Building from git clone

-----------------------

``` shell

$ ./bootstrap

$ ./configure

```

The bootstrap script will use autotools to set up the build environment

and create the `configure` script.

Use `./configure --help` for options. Use `--prefix` to make an install in

your home directory. This is necessary to test python scripts. The systemd

user unit directory should be set to avoid writing to the system location.

Systemd will look for the unit files in `~/.config/systemd/user` so this

directory can be used as a target if the unit files will be used. Otherwise

the location can be set to `no` to disable the systemd files.

Example:

``` shell

$ ./configure --prefix=$HOME/redshift/root \

--with-systemduserunitdir=$HOME/.config/systemd/user

```

Now, build the files:

``` shell

$ make

```

The main redshift program can be run at this point. To install to the

prefix directory run:

``` shell

$ make install

```

You can now run the python script. Example:

``` shell

$ $HOME/redshift/root/bin/redshift-gtk

```

Dependencies

------------

\* autotools, gettext

\* intltool, libtool

\* libdrm (Optional, for DRM support)

\* libxcb, libxcb-randr (Optional, for RandR support)

\* libX11, libXxf86vm (Optional, for VidMode support)

\* Glib 2 (Optional, for GeoClue2 support)

\* python3, pygobject, pyxdg (Optional, for GUI support)

\* appindicator (Optional, for Ubuntu-style GUI status icon)

Ubuntu users will find all these dependencies in the packages listed in ``.travis.yml``.

Coding style for C code

-----------------------

Redshift follows roughly the Linux coding style

<http://www.kernel.org/doc/Documentation/CodingStyle>. Some specific rules to

note are:

\* Lines should not be longer than 80 characters in new code. If lines are

longer than this the code could likely be improved by moving some parts to a

smaller function.

\* All structures are typedef'ed.

\* Avoid Yoda conditions; they make the logic unnecessarily hard to comprehend.

\* Avoid multiline if-statements without braces; either use a single line or add

the braces.

\* Use only C-style comments (`/\* \*/`).

Creating a pull request

-----------------------

1. Create a topic branch for your specific changes. You can base this off the

master branch or a specific version tag if you prefer (`git co -b topic master`).

2. Create a commit for each logical change on the topic branch. The commit log

must contain a one line description (max 80 chars). If you cannot describe

the commit in 80 characters you should probably split it up into multiple

commits. The first line can be followed by a blank line and a longer

description (split lines at 80 chars) for more complex commits. If the commit

fixes a known issue, mention the issue number in the first line (`Fix #11:

...`).

3. The topic branch itself should tackle one problem. Feel free to create many

topic branches and pull requests if you have many different patches. Putting

them into one branch makes it harder to review the code.

4. Push the topic branch to Github, find it on github.com and create a pull

request to the master branch. If you are making a bug fix for a specific

release you can create a pull request to the release branch instead

(e.g. `release-1.9`).

5. Discussion will ensue. If you are not prepared to partake in the discussion

or further improve your patch for inclusion, please say so and someone else

may be able to take on responsibility for your patch. Otherwise we will

assume that you will be open to criticism and suggestions for improvements

and that you will take responsibility for further improving the patch. You

can add further commits to your topic branch and they will automatically be

added to the pull request when you push them to Github.

6. You may be asked to rebase the patch on the master branch if your patch

conflicts with recent changes to the master branch. However, if there is no

conflict, there is no reason to rebase. Please do not merge the master back

into your topic branch as that will convolute the history unnecessarily.

7. Finally, when your patch has been refined, you may be asked to squash small

commits into larger commits. This is simply so that the project history is

clean and easy to follow. Remember that each commit should be able to stand

on its own, be able to compile and function normally. Commits that fix a

small error or adds a few comments to a previous commit should normally just

be squashed into that larger commit.

If you want to learn more about the Git branching model that we use please see

<http://nvie.com/posts/a-successful-git-branching-model/> but note that we use

the `master` branch as `develop`.

Contributing translations

-------------------------

You can contribute translations directly at

[Launchpad Translations for Redshift](https://translations.launchpad.net/redshift).

Updated translations will be pulled back into the `po` files on Github

before a release is made.

Creating a new release

----------------------

1. Select a commit in master to branch from, or if making a bugfix release

use previous release tag as base (e.g. for 1.9.1 use 1.9 as base)

2. Create release branch `release-X.Y`

3. Apply any bugfixes for release

4. Import updated translations from launchpad and commit. Remember to update

`po/LINGUAS` if new languages were added

5. Update version in `configure.ac` and create entry in NEWS

6. Run `make distcheck`

7. Commit and tag release (`vX.Y` or `vX.Y.Z`)

8. Push tag to Github and also upload source dist file to Github

Also remember to check before release that

\* Windows build is ok

\* Build files for distributions are updated

Build Fedora RPMs

-----------------

Run `make dist-xz` and copy the `.tar.xz` file to `~/rpmbuild/SOURCES`. Then run

``` shell

$ rpmbuild -ba contrib/redshift.spec

```

If successful this will place RPMs in `~/rpmbuild/RPMS`.

Cross-compile for Windows

-------------------------

Install MinGW and run `configure` using the following command line. Use

`i686-w64-mingw32` as host for 32-bit builds.

``` shell

$ ./configure --disable-drm --disable-randr --disable-vidmode --enable-wingdi \

--disable-quartz --disable-geoclue2 --disable-corelocation --disable-gui \

--disable-ubuntu --host=x86\_64-w64-mingw32

```

Notes

-----

\* verbose flag is (currently) only held in redshift.c; thus, write all

verbose messages there.