

i have a weird fedora 43 btrfs subvolumes layout for timeshift

how can i backup?  
is using live cd only option?

You can back up a *weird* Timeshift+Btrfs subvolume layout on Fedora 43 without relying on a live ISO, but whether you **should** depends on how broken the layout is and what you want to preserve.

Here's the straight rundown.

---

## ✓ 1. You *do not* need a live CD just to back up a Btrfs system

As long as your system boots normally and the subvolumes mount without corruption, you can back up from the running system. Btrfs is copy-on-write; snapshots are consistent even when created live.

Tools that work without a live ISO:

- `btrfs send` / `btrfs receive`
- `timeshift --create` (if your layout is valid enough for Timeshift to run)
- Manual `rsync` backup of subvolumes
- `snapper` (if installed, but likely not your case)

Backing up from a live ISO is usually only necessary when:

- the filesystem is damaged,

- the layout is too misconfigured to allow a clean snapshot,
  - you want a cold, unmounted backup for safety.
- 

## ✓ 2. If Timeshift layout is “weird,” avoid relying on Timeshift alone

Timeshift assumes a specific subvolume structure:

```
python  
  
@ (root)  
@home  
@var/*
```

If yours differs—even slightly—Timeshift may create incomplete backups.

If you’re unsure, **don’t rely on Timeshift as your only backup.**

---

## ✓ 3. Safest robust backup method: Btrfs send/receive

This bypasses Timeshift completely and backs up exactly what exists in your filesystem.

### Step 1—Create a read-only snapshot of your root

```
bash
```

```
sudo btrfs subvolume snapshot -r / /.snapshots/root-$(date +%F)
```

### Step 2—(Optional) snapshot home

```
bash
```

```
sudo btrfs subvolume snapshot -r /home /.snapshots/home-$(date +%F)
```

### Step 3 — Send to another drive

The receiving filesystem **must** be Btrfs:

```
bash
```

```
sudo btrfs send /.snapshots/root-2025-02-23 | sudo btrfs receive  
/mnt/backup
```

This is a bit-for-bit backup of the snapshot.

---

### 4. If you want to back up to a non-Btrfs drive

Use `rsync` but make sure to exclude Btrfs system dirs:

```
bash
```

```
sudo rsync -aAXv \  
    --exclude=\  
    {"/dev/*", "/proc/*", "/sys/*", "/tmp/*", "/run/*", "/mnt/*", "/media/*", ".s  
    \  
    / /mnt/backup/
```

This can be restored, but requires rebuilding bootloader and fstab later.

---

### When a live ISO is your best move

Use a live system if:

- You suspect your layout is broken or mounting weirdly.
- You want a guaranteed clean backup with nothing running on it.
- You want to restructure subvolumes while backing up (common when coming from weird Timeshift setups).

In those cases, a live ISO avoids background processes writing data while you back up.

---

If you want, you can paste your `btrfs subvolume list -p /` output

I can tell you exactly:

- what's wrong with the layout,
- which subvolumes matter for backup,
- whether Timeshift will behave correctly,
- the safest backup method for your exact structure.

Just paste the output and we go from there.

i want system backp in case while customisation or  
some package corrupt i will be safe

exclude personal files they are easier to recover from  
live cd

Got it — you want **system-only backup**, no personal files, and the goal is to be able to roll back quickly if you break the OS while customizing or installing packages.

The cleanest way to do that on Btrfs (especially with a non-standard Timeshift layout) is **read-only snapshots + send/receive**, or a **dedicated OS-only subvolume snapshot**. You *don't* need a live CD unless your system is unbootable.

Here are the practical options, bluntly:

---

## ✓ Option 1 — Btrfs read-only snapshot + send (best, fast, reliable)

This gives you a full system backup without walking through every file.

Personal data is automatically excluded if your home is a separate subvolume.

### 1. Create a read-only snapshot of your root OS

bash

```
sudo btrfs subvolume snapshot -r / /.snap-os-$(date +%F)
```

If your OS root is actually `@` or something else, use its mount path instead:

bash

```
sudo btrfs subvolume snapshot -r /@ /.snap-os-$(date +%F)
```

### 2. Send the snapshot to your backup drive

Backup drive must be Btrfs for this method.

bash

```
sudo btrfs send /.snap-os-2025-02-23 | sudo btrfs receive
```

### 3. Restore if needed

Boot a live CD **only when restoring**, not when backing up.

bash

```
sudo btrfs subvolume delete @# or your root subvol  
sudo btrfs receive /mnt/root_device / < backup_stream
```

This gives you a clean, exact restoration of your OS.

---



### Option 2 — Timeshift (if your layout works with it)

Timeshift ignores personal files by default, so it meets your requirements.

**But:** with a weird Fedora layout, Timeshift may not capture everything needed to boot. If Timeshift already works on your system and makes snapshots without errors, you can keep using it — just don't assume it will restore perfectly unless your subvolumes follow its expected pattern.

If you're unsure, avoid relying solely on Timeshift.

---



### Option 3 — Rsync system-only backup (works even with odd layouts)

If your backup drive is NOT Btrfs, use this.

This copies only the OS, excluding home and runtime files:

```
bash
```

```
sudo rsync -aAXHv \  
    --exclude=\  
    {"/home/*", "/dev/*", "/proc/*", "/sys/*", "/tmp/*", "/run/*", "/mnt/*", "/med  
    \  
    / /mnt/backup/
```

This is more “old school” and less clean than Btrfs send, but it works and is simple.

On restore, you would reinstall bootloader + regenerate initramfs — not hard, but not elegant.

---

## ✖ What won’t solve your problem:

- Backing up from a live CD every time → unnecessary and slows you down.
  - Backing up personal files → you said you don’t care; avoid them.
  - Raw disk imaging → pointless with Btrfs snapshots available.
- 



## 🔥 Recommendation (straightforward and resilient)

Because you want *system safety while customizing packages*, this is your best setup:

\*\*Use Btrfs read-only snapshots for quick rollback

- Btrfs send to external drive for real backups.\*\*

The snapshot → rollback cycle is nearly instant.

The send → backup cycle is safe and offline.

---

## If you want maximum safety, send me:

CSS

```
btrfs subvolume list -p /
```

I'll tell you exactly which subvolumes need to be backed up and the exact commands for *your specific layout*.

list subvolumes command

To list all Btrfs subvolumes on your system (including parent IDs), use:

bash

```
sudo btrfs subvolume list -p /
```

That's the most informative version.

The `-p` flag shows each subvolume's **parent ID**, which is important when diagnosing weird layouts.

If you want extra clarity, these variants help too:

### Show subvolume paths only (clean)

bash

```
sudo btrfs subvolume list -o /
```

Show everything, including IDs, UUIDs, generation, flags

```
bash
```

```
sudo btrfs subvolume list -t /
```

If your root is mounted as @ (common on Fedora with Timeshift)

Run it on the device directly:

```
bash
```

```
sudo btrfs subvolume list -p /mnt
```

(where `/mnt` is the mountpoint of your root partition)

---

If you paste the output here, I can tell you:

- which subvolume is the *real* root,
- whether Timeshift is safe to use,
- the correct snapshot + backup commands,
- and whether anything in your layout is risky.

Drop the output and I'll map it out cleanly.

```
pd@fedora:/$ sudo btrfs subvolume list -p /
[sudo] password for pd:
ID 256 gen 36565 parent 5 top level 5 path root
```

```
ID 257 gen 36565 parent 5 top level 5 path home
ID 258 gen 36530 parent 256 top level 256 path
var/lib/machines
ID 259 gen 17757 parent 256 top level 256 path
.snapshots
ID 260 gen 17741 parent 257 top level 257 path
home/.snapshots
ID 261 gen 8791 parent 260 top level 260 path
home/.snapshots/1/snapshot
ID 262 gen 8791 parent 259 top level 259 path
.snapshots/1/snapshot
ID 263 gen 8791 parent 260 top level 260 path
home/.snapshots/2/snapshot
ID 264 gen 8791 parent 259 top level 259 path
.snapshots/2/snapshot
```

Here's the straight assessment of your layout and the safest backup approach.

---

## ✓ Your Btrfs layout (interpreted)

pgsql

ID 256 path root	← your <b>real</b> root filesystem (@)
ID 257 path home	← home <b>as</b> separate subvolume
ID 258 path var/lib/machines	← systemd-nspawn container area
ID 259 path .snapshots	← snapper-style snapshots <b>of</b>
root	
ID 260 path home/.snapshots	← snapper-style snapshots <b>of</b>
home	

**Conclusion:**

Your system uses **Snapper-style** snapshots, not Timeshift layout.

Timeshift expects:

```
python
```

```
@      (= /)
```

```
@home
```

but your system does **not** use those names.

So Timeshift will *not* behave correctly with this layout.

Good news: your layout is clean and suitable for proper Btrfs snapshots and send/receive.

---

✓ **Correct way to back up your system *only* (excluding personal data)**

Since your system root is:

```
lua
```

```
/
```

```
→ ID 256 (path: root)
```

you want to snapshot ID 256 (root) only.

Your home is a separate subvolume (ID 257), so it will *not* be included automatically — perfect.

---

## 1. Create a read-only snapshot of the root OS

bash

```
sudo btrfs subvolume snapshot -r / /.sys-backup-$(date +%F)
```

This snapshot contains the OS only, not personal files.

---

## 2. (Optional but recommended) Store snapshots in a dedicated directory

If you want things tidy, create:

bash

```
sudo mkdir /.sys-snapshots
sudo btrfs subvolume snapshot -r / /.sys-snapshots/backup-$(date
+%F)
```

---

## 3. Backup snapshot using Btrfs send (BEST)

Backup drive *must* be Btrfs:

bash

```
sudo btrfs send /.sys-snapshots/backup-2025-02-23 \  
| sudo btrfs receive /mnt/backup
```

This gives you a **byte-accurate system backup**, fast and reliable.

---



## Alternative if your backup drive is NOT Btrfs (rsync)

bash

```
sudo rsync -aAXhv \  
--exclude=\  
{/home/*, /dev/*, /proc/*, /sys/*, /tmp/*, /run/*, /mnt/*, /med  
\  
/ /mnt/backup/
```

Personal files in `/home` are excluded automatically.

---



## Restore procedure (when needed)

You only need a live CD for **restore**, not for backups.

1. Boot live ISO
2. Mount your Btrfs root
3. Delete broken root subvolume
4. Receive backup snapshot
5. Make it the new root
6. Reinstall bootloader (Fedora: `grub2-install` + `grub2-mkconfig`)

## ⭐ Recommendation for your exact layout

Since you use Snapper-style subvolumes:

**Use Btrfs read-only snapshot + Btrfs send.**

Don't use Timeshift — wrong expectations and naming scheme.

---

If you'd like, I can generate **exact restore commands** based on your partition name.

```
pd@fedora:/$ sudo btrfs subvolume snapshot -r /
./sys-backup-$(date +%F)
Create readonly snapshot of '/' in './sys-backup-2025-
12-07'
```

Good — that means your root filesystem **successfully created a read-only snapshot**:

```
bash
```

```
/ .sys-backup-2025-12-07
```

That snapshot is a complete, consistent backup of your **system only** (no personal files, since `/home` is its own subvolume).

Now you have a safe point you can always send to an external drive or use for rollback.

---



**Next step: copy the snapshot to your backup drive**

If your backup drive is **Btrfs**, mount it:

```
bash
```

```
sudo mount /dev/sdX1 /mnt/backup
```

Then:

```
bash
```

```
sudo btrfs send /.sys-backup-2025-12-07 | sudo btrfs receive  
/mnt/backup
```

This is the ideal method — creates a perfect, fast external backup.

---

## If your backup drive is NOT Btrfs

Use rsync to copy only the system snapshot:

```
bash
```

```
sudo rsync -aAXhv /.sys-backup-2025-12-07/ /mnt/backup/sys-backup/
```

Don't rsync the whole `/`; only the snapshot — cleaner, predictable, excludes personal files.

---

## Restoring from this snapshot (when something breaks)

You don't need to restore immediately; this is just for reference:

1. Boot live ISO
2. Mount your root partition
3. Delete the existing `root` subvolume
4. Receive or copy back the snapshot
5. Rename the snapshot to the correct root name
6. Reinstall bootloader

If you want, I can write the exact restore commands for **your layout**.

---

## 👍 You're in a safe state now

You can customize Fedora, install packages, break things — and you can always:

- rollback locally (if you convert the snapshot to root), or
- restore from external backup

Your layout supports clean, predictable recovery.

If you want, I can show you how to:

- automate daily system-only snapshots,
- prune old snapshots,
- or send incremental backups.

is it good if i keep backup on same drive?

Short answer: **It's fine for quick rollbacks, but not safe as your only backup.**

Let me break it down honestly:

---

## Good reasons to keep a backup on the same drive

- Instant rollback if a package or config breaks
- No external drive needed
- No downtime
- Btrfs snapshots are extremely space-efficient
- Very fast creation and restoration

For **system tinkering**, same-drive snapshots are perfect.

---

## But it's NOT a real backup

If the physical SSD/HDD fails, you lose:

- the system
- the snapshot
- everything

Also lost if:

- filesystem gets corrupted
- accidental `btrfs subvolume delete`
- malware or a bad script nukes your root
- a botched distro upgrade damages the whole filesystem

Same-drive snapshots protect you only from **software mistakes**, not hardware failure.

---

## Best practice (realistic)

For your situation:

## 1 Keep snapshots on the same drive

Use them for quick, everyday safety during experiments.

## 2 Occasionally send important system snapshots to an external Btrfs drive

This protects against total drive death.

Example:

```
bash
```

```
sudo btrfs send /.sys-backup-2025-12-07 | sudo btrfs receive  
/mnt/backup
```

This gives you both:

- **fast rollback** (local snapshots)
- **disaster recovery** (external clone)



## Recommendation for your workflow

Because you're customizing Fedora aggressively, it's absolutely fine to keep 3–4 recent system snapshots on the same disk.

Just don't trust them as *your only* backup.

---

If you want, I can help you:

- set up automatic cleanup (older snapshots deleted safely)
- set up periodic external backups

- or create a rollback script that switches root to a chosen snapshot in one command.

backup commands

Here are **clean, correct backup commands** for your layout.

I'll give you two sections:

1. **Local snapshots (same drive)** – fastest, for rollback
2. **External backup (real backup)** – protects against drive failure

Use whichever you need.

---



## 1. Create SYSTEM-ONLY snapshot (same drive)

You already used this, but here it is clean:

bash

```
sudo btrfs subvolume snapshot -r / /.sys-backup-$(date +%F)
```

List your snapshots:

bash

```
sudo btrfs subvolume list /
```

Delete an old system snapshot:

bash

```
sudo btrfs subvolume delete /.sys-backup-2025-12-07
```

This protects you from package breaks, config mistakes, etc.

---

## ✓ 2. TRUE BACKUP to external Btrfs drive

Mount your external drive:

bash

```
sudo mount /dev/sdX1 /mnt/backup
```

### Create snapshot (if not created already)

bash

```
sudo btrfs subvolume snapshot -r / /.sys-backup-$(date +%F)
```

### Send snapshot to backup drive

(Preferred, safest Btrfs method)

bash

```
sudo btrfs send /.sys-backup-2025-12-07 | sudo btrfs receive  
/mnt/backup
```

Check it:

bash

```
sudo btrfs subvolume list /mnt/backup
```

### Incremental backup (optional, much faster)

Requires an older snapshot on backup drive:

```
bash
```

```
sudo btrfs send -p /.sys-backup-OLD /.sys-backup-NEW | sudo btrfs  
receive /mnt/backup
```

---

## ✓ 3. Backup to NON-Btrfs drive (USB formatted ext4, NTFS, etc.)

Use rsync and copy only system snapshot:

```
bash
```

```
sudo rsync -aAXhv /.sys-backup-2025-12-07/ /mnt/backup/sys-backup/
```

This avoids copying your personal files.

---

### 🔧 Optional: Automate daily system snapshots

```
bash
```

```
sudo systemctl enable --now snapper-timeline.timer  
sudo systemctl enable --now snapper-cleanup.timer
```

Or a simple cron entry if you prefer your own snapshot directory.

---

## ⭐ If you want restore commands too

Tell me your root partition (example: `/dev/nvme0n1p3`), and I'll give you exact restore steps that match your system.

how can i backup automatically within same disk

To **automatically back up your system (OS only) on the same disk**, the cleanest and most reliable tools for your layout are:

### 🔥 Option 1 — Use Snapper (recommended for your layout)

Your system already uses **snapper-style subvolumes**:

arduino

root → .snapshots

home → home/.snapshots

So enabling Snapper automation is almost zero effort.

---

### ✅ Enable automatic system snapshots

Snapper config for root is usually named `root`.

Check if it's present:

```
bash
```

```
sudo snapper list-configs
```

If you see:

```
nginx
```

```
root / ...
```

then just enable automation:

### Enable timeline (hourly) snapshots

```
bash
```

```
sudo systemctl enable --now snapper-timeline.timer
```

### Enable automatic cleanup of old snapshots

```
bash
```

```
sudo systemctl enable --now snapper-cleanup.timer
```

That's all.

Snapper will now:

- create periodic snapshots (usually hourly + daily + monthly)
- automatically delete old ones
- never include personal data (home is separate)

Your layout is *already* aligned with Snapper. This is the best long-term automated solution.

## 🔥 Option 2 — Your own automatic snapshots via systemd timer

If you want simple, custom, predictable snapshots outside snapper:

### 1. Create a script

```
/usr/local/bin/system-snapshot.sh :
```

```
bash
```

```
#!/bin/bash
DATE=$(date +%F-%H%M)
btrfs subvolume snapshot -r / /.sys-backup-$DATE
```

Make it executable:

```
bash
```

```
sudo chmod +x /usr/local/bin/system-snapshot.sh
```

---

### 2. Create a systemd service

```
/etc/systemd/system/system-snapshot.service :
```

```
ini
```

```
[Unit]
Description=Automatic Btrfs System Snapshot

[Service]
```

```
Type=oneshot  
ExecStart=/usr/local/bin/system-snapshot.sh
```

### 3. Create a systemd timer

```
/etc/systemd/system/system-snapshot.timer :
```

```
ini
```

```
[Unit]  
Description=Run system snapshot every day  
  
[Timer]  
OnCalendar=daily  
Persistent=true  
  
[Install]  
WantedBy=timers.target
```

Enable it:

```
bash
```

```
sudo systemctl enable --now system-snapshot.timer
```

This will create a **daily read-only snapshot** of your OS.

---

🔥 Option 3 — Automatic snapshots triggered by package updates

Use pacman hooks (Arch) or dnf plugins, but in Fedora you can use:

### Install:

```
bash
```

```
sudo dnf install python3-dnf-plugin-snapper
```

This plugin auto-creates snapshots before and after package operations.

### Enable:

```
bash
```

```
sudo snapper setup-config --no-answer=root
```

Fedora's Snapper integration is not as polished as SUSE's, but it works.

---



## Which option should YOU choose?

Given your exact subvolume layout:



### Best choice for you: Snapper automation

Because:

- your layout already matches Snapper's structure,
- you already have `.snapshots` folders,
- Snapper cleans up old snapshots automatically,
- Snapper lets you rollback cleanly,
- No custom scripts needed.

This gives you safe, continuous, automatic system backups **on the same disk**.

# If you want, I can configure Snapper properly for your system

Just tell me the output of:

```
bash
```

```
sudo snapper list-configs
```

```
pd@fedora:/$ sudo snapper list-configs
Config | Subvolume
_____|_____
pd@fedora:/$
```

You have **no Snapper configs** set up yet.

That's fine — you can enable automatic system snapshots cleanly.

Since your root filesystem is the subvolume named `root` (ID 256), you can create a proper Snapper config for it manually.

Here's the exact setup for **automatic system-only backups on the same disk**, customized for your layout.

---

## ✓ Step 1 — Create the Snapper config for your root

Run:

```
bash
```

```
sudo snapper -c root create-config /
```

Snapper will automatically detect the `.snapshots` directory (ID 259) and use it.

After this, check:

```
bash
```

```
sudo snapper list-configs
```

You should now see:

```
nginx
```

```
root | /
```

---

## ✓ Step 2 — Enable automatic snapshots (timeline)

```
bash
```

```
sudo systemctl enable --now snapper-timeline.timer
```

This creates:

- hourly snapshots
  - plus daily/weekly/monthly based on settings
- 

## ✓ Step 3 — Enable automatic cleanup of old snapshots

```
bash
```

```
sudo systemctl enable --now snapper-cleanup.timer
```

This prevents your disk from filling up.

---

## ✓ Step 4 — (Optional but recommended) Adjust snapshot retention

Edit:

```
bash
```

```
sudo nano /etc/snapper/configs/root
```

Useful defaults:

```
ini
```

```
TIMELINE_CREATE="yes"
TIMELINE_CLEANUP="yes"

TIMELINE_MIN_AGE="1800"          # 30 minutes min gap
TIMELINE_LIMIT_HOURLY="8"        # keep last 8 hours
TIMELINE_LIMIT_DAILY="7"
TIMELINE_LIMIT_WEEKLY="4"
TIMELINE_LIMIT_MONTHLY="3"
TIMELINE_LIMIT_YEARLY="0"
```

This keeps enough snapshots without filling the disk.

## 👍 At this point, automation is fully active

Snapper will now:

- automatically create root snapshots
- automatically prune old ones
- store them in `.snapshots` (fits your layout perfectly)
- keep home data separate (no risk of including personal files)

You now have reliable, automatic system-only backups within the same disk.

---

If you want, I can also show you:

- how to rollback to any snapshot safely
- how to clone selected snapshots to an external disk
- how to trigger snapshots before every `dnf upgrade` automatically

```
pd@fedora:/$ sudo snapper list-configs
Config | Subvolume
-----
pd@fedora:/$ sudo snapper -c root create-config /
Creating config failed (creating btrfs subvolume
.snapshots failed since it already exists).
pd@fedora:/$ sudo snapper list-configs
Config | Subvolume
-----
```

Snapper refuses to create the config because your system **already has a** `.snapshots` **subvolume**, but **not in the structure that snapper expects**.

Snapper requires this exact layout:

swift

```
/           ← subvolume for root  
.snapshots    ← directory (not a subvolume)  
.snapshots/ID  ← each snapshot is a subvolume
```

But your layout is:

kotlin

```
ID 259 path .snapshots      ← this is a SUBVOLUME, not a directory
```

Snapper can't manage that.

That's why the config creation fails and `list-configs` remains empty.

So you must choose between:

---

## 🚧 Option A — Convert existing `.snapshots` subvolume to Snapper-compatible layout

This is possible but requires restructuring.

If you want the *cleanest, fully automated Snapper setup*, I can guide you step-by-step.

---

## 🚧 Option B — Don't force Snapper; instead use simple, robust Btrfs snapshot automation

This works **perfectly** with your layout, requires no restructuring, and is 100% safe.

Given you want **automatic system backups on the same drive**, Option B is by far the easiest and safest.

So I recommend implementing **systemd-based automatic snapshots**, not Snapper.

---

## Automatic backups using Btrfs + systemd (recommended for your layout)

This avoids all Snapper structural requirements and works immediately.

---

### 1 Create snapshot script

Create:

```
bash

sudo nano /usr/local/bin/system-snapshot.sh
```

Paste:

```
bash

#!/bin/bash
DATE=$(date +%F-%H%M)
btrfs subvolume snapshot -r / /.sys-backup-$DATE
```

Save, exit.

Make executable:

```
bash
```

```
sudo chmod +x /usr/local/bin/system-snapshot.sh
```

---

## 2 Create systemd service

Create:

```
bash
```

```
sudo nano /etc/systemd/system/system-snapshot.service
```

Paste:

```
ini
```

```
[Unit]
Description=Automatic Btrfs System Snapshot
```

```
[Service]
Type=oneshot
ExecStart=/usr/local/bin/system-snapshot.sh
```

---

## 3 Create systemd timer

Create:

```
bash
```

```
sudo nano /etc/systemd/system/system-snapshot.timer
```

Paste:

```
ini

[Unit]
Description=Run daily Btrfs system snapshot

[Timer]
OnCalendar=daily
Persistent=true

[Install]
WantedBy=timers.target
```

Enable it:

```
bash

sudo systemctl enable --now system-snapshot.timer
```



**Done**

Your system will now automatically create daily, system-only snapshots on the same disk.

This method:

- requires no Snapper
- works with your existing layout

- avoids subvolume conflicts
  - avoids risk
  - is simple, transparent, and robust
- 

## If you want hourly or triggered snapshots instead

Tell me and I'll tune the timer for:

- hourly
- on boot
- before `dnf upgrade`
- after `dnf upgrade`

Whatever you prefer.

i dont want custom scripts as its make it harder to  
debug

Understood — no custom scripts and no hacky workarounds.

If you want automatic system-only snapshots on the same disk, and you don't want to maintain your own scripts, then:



**You have exactly TWO real, supported choices:**

1. Fix your `.snapshots` layout so Snapper can work
2. Use Timeshift (which works *with* a subvolume at `.snapshots`)

Snapper is failing because your current layout is incompatible, not because Snapper is broken.

If you want automation with **no custom scripting**, Snapper or Timeshift are the only real options.

Let's evaluate them:

---

## 🔍 Why Snapper fails on your system now

You have:

```
pgsql
```

```
ID 259 path .snapshots ← This is a subvolume
```

Snapper requires:

```
CSS
```

```
/.snlapshots ← MUST be a directory, not a subvolume
```

Snapper checks:

- "Does .snapshots exist?" → yes
- "Is it a directory?" → **no, it's a subvolume** → Snapper aborts

This won't fix itself.

Snapper will *never* create a config until this is corrected.

---

⭐ If you want Snapper automation (best long-term),  
you must do:

1. Temporarily move your existing `.snapshots` subvolume aside

2. Let Snapper create a new `.snapshots` directory
3. (Optional) Re-import old snapshots

These are official, supported operations — NO custom scripts.

If you want this path, I can give you step-by-step safe commands (nothing destructive unless you confirm).

---

## ★ If you prefer Timeshift automation instead (easier)

Timeshift **does not care** that `.snapshots` is a subvolume.

It works perfectly with your layout.

You can enable automatic snapshots using Timeshift's built-in scheduling:

1. Open GUI:



timeshift-launcher

2. Go to:

**Settings → Schedule**

Turn on:

- ♦ Hourly
- ♦ Daily
- ♦ Boot snapshots

Whatever you want.

**That's it — NO scripting, NO restructuring.**

Timeshift snapshots are stored in `.snapshots` and work fine with Fedora+Btrfs.

---



## Which option should YOU choose?

**If you want:**

- stable automatic snapshots
- clean rollback
- no custom scripts
- minimal maintenance

Then:

**→ Use Timeshift automatic snapshots.**

It works with your current layout *as-is*.

**If you want:**

- SUSE-style rollback
- Rich snapshot management
- Pre/post-updates snapshots
- Integration with system tools

Then:

**→ Fix `.snapshots` and enable Snapper.**

Snapper is more flexible, but requires a one-time layout fix.

---

**If you tell me:**

Do you want Timeshift automation or Snapper automation?

I'll give you the exact next steps — clean, safe, fully supported, and with no custom scripts.

↳ \* \*\*Temporarily move your existing `snapshots` subvolume aside\*\*

1

Great — you chose **Option 1: Fix the layout so Snapper can work.**

That means you want:

- Automatic system snapshots
- NO custom scripts
- Proper integration with Fedora + Btrfs
- Supported, clean layout

To do that, we must temporarily move the existing `.snapshots` SUBVOLUME aside, then let Snapper create a proper `.snapshots` DIRECTORY.

This is a safe, reversible process *as long as you follow exactly.*

---

## ✓ Step 1— Check if `.snapshots` is empty or used

Show what's inside:

bash

```
ls -al /.snapshots
```

You likely have entries like:

```
1/
```

```
2/
```

Those correspond to subvolume IDs:

```
pgsql
```

```
ID 259  path .snapshots
ID 262  path .snapshots/1/snapshot
ID 264  path .snapshots/2/snapshot
```

We are NOT deleting anything.

We only move the subvolume.

---

## Step 2 — Move the existing `.snapshots` subvolume aside

Run:

```
bash
```

```
sudo mv /.snapshots /.snapshots-old
```

Because `.snapshots` is a subvolume, `mv` will instantly rename it — this is safe and lossless.

Now you have:

sql

```
/.snapshots-old           ← old Snapper-style snapshots live here  
(no .snapshots exists) ← Snapper now can create one
```

---

## ✓ Step 3 — Create Snapper config

Now Snapper will succeed:

bash

```
sudo snapper -c root create-config /
```

This time it will create:

bash

```
/.snapshots   ← directory (NOT subvolume)
```

It will then populate it like:

bash

```
/.snapshots/1/
```

Now:

bash

```
sudo snapper list-configs
```

Should show:

nginx

root | /

Congrats — Snapper is now fully functional.

---

## 🔥 Optional but important:

If you want Snapper to **manage only new snapshots**:

→ You're already done.

If you want Snapper to **see your old snapshots**, tell me and I'll provide the import steps. (It's optional and more advanced.)

---

## ⚙️ Step 4 — Enable automatic snapshots

Now turn on automation:

bash

```
sudo systemctl enable --now snapper-timeline.timer  
sudo systemctl enable --now snapper-cleanup.timer
```



You now have:

- ✓ Clean Fedora+Btrfs layout
  - ✓ Snapper fully working
  - ✓ Automatic snapshots
  - ✓ Automatic cleanup
  - ✓ No custom scripts
  - ✓ System-only backups (home excluded automatically)
- 

## If you want, I can give you:

- Recommended Snapper retention settings
- Rollback instructions
- Before/after dnf upgrade snapshot integration

Just tell me.

```
pd@fedora:/$ sudo ls -al /.snapshots
total 0
drwxr-x---. 1 root root 4 Nov 20 18:05 .
dr-xr-xr-x. 1 root root 190 Dec  7 12:04 ..
drwxr-xr-x. 1 root root 32 Nov 20 18:00 1
drwxr-xr-x. 1 root root 32 Nov 20 18:05 2
```

Perfect — this confirms exactly what we expected:

```
kotlin
```

```
/.snapshots           ← this is currently a **subvolume**
|--- 1/
```

Those directories contain **snapshots**, but they were not created by Snapper (because Snapper isn't configured). They were likely created by some other tool or early setup.

To make Snapper work **without custom scripts**, the `.snapshots` must be a **normal directory**, not a subvolume.

So we will:

1. **Move the subvolume aside** (safe, instant)
2. **Let Snapper create a new, proper `.snapshots` directory**
3. **Enable automatic snapshots**

This does **not** delete anything. It only renames the existing subvolume.

---

## ✓ Step 1—Move `.snapshots` subvolume aside

Run exactly:

```
bash
```

```
sudo mv /.snapshots /.snapshots-old
```

Because `.snapshots` is a subvolume, `mv` will simply rename it — no data copy, no risk.

After this you will have:

```
bash
```

```
./snapshots-old/1/  
./snapshots-old/2/
```

And now **no** `/ .snapshots` exists, which is what Snapper needs.

---

## ✓ Step 2 — Create Snapper root config

Now Snapper will successfully create the right layout:

```
bash  
  
sudo snapper -c root create-config /
```

This will create:

```
kotlin  
  
/.snapshots/           ← this time as a directory
```

Snapper will then populate:

```
bash  
  
/.snapshots/1/
```

Check:

```
bash  
  
sudo snapper list-configs
```

Expected output:

```
nginx
```

```
root | /
```

---

## ✓ Step 3 — Enable automatic snapshots

```
bash
```

```
sudo systemctl enable --now snapper-timeline.timer  
sudo systemctl enable --now snapper-cleanup.timer
```

Now your system will automatically create and clean system snapshots **every day/hour** without custom scripts.

---

## ⚠ About your old snapshots

They now live in:

```
bash
```

```
/.snapshots-old
```

You have **three options**, depending on what you want:

### Option A — Keep them as historical backups (safe, default)

Do nothing.

They will stay untouched and won't interfere with Snapper.

### Option B — Delete them (only if you don't care about them)

```
bash
```

```
sudo btrfs subvolume delete /.snapshots-old/1/snapshot  
sudo btrfs subvolume delete /.snapshots-old/2/snapshot  
sudo btrfs subvolume delete /.snapshots-old
```

### Option C — Import them into Snapper (advanced)

Possible, but only worth doing if you need these old snapshot states.

I can guide you if you want this.

---



### After Step 1–3, you will have:

- ✓ Fully working Snapper
  - ✓ Automatic system snapshots
  - ✓ Automatic cleanup
  - ✓ Fully supported layout
  - ✓ No custom scripts
  - ✓ Only the OS backed up (home excluded by design)
- 

👉 When you're ready, run Step 1 (`mv /.snapshots`) and show me the result of:

```
bash
```

```
sudo snapper -c root create-config /
```

I'll confirm that everything is set correctly.

```
pd@fedora:/$ sudo snapper -c root create-config /
pd@fedora:/$ sudo snapper list-configs
Config | Subvolume
-----|-----
root  | /
pd@fedora:/$
```

how to backup home config?

Good — Snapper is now fully configured for **root**, and automatic system snapshots will work cleanly.

Now to your question:

## How to back up your *home config* (not personal files)

When people say “home config,” they usually mean:

- hidden dotfiles ( `~/.bashrc` , `~/.config/...` , etc.)
- app settings
- desktop configs
- plugin settings
- terminal/editor/browser configs
- etc.

You want these backed up **but not big personal folders** like Downloads, Pictures, videos, etc.

This is doable in two clean and officially supported ways — **no custom scripts** needed.

## OPTION 1— Snapper home config (best integrated with your system)

You already have a **home** subvolume:

```
arduino
```

```
ID 257 path home
```

Snapper supports multiple configs, so you can create a **home Snapper config** just like root.

### ★ Step 1: Create snapper config for /home

Run:

```
bash
```

```
sudo snapper -c home create-config /home
```

Then:

```
bash
```

```
sudo snapper list-configs
```

Expected:

```
arduino
```

```
root  | /
home | /home
```

Snapper will create:

```
arduino
```

```
/home/.snapshots/
```

You *already* have one ( `ID 260` ), so if Snapper complains, same trick applies as root:

```
mv /home/.snapshots /home/.snapshots-old .
```

But usually it works because many Fedora builds leave `.snapshots` untouched.

---

## ★ Step 2: Enable automation for home config

```
bash
```

```
sudo systemctl enable --now snapper-timeline.timer  
sudo systemctl enable --now snapper-cleanup.timer
```

These timers work for **all** Snapper configs (root + home).

After this:

Snapper will automatically create snapshots for:

- `/` (system)
- `/home` (your config + settings)

... but **not your big personal data**, unless stored in hidden folders.

Snapper ignores nothing; it snapshots everything in `/home`, including dotfiles and documents.

If you don't want documents, see Option 2 below.

---

## OPTION 2 — Use a separate home-config-only subvolume

If you want to **exclude large personal files** and only back up dotfiles, this is the clean approach:

1. Create a separate subvolume for home config:

arduino

/home/.config-backup

2. Bind-mount your dotfiles into it
3. Use Snapper only on that subvolume

This avoids backing up:

- Downloads
- Videos
- Steam libraries
- VM images
- etc.

If you want this setup, I can map out the exact structure.

---

## OPTION 3 — Don't back up home with Snapper at all

If your settings are minimal or synced (e.g. via Git, cloud), you can skip backing up home entirely.

Given you said "home configs are easier to restore,"  
you may actually not need Snapper on `/home`.

Snapper on `/` is usually enough.

---

## 👍 Recommendation for YOU

Given your earlier messages:

- you want system stability
- you don't care about personal files
- you want **no custom scripts**
- your home settings matter but are not huge

The simplest and cleanest method is:

→ **Create Snapper config for `/home` just like `/`**

It gives:

- ✓ Automatic backup of configs
- ✓ Automatic cleanup
- ✓ Integration with existing system tools
- ✓ No maintenance

So just run:

bash

```
sudo snapper -c home create-config /home
sudo systemctl enable --now snapper-timeline.timer
sudo systemctl enable --now snapper-cleanup.timer
```

## If you want me to check your home layout

Run:

```
bash
```

```
sudo btrfs subvolume list -p /home
```

and I'll confirm whether `.snapshots` in `/home` needs to be moved like root did.

```
pd@fedora:/$ sudo snapper -c home create-config  
/home  
Creating config failed (creating btrfs subvolume  
.snapshots failed since it already exists).
```

Snapper is failing for the same reason as root earlier:

```
arduino
```

```
/home/.snapshots      ← is a *subvolume*  
Snapper requires       ← a *directory*
```

So we must do the same fix you used for `/`:

**Rename the existing `.snapshots` subvolume so Snapper can create its own.**

This is safe and lossless.

Here's the correct sequence.

## 1. Confirm what's inside /home/.snapshots

Just to be aware:

```
bash
```

```
sudo ls -al /home/.snapshots
```

You will likely see:

```
1/
```

```
2/
```

These correspond to:

```
bash
```

```
ID 260 path home/.snapshots
```

```
ID 261 path home/.snapshots/1/snapshot
```

```
ID 263 path home/.snapshots/2/snapshot
```

These are *not* Snapper-managed.

---

## 2. Move the entire .snapshots subvolume aside

```
bash
```

```
sudo mv /home/.snapshots /home/.snapshots-old
```

Because it's a subvolume, this is **instant** and **safe** — nothing is deleted.

After this:

sql

```
/home/.snapshots-old/ ← contains old snapshots  
/home/.snapshots       ← does not exist anymore
```

Perfect for Snapper.

---

### ✓ 3. Now create the Snapper config again

bash

```
sudo snapper -c home create-config /home
```

This time it will succeed.

Snapper will create:

swift

```
/home/.snapshots      ← directory  
/home/.snapshots/1/
```

---

### ✓ 4. Verify Snapper sees both configs

bash

```
sudo snapper list-configs
```

You should now see:

```
arduino
```

```
root | /
home | /home
```

---

## ✓ 5. Automatic snapshot timers work for ALL configs

Since you already enabled these:

```
bash
```

```
sudo systemctl enable --now snapper-timeline.timer
sudo systemctl enable --now snapper-cleanup.timer
```

Snapper will now automatically manage:

- system snapshots ( / )
- home snapshots ( /home )

No scripts, no custom tooling.

---

### ⚠ About the old `/home/.snapshots-old` folder

Your options:

#### A — Keep it forever (safe default)

Nothing will touch it. Acts like a manual backup.

#### B — Delete when you're ready

(Only when you're sure you don't need the old snapshots.)

## C — Import old snapshots into Snapper

Possible but not usually worth it.

If you want to cleanly delete them later, tell me and I'll give you the correct `btrfs subvolume delete` order.

---

### 👉 After these steps, you'll have:

- Automatic system snapshots
  - Automatic home-config snapshots
  - Automatic cleanup
  - Proper supported Btrfs+Snapper layout
  - No custom scripts
  - Safe rollback for both system and user config
- 

If you'd like, I can show you:

- how to rollback `/home` safely
- how to rollback only selected dotfiles
- how to exclude large folders inside `/home` from snapshots (cleanly)

i have this pika backup already - wanted to show them to you  
ls: cannot open directory '': Permission denied

```
root@fedora:/run/user/1000/pika-
backup/mount/186178ca/661f20-e924bcda/home/pd#
```

It looks like you **mounted a Pika Backup archive** under:

```
swift
```

```
/run/user/1000/pika-backup/mount/186178ca/661f20-e924bcda/home/pd
```

...and you tried to run:

```
bash
```

```
ls .
```

as **root**, and got:

```
arduino
```

```
ls: cannot open directory '.' : Permission denied
```

This is **normal** and expected with Pika Backup (Borg backup).

Here's why:

---

## ✓ Why you get “Permission denied” even as root

Borg/Pika archives mount using **FUSE** with very strict user-based permissions.

The mount belongs to your *normal user* ( uid 1000 ), not root.

So root cannot read inside the archive unless you explicitly allow it.

Borg/Pika mounts default to:

- Only the user who mounted it may read it
- root has no access, because FUSE does not automatically grant root privileges

This is intended — it prevents backup archives from leaking sensitive data to other users.

---

## HOW TO FIX IT (2 OPTIONS)

### Option 1 — View the files as your normal user (recommended)

Don't use root.

Just open the folder normally:

```
bash
```

```
cd /run/user/1000/pika-backup/mount/186178ca/661f20-e924bcda/home/pd  
ls
```

Or use a GUI file manager (Nautilus).

This is the intended way to browse Pika Backup archives.

---

### Option 2 — Allow root to access the mount

If you really need root to browse it, you must mount with:

```
lua
```

```
borg mount --options=allow_other
```

BUT Pika Backup **does not expose this option in its GUI** for safety reasons.

So with Pika Backup GUI mounts, **root simply cannot access them**.

If you want to mount manually (advanced):

bash

```
sudo borg mount --options=allow_other /path/to/repo /mnt/borg
```

...but this is only needed if you are doing advanced recovery.

---

## What you are seeing in that directory

Your pasted path suggests Pika Backup restored or mounted a **backup of your /home/pd directory**, not a system snapshot.

- It is **not** a system backup like Snapper
- It includes **personal files**, configs, dotfiles
- It does **not** include the OS, `/etc`, `/usr`, or the filesystem structure

Pika Backup = **user data backup**

Snapper = **system + home config snapshots**

You can use both, they don't conflict.

---

## If you want, you can show me:

Just run as **normal user**:

bash

```
ls -al /run/user/1000/pika-backup/mount/186178ca/661f20-e924bcda/home/pd
```

I can tell you:

- what type of data is in your Pika backup
- whether it's redundant with Snapper
- what role each backup tool is playing

Pika Backup and Snapper are very different tools — and they complement each other well if configured correctly.

```
pd@fedora:/run/user/1000/pika-
backup/mount/186178ca/661f20-e924bcda/home/pd$ 
ls -a
. 0096_sudoku.txt.out Applications .bash_logout
.bashrc .config .gemini .gnome .local .nv
.pki Public .recoll Templates .var
.. .antigravity .bash_history .bash_profile
bbreeds.out .dotnet .gitconfig install.sh .mozilla
.nvidia-settings-rc .profile .python_history rpmbuild
tmp .vscode
```

am i safe?

missing something in case i reinstall

Let's break down what you have and whether it's enough for a **complete restore of your user environment after a clean reinstall**.

You listed:

```
pgsql

. . . .antigravity .bash_history .bash_profile .bash_logout
.bashrc
0096_sudoku.txt.out Applications bbreeds.out .config .dotnet
.gemini .gitconfig .gnome install.sh .local .mozilla
.nv .nvidia-settings-rc .pki Public .python_history
.recoll Templates rpmbuild tmp .vscode .var
```

This is a **very typical full backup of your home directory**, including:

✓ All your user config

- .bashrc , .bash\_profile , .bash\_history
- .config/ (the critical one)
- .local/
- .gnome , .mozilla , .vscode , .gitconfig , .pki , etc.

✓ App settings and preferences

Everything from GNOME settings → VS Code → Firefox → shell → CLI tools.

✓ User metadata

- .var/
- .dotnet/
- .nvidia-settings-rc
- .python\_history
- .nv/

✓ User projects or files

- Applications
- Public
- Templates
- rpmbuild
- Random `.out` files and scripts.

## ✖ What you do NOT have (and what you *need* to reinstall manually)

Your Pika backup *does not* include:

- system packages
- system configs in `/etc`
- system-level customizations
- flatpak system installs
- system groups/users you added
- kernel parameters
- bootloader
- systemd user services (the service files, but not the data)

This is **normal** — Pika Backup is a *user data* backup tool, not a *system backup* tool.

You are now covered for **all your user settings**, but not the OS itself.

---

## ⭐ Are you SAFE?

✓ YES — Your user environment can be restored perfectly

With what you have, after a reinstall:

- Firefox, VS Code, GNOME, terminal settings → restored
- All dotfiles → restored
- Your GNOME extensions, preferences, layouts → restored

- Your CLI environment → restored
- Your app data (VSC plugins, Firefox profiles, etc.) → restored
- Workspace files → restored

## ✗ You are NOT covered for system-level recovery

Things *not* included:

- Installed RPM packages
- System configs (`/etc/fstab`, `/etc/sudoers`, `/etc/hostname`,  
`/etc/NetworkManager/...`)
- Btrfs layout
- Bootloader
- Drivers and kernel
- Services you enabled
- DNF package history

If something breaks the OS, you cannot restore it with Pika — but now you have **Snapper** to handle that.

---

## ✓ Combined protection (YOU NOW HAVE IT):

### Snapper

- Protects OS, `/` and system-level changes
- Automatic rollback
- Recovers from broken update, broken config, bad package

### Pika Backup

- Protects **your user profile**, all app settings, dotfiles, browser, and editor config
- Restores your working environment after clean reinstall

→ Snapshot of your /home/pd

This is the ideal setup for Fedora.

---

## ! What is still missing (optional but recommended)

If you're redoing the whole system someday, these help:

### ✓ List your installed DNF packages

Run this once and save the file:

```
bash
```

```
dnf history userinstalled > ~/installed-packages.txt
```

This gives you a complete list of packages you explicitly installed.

Later restore via:

```
bash
```

```
sudo dnf install $(cat installed-packages.txt)
```

### ✓ Backup your flatpaks (if you use them)

Save installed flatpaks:

```
bash
```

```
flatpak list --app --columns=application > ~/flatpaks.txt
```

Restore later:

```
bash
```

```
flatpak install flathub $(cat flatpaks.txt)
```

---

## ★ Summary: Are you safe?

- ✓ Your *home* is fully backed up
- ✓ Your system is now protected with Snapper
- ✓ Your configurations will survive a reinstall

### ! Only missing:

- reinstalling packages automatically
- system configs in `/etc/` (optional to back up manually)

If you want, I can show you:

- how to back up `/etc` safely
- how to rebuild your system from scratch using these backups
- how to test your Snapper rollback
- how to automate package list backups

Just tell me what you want next.

↳ ### Snapper → Protects \*\*OS\*\*, `/` and system-level changes → Automatic rollback →  
Recovers from broken update, broken config, bad package

help me backup and setup config

Alright — let's set up a **complete, reliable backup strategy** for your Fedora+Btrfs system that covers:

- ✓ **System protection (Snapper — OS rollback)**
- ✓ **User config protection (Pika Backup — your home directory)**
- ✓ **Reinstall protection (package lists + optional /etc backup)**
- ✓ **Clean, debuggable setup (NO custom scripts)**

You already have Snapper working for `/` and a Pika Backup archive for `/home/pd`.

Now we finish setting everything up so your system is **fully restorable** even after a clean reinstall.

---

## ✓ 1. Snapper: System Protection Setup

You already have:

```
lua
```

```
snapper -c root create-config /
```

So Snapper is active.

Now confirm your automatic timers:

```
bash
```

```
sudo systemctl status snapper-timeline.timer  
sudo systemctl status snapper-cleanup.timer
```

If either is not active, enable them:

```
bash
```

```
sudo systemctl enable --now snapper-timeline.timer  
sudo systemctl enable --now snapper-cleanup.timer
```

Snapper will now automatically create snapshots:

- hourly
- daily
- weekly
- monthly
- before/after a `snapper` operation

This protects the **OS**, not your personal files.

---

## 2. Enhance Snapper with pre/post DNF upgrade snapshots

Fedora supports automatic snapshots before OS updates.

Install the plugin:

```
bash
```

```
sudo dnf install python3-dnf-plugin-snapper
```

Enable it:

```
bash
```

```
sudo nano /etc/dnf/plugins/snapper.conf
```

Ensure:

```
ini
```

```
enabled = 1
```

Now every time you run:

```
bash
```

```
sudo dnf upgrade
```

Snapper will automatically create:

- snapshot before update
- snapshot after update

If the update breaks your system → rollback instantly.

This protects you from package corruption, bad configs, driver issues, etc.

---

### ✓ 3. Backup Package List (critical for reinstall)

This lets you reinstall ALL applications you installed manually.

Run:

```
bash
```

```
dnf history userinstalled > ~/installed-packages.txt
```

And also Flatpak list if you use them:

```
bash
```

```
flatpak list --app --columns=application > ~/flatpaks.txt
```

Keep `installed-packages.txt` and `flatpaks.txt` in your Pika backup.

### Restore after reinstall:

DNF:

```
bash
```

```
sudo dnf install $(cat installed-packages.txt)
```

Flatpak:

```
bash
```

```
flatpak install flathub $(cat flatpaks.txt)
```



## 4. Optional: Backup `/etc/` (system configs)

This includes:

- network configs
- hostname
- fstab
- systemd services
- grub settings
- system-level customizations

Back up with Pika Backup by adding `/etc` as an extra folder *once*, OR manually:

bash

```
sudo tar czvf ~/etc-backup.tar.gz /etc
```

Store `etc-backup.tar.gz` inside your home directory so Pika Backup saves it.

This is not required for rollback (Snapper covers OS state), but helpful for full reinstall.

---

## ✓ 5. Pika Backup: Keep backing up Home

You already confirmed your Pika archive contains:

- `.config`
- `.local`
- `.gnome`
- Browsers
- VSCode
- Shell settings
- Dotfiles
- App data

After reinstall, restoring this gives you your whole desktop environment exactly as it was.

### Suggested Pika Backup schedule:

- Daily or weekly incremental backup
- Keep a few months of snapshots
- Verify that “home/pd” is selected

## 🎯 At this point, after a reinstall you can restore:

- ✓ System state → Snapper rollback (if restoring on same install)
- ✓ User settings → Pika Backup restore
- ✓ Installed apps → `installed-packages.txt`
- ✓ Flatpaks → `flatpaks.txt`
- ✓ System configs → `etc-backup.tar.gz` (optional)

This is a complete and robust setup — no custom scripts, easy to debug, fully supported.

---

## 🔥 Optional Next Steps

If you want, I can help you with:

- making Snapper rollback *bootable*
- excluding large folders from Pika
- backing up Btrfs layout itself
- testing a full rollback
- building a reinstall playbook for future-proofing

Just tell me what direction you want next.

```
pd@fedora:~$ sudo systemctl status snapper-
timeline.timer
● snapper-timeline.timer - Timeline of Snapper
```

## Snapshots

Loaded: loaded (/usr/lib/systemd/system/snapper-timeline.timer; enabled; preset: disabled)  
Active: active (waiting) since Sun 2025-12-07  
12:17:26 IST; 12s ago  
Invocation: af9a5f414eda4db296bbdf904de649c7  
Trigger: Sun 2025-12-07 13:00:00 IST; 42min left  
Triggers: ● snapper-timeline.service  
Docs: man:snapper(8)  
man:snapper-configs(5)

Dec 07 12:17:26 fedora systemd[1]: Started snapper-timeline.timer - Timeline of Snapper Snapshots.

pd@fedora:~\$ sudo systemctl status snapper-cleanup.timer  
● snapper-cleanup.timer - Daily Cleanup of Snapper Snapshots

Loaded: loaded (/usr/lib/systemd/system/snapper-cleanup.timer; enabled; preset: disabled)

Active: active (waiting) since Sun 2025-12-07  
12:17:31 IST; 20s ago

Invocation: 942b7b13221546a3be123ef519f4501c

Trigger: Sun 2025-12-07 13:17:31 IST; 59min left

Triggers: ● snapper-cleanup.service

Docs: man:snapper(8)

man:snapper-configs(5)

Dec 07 12:17:31 fedora systemd[1]: Started snapper-cleanup.timer - Daily Cleanup of Snapper Snapshots.

Perfect — your timers are **active, enabled, and scheduled**:

- **snapper-timeline.timer** → will create automatic hourly/daily/weekly/monthly snapshots
- **snapper-cleanup.timer** → will automatically prune old snapshots

This means **your system snapshot automation is fully operational.**

Now let's finalize your backup + restore setup so you are *completely safe* for reinstall, rollback, updates, and user-config recovery.

Below is the clean, minimal, maintainable setup.

---

## 1. Enable pre/post-update snapshots (critical for Fedora safety)

This gives you automatic Snapper snapshots **before DNF upgrades**, so you can undo a broken update instantly.

Install plugin:

```
bash
sudo dnf install python3-dnf-plugin-snapper
```

Check config:

```
bash
grep enabled /etc/dnf/plugins/snapper.conf
```

You want:

```
ini
```

```
enabled = 1
```

Now **every time you update**, Snapper will automatically create:

- Pre-update snapshot
- Post-update snapshot

Rollback is instant.

---

## ✓ 2. Make a list of user-installed RPM packages

This protects you if you ever reinstall Fedora.

Run:

```
bash
```

```
dnf history userinstalled > ~/installed-packages.txt
```

This file contains every package you manually installed.

**IMPORTANT:**

This file **must live inside your home folder** so Pika Backup saves it automatically.

**Restore after reinstall:**

```
bash
```

```
sudo dnf install $(cat installed-packages.txt)
```

Easy, reproducible, reliable.

### 3. Save Flatpaks list (if you use them)

bash

```
flatpak list --app --columns=application > ~/flatpaks.txt
```

Restore:

bash

```
flatpak install flathub $(cat flatpaks.txt)
```

Pika Backup will also store this file automatically.

---

### 4. Optional but recommended: Backup /etc

This gives you:

- Network config
- Hostname
- fstab
- custom systemd units
- sudoers changes
- firewall config

Make a compressed archive:

bash

```
sudo tar czvf ~/etc-backup.tar.gz /etc
```

Because this file is stored in your home folder, **Pika Backup will include it**.

## ✓ 5. Your Home directory backup is already excellent

From your previous Pika archive listing:

You have:

- `.config`
- `.local`
- `.mozilla`
- `.gnome`
- `.vscode`
- `.bashrc`, `.profile`, `.gitconfig`
- All settings for your desktop, CLI tools, apps

This means:

After reinstall + restore, 90% of your environment comes back exactly as before.

Perfect.

---



## SUMMARY: YOU ARE NOW PROTECTED AGAINST...

### ✓ OS breakage

Snapper (root config) + dnf plugin → automatic pre/post update snapshots, hourly timeline snapshots, rollback instant.

### ✓ User environment loss

Pika Backup gives you a complete and restorable home directory.

### ✓ Clean reinstall

- `installed-packages.txt`
- `flatpaks.txt`
- `etc-backup.tar.gz`

Together restore your system to the same configuration.

### ✓ No custom scripts

Everything uses standard Fedora tools.

---

## ★ If you want, I can generate a reinstall recovery checklist such as:

- What to install first
- How to restore Pika archive
- How to restore Snapper rollback
- How to restore package list
- How to restore user config

Just tell me: **Do you want the recovery guide?**

[check snapper config](#)

To **check your Snapper configuration cleanly**, use the commands below.

These will show whether both configs (root + home) are valid, active, and using proper subvolumes.

## 1. Show all Snapper configs

bash

```
sudo snapper list-configs
```

Expected output:

arduino

Config	Subvolume
root	/
home	/home

← only **if** you created it

If `home` is missing, tell me and I'll help fix it.

---

## 2. Show full details of the root config

bash

```
sudo snapper -c root get-config
```

This shows:

- timeline enabled?
- cleanup enabled?
- snapshot limits
- snapshot type settings
- important paths

Key values should show:

ini

```
TIMELINE_CREATE="yes"  
TIMELINE_CLEANUP="yes"
```

---

### 3. Show full details of the home config (if used)

bash

```
sudo snapper -c home get-config
```

Expected similar output.

If this fails, your home config needs repair.

---

### 4. Verify that Snapper snapshots work at filesystem level

Root:

bash

```
sudo btrfs subvolume list / | grep '.snapshots'
```

You should see something like:

css

```
ID xxx top level 5 path .snapshots  
ID xxx top level ... path .snapshots/1/snapshot  
ID xxx top level ... path .snapshots/2/snapshot
```

### Home:

```
bash
```

```
sudo btrfs subvolume list /home | grep '.snapshots'
```

If `/home/.snapshots` wasn't replaced correctly, I'll help you fix it.

---

## 5. Manually trigger a snapshot (to ensure it works)

### Root:

```
bash
```

```
sudo snapper -c root create -d "test"
```

### List it:

```
bash
```

```
sudo snapper -c root list
```

You should see a new snapshot at the bottom.

### Home:

```
bash
```

```
sudo snapper -c home create -d "test"
```

List:

```
bash
```

```
sudo snapper -c home list
```

---

👉 Please run these and paste the results:

**Minimum needed:**

1. `sudo snapper list-configs`
2. `sudo snapper -c root get-config`
3. `sudo btrfs subvolume list / | grep .snapshots`

If you created a Snapper config for `/home`, also run:

4. `sudo snapper -c home get-config`
5. `sudo btrfs subvolume list /home | grep .snapshots`

I'll read the output and tell you **exactly** if everything is correct, or if something needs fixing.

```
pd@fedora:~$ sudo snapper list-configs
Config | Subvolume
-----|-----
root  | /
pd@fedora:~$ sudo snapper -c root get-config
Key          | Value
```

```
ALLOW_GROUPS          |  
ALLOW_USERS           |  
BACKGROUND_COMPARISON | yes  
EMPTY_PRE_POST_CLEANUP | yes  
EMPTY_PRE_POST_MIN_AGE | 3600  
FREE_LIMIT            | 0.2  
FSTYPE                | btrfs  
NUMBER_CLEANUP         | yes  
NUMBER_LIMIT           | 50  
NUMBER_LIMIT_IMPORTANT | 10  
NUMBER_MIN_AGE         | 3600  
QGROUP                 |  
SPACE_LIMIT             | 0.5  
SUBVOLUME              | /  
SYNC_ACL                | no  
TIMELINE_CLEANUP        | yes  
TIMELINE_CREATE          | yes  
TIMELINE_LIMIT_DAILY     | 10  
TIMELINE_LIMIT_HOURLY    | 10  
TIMELINE_LIMIT_MONTHLY   | 10  
TIMELINE_LIMIT_QUARTERLY | 0  
TIMELINE_LIMIT_WEEKLY    | 0  
TIMELINE_LIMIT_YEARLY    | 10  
TIMELINE_MIN_AGE         | 3600  
pd@fedora:~$
```

Your Snapper root configuration is healthy, correct, and fully operational.

Let's break it down and confirm each part so you know exactly what's working — and what's optional to improve.

## 1. Snapper root config is active

nginx

```
Config | Subvolume
root   | /
```

- ✓ Root snapshots are configured correctly
  - ✓ Only root config exists (you did not configure `/home`, which is fine)
- 

## 2. Timeline snapshots ARE enabled

From your config:

ini

```
TIMELINE_CREATE = yes
TIMELINE_CLEANUP = yes
```

- ✓ Automatic hourly/daily/monthly snapshots will run
- ✓ Old snapshots will be cleaned automatically
- ✓ The timers you enabled are using these settings

This is exactly what we want.

---

## 3. Snapshot limits are sane

Your snapshot limits:

ini

```
HOURLY = 10  
DAILY = 10  
MONTHLY = 10  
YEARLY = 10  
QUARTERLY = 0  
WEEKLY = 0
```

This means Snapper will keep:

- up to **10 hourly snapshots**
- up to **10 daily snapshots**
- up to **10 monthly snapshots**
- up to **10 yearly snapshots**

This is a safe, balanced retention policy and will not fill the disk.

---

## ✓ 4. Free space protection is active

ini

```
SPACE_LIMIT = 0.5
```

Snapper will start deleting old snapshots if the filesystem goes below **50% free space**.

This prevents accidental disk filling.

ini

```
FREE_LIMIT = 0.2
```

Snapper tries to maintain **20% free space** minimum.

## ⚠️ Not configured but optional:

You have not configured Snapper for `/home`, which means:

- Your `home/config` is NOT snapshotted
- Pika is your only home backup
- Snapper time machine for `/home` is not active

This is fine if you prefer **Pika only for home**, but you will lose dotfile rollback ability.

If you want Snapper-for-home added, we can set it up safely.

---

## ⭐ Should you configure Snapper for `/home`?

Here's how to decide:

### If you want:

- rollback of configs
  - rollback of GNOME settings
  - rollback of `.config`
  - rollback of `.local`
- Enable Snapper for `/home`.

### If you prefer:

- Pika Backup handles home
  - Snapper handles only the system
- Do NOT enable Snapper for `/home`.

Both setups are valid.

Most users use Pika for /home, Snapper for /.

This avoids double snapshotting.

---

## 🎯 Your current setup (based on everything you've shown me)

- ✓ System snapshots (root) — correctly configured
  - ✓ Automatic timeline snapshots — working
  - ✓ Cleanup — working
  - ✓ Pika Backup — backing up /home fully
  - ✓ Snapper does not snapshot /home — fine for your workflow
  - ✓ Backup strategy is complete and functional
- 

## 🧪 OPTIONAL: Validate Snapper working by creating a test snapshot

Try:

```
bash
```

```
sudo snapper -c root create -d "manual test"  
sudo snapper -c root list
```

You should see a new snapshot at the bottom.

## If you want, I can now help you:

1. Create Snapper config for `/home` (optional)
2. Tune retention settings for minimal disk usage
3. Configure Pika Backup to skip large folders
4. Test recovery → how to rollback Snapper snapshot safely
5. Build a full reinstall recovery checklist

Just tell me what you want next.

didnt i backed up home/pd dotfiles?

Yes — you *did* back up your entire `/home/pd` **dotfiles** with *Pika Backup*, and the backup you showed earlier confirms it 100%.

Let me walk you through this clearly so you understand exactly what you have covered and what you don't need Snapper for.

---

## Your Pika Backup already contains ALL critical dotfiles

You previously listed this from your Pika backup:

lua

```
.config/    .local/    .gnome/    .mozilla/    .vscode/
.bashrc     .profile    .gitconfig  .pki/        .var/
.nv/        .python_history
```

This is everything that defines your **user environment**.

- ✓ Desktop settings
- ✓ App configs
- ✓ GNOME settings
- ✓ Browser profiles
- ✓ Terminal/shell configs
- ✓ VSCode extensions
- ✓ Tool configs (git, python, dotnet, etc.)
- ✓ Local user caches and metadata
- ✓ Anything stored in `~/.config`, `~/.local/share`, etc.

**This means:**

- Your user environment can be fully restored after reinstall
  - No need for Snapper snapshots of `/home` unless you want *rollback* (not restore)
- 

## ! Snapper ≠ Pika

These tools target **different problems**:

**Snapper:**

- Protects **the system** (`/`)
- Lets you *rollback* instantly if updates break Fedora
- Useful for undoing config mistakes at OS level

**Pika Backup / Borg:**

- Protects **your home directory** and dotfiles
- Covers persistent config and personal data
- Works across reinstalls

- Designed for backup, not rollback
- 

## ⭐ So to answer cleanly:

✓ YES — You backed up 100% of your dotfiles

With Pika you can restore:

- All GNOME settings
- All app preferences
- All desktop customizations
- All shell settings
- All editor settings
- All user-level programs configs
- Everything under `/home/pd` (unless excluded)

Your personal environment is safe.

---

## 🚫 What you did NOT back up:

- System-wide configs in `/etc`
- System packages
- System settings outside your home
- Bootloader
- Drivers
- Systemd services outside `$HOME/.config/systemd/user`

These are handled by Snapper (rollback) or optional `/etc` backup.

---

# Do you NEED Snapper on /home ?

Most users do not.

Why?

Because Pika already gives:

- ✓ real backups
- ✓ version history
- ✓ off-disk safety
- ✓ restorable after reinstall
- ✓ simpler than Snapper home configs
- ✓ avoids double snapshotting (saves space)

Snapper-on-home is only useful if you want:

- "oh no I changed a config file—instantly roll back"
- time-machine style undo inside \$HOME

You have Pika, so you can ignore Snapper for /home .

---

## Your current setup is clean and correct:

**Snapper → Protects the OS**

**Pika Backup → Protects your home directory dotfiles**

**Installed package list → Protects reinstall process**

**Optional /etc tarball → Protects system configs**

If you want, I can help you:

- test restoring your dotfiles from your Pika archive
- verify nothing important is missing
- add recommended exclusions to avoid huge backups
- optimize Snapper so it uses minimal space
- build a full “reinstall recovery plan”

Just tell me what you want next.

can i use snapper on other folders like  
afs bin boot builddir dev etc home lib lib64 media  
mnt opt proc root run sbin srv sys tmp usr var

Short answer: **NO — you cannot use Snapper directly on arbitrary directories like `/etc`, `/usr`, `/var`, `/bin`, etc.**

Snapper **only works on Btrfs subvolumes**, and those folders are *not* subvolumes.

Here's the full explanation so you understand what is possible and what is not.

---



## 1. Snapper can ONLY manage Btrfs subvolumes

Snapper needs:

- a real Btrfs subvolume
- its own `.snapshots` directory inside that subvolume

Example of valid Snapper targets:

- `/` (root subvolume) → ✓ working
- `/home` → possible if `/home` is a subvolume
- `/var/lib/machines` → possible (subvolume)
- `/opt/myproj` → possible if you create it as a subvolume first

Folders that are **not** subvolumes **cannot** be managed by Snapper.

---

## ✗ 2. These system paths are NOT subvolumes

Snapper CANNOT manage these:

```
bash
```

```
/afs  
/bin  
/boot  
/builddir  
/dev  
/etc  
/lib  
/lib64  
/media  
/mnt  
/opt  
/proc  
/root  
/run  
/sbin  
/srv  
/sys
```

```
/tmp  
/usr  
/var
```

Why?

Because these are **regular directories inside the same root subvolume**.

Snapper already handles them **as part of your root snapshot**.

Your root Snapper config **already covers every one of those paths** except `/home`.

So:

- You **do not** need a separate Snapper config for `/etc`
  - You **do not** need one for `/usr`
  - You **do not** need one for `/var`
- ...they are already included in the root Snapper snapshots.

---

## ✓ 3. What Snapper *already* protects for you

Your existing root Snapper config covers all system-level directories:

```
bash  
  
/bin      → yes  
/boot     → yes (if it's not a separate partition)  
/etc      → yes  
/usr      → yes  
/lib      → yes  
/var      → mostly yes
```

```
/root      → yes  
/opt       → yes
```

This means:

**If a package breaks something → Snapper rollback fixes it**

**If an update corrupts /etc → Snapper rollback fixes it**

**If you edit a config wrong → Snapper rollback fixes it**

You already have this protection.

---

## ✓ 4. Snapper *can* operate on non-root subvolumes IF you create them

You *could* create separate Btrfs subvolumes if you want Snapper to track them independently.

For example:

```
bash  
  
sudo btrfs subvolume create /opt  
sudo snapper -c opt create-config /opt
```

But this is rarely needed and can complicate boot logic, layout, and cleanup.

Fedora's best practice is:

**Snapper for:**

- / (system)

- `/home` (optional)

### Not Snapper for:

- `/etc`, `/usr`, `/var`, etc. (included in root snapshot anyway)
  - System folders outside `/home`
- 

## ★ Recommended Setup for You (based on your goals)

You already have:

- ✓ Snapper for `/` (system rollback)
- ✓ Pika Backup for `/home/pd` (user data)
- ✓ NO Snapper for `/home` (fine, Pika covers it)

This setup is clean, easy to debug, and restores the entire system:

- Snapper covers all system folders you listed
- Pika covers all your dotfiles and personal settings

This is the correct modern Fedora+Btrfs configuration.

---

## ■ If you want, you can Snapper a custom folder — but only if made a subvolume

If you tell me which **specific folder** you want Snapper to protect separately, I can show you how to convert it safely into a Btrfs subvolume and create a Snapper config for it.

Examples that make sense:

- `/var/lib/libvirt`
- `/var/lib/containers`
- `/opt/my-project`

But again: **you do not need to snap anything that's already inside the root subvolume.**

---

## If you want:

I can show you exactly **what directories are already included in your root Snapper snapshots**, and what separate Snapper configs would actually help you.

Just tell me:

**Which folder do you want Snapper to protect?**

which folder are needed to backup if i reinstall full os?

If you do a **full OS reinstall**, there are only *three categories* of data you need to back up in order to restore your entire environment exactly as it was:

### 🔥 1. Your Home Directory (critical for your personal environment)

This includes:

- `~/.config/`
- `~/.local/share/`
- `~/.cache/` (optional)

- `~/.mozilla/`, `~/.vscode/`, `.gnome/`, `.var/`, `.local/`
- Shell configs (`.bashrc`, `.profile`, `.bash_history`)
- Application configs
- GNOME desktop settings
- Editor + IDE configs
- SSH keys (if used)
- GPG keys (if used)
- Any scripts/programs in your home folder

You already backed these up with **Pika Backup** → ✓ You are safe.

**This is the MOST important category.**

Your entire user experience is stored here.

---

## 🔥 2. Package list (critical for restoring all apps you installed)

Fedora does **not** automatically reinstall your manually installed apps.

So you need this:

```
bash
```

```
dnf history userinstalled > ~/installed-packages.txt
```

Store `installed-packages.txt` in your home folder (so Pika Backup captures it).

**After reinstall:**

```
bash
```

```
sudo dnf install $(cat installed-packages.txt)
```

Boom → all your apps return exactly as before.

If you use Flatpak, also save:

```
bash
```

```
flatpak list --app --columns=application > ~/flatpaks.txt
```

Restore after reinstall:

```
bash
```

```
flatpak install flathub $(cat flatpaks.txt)
```

---

### 🔥 3. System configuration files (optional but very useful)

Most users skip this, but if you have custom system configs:

- /etc/fstab
- /etc/hostname
- /etc/hosts
- /etc/default/grub
- /etc/sudoers.d/\*
- /etc/NetworkManager/system-connections/\*
- Custom systemd units in /etc/systemd/system

Back them up like this:

```
bash
```

```
sudo tar czvf ~/etc-backup.tar.gz /etc
```

Your Pika backup will store this file automatically.

You only restore the parts you need, not the whole `/etc`.

---



## BONUS: Which folders you DON'T need to back up

Many people waste time backing up unnecessary system folders.

Here is what you **do NOT need** to back up when reinstalling:

```
bash
```

```
/bin  
/boot  
/dev  
/lib  
/lib64  
/media  
/mnt  
/opt  
/proc  
/root    (unless you store files there, unusual)  
/run  
/sbin  
/srv  
/sys  
/tmp
```

```
/usr  
/var (except databases or VMs)
```

All of these will be recreated by the Fedora installer.

Snapper backs up these anyway for rollback, **but they are not needed for a reinstall.**

---

👉 **FINAL ANSWER: The ONLY folders you actually need to keep before reinstalling Fedora are:**

✓ `/home/USERNAME`

(Pika backup already covers this)

✓ A list of installed packages

(Use `installed-packages.txt`)

✓ Optional: backup of `/etc`

(`etc-backup.tar.gz`)

That's it.

If you back up these three things, you can rebuild your entire OS and user environment completely.

---

🎯 **If you want, I can generate:**

- a full reinstall recovery guide

- a script to **reapply your DNF + Flatpak packages**
- a checklist you can keep for future reinstalls
- instructions for restoring your Pika archive safely

Just tell me what you'd like next.