Running A Program At Start UP

A Beginner's Guide

Table of Contents

1 Colophon....5
2 Introduction....6
2.1 What's Not Covered....6
2.2 Requirements....6
2.3 Conventions....7
3 Selecting A Method....8
3.1 The Difference Between Boot Or Login....8
3.2 Available Methods....8
3.3 Selecting A Method....8
4 Command Line Programs and Scripts....10
4.1 Using .bashrc Or .profile....10
4.1.1 Advatages....10
4.1.2 Disadvantages....10
4.1.3 Basic Usage....10
4.1.4 Advanced Usage....11
4.1.4.1 Programs That Do Not Exit Immediately....11
4.1.4.2 Capturing Output And Errors....11
4.2 Using /etc/rc.local....12
4.2.1 Advantages....12
4.2.2 Disadvantages....12
4.2.3 Basic Usage....13
4.2.4 Advanced Usage....14
4.2.4.1 Programs That Do Not Exit Immediately....14
4.2.4.2 Capturing Output And Errors....14
4.2.4.3 Running As A User Other Than Root....15
4.3 Using cron....16
4.3.1 Advantages....16
4.3.2 Disadvantages....16
4.3.3 Basic Usage....17
4.3.4 Advanced Usage....17
4.3.4.1 Using root's crontab....17
4.3.4.2 Running More Than One Program....17
4.3.4.3 Capturing Output And Errors....17
4.4 Using A Systemd Service....18
4.4.1 Advantages....18
4.4.2 Disadvantages....18
4.4.3 Creating A .service File....18
4.4.3.1 The [Unit] Section....19
4.4.3.2 The [Service] Section....20
4.4.3.3 The [Install] Section....21
4.4.4 Installing A Service....22
4.4.5 Advanced Usage....23
4.4.5.1 Per User Services....23
4.4.5.2 Starting A Service After The Network Is Up....23
4.4.5.3 Handling Input And Output....24

4.4.5.4 Further Reading....25
5 Desktop (GUI) Programs - X11....26
5.1 When The Full Desktop Is Required....26
5.1.1 Autostart Via The autostart File....26
5.1.1.1 Advantages....26
5.1.1.2 Disadvantages....26
5.1.1.3 Usage....27
5.1.2 Autostart Via A .desktop File....28
5.1.2.1 Advantages....28
5.1.2.2 Disadvantages....28
5.1.2.3 Usage....28
5.1.3 Via A Systemd Services....29
5.1.3.1 Advantages....29
5.1.3.2 Disadvantages....29
5.1.3.3 Creating A .service File....29
5.1.3.4 Installing The Service....29
5.1.3.5 Advanced Usage....29
5.2 When The Full Desktop Is Not Required....30
5.2.1 Autostart Via The autostart File....30
5.2.1.1 Advantages....30
5.2.1.2 Disadvantages....30
5.2.1.3 Usage....31
5.2.2 Autostart Via A .desktop File....32
5.2.2.1 Advantages....32
5.2.2.2 Disadvantages....32
5.2.2.3 Usage....33
5.2.3 Via A Systemd Service And Automatic Login....34
5.2.3.1 Advantages....34
5.2.3.2 Disadvantages....34
5.2.3.3 Disable The Default Autostart....34
5.2.3.4 Creating A .service File....34
5.2.3.5 Installing The Service....35
5.2.3.6 Advanced Usage....35
5.2.4 Without Automatic Login....36
5.2.4.1 Advantages....36
5.2.4.2 Disadvantages....36
5.2.4.3 OS Configuration....36
5.2.4.4 Usage....37
6 Desktop (GUI) Programs - Wayland/Wayfire....38
6.1 When The Full Desktop Is Required....38
6.1.1 Autostart Via wayfire.ini....38
6.1.1.1 Advantages....38
6.1.1.2 Disadvantages....38
6.1.1.3 Usage....39
6.1.2 Autostart Via A .desktop File....40
6.1.2.1 Advantages....40
6.1.2.2 Disadvantages....40
6.1.2.3 Usage....40

6.2 When The Full Desktop Is Not Required....41
7 Hints And Tips....42
8 Troubleshooting....43
8.1 First Find The Error....43
8.1.1 .bashrc and .profile....43
8.1.2 /etc/rc.local....43
8.1.3 Cron....43
8.1.4 Systemd Services....43
8.1.5 Autostart Via The autostart File....44
8.1.6 Autostart Via A .desktop File....44
8.1.7 GUI Programs Without The Full Desktop....44
8.2 Once You Have The Error....45
8.3 Common Issues, Their Causes, And Potential Solutions....46
8.3.1 It works in Thonny or in a logged in shell but not when started at system boot....46
8.3.2 Log File Is Empty....46
8.3.3 Command or file not found....46
8.3.4 My Program Can’t Find A file It Needs....46
8.3.5 Output Files Are Not Created Or Updated Or Are Written In The Wrong Place....47
8.3.6 Command Or Program Cannot Access The Network....47
8.3.7 Python Complains That a Method, Class, etc. Is Unknown....47
8.3.8 Python Cannot Import A Module....47
8.3.9 Python Cannot Import A Custom Module....48
8.3.10 Cannot Open Display....48
8.3.11 Python and Virtual Environments....48
9 Change Log....49
9.1 2023-11-21....49

1 Colophon

2 Introduction

This is a guide to running programs at start up. While aimed at the Raspberry Pi and Raspberry Pi OS it applies equally to any Linux running systemd.

It is assumed that the reader has a basic familiarity with the Linux command line and at least one text editor. Desktop users will need to open a terminal to execute many of the commands in this guide.

2.1 What's Not Covered

- SysV init.

2.2 Requirements

- Raspberry Pi (any model) and the normal accessories.
- A program you wish to run at system start

2.3 Conventions

```
Text like this indicates input to or output from the command line.
```

`Text like this` also refers to full or partial commands but is not generally intended to be entered into the command line as is.

“SD card” refers equally to full size and micro SD cards.

“RPiOS” refers to Raspberry Pi OS.

“CWD” and “cwd”: Current Working Directory.

All example code uses the user name “pi” and group name “pi”. Replace as necessary.

3 Selecting A Method

3.1 The Difference Between Boot Or Login

The difference between starting a program during boot or on login can be subtle but is important.

Given the default behaviour of RPiOS[1] of booting to a logged in desktop the two are often seen as the same thing. They are not.

Programs started during boot will be started once. Programs started on login will be started on every login, regardless of source or whether the program is already running. On login programs started as part of your command prompt initialisation will also be started each time a terminal window is opened on the desktop.

3.2 Available Methods

- During boot:
  - systemd service
  - systemd user service
  - cron
  - `/etc/rc.local`
- At login:
  - `$HOME/.profile`
  - `$HOME/.bashrc`
  - autostart (desktop only)

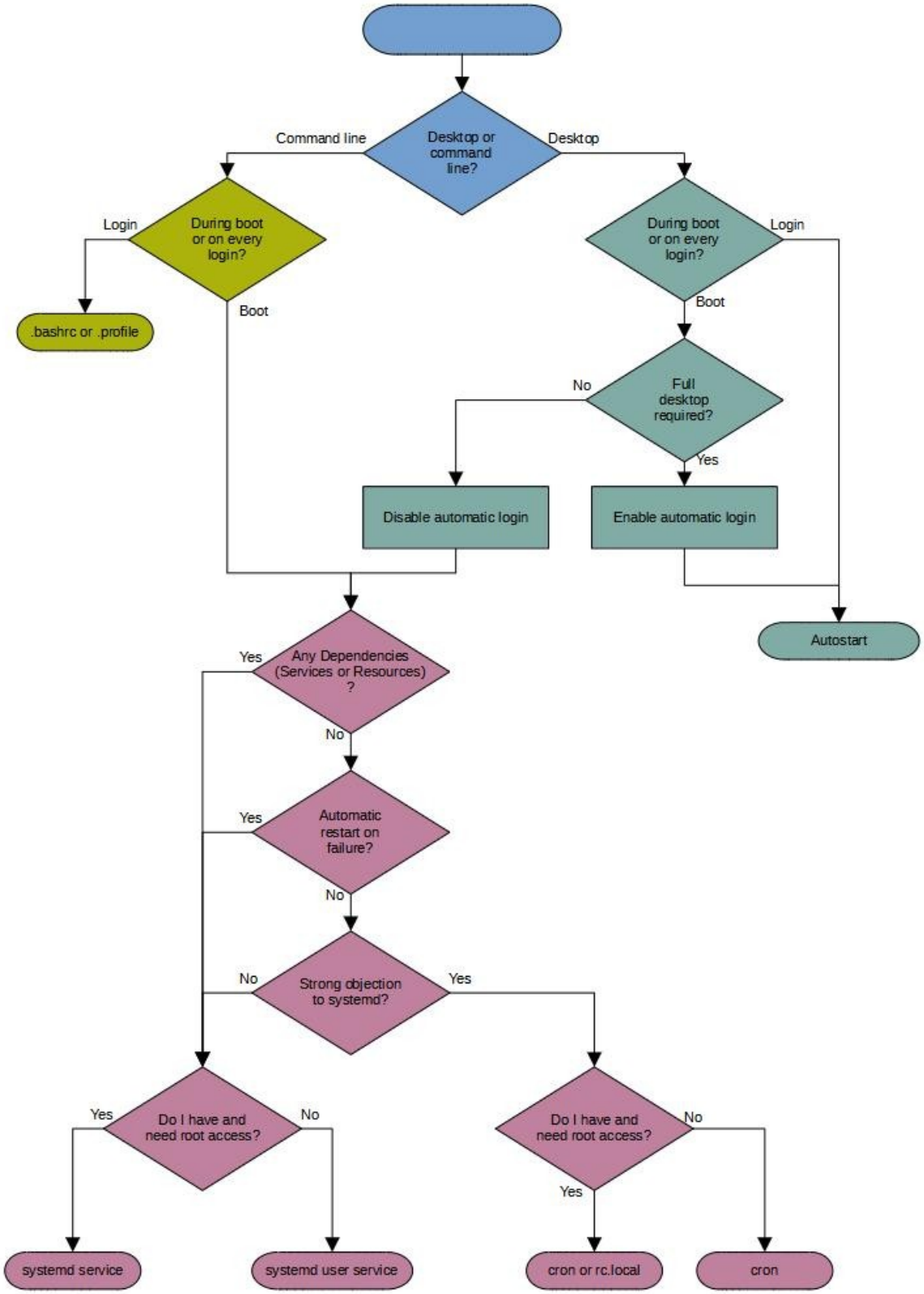
System wide equivalents to .profile, .bashrc, and autostart exist but using them is not recommended as doing so impacts all users.

A systemd service is the preferred option as it can manage both dependencies and restarting a program should it fail.

3.3 Selecting A Method

See the following flow chart.

1 Raspberry Pi OS Lite excluded.


Desktop or command line?
Command line
Desktop
During boot or on every login?
Login
Boot
.bashrc or .profile
During boot or on every login?
Login
Boot
Full desktop required?
No
Yes
Disable automatic login
Enable automatic login
Autostart
Any Dependencies (Services or Resources) ?
Yes
No
Automatic restart on failure?
Yes
No
Strong objection to systemd?
No
Yes
Do I have and need root access?
Yes
No
systemd service
systemd user service
Do I have and need root access?
No
Yes
cron or rc.local
cron

4 Command Line Programs and Scripts

Methods show here are suitable for command line programs and scripts that do not require input from or output to the console.

4.1 Using .bashrc Or .profile

Both .bashrc and .profile can be used to start a program on login. On RPiOS .bashrc is preferred.

.bashrc and .profile are located in the user's home directory but will not be show by `ls` without the `-a` option.[2]

4.1.1 Advatages

- It's easy.
- It's a shell script so can use all features of such scripts.
- It's per user.

4.1.2 Disadvantages

- It's per user.
- It runs on every login whether local, remote (e.g. over ssh), command line, or desktop.
- It runs every time a terminal windows is opened in the desktop or a new tab is opened in an existing terminal window.
- If the user does not login the program will not be started.
- Programs started as foreground tasks will block the terminal/login.
- There is little to no process control, only that provided by the shell.[3]
- No automatic restart on exit.

4.1.3 Basic Usage

1. Open .bashrc in your preferred text editor.
2. Add the program to the end of the file.
3. Save and close.
4. Log out.
5. Log in.

2 Files with names starting with "." are considered hidden.

3 `kill`, `jobs`, `fg`, `bg`, `nice`, `ps`, etc.

4.1.4 Advanced Usage

4.1.4.1 Programs That Do Not Exit Immediately

If the program you're starting does not exit immediately and does not interact with the terminal it should be started as a background job. Failure to do so will prevent use of the terminal.

A command can be run in the background by appending `&` to it. `&` must come after all arguments and after any input or output redirection.

For example:

```
/usr/bin/env &
```

4.1.4.2 Capturing Output And Errors

To assist in troubleshooting it is often useful to capture the output and/or error messages[4] from a program to a file. This is particularly help when a system is headless[5] or programs are started in the background. Redirection is done using the standard linux syntax:

- `>/path/to/file` to redirect output replacing the contents of `file`.
- `>>/path/to/file` to redirect output adding to the contents of `file`.
- `2>/path/to/file` to redirect errors replacing the contents of `file`.
- `2>>/path/to/file` to redirect errors adding to the contents of `file`.
- `2>&1` to redirect errors to the same place as output. This will add or replace contents depending on how the output has been redirected.

For example:

```
/usr/bin/env >/tmp/env.log 2>&1
```

Output redirection will prevent output from being displayed by the terminal.

4 Also know as stdout and stderr.
5 Has no monitor attached.

4.2 Using /etc/rc.local

4.2.1 Advantages

- It’s easy.
- It’s a shell script so can use all features of such scripts.

4.2.2 Disadvantages

- Can only be edited by root or with sudo.
- rc.local is run by root so everything started by it is also run by root[6].
- Any error in rc.local or in anything it starts will prevent the rest of rc.local from running.
- Each item started by rc.local must return before the next is started.[7]
- Any output is sent to the console unless redirected.
- No input is received.
- No automatic restart if the program exits.
- Services and resources the program depends on may not be available.
- System wide and user specific .bashrc and .profile are not run.
- Current working directory is `/`
- You must use the full path to any programs/scripts that are not in $PATH unless they are shell built ins or in `/`
- You must use the full path to any files passed to the program/script you’re starting unless they are in `/`
- The program/script you’re starting must use the full path to any files it references within its code unless these files are in `/`

6 Judicious use of su and sudo can change this.
7 Unless started as a background task.

4.2.3 Basic Usage

1. Open /etc/rc.local in your preferred text editor. You will need to be root or use `sudo`.
2. Find the line that reads

   ```
   exit 0
   ```

3. Insert the command(s) required above that line.
4. Save and close the file.

Commands will run at the next (re)boot.

A trivial example based on RPiOS' default rc.local:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi
## trivial example – print environment to console:
/usr/bin/env

exit 0
```

4.2.4 Advanced Usage

4.2.4.1 Programs That Do Not Exit Immediately

If the program you're starting does not exit immediately it must be started as a background job. Failure to do so has two main effects:

1. Start up is delayed and may result in a timeout and a failed boot.
2. The next command in rc.local will not be run until the current one exits[8].

A command can be run in the background by appending `&` to it. `&` must come after all arguments and after any input or output redirection.

Using the same trivial example as in 4.2.3:

```
/usr/bin/env &
```

4.2.4.2 Capturing Output And Errors

To assist in troubleshooting it is often useful to capture the output and/or error messages[9] from a program to a file. This is particularly help when a system is headless[10]. Redirection is done using the standard linux syntax:

- `>/path/to/file` to redirect output replacing the contents of `file`.
- `>>/path/to/file` to redirect output adding to the contents of `file`.
- `2>/path/to/file` to redirect errors replacing the contents of `file`.
- `2>>/path/to/file` to redirect errors adding to the contents of `file`.
- `2>&1` to redirect errors to the same place as output. This will add or replace contents depending on how the output has been redirected.

Using the same trivial example as in 4.2.3:

```
/usr/bin/env >/tmp/env.log 2>&1
```

8 If it never exits, the next command will never be run.
9 Also know as stdout and stderr.
10 Has no monitor attached.

4.2.4.3 Running As A User Other Than Root

Programs should not but run by root unless the absolutely need to be in order to function correctly. `sudo` can be used to run them under a different user.

Continuing the example from 4.2.3:

```
sudo -u pi -i /usr/bin/env
```

`-u username` tells `sudo` which user to run the command as. In the above case the pi user.

`-i` is optional and tells `sudo` to run the command as if under a login shell. Runs .bashrc (and others),sets working directory to the user's home directory, etc.

For more on `sudo`, see

```
man sudo
```

4.3 Using cron

Cron can do more than run a program at boot however that's beyond the scope of this guide.

4.3.1 Advantages

- It's easier than a systemd service.
- It can (and should) be configured per user.

4.3.2 Disadvantages

- Configuration files must be edited via the `crontab` command.
- Any output is discarded[11] unless redirected.
- No input is received.
- No automatic restart if the program exits.
- Services and resources the program depends on may not be available.
- "~" is not expanded.
- A different shell to that used by a logged in session is used: /bin/sh instead of /bin/bash
- Different `$PATH` to a logged in session:

```
/usr/bin:/bin
```

- System wide and user specific .bashrc and .profile are not run.
- Current working directory is the user's home directory.
- You must use the full path to any programs/scripts that are not in $PATH unless they are shell built ins or in the user's home directory.
- You must use the full path to any files passed to the program/script you're starting unless they are in the user's home directory.
- The program/script you're starting must use the full path to any files it references within its code unless these files are in the user's home directory.

11 If a suitable Mail Transfer Agent has been installed and configured output will instead be emailed to the appropriate user. Installing and configuring an MTA is outside the scope of this guide.

4.3.3 Basic Usage

- Open your crontab for editing:

```
crontab -e
```

- Add a new line ensuring it starts with @reboot. For example:

```
@reboot /usr/bin/env
```

- Save and close

At the next boot/reboot you command will run. The example given above will appear to do nothing as output from it will be discarded.

4.3.4 Advanced Usage

4.3.4.1 Using root's crontab

Using root's crontab is simple but should be avoided unless the program you wish to run must be run as root. Simply use `sudo crontab -e` rather than `crontab -e`.

The working directory for programs started from root's crontab is `/root`.

4.3.4.2 Running More Than One Program

If the programs are entirely independent use multiple lines in your crontab, one per program.

For simple situations, normal shell syntax can be used to chain commands:

- A ; B Do A then B.
- A && B Do A, if A succeeds do B
- A || B Do A, if A fails do B
- A && B || C Do A, if A succeeds do B. If A failed do C

For more complex situations, put your commands into a shell script and call that from cron.

4.3.4.3 Capturing Output And Errors

See 4.2.4.2

4.4 Using A Systemd Service

4.4.1 Advantages

- It’s flexible.
- Program start can be delayed until the required dependencies are available.
- Programs can be automatically restarted should they exit.
- Programs can run as normal users.
- Programs are controlled through the same tools (`systemctl` and `journalctl`) as system services.

4.4.2 Disadvantages

- It’s more complex than `cron` or `rc.local`.
- Creating or modifying service files must be done as root or with `sudo`.
- The default user and group are root.
- The default CWD is /
- By default all output is discarded.
- By default no input is received.

4.4.3 Creating A .service File

A systemd service is defined in a service file. Service files are plain text and have a file name ending in `.service` for example `my-service.service` with contents of:

```
[Unit]
Description=pwd test

[Service]
ExecStart=pwd

[Install]
WantedBy=multi-user.target
```

The [Unit] section contains generic information about the service.

The [Service] section contains information specific to the service and its type.

The [Install] section is used when services are enabled or disabled.

4.4.3.1 The [Unit] Section

If the [Unit] or any of its fields are omitted sensible defaults will be used. Valid fields include:

`Description=`	Human readable name for the service
`Wants=`	Space separated list of units wanted by this service. Systemd will attempt to start these units if they are not already running when this service is started.
`After=`	Space separated list of units that must be started before this one.

Please refer to the systemd documentation for full details:
https://www.freedesktop.org/software/systemd/man/systemd.unit.html

4.4.3.2 The [Service] Section

The [Service] section cannot be omitted. Valid fields include:

`Type=`	One of `simple`, `exec`, `forking`, `oneshot`, `dbus`, `notify`, or idle
	`simple` is the default. The main process is expected to keep running.
	`oneshot` is used for processes that are expected to exit and allows them to do so without causing the service to enter a failed state.
	Other service types are outside the scope of this guide.
`RemainAfterExit=`	`yes` or `no`. If yes the service will be considered active even when all of its processes have exited.
	Defaults to `no`
	Most useful with `Type=oneshot`
`ExecStart=`	Command and its arguments to be executed when the service is started.
	A single command only.
	The first argument must be either an absolute path to an executable or a simple file name.
`Restart=`	One of `no`, `on-success`, `on-failure`, `on-abnormal`, `on-abort`, or `always`
	Specifies whether and under what circumstances the main process will be restarted.
	Defaults to `no`
`User=`	Username to run the service as. Defaults to root.
`Group=`	Group to run the service as. Defaults to root.
`WorkingDirectory=`	Absolute path to the directory to use as the working directory for the service or `~`
	`~` expands to the home directory of the user specified in `User=`
	Defaults to `/`

Please refer to the systemd documentation for full details:
https://www.freedesktop.org/software/systemd/man/systemd.service.html

4.4.3.3 The [Install] Section

The [Install] section cannot be omitted. Valid fields include:

`WantedBy=` Space separated list of units that want this service.

For most custom run at system start services `WantedBy=multi-user.target` can be used.

Please refer to the systemd documentation for full details:
https://www.freedesktop.org/software/systemd/man/systemd.unit.html

4.4.4 Installing A Service

1. Create your service file.
2. Copy it to /etc/systemd/system:

```
sudo cp my-service.service /etc/systemd/system/
```

3. Update systemd's internal data:

```
sudo systemctl daemon-reload
```

4. Enable your service

```
sudo systemctl enable my-service
```

5. Start it

```
sudo systemctl start my-service
```

4.4.5 Advanced Usage

4.4.5.1 Per User Services

Systemd allows unprivileged users to create and run their own services though any options that require root privileges (e.g. `User=`) cannot be used.

1. Enable “linger” on the user account. This is a one time task and only needed where it is desirable for the user’s services to run when they are not logged in.

   ```
   sudo loginctl enable-linger pi
   ```

2. Create the directory to hold your services. This only needs to be done once.

   ```
   mkdir -p ~/.config/systemd/user
   ```

3. Create your .service file.
4. Copy or move it to `~/.config/systemd/user`
5. Enable, disable, start, stop, restart, etc. using `systemctl --user` rather than `sudo systemctl`.

4.4.5.2 Starting A Service After The Network Is Up

Add the following to the [Unit] section of your .service file:

```
After=network-online.target
Wants=network-online.target
```

See also https://www.freedesktop.org/wiki/Software/systemd/NetworkTarget/

4.4.5.3 Handling Input And Output

By default anything sent by the service's main process that would normally be sent to the terminal is instead sent to systemd's journal. Any input that would be read from the terminal is instead read from `/dev/null`. Normal shell redirection (>, <, etc.) is not supported.

The following entries in the [Service] section can be used to change this behaviour.

`StandardInput=` One of `null`, `tty`, `tty-force`, `tty-fail`, or `file:path`

`null` reads from `/dev/null`

`tty` reads from the terminal specified in `TTYPath=` (see below) and will wait for the terminal to become available if another process has control of it.

`tty-force` behaves as tty expect it will take control of the specified terminal.

`tty-fail` behaves as tty except the service will fail if it cannot take control of the terminal.

`file:path` reads from the specified file. `Path` must be an absolute path.

Defaults to `null`.

`StandardOutput=` One of `inherit`, `null`, `tty`, `journal`, `journal+console`, or `file:path`

`inherit` sends output to the same place that has been configured for StandardInput.

`null` sends all output to /dev/null

`journal` sends all output to the systemd journal

`journal+console` sends all output to the systemd journal and duplicates it to the system console.

`file:path` sends all output to the specified file. `Path` must be an absolute path.

Defaults to `journal`.

`StandardError=` As `StandardOutput=` except that `inherit` comes from `StandardOutput=` rather than StandardInput=.

Defaults to `journal`.

`TTYPath=` Device node to use for `tty*` options.

Defaults to `/dev/console`

Please refer to the systemd documentation for full details:
https://www.freedesktop.org/software/systemd/man/systemd.exec.html

4.4.5.4 Further Reading

This has just touched the surface of what can be done with systemd. A full set of systemd manual pages can be found here: https://www.freedesktop.org/software/systemd/man/

5 Desktop (GUI) Programs - X11

For all methods in this section first ensure X11 and the desktop are installed. The default RPiOS desktop is assumed.

5.1 When The Full Desktop Is Required

(Or when you can live with the security implications of using it)

5.1.1 Autostart Via The autostart File

5.1.1.1 Advantages

- It's relatively easy.
- It can (and should) be configured per user.

5.1.1.2 Disadvantages

- The user must be logged in to the desktop.
- It runs whenever the user logs in to the desktop or whenever the desktop is started from the command line.
- If the user does not login to or start the desktop the program will not be started.
- Services and resources the program depends on may not be available.
- There is little to no process control, only that provided by the shell.[12]
- Current working directory is the user's home directory.
- You must use the full path to any programs/scripts that are not in $PATH unless they are shell built ins or in the user's home directory.
- You must use the full path to any files passed to the program/script you're starting unless they are in the user's home directory.
- The program/script you're starting must use the full path to any files it references within its code unless these files are in the user's home directory.
- The autostart file is not a shell script.[13] Should you need multi-line commands or other features of shell scripts[14] you will need to provide a wrapper script.

12 `kill`, `jobs`, `fg`, `bg`, `nice`, `ps`, etc.
13 See https://wiki.lxde.org/en/LXSession#autostart_configuration_file
14 Such as escaping, quoting, redirection, etc.

5.1.1.3 Usage

1. Create the user's session configuration directory:

```
mkdir -p /home/pi/.config/lxsession/LXDE-pi
```

2. Copy the default autostart file into the directory created above:

```
cp /etc/xdg/lxsession/LXDE-pi/autostart /home/pi/.config/lxsession/LXDE-
   pi/
```

3. Open `/home/pi/.config/lxsession/LXDE-pi/autostart` in your preferred text editor. This does not require root or sudo.

4. Add the commands to the end of the file one program per line. Start a line with `@` if you want the windows manager to restart a program should it fail. For example:

```
@mousepad
```

5. Save and close.

6. Logout and back in or reboot.

5.1.2 Autostart Via A .desktop File

5.1.2.1 Advantages

- It’s relatively easy.

5.1.2.2 Disadvantages

- The user must be logged in to the desktop.
- It runs whenever the user logs in to the desktop or whenever the desktop is started from the command line.
- If the user does not login to or start the desktop the program will not be started.
- Services and resources the program depends on may not be available.
- There is little to no process control, only that provided by the shell.[15]
- Current working directory is the user’s home directory.
- You must use the full path to any programs/scripts that are not in $PATH unless they are shell built ins or in the user’s home directory.
- You must use the full path to any files passed to the program/script you’re starting unless they are in the user’s home directory.
- The program/script you’re starting must use the full path to any files it references within its code unless these files are in the user’s home directory.

5.1.2.3 Usage

1. Create the autostart directory:

```
mkdir -p /home/pi/.config/autostart
```

2. Open a new text file in your preferred editor.
3. Add the following three lines:

```
[Desktop Entry]
Type=Application
Exec=mousepad
```

   Replace `mouespad` with the command of your choice.

4. Save in `/home/pi/.config/autostart` with a name ending in `.desktop` e.g. `mousepad.desktop`
5. Logout and login or reboot.

15 `kill`, `jobs`, `fg`, `bg`, `nice`, `ps`, etc.

5.1.3 Via A Systemd Services

5.1.3.1 Advantages

- See 4.4.1

5.1.3.2 Disadvantages

- A user must be logged in to the desktop.
- The service (or its main process) must be run by the same user.[16]
- See 4.4.2

5.1.3.3 Creating A .service File

See 4.4.3

A service to start a desktop/GUI is likely to fail without the following:

1. In the [Unit] section:

```
After=graphical.target
```

2. In the [Service] section:

```
User=pi
Group=pi
Environment="DISPLAY=:0"
```

   Replace “pi” and “:0” as required.[17]

3. In the [Install] section:

```
WantedBy=graphical.target
```

5.1.3.4 Installing The Service

See 4.4.4

5.1.3.5 Advanced Usage

See 4.4.5

16 Or security must be severely relaxed on the X server. That’s outside the scope of this guide.
17 “:0” is the first X server (desktop) on the local machine.

5.2 When The Full Desktop Is Not Required

5.2.1 Autostart Via The autostart File

5.2.1.1 Advantages

- It's relatively easy.
- It can (and should) be configured per user.

5.2.1.2 Disadvantages

- A user must be logged in to the desktop.
- The window manager is still running. Right clicking on the desktop will open its menu allowing additional tasks to be started. Keyboard shortcuts e.g. `ctrl-alt-t` are still active.
- It runs whenever the user logs in to the desktop or whenever the desktop is started from the command line.
- If the user does not login to or start the desktop the program will not be started.
- Services and resources the program depends on may not be available.
- There is little to no process control, only that provided by the shell.[18]
- Current working directory is the user's home directory.
- You must use the full path to any programs/scripts that are not in $PATH unless they are shell built ins or in the user's home directory.
- You must use the full path to any files passed to the program/script you're starting unless they are in the user's home directory.
- The program/script you're starting must use the full path to any files it references within its code unless these files are in the user's home directory.
- The autostart file is not a shell script.[19] Should you need multi-line commands or other features of shell scripts[20] you will need to provide a wrapper script.

18 `kill`, `jobs`, `fg`, `bg`, `nice`, `ps`, etc.
19 See https://wiki.lxde.org/en/LXSession#autostart_configuration_file
20 Such as escaping, quoting, redirection, etc.

5.2.1.3 Usage

1. Create the user's session configuration directory:

   ```
   mkdir -p /home/pi/.config/lxsession/LXDE-pi
   ```

2. Open a new file in your preferred text editor.

3. Add the commands to the end of the file one program per line. Start a line with `@` if you want the windows manager to restart a program should it fail. For example:

   ```
   @mousepad
   ```

4. Save it as `/home/pi/.config/lxsession/LXDE-pi/autostart` and close.

5. Logout and back in or reboot.

5.2.2 Autostart Via A .desktop File

5.2.2.1 Advantages

- It’s relatively easy.

5.2.2.2 Disadvantages

- The user must be logged in to the desktop.
- The window manager is still running. Right clicking on the desktop will open its menu allowing additional tasks to be started. Keyboard shortcuts e.g. `ctrl-alt-t` are still active.
- It runs whenever the user logs in to the desktop or whenever the desktop is started from the command line.
- If the user does not login to or start the desktop the program will not be started.
- Services and resources the program depends on may not be available.
- There is little to no process control, only that provided by the shell.[21]
- Current working directory is the user’s home directory.
- You must use the full path to any programs/scripts that are not in $PATH unless they are shell built ins or in the user’s home directory.
- You must use the full path to any files passed to the program/script you’re starting unless they are in the user’s home directory.
- The program/script you’re starting must use the full path to any files it references within its code unless these files are in the user’s home directory.

21 `kill`, `jobs`, `fg`, `bg`, `nice`, `ps`, etc.

5.2.2.3 Usage

1. Create the user's session configuration directory:

```
mkdir -p /home/pi/.config/lxsession/LXDE-pi
```

2. Create the autostart directory:

```
mkdir -p /home/pi/.config/autostart
```

3. Create an empty `/home/pi/.config/lxsession/LXDE-pi/autostart` file. This overrieds the system default.

```
Touch /home/pi/.config/lxsession/LXDE-pi/autostart
```

4. Open a new text file in your preferred editor.

5. Add the following three lines:

```
[Desktop Entry]
Type=Application
Exec=mousepad
```

   Replace `mouespad` with the command of your choice.

6. Save in `/home/pi/.config/autostart` with a name ending in `.desktop` e.g. `mousepad.desktop`

7. Logout and login or reboot.

5.2.3 Via A Systemd Service And Automatic Login

5.2.3.1 Advantages

- See 4.4.1

5.2.3.2 Disadvantages

- A user must be logged in to the desktop.
- The window manager is still running. Right clicking on the desktop will open its menu allowing additional task to be started. Keyboard shortcuts e.g. `ctrl-alt-t` are still active.
- The service (or its main process) must be run by the same user.[22]
- See 4.4.2

5.2.3.3 Disable The Default Autostart

1. Create the autostart directory:

```
mkdir -p /home/pi/.config/autostart
```

2. Create an empty/home/pi/.config/lxsession/LXDE-pi/autostart file.

```
touch /home/pi/.config/lxsession/LXDE-pi/autostart
```

5.2.3.4 Creating A .service File

See 4.4.3

A service to start a desktop/GUI is likely to fail without the following:

1. In the [Unit] section:

```
After=graphical.target
```

2. In the [Service] section:

```
User=pi
Group=pi
Environment="DISPLAY=:0"
```

   Replace “pi” and “:0” as required.[23]

3. In the [Install] section:

```
WantedBy=graphical.target
```

22 Or security must be severely relaxed on the X server. That’s outside the scope of this guide.
23 “:0” is the first X server (desktop) on the local machine.

5.2.3.5 Installing The Service

See 4.4.4

5.2.3.6 Advanced Usage

See 4.4.5

5.2.4 Without Automatic Login

5.2.4.1 Advantages

- More secure.
- The window manager is not running.
- Can use `cron`, `rc.local`, or a systemd service.

5.2.4.2 Disadvantages

- Some OS configuration is required.
- The window manager is not running.
- The started process must keep running or the X server will exit.

5.2.4.3 OS Configuration

Your Pi will need an internet connection to perform these steps.

1. Update your package lists:

   ```
   sudo apt update
   ```

2. Optional but recommended. Upgrade your installed packages:

   ```
   sudo apt full-upgrade
   ```

3. Install the required package:

   ```
   sudo apt install xserver-xorg-legacy
   ```

4. Configure it:

   ```
   sudo dpkg-reconfigure xserver-xorg-legacy
   ```

   Select `Anybody` when prompted.

5. Using sudo raspi-config, configure your Pi/OS to boot to the command line without automatic login.

6. Optional. Do not perform this step unless you have enabled ssh. Disable login on the console:

   ```
   sudo systemctl disable getty@tty1
   ```

5.2.4.4 Usage

See 4.2, 4.3, and 4.4 but:

Ensure your command contains the full path to the executable then prefix it with `/usr/bin/startx`. For example:

rc.local

```
/usr/bin/startx /usr/bin/mousepad &
```

or

```
sudo -u pi /usr/bin/startx /usr/bin/mousepad &
```

cron

```
@reboot /usr/bin/startx /usr/bin/mousepad
```

Systemd service

```
[Unit]
Description=test service

[Service]
Type=simple
User=pi
Group=pi
Restart=always
ExecStart=/usr/bin/startx /usr/bin/mousepad

[Install]
WantedBy=multi-user.target
```

In this OS configuration, `graphical.target` is never reached so cannot be used.

6 Desktop (GUI) Programs - Wayland/Wayfire

For all methods in this section first ensure Wayland, Wayfire, and the desktop are installed. The default RPiOS desktop is assumed.

6.1 When The Full Desktop Is Required

(Or when you can live with the security implications of using it)

6.1.1 Autostart Via wayfire.ini

6.1.1.1 Advantages

- It's relatively easy.
- It can (and should) be configured per user.

6.1.1.2 Disadvantages

- The user must be logged in to the desktop.
- It runs whenever the user logs in to the desktop or whenever the desktop is started from the command line.
- If the user does not login to or start the desktop the program will not be started.
- Services and resources the program depends on may not be available.
- There is little to no process control, only that provided by the shell.[24]
- Current working directory is the user's home directory.
- You must use the full path to any programs/scripts that are not in $PATH unless they are shell built ins or in the user's home directory.
- You must use the full path to any files passed to the program/script you're starting unless they are in the user's home directory.
- The program/script you're starting must use the full path to any files it references within its code unless these files are in the user's home directory.

24 `kill`, `jobs`, `fg`, `bg`, `nice`, `ps`, etc.

6.1.1.3 Usage

1. Open $HOME/.config/wayfire.ini in your preferred text editor.
2. Locate the [autostart] section. If one does not exists it can be added at the end of the file.
3. For each item you wish to start add a line in the following format:

   ```
   unique id = some command
   ```

   e.g.

   ```
   terminal = lxterminal
   ```

Wayfire does not appear to support the same `@` prefix feature[25] that LXDE does so you'll need to handle exit detection and restart your self, for example by wrapping the actual command in a small bash script such as this:

```
#!/bin bash
while true; do
    #your command goes here
    lxterminal
done
```

25 Automatic restart if the process exits.

6.1.2 Autostart Via A .desktop File

6.1.2.1 Advantages

- It’s relatively easy.

6.1.2.2 Disadvantages

- The user must be logged in to the desktop.
- It runs whenever the user logs in to the desktop or whenever the desktop is started from the command line.
- If the user does not login to or start the desktop the program will not be started.
- Services and resources the program depends on may not be available.
- There is little to no process control, only that provided by the shell.[26]
- Current working directory is the user’s home directory.
- You must use the full path to any programs/scripts that are not in $PATH unless they are shell built ins or in the user’s home directory.
- You must use the full path to any files passed to the program/script you’re starting unless they are in the user’s home directory.
- The program/script you’re starting must use the full path to any files it references within its code unless these files are in the user’s home directory.

6.1.2.3 Usage

1. Create the autostart directory:

```
mkdir -p /home/pi/.config/autostart
```

2. Open a new text file in your preferred editor.
3. Add the following three lines:

```
[Desktop Entry]
Type=Application
Exec=mousepad
```

   Replace `mouespad` with the command of your choice.

4. Save in `/home/pi/.config/autostart` with a name ending in `.desktop` e.g. `mousepad.desktop`
5. Logout and login or reboot.

26 `kill`, `jobs`, `fg`, `bg`, `nice`, `ps`, etc.

6.2 When The Full Desktop Is Not Required

I have yet to find an easy, safe, and reliable method to do this under wayland/wayfire. My recommendation is that you switch from Wayland to X11 (`sudo raspi-config`, Advanced Options, A6 Wayland) and see 5.2.

7 Hints And Tips

- If you don’t know the full path to a program `which program-name` e.g. `which mousepad` will usually return it.
- When troubleshooting, the first step should always be to capture and log all error messages.
- For python programs:
  - Ensure you run them with the correct version of python. `python` for python 2 programs, `python3` for python 3.
  - When running as a different user ensure all additional modules have been installed for all users. I.E. with `sudo pip` rather than just `pip`[27].
- When using any of the methods in 5.2.4 only a single program can be started. If more than one are needed, wrap them with a shell script and have `startx` call that instead.

27 pip3 for python 3

8 Troubleshooting

8.1 First Find The Error

Without knowing the error being thrown any attempt at fixing the problem is based on guesses and will likely take much more time than necessary.

When specifying log file names always use the full path to the log file.

8.1.1 .bashrc and .profile

Output and errors go to the terminal, ssh session, or console login that caused them to be run. This is often inconvenient and does not provide a permanent record of the error.

See 4.1.4.2 for details on capturing output and errors.

8.1.2 /etc/rc.local

Output and errors go to the text console[28] and are interleaved with all other output. This is often inconvenient and does not provide a permanent record of the error.

See 4.2.4.2 for details on capturing output and errors.

8.1.3 Cron

On a default Raspberry Pi OS installation all output and errors from a cron job are discarded.

See 4.2.4.2 for details on capturing output and errors.

8.1.4 Systemd Services

By default output and errors go to the systemd journal. This can be viewed using the `journalctl` command. It may be necessary to run `journalctl` with sudo to see the entire log.

Systemd services do not support shell style output redirection. See 4.4.5.3 for details.

Python programs run via systemd must pass the `-u` command line option to the interpreter or output and errors will be lost. For example:

```
ExecStart=/usr/bin/python3 -u /home/pi/myprogram.py
```

28 And serial console if enabled

8.1.5 Autostart Via The autostart File

The autostart file is not a shell script and does not support output redirection. Use a small wrapper shell script instead of calling the program directly.

For example:

myprogram.sh:

```
#!/bin/bash
/usr/bin/python3 -u /home/pi/myprogram.py >/home/pi/myprogram.log 2>&1
```

autostart:

```
/home/pi/myprogram.sh
```

8.1.6 Autostart Via A .desktop File

As with the autostart file output redirection is not supported. Use a small wrapper shell script instead of calling the program directly.

For example:

myprogram.sh:

```
#!/bin/bash
/usr/bin/python3 -u /home/pi/myprogram.py >/home/pi/myprogram.log 2>&1
```

my.program.desktop:

```
[Desktop Entry]
Type=Application
Exec=/home/pi/myprogram.sh
```

8.1.7 GUI Programs Without The Full Desktop

The method to capture output and errors varies depending on the method used to start the program. Refer to the appropriate sub section above.

8.2 Once You Have The Error

You have a place to start debugging from.

8.3 Common Issues, Their Causes, And Potential Solutions

This is not an exhaustive list and is in no particular order.

8.3.1 It works in Thonny or in a logged in shell but not when started at system boot.

Applies to:	All methods
Cause:	Differences in environment
Fix:	Start by finding the error message

Many of the problems below can be traced back to this.

8.3.2 Log File Is Empty

Applies to:	All methods
Cause:	1. Your command or program has generated no output 2. No attempt has been made to start your command or program: Your crontab entry, systemd service, etc is invalid Your systemd service is not enabled Your line in rc.local is below the exit 0 line A previous line in rc.local never returns. A user has yet to log in so .bashrc and/or .profile have not been run.
Fix:	1. Add some debugging output 2. Depends on the cause

8.3.3 Command or file not found

Applies to:	All methods
Cause:	The command or file you are trying to run is not in the default working directory or in a directory contained in the default $PATH
Fix:	Use the full path to the command or file

8.3.4 My Program Can’t Find A file It Needs

Applies to:	All methods
Cause:	1. Not using the full path to the file in your command or program. 2. Running as a different user
Fix:	1. Use the full path to the file 2. Run as the development user or ensure that the new user has correct permissions on the file and directory.

8.3.5 Output Files Are Not Created Or Updated Or Are Written In The Wrong Place

Applies to:	All methods
Cause:	1. Not using the full path to the file in your command or program. 2. Running as a different user
Fix:	1. Use the full path to the file 2. Run as the development user or ensure that the new user has correct permissions on the file and directory.

8.3.6 Command Or Program Cannot Access The Network

Applies to:	cron, rc.local, systemd services
Cause:	Starting thing before the network is available on online.
Fix:	1. Modify your code to include suitable error trapping and handling.[29] 2. Change your start methods and dependencies so that it occurs after the network is available. 3. Configure your OS to wait for a network connection at boot. 4. All of the above.

8.3.7 Python Complains That a Method, Class, etc. Is Unknown

Applies to:	All methods
Cause:	Running under the wrong version of python.
Fix:	Run using the correct version of python

8.3.8 Python Cannot Import A Module

Applies to:	All methods
Cause:	1. Running under the wrong python version 2. Module installed for the wrong python version 3. Module installed in a virtual environment 4. Module installed for a single user
Fix:	1. See 8.3.7 2. (Re)install the module for the correct python version 3., 4. Install the module for all users system wide

29 Outside the scope of this guide but this is he rpeferred action as network outages can occur at any time.

8.3.9 Python Cannot Import A Custom Module

Applies to:	All methods
Cause:	1. Module is not in the current working directory 2. Module location is not on the current PYTHONPATH 3. Virtual environment not active
Fix:	1. Change the current working directory before launching your python script or within your script but before attempting the import. 2. Add the module's location to PYTHONPATH before attempting the import.[30] 1., 2. Package your python script and associated modules into a .zip file and pass the zip file to the python interpreter 3. See 8.3.11

8.3.10 Cannot Open Display

Applies to:	cron, rc.local, systemd services, .bashrc, .profile
Cause:	1. Starting the program before the desktop/GUI has started 2. Not specifying which display to connect to. 3. Both.
Fix:	1. Modify the method use as detailed in 5 2. Set the DISPLAY environment variable before starting the program e.g. `DISPLAY=:0 python3 myprogram.py`[31]

8.3.11 Python and Virtual Environments

While not a new python feature, changes made to RPiOS in the Bookworm release make it more likely that they are in use.
This is mostly likely to be seen by a failure to load a python module that was installed with `pip`.

Applies to:	All methods
Cause:	1. Virtual environment not active 2. Running the system python instead of the venv python 3. Wrong shebang in the python script
Fix:	1. Activate the virtual environment before launching your python programme 2, 3. Use the python interpreter from the virtual environment instead of the system one. E.g use /home/pi/.venv/bin/python instead of /usr/bin/python

A full discussion of virtual environment and their impact is outside the scope of this guide. Some advice is given in https://github.com/thagrol/Guides/blob/main/bookworm.pdf.

30 Detailed instructions are outside the scope of this guide. Refer to the documentation for your chosen python version.

31 Other methods are possible including passing it as a command line option to some commands.

9 Change Log

9.1 2023-11-21

Retitled section 6.

Added new section 7

Renumber old sections 7 and upwards

Added section 8.3.11

Added change log