
Venus OS: the Victron Energy Unix like distro with a linux kernel

The problematic part with this name is that it is from the Roman mythology and not, as most of our products, from the Greek. Phoenix is already taken though by a charger...

This readme documents how to compile and build Venus OS from source.

First, make sure that that is really what you want or need. It takes several hours to compile, lots of disk space and results in an image and sdk which are both already available for download as binaries `swu` and `sdk`.

Even when you are developing on one of the parts of Venus OS, for example one of its drivers, or the gui, its still not necessary to build the full Venus OS from source.

Make sure to read the Venus OS wiki first.

So, if you insist: this repo is the starting point to build Venus. It contains wrapper functions around `bitbake` and `git` to fetch, and compile sources.

For a complete build you need to have access to private repos of Victron Energy. Building only open-source packages is also possible (but not checked automatically at the moment).

Venus uses OpenEmbedded as build system.

Getting started

Building Venus requires a Linux. At Victron we use Ubuntu for this.

```
1 # clone this repository
2 git clone https://github.com/victronenergy/venus.git
3 cd venus
4
5 # install host packages (Debian based)
6 sudo make prereq
7
8 # fetch needed subtrees
9 # use make fetch-all instead, if you have access to all the private
   repos.
10 make fetch
```

That last fetch command has cloned several things into the `./sources/` directory. First of all there is `bitbake`, which is a make-like build tool part of OpenEmbedded. Besides that, you'll find `openembedded-core` and various other layers containing recipes and other metadata defining Venus.

Now its time to actually start building (which can take many hours). Select one of below example commands:

```
1 # build all, this will take a while though... it builds for all
  MACHINES as found
2 # in conf/machines.
3 make venus-images
4
5 # build for a specific machine
6 make ccgx-venus-image
7 make beaglebone-venus-image
8
9 # build the swu file only
10 make ccgx-swu
11
12 # build from within the bitbake shell.
13 # this will have the same end result as make ccgx-swu
14 make ccgx-bb
15 bitbake venus-swu
```

Configs

Above Getting Started instructions will automatically select the config that is used for Venus OS as distributed. Alternative setups can also be used, e.g. to build for a newer OE version:

```
1 make CONFIG=rocko fetch-all
```

To see which config your checkout is using, look at the `./conf` symlink. It will link to one of the configs in the `./configs` directories.

For each config there are a few files:

- `repos.conf` contains the repositories which need to be checked out. It can be rebuild with `make update-repos.conf`.
- `metas.whitelist` contains the meta directory which will be added to `bblayers.conf`, but only if they are actually present.
- `machines` contains a list of machines that can be build in this config

To add a new repository, put it in `sources`, then checkout the branch you want and set an upstream branch. The result can be made permanent with: `make repos.conf`.

Don't forget to add the directories you want to use from the new repository to `metas.whitelist`.

Using the repos command

Repos is just like `git submodule foreach -q git`, but shorter, so you can do:

```
./repos push origin ./repos tag xyz
```

It will push all, tag all etc. Likewise you can revert to a certain revision with:

```
./repos checkout tagname
```

managing git remotes and branches

```
1 # patches not in upstream yet
2 ./repos cherry -v
3
4 # local changes with respect to upstream
5 ./repos diff @{u}
6
7 # local changes with respect to the push branch
8 ./repos diff 'origin/`git rev-parse --abbrev-ref HEAD`'
9 or if you have git 2.5+ ./repos diff @{push}
10
11 ./repos log @{u}..upstream/`git rev-parse --abbrev-ref @{u} | grep -o "[a-Z0-9]*$" --oneline
12
13 # rebase your local checkout branches on upstream master
14 ./repos fetch origin
15 ./repos rebase 'origin/$checkout_branch'
16
17 # checkout the branches as per used config
18 ./repos checkout '$checkout_branch'
```

Releasing

```
1 # tag & push venus repo as well as all repos.
2
3 git tag v2.21
4 git push origin v2.21
5
6 ./repos tag v2.21
7 ./repos push origin v2.21
```

Maintenance releases

How to create a new maintenance branch The base branch on which the maintenance releases will be based is to be prefixed with a **b**.

This example shows how to create a new maintenance branch. The context is that master is already working on v2.30. Latest official release was v2.20. So we make a branch named b2.20 in which the

first release will be v2.21; later if another maintenance release is necessary v2.22 is pushed on top; and so forth.

```
1 # clone & make a branch in the venus repo
2 git clone git@github.com:victtronenergy/venus.git venus-b2.20
3 cd venus-b2.20
4 git checkout v2.20
5 git checkout -b b2.20
6
7 # fetch all the meta repos
8 make fetch-all
9
10 # clone, prep and push them
11 ./repos checkout v2.20
12 ./repos checkout -b b2.20
13 ./repos push --set-upstream origin b2.20
14
15 # update the used config to the new branch
16 make update-repos.conf
17 git commit -a -m "pin dunfell branches to b2.20"
18
19 # update the raspbian config to the new branch
20 [
21     Now manually update the raspbian config file, and commit that as well
22     .
23     See some earlier branch for example.
24 ]
25 git commit -a -m "pin raspbian branches to b2.20"
26
27 # Update gitlab-ci.yml
28 [
29     Now, modify .gitlab-ci.yml. See a previous maintenance branch for
30     how that is done.
31 ]
32
33 git commit -a -m "Don't touch SSTATE cache & build from b2.20"
34
35 # Push the new branch and changes to the venus repo
36 # Note that this causes a (useless) CI build to start on the builder
37 #   once
38 # it syncs. Easily cancelled in the gitlab ui.
39 git push --set-upstream origin b2.20
```

Now you're all set; and ready to start cherry-picking.

Full cherry-picks vs backporting patches Be aware that there are two ways to backport a change. One is to take a complete commit from the meta repositories; and the other one is to add patches from

the source repository. Where you can, apply method one. But in case the repository, for example mk2-dbus or the gui, has had lots of commits out of which you need only one; then you have to take just the patch.

The master rule when deciding against- or for inclusion Changes need to be either really small, well tested or very important

The eight golden rules of maintaining maintenance branches

1. only take changes from master: cherry-picking
2. don't add changes or new versions that are not in master yet
3. `git cherry-pick -x` appends a nice (cherry-picked from [ref]) line to the commit message
4. add and/or increase the PR when adding patches
5. drop the PR again when going to a clean version
6. when adding patches; add a `backported from` note just like this one to the commit message
7. go through the todo where the team is working on master, and add (`**backported to v2.22**`) or where applicable (`**backported to v2.22 as a patch**`) to each and every patch and version that's been backported.
8. double verify everything by cross referencing the todo, the commits logs from master as well as your own commit log.

Building a maintenance release To build, create a pipeline on the mirrors/venus repo, and run it for the maintenance branch. No variables needed.

Various notes

1. Linux update If you encounter problems like this: `* Solver encountered 1 problem(s): * Problem 1/1: * - nothing provides kernel-image-4.14.67 needed by packagegroup-machine-base-1.0-r83.einstein`

if can be fixed with: `make einstein-bb bitbake -c cleanall packagegroup-machine-base`
and thereafter try again