not be taking over a TTY. Congratulations! You have a running terminal multiplexer and some other useful programs ready to start at boot!

## Window manager

To run a window manager as a systemd service, you first need to run **#Xorg as a systemd user service**. In the following we will use awesome as an example:

```
~/.config/systemd/user/awesome.service
```

```
[Unit]
Description=Awesome window manager
After=xorg.target
Requires=xorg.target

[Service]
ExecStart=/usr/bin/awesome
Restart=always
RestartSec=10

[Install]
WantedBy=wm.target
```

**Note:** The `[Install]` section includes a `WantedBy` part. When using `systemctl --user enable` it will link this as `~/.config/systemd/user/wm.target.wants/`*`window_manager`*`.service`, allowing it to be started at login. Is recommended to enable this service, not to link it manually.

## Kill user processes on logout

Arch Linux builds the systemd (https://www.archlinux.org/packages/?name=systemd) package with `--without-kill-user-processes`, setting `KillUserProcesses` to `no` by default. This setting causes user processes not to be killed when the user completely logs out. To change this behavior in order to have all user processes killed on the user's logout, set `KillUserProcesses=yes` in `/etc/systemd/logind.conf`.

Note that changing this setting breaks terminal multiplexers such as **tmux** and **screen**. If you change this setting, you can still use a terminal multiplexer by using `systemd-run` as follows:

```
$ systemd-run --scope --user command args
```

For example, to run `screen` you would do:

```
$ systemd-run --scope --user screen -S foo
```

## See also

- **KaiSforza's Bitbucket wiki (https://bitbucket.org/KaiSforza/systemd-user-units/wiki/Home)**
- **Zoqaeski's units on GitHub (https://github.com/zoqaeski/systemd-user-units)**
- **Arch forum thread about changes in systemd 206 user instances (https://bbs.archlinux.org/viewtopic.php?id=167115)**

Retrieved from "https://wiki.archlinux.org/index.php?title=Systemd/User&oldid=506607"