Nix on Sailfish X (Sailfish OS for Sony Xperia)

DROWNING IN SOFTWARE FROM NIXPKGS,
THE LARGEST SOFTWARE PACKAGE SET

Date put out	Time burden	Word score	Keywords
2026-05-30	10–16 minutes	3208	nix sailfish os mobile os how to

Writer	Crafted in	License
toastal ＜https://toast.al＞	[illegible]	CC-BY-NC-SA-4.0 *(Creative Commons Attribution Non Commercial Share Alike 4.0 International)* ＜https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode＞

Sailfish OS is a partially-open, GNU *(GNU's not Unix!)*/Linux-based mobile OS *(operating system)* to compete with the U.S. *(United States of America)*-based, privacy-draining, you-no-longer-truly-own-your-device,nannying Android / iOS duopoly. That was a mouthful, but we need to address the fact that *we* need to be taking back our freedoms, privacy, sovereignty from these megacorporations. Step one is planning your duopoly off-ramp—& while there are quite a few Linux options, Sailfish OS is a viable one of those options & the one I chose (since I like Sony hardware, & you will want microSD support for a Linux phone—if not for your phones generally until the OEM *(original equipment manufacturer)*s too them away from us). We will be looking to augment the small package set Jolla provides by installing Nix ＜https://nixos.org＞ + *Nixpkgs*, which will give us access to the largest package set for software out there where much of it is compiled for `aarch64` consumption! This is the set up I chose

(& would recommend given it's what I did), but readers are to take part of the *free* part of free software & make modifications.

Quick background

I actually set this up on my device a while back & I loved the novelty I showing folks *proper* Nix running on a phone (the NixOS phone project doesn't support much unfortunately), but in hindsight I would have made some changes. The Sailfish OS 5.1.0.7 update (which I was so excited to get the notification for!) ultimately went bad for me—& it seems enough other users also had issuse that Jolla paused the update & then created a 5.1.0.8 patch to help with the storage calculation situation. I was able to get back up & running without losing any data, but I decided to grow space for the root partition by culling space / rebuilding the nix partition as I wanted to start over since I mistakenly used the Determinate Nix installer instead of the official one & got stranded on their proprietary daemon fork after an upgrade since they stopped supporting the upstream Nix installation[*]. Needless to say, I would be better off just starting over on Nix… while actually taking notes.

Laying the seabed

Device layout, file systems, & where to put Nix

On the Sailfish side, we know that the device setup uses LVM *(Logical Volume Manager)* for a logical volume layout separating `root` & `home`, with `home` being LUKS *(Linux Unified Key Setup)*-encypted, & both on ext4 *(fourth extended filesystem)*. But within this layout `/nix` will need a location. The Nix folder has some special requirements that deserve some thought:

- The Nix store should never have plaintext sensitive data, so no need for encryption overhead
- Nix, without setting bounds, will balloon & eat a *ton* of space on disk
- Broadly, Nix is write once, read many I/O *(input/output)* with append-only/immutable behavior instead of mutating files
- Heaps of tiny `*.drv` & `.nar` files

With this workload, I will opt for a separate LVM *(Logical Volume Manager)* volume to prevent any runaway chance of Nix growing beyond a size I want, while using F2FS *(Flash-Friendly File System)*

<https://www.kernel.org/doc/html/latest/filesystems/f2fs.html> with neither fscrypt nor LUKS *(Linux Unified Key Setup)* encryption. While ext4 *(fourth extended filesystem)* is a safe default here (& actually has tools Sailfish repositories ship with), & the differences have been narrowing, I still can't quit F2FS *(Flash-Friendly File System)* in this scenario since:

- It's optimized for NAND flash like eMMC & UFS
- `inline_data` can store `.drv` & `.nar` files in the inode
- `inline_dentry` can help query the ocean that is `/nix/store`
- `atgc` can help identify hot vs. *(versus)* cold data to clean / move data in a manner to reduce writes
- `gc_merge` to only GC when FS isn't under pressure or even more optimally, script a hybrid TRIM sach as after a Nix garbage collection or big `home-manager` build after `nixtamal refresh`
- in the future if we get a Linux kernel update beyond 5.4.x we could add compression like `lz4` to further reduce writes & checksum compression

The downsides:

- F2FS *(Flash-Friendly File System)* can grow (offline) but not shrink (hence why we are starting over, but had I not needed to claim space for the root expansion, I wouldn't be touching anyhow)
- Neither Sailfish nor even Chum provide `f2fs-tools` so we will need to build manually; it doesn't demand a big build time or toolchain, it's just annoying (but we can build with proper `mtune` as a silver lining—as well as having tooling to format any microSD to something clearly optimized for it)
- Recovery tools are not as good (but since we aren't *relying* on Nix, it's a 🤷)
- Not as good boring as ext4 *(fourth extended filesystem)*

Meh:

- Performance: it will differ depending on task & drive, but every time I see ext4 *(fourth extended filesystem)* & F2FS *(Flash-Friendly File System)* compete in a benchmark, it's a neglible real-world wash—with both regularly getting perf *(performance)* bumps about every other kernel

The TL;DR *(too long; didn't read)* being: I'll take the cost of building `f2fs-tools` knowing the kernel will have no issuse with F2FS *(Flash-Friendly File System)* & it ultimately should reduce long-term wear on the drive even with drives these style of drives having some in-built ability to help reduce writes.

Discovery of our Recovery

> IMPORTANT
>
> You should already have finished installing following [Sailfish X Guide page <https://jolla.com/sailfishxinstall/>](https://jolla.com/sailfishxinstall/).

Just like seasoned Android ROM *(read-only memory)* fans recognize, given that these were once Android devices, we will need to do some recovery in the form of flashing some images. These images will come in ZIP achive from either the [trials <https://shop.jolla.com/downloads/>](https://shop.jolla.com/downloads/) page or via purchasing a license.

> NOTE
>
> For whatever odd reason, they only let you purchase a license if you are in Europe? 🤨

Unzipping the archive, & changing into its extract directory, we get...

```
$ find . -type f -iname "*.img"
./vendor_boot.img
./hybris-boot.img
./hybris-recovery.img
./dtbo.img
```

Take a wild guess which one of these images is gonna be used. Jolla's published a [Recovery Guide <https://docs.sailfishos.org/Support/Help_Articles/Recovery_Mode/>](https://docs.sailfishos.org/Support/Help_Articles/Recovery_Mode/) too you should follow.

```
$ fastboot flash boot_a hybris-boot.img
$ fastboot flash boot_b hybris-boot.img
$ fastboot reboot
```

This puts our device into a state where once attached to a capable machine with `telnet`, we can phone in with `telnet 10.42.66.66` (or whatever the address the tiny font on the display says). Once the `telnet` is executed, whisper you best ["I'm in" Trinity impression]

<https://www.youtube.com/watch?v=1eRxp_r9Qx4&t=53s>, & read the menu options on screen. I might recommend picking the file system check option first just to make sure all is well & it happen quickly. Otherwise/after, choose the shell option. You'll be dropped into a BusyBox environment with very minimal tooling, but it will be enough since we can,

```
# chroot /rootfs
```

chroot ourselves into Sailfish OS environment with its root tools. Now with more tools at our disposal:

```
# lsblk -o NAME,SIZE,FSTYPE,MOUNTPOINT /dev/sda75
NAME              SIZE FSTYPE MOUNTPOINT
sda75           101.8G LVM2_m
├─sailfish-root     5G ext4   /
└─sailfish-home  96.8G crypto
  └─luks-$UUID   96.8G ext4   /home
# lvs
LV   VG       Attr       LSize
root sailfish -wi-ao---- 5-ish-g
home sailfish -wi-ao---- rest-of-disk-g
# dmsetup ls
sailfish-root   (253:0)
luks-$GUID      (253:3)
sailfish-home   (253:1)
```

We can now see we have LVM *(Logical Volume Manager)* set up which is good since we can add a new logical volume. Our home uses LUKS *(Linux Unified Key Setup)* encryption. Both using ext4 *(fourth extended filesystem)*, in this case we get important ext4 *(fourth extended filesystem)*-related wins in that we can not only shrink/grow but do this online; this matters since we need to be `chroot`ed into `/rootfs` as the recovery environment's BusyBox sadly isn't shipping with the file-system-related tools we would need to do the operation (tho I guess you might be able to find the binaries & run them from the mount, but I would stick to `chroot` as this is well-worn ground).

NOTE

If the ability to grow/shrink online is something that matters to you or sounds too useful, just make your Nix partition ext4 *(fourth extended filesystem)* instead of F2FS *(Flash-Friendly File System)* —you'd get no

judgment from me, just remember to read the manual & tune it for Nix workloads. I might judge you for choosing ext4 *(fourth extended filesystem)* over F2FS *(Flash-Friendly File System)* on the microSD though… 🙂 … where, fun fact: yes, Sailfish OS will properly prompt you in the UI *(user interface)* if you use fscrypt to encrypt the partition.

Making space

TIP

Since we are resizing the home partition, you should **really** consider expanding the space here to grow your root partition too since 5 GB *(gigabyte)* is a tight squeeze after installing like 2 app *(application)*s.

First, how much space? I plan to expand `root` 10 GB *(gigabyte)*, shrink `home` to 72 GB *(gigabyte)*, & give the rest, 19.8 GB *(gigabyte)*, to `nix`. Knowing I can expand either of `root` or `nix` in the future, this should be good enough for now. Meaning the goal is:

```
# lsblk -o NAME,SIZE,FSTYPE,MOUNTPOINT /dev/sda75
NAME              SIZE FSTYPE      MOUNTPOINT
sda75           101.8G LVM2_member
├─sailfish-root    10G ext4        /
├─sailfish-home    72G crypto_LUKS
│ └─luks-$UUID     72G ext4        /home
└─sailfish-nix   19.8G f2fs        /nix
```

Reducing `home`

```
# e2fsck -f /dev/mapper/sailfish-home
# resize2fs /dev/mapper/sailfish-home 72G
# lvreduce -L 72G /dev/sailfish/home
# cryptsetup resize sailfish-home
```

Extending `root`

```
# lvextend -L 10G /dev/sailfish/root
# resize2fs /dev/sailfish/root
```

Creating `nix`

```
# lvcreate -l 100%FREE -n nix sailfish
```

With the space freed & a volume, it's going to be more comfortable/easier to finish by rebooting into the full OS where it'll be easier to get the rest of the tooling.

Mission Possible: Building `f2fs-tools`

> *Ah Shit, Here We Go Again*
>
> —Carl "CJ" Johnson, straight busta

Time to build some software the way our forefathers did…

```
# pkcon refresh
# pkcon update
# pkcon install git
# cd /opt
# git clone https://git.kernel.org/pub/scm/linux/kernel/git/jaegeuk/f2fs-tools.git
# cd f2fs-tools
```

After reading the README & skipping SELinux since it isn't needed for us (well, & there is a version mismatch between Chum & Jolla repo *(repository)*s, which is what we would be avoiding with Nix—can't make this up)…

```
# pkcon install autoconf make libtool libuuid-devel
# ./autogen.sh
# ./configure --without-selinux CFLAGS="-O2 -mcpu=cortex-a55+crypto"
# make -C lib -j4
# make -C fsck -j4
# make -C mkfs -j4
# make -C lib install
# make -C fsck install
# make -C mkfs install
# fsck.f2fs -V
fsck.f2fs 1.16.0 (2026-01-01)
```

```
# mkfs.f2fs -V
mkfs.f2fs 1.16.0 (2026-01-01)
```

Mission: complete

Setting up F2FS *(Flash-Friendly File System)*

```
# mkfs.f2fs -O extra_attr,flexible_inline_xattr,inode_checksum,sb_checksum,compressior

    F2FS-tools: mkfs.f2fs Ver: 1.16.0 (2026-01-01)

Info: Debug level = 0
Info: Label = nix
Info: Trim is enabled
Info: Enable Compression
Info: [/dev/sailfish/nix] Disk Model: MT128GAXAU2U227X
Info: Segments per section = 1
Info: Sections per zone = 1
Info: sector size = 4096
Info: total sectors = 5191680 (20280 MB)
Info: block size = 4096
Info: zone aligned segment0 blkaddr: 512
Info: format version with
  "Linux version 5.4.210-sodp (abuild@phost23) (Android (10087095, +pgo, +bolt, +lto,
Info: [/dev/sailfish/nix] Discarding device
Info: This device doesn't support BLKSECDISCARD
Info: Discarded 20280 MB
Info: Overprovision ratio = 1.040%
Info: Overprovision segments = 109 (GC reserved = 103)
Info: format successful
```

NOTE
Setting compression flag now in hopes (tho doubtful) that there may be a kernel update in the future to allow it

Give it a mount...

```
# mount -t f2fs /dev/mapper/sailfish-nix -o rw,noatime,nodiratime,compress_algorithm=l
# lsblk -o NAME,SIZE,FSTYPE,MOUNTPOINT,UUID /dev/sda75
NAME              SIZE FSTYPE      MOUNTPOINT UUID
sda75           101.8G LVM2_member            …
├─sailfish-root    10G ext4        /          …
```

```
├─sailfish-home    72G crypto_LUKS            …
│ └─luks-$UUID     72G ext4        /home      …
└─sailfish-nix   19.8G f2fs        /nix       $NIX_UUID
```

…copy that UUID & adding to `fstab`

```
# vi /etc/fstab
```

```
UUID=$NIX_UUID /nix f2fs defaults,rw,noatime,nodiratime,compress_algorithm=lz4,inline_
```

Doing a Nix

“Which Nix?”

Nix, Lix, Snix, Determinate Nix are some of the popular names, but maybe you never thought you should even ask. Snix isn’t quite ready yet. Determinate Nix is proprietary / not geared towards my use cases. I would like to try out Lix again, but lacks experimental `blake3-hashes`, so I will have to pass again. Which leaves us with Nix. Kinda uninteresting result, but it does mean we think about the which *installer*—more specifically: **do not use the Determinate Nix installer if you don’t intend to use Determinate Nix**. I was advocated for hard in the past for legitimately providing a nicer UX *(user experience)* but ultimately leaves users with their fork now. So we use the basic installer: https://nixos.org/download/ <https://nixos.org/download/>.

Installer, I hardly kn…

This threw me for a loop… you will need to GNU *(GNU’s not Unix!)* Bash (not the BusyBox symlink) to for the installer since the official installer uses Bashisms (plague of many scripts). You might need this too (you can undo later, but this is a bit sketch as a symlink to begin with):

```
$ pkcon remove busybox-symlinks-bash
$ pkcon install gnu-bash
```

Multiuser setup:

```
# curl --proto '=https' --tlsv1.2 -L -o nix-installer.sh https://nixos.org/nix/install
# cat nix-installer.sh
… to make sure nothing unexpected is here
…
# chmod +x nix-installer
# ./nix-installer --daemon
…
Alright! We're done!
Try it! Open a new terminal, and type:

  $ nix-shell -p nix-info --run "nix-info -m"

Thank you for using this installer. If you have any feedback or need
help, don't hesitate:

You can open an issue at
https://github.com/NixOS/nix/issues/new?labels=installer&template=installer.md

Or get in touch with the community: https://nixos.org/community

---- Reminders ---------------------------------------------------------------
[ 1 ]
Nix won't work in active shell sessions until you restart them.



# nix-shell -p nix-info --run "nix-info -m"
Stack size hard limit is 2097152, which is less than the desired 62914560. If possible
 - system: `"aarch64-linux"`
 - host os: `Linux 5.4.210-sodp, Sailfish OS, 5.1.0.8 (Pispala), nobuild`
 - multi-user?: `yes`
 - sandbox: `yes`
 - version: `nix-env (Nix) 2.34.7`
 - channels(root): `"nixpkgs"`
 - nixpkgs: `/nix/store/j45fm5ffm7agj6pbsvkqbq7778rrxadc-nixpkgs/nixpkgs`
```

Add Nix experimental features that offer a better experience, then we need to enable + restart the systemd service since our settings changed & we want the daemon running on boot

```
# echo "extra-experimental-features = nix-command fetch-tree blake3-hashes" >>/etc/ni
# systemctl enable nix-daemon.service
# systemctl restart nix-daemon.service
```

□□□□

`home-manager`

Yes, still using `home-manager`. Even if it has its flaws, managing dotfiles like this as a good use case (text editor, CLI *(command line interface)* tools, fonts). I have a local config checkout @ `~/nixcfg`.

```
# su defaultuser
$ cd ~/nixcfg
$ nix-build host/$HOSTNAME/home.nix
$ result/activate
```

Sadly, the `home-manager --file` option doesn't work like *I* prefer, so I will just build the activation script, then run it. In the future, I should make a script that runs this… but also disables the F2FS *(Flash-Friendly File System)* GC until activation script is run.

`system-manager`

Would like to use it again to handle system-level config/packages, but it only works with experimental flakes. Having touched enough flaked Nix code I saw its severe design flaws, but also how it tends to get Nix users writing Nix code that isn't very good (or compatible) either & eventually just stripped it out. This isn't to say `system-manager`'s Nix code is bad like I just mentioned, it's just there wasn't any way to use the `system-manager` binary with stable Nix code—requiring, as far as I can tell, that configuration be a non-standard extension of the flake manifest file. Maybe if they address classic Nix support (or I look deeper into how to run it without flakes despite their locked-in design), I would reconsider. If you know of an alternative contact me </contact>.

Why not Chum? It has a larger package repository…

Chum <https://sailfishos-chum.github.io> is "a software repository by the community". Noble goal—especially since some packages even use `cortex-a55` mtune flags, unlike Nixpkgs conservative, generic optimization (not even `x86_64-v2`!)—but it has some issue:

1. My first query, `f2fs-tools`, returns nothing (mind you, this is after `pkcon search f2fs-tools` on the Jolla repo returned nothing) so it's not all-encompassing

2. This will never allow declarative config & will need to fight issues that other distro *(distribution)*s do with multiple libary version needed. Once you have tried Nix + NixOS, you will try to avoid stateful config.
3. It's duplicating a lot of the efforts of Nixpkgs, but with a much smaller community to keep everything update.
4. JFC, I want *less* projects that require me to have an account with Microslop-owned proprietary Git forge / social media platform, in GitHub… I've been philosphically opposed since the buy-out, but with their recent hacks, terrible uptime, YAML *(Yet Another Markup Language)*-based bad CI *(continuous integration)*, & focusing on marketing bits like growth hacking <https://www.timlrx.com/blog/growth-hacking-github-how-to-get-github-stars/> (a.k.a. *(also known as)* star hacking <https://awesomeagents.ai/news/github-fake-stars-investigation/>) over the actual code, why are projects still adopting this garbage? I already have an account, required to participate in Nixpkgs since it won't/can't move off[†], but if I can avoid exposure to a Microslop-hosted project, I will. It's also counter to the primary reason layfolk are of choosing a Sailfish OS phone at present by being EU *(European Union)*-based & an alternative to the megacorporate status quo …so then a community-run project hosts code/bug tracking on a platform that requires accounts with that class of companies ruining society (possibly leading to civil unrest with this AI *(artificial intelligence)* job displacement stuff 🤦)? Jolla is guilty of it too, but at least as a for-profit at the end of the day, I am not surprised, but being community-based & not *that* old, it doesn't have a legacy excuse to exist on that platform… fuck no, stand for something. (I'd say `/rant` but we both know the rant won't stop until the situation improves, & if users don't raise a voice, everyone will think these sorts of choices are *okay* when they aren't —but there's still time to course correct).

So since it doesn't have the software I will be needing & the philosophy sucks, we will be passing for something I know will work.

Takeaway

Nix (& Guix) can coexist on existing operating systems. This is a great power since NixOS (& Guix System) aren't the right option for many use cases since the OS might be tied for firmware or other reasons—such as a phone. This means you can still get 60%[‡] of the benefits of these declatative OS *(operating system)*s without messing with the core. In my case, it lets me bring all of my familar CLI *(command line interface)* tools to the computer in

my pocket despite the megacorporate OS *(operating system)*s trying to get users to treat it as a kiosk to proprietary services. I find this empowering — a way to take back some of my freedom in a familar way, while sharing configurations between the 5 computers I active run so I never feel ‘naked’ once I SSH *(Secure Shell)* into those machine when I need to, with the added bonus of getting to `nix-shell -p …` basically *any* missing piece of software.

---

[*] Was the Derminate Nix Installer a rug pull? Is it no longer a ‘rug pull’ since a 1 year notice was given? Was dropping support always the goal, but the official Nix installation not an olive branch but a Trojan horse to get their proprietary Nix fork into your org? 🤔 𝒥𝔲𝔰𝔱 𝕒𝐬𝕜𝒊𝔫′ 𝑞𝔲𝑒𝔰𝐭𝒊𝜎𝒏𝐬…

[†] Nixpkgs can’t move partially from vendor locking-into the Microslop tools, but also it’s a giant monorepo instead of being split up (I applaud *Aux* <https://auxolotl.org/> for at least trying to break up Nixpkgs) so it needs so much extra infrastructure resources (hard problem) which also relies on the shoddy, inefficient *centralized pull request model* that doesn’t scale to other platforms. There absolutely is a sect of the Nixpkgs user base that definitely wants to go — & the risks have been assessed — but we are stuck in a `WONTFIX` limbo given the effort to change the entire workflow to work on something that can be self-hosted or hosted in a safer-for-sovereignty place. But this is **precisely** the sort of *cautionary tale* of why you *don’t* get yourself stuck in their situation from jump.

[‡] 60% is arbitrary… but I really think many, like the `nix-darwin` crowd, underestimate how much extra value you get once you let Nix take over the *entire* system, rather than just a convenient/reproducible way to get a lot of software.

If you liked the post or found it useful, you can help me with a spare buck </funding> or reach out </contact> if you know of any remote job openings or contract work.

𐑦𐑓 𐑿 𐑤𐑲𐑒𐑑 𐑞 𐑐𐑴𐑕𐑑 𐑹 𐑓𐑬𐑯𐑛 𐑦𐑑 𐑿𐑕𐑓𐑫𐑤, 𐑿 𐑒𐑨𐑯 𐑣𐑧𐑤𐑐 𐑥𐑰 𐑢𐑦𐑞 𐑩 𐑕𐑐𐑺 𐑚𐑳𐑒 </funding>
𐑹 𐑮𐑰𐑗 𐑬𐑑 </contact> 𐑦𐑓 𐑿 𐑯𐑴 𐑝 𐑧𐑯𐑰 𐑮𐑰𐑥𐑴𐑑 𐑡𐑪𐑚 𐑴𐑐𐑩𐑯𐑦𐑙𐑟 𐑹 𐑒𐑪𐑯𐑑𐑮𐑨𐑒𐑑 𐑢𐑻𐑒.