Hardware

- Check disk I/O speed
- Clone damaged disk

Check disk I/O speed

우분투에서 디스크의 읽기 및 쓰기 성능을 확인하려면 **dd 명령어, hdparm 명령어, 또는 fio 도구를 사용할 수 있습니다**. 가장 쉽고 빠른 방법은 기본 도구인 `dd`와 `hdparm`을 활용하는 것입니다. [1, 2, 3, 4, 5]

1. dd 명령어로 쓰기 및 읽기 속도 측정하기

`dd` 명령어는 시스템에 파일을 직접 쓰고 읽어보며 간단하게 성능을 측정합니다. [1, 2]

쓰기 속도 테스트:

bash

```
sudo dd if=/dev/zero of=/testfile bs=1G count=1 oflag=direct
```

Use code with caution.

(1GB 크기의 더미 파일을 만들어 디스크에 직접 기록하는 속도를 측정합니다.)

읽기 속도 테스트:

bash

```
sudo dd if=/testfile of=/dev/null bs=1G count=1 iflag=direct
```

Use code with caution.

(앞서 생성한 테스트 파일을 다시 읽어들이는 속도를 측정합니다.)

테스트 완료 후 파일 삭제:

bash

```
sudo rm /testfile
```

Use code with caution.

[1]

---

2. hdparm 명령어로 읽기 속도 측정하기

`hdparm`은 하드디스크와 SSD의 캐시 및 버퍼 읽기 성능을 빠르게 확인하는 전용 도구입니다. [1, 2]

설치 방법:

bash

```
sudo apt update && sudo apt install hdparm
```

Use code with caution.

속도 측정 실행:

bash

```
sudo hdparm -Tt /dev/sda
```

Use code with caution.

(`/dev/sda` 부분에는 확인하려는 실제 디스크 장치명을 입력합니다.)

---

3. 실시간 디스크 사용량 확인 (iostat)

현재 시스템에서 발생하는 실시간 입출력(I/O) 부하를 확인하려면 `sysstat` 패키지를 이용합니다. [1, 2]

설치 방법:

bash

```
sudo apt install sysstat
```

Use code with caution.

실시간 모니터링 실행:

bash

```
iostat -xz 2
```

Use code with caution.

Clone damaged disk

Reallocated Sectors: 90 suddenly increase and gives errors I/O input error. I wish to backup data to another disk any change to lost data?

Yes, there is a strong chance that some data is already lost or at immediate risk.

A sudden spike in reallocated sectors accompanied by I/O errors indicates active physical degradation on the disk media. Standard file copiers (like Windows Explorer or basic `cp`) will hang, freeze, or fail completely when encountering these sectors, worsening drive health in the process.

Physical bad sectors prevent head read operations. Source: Adroit Data Recovery Centre

What Is Happening to Your Disk

- **Reallocated Sectors (90):** The drive controller identified damaged blocks, retired them, and remapped them to spare space. Files stored in those specific sectors prior to reallocation may already be corrupted.
- **I/O Errors:** The drive's read/write head is repeatedly trying to read sectors that have failed but haven't been remapped yet (pending sectors). Every read attempt risks head contact and further mechanical wear.

Critical Next Steps (Do Not Use Standard Copying)

Do **not** run `chkdsk`, `fsck`, or drive diagnostic tools like S.M.A.R.T. extended self-tests. These perform heavy read/write operations that can destroy a failing drive.

1. Prioritize Crucial Files First If the drive is still mounted, manually copy your absolute highest-value files (documents, personal records) individually to a healthy drive before doing anything else.

2. Clone the Drive with a Fault-Tolerant Tool To rescue the maximum amount of data without hanging your system on bad blocks, clone the entire drive to a healthy disk of equal or larger size using a tool designed to skip bad sectors:

- **Linux / Command Line:** Use `ddrescue`. It reads the healthy parts of the drive first, notes the bad sectors, and loops back later to attempt non-destructive retries.

Bash

```
sudo ddrescue -d -r1 /dev/sdX /dev/sdY mapfile.log
```

(Replace `sdX` with your dying drive and `sdY` with your target drive).

- **Windows (GUI):** Use tools like **HDD Raw Copy Tool** or **R-Studio** to create an image file while ignoring I/O errors.

3. Unmount and Replace Once the clone completes, disconnect the failing drive immediately. Perform any file system repair or data recovery operations on the *cloned copy*, never the original failing drive.

If the drive begins making clicking noises or ceases to mount entirely during this process, power it off immediately; at that point, only a professional data recovery lab with cleanroom facilities can safely extract the remaining magnetic data.

Depending on your Operating System and setup, we can tailor the exact recovery command:
Show exact ddrescue commands for Linux rescue
Recommend Windows tools for rescuing dying drives

oh my os is ubunut and Can I clone my data to another machine using tailscale?

Yes, you can clone the drive over Tailscale, but because of the failing state of the drive (I/O errors), you must use a tool that streams block-level data while skipping bad sectors over an SSH tunnel. Standard network copy tools (`scp`, `rsync`) will freeze or fail when they hit bad blocks.

Step-by-Step Recovery over Tailscale

1. Install GNU `ddrescue` on the Local Machine

Install the fault-tolerant recovery tool on the Ubuntu machine with the dying drive:

Bash

```
sudo apt update && sudo apt install gddrescue -y
```

2. Identify Drive Device Names

Find the device node for your failing drive (e.g., `/dev/sdb` or `/dev/nvme0n1`):

Bash

```
lsblk
```

(Verify carefully—do not mistake your system root drive for the dying drive).

3. Stream `ddrescue` to the Remote Machine over Tailscale

Run `ddrescue` on the local machine and pipe its output through SSH to write directly into an image file on the remote machine across your Tailnet:

Bash

```
sudo ddrescue -d -r1 /dev/sdX - remote_user@REMOTE-TAILSCALE-
IP:/path/to/backup/drive_rescue.img mapfile.log
```

- **`/dev/sdX`**: Replace with your dying drive identifier.
- **`-`**: Outputs the rescued raw data directly to `stdout` instead of a local file.
- **`remote_user@REMOTE-TAILSCALE-IP`**: Use the Tailscale IP address or MagicDNS hostname of the destination machine.
- **`/path/to/backup/drive_rescue.img`**: The destination file path on the receiving machine (ensure the receiving disk has enough free space to hold the full capacity of the source drive).

Important Considerations for Network Recovery

- **Bandwidth & Speed limits:** Rescuing a drive over a VPN will be bottlenecked by your upload speed. If the drive has hundreds of gigabytes, this will take significant time. If both machines are on the same local network, use the receiving machine's LAN IP inside the command to maximize speed while keeping Tailscale active.
- **Resume Capability:** Piping `ddrescue` over SSH output directly into a raw file makes mapfile tracking remote-side trickier if the network drops. If network dropouts are a concern, mount the target machine's destination folder locally using `SSHFS` over Tailscale first:
  Bash

```
sudo apt install sshfs -y
mkdir -p /mnt/remote_backup
sshfs remote_user@REMOTE-TAILSCALE-IP:/path/to/backup /mnt/remote_backup
sudo ddrescue -d -r1 /dev/sdX /mnt/remote_backup/drive_rescue.img rescue.log
```

This allows `ddrescue` to save a local mapfile, meaning you can pause or resume the recovery seamlessly if the connection breaks.

Need help setting up SSHFS over Tailscale to preserve your rescue progress log?

ssh port is 2010

To use port `2010`, pass the `-P` parameter (capital **P**) inside the `sshfs` mount command or the `-p` parameter (lowercase **p**) if piping directly through `ssh`.

Using the **SSHFS method** is strongly recommended here—it keeps your `rescue.log` mapfile local so you can safely stop and resume the clone if your Tailscale connection drops.

Method 1: Recommended (SSHFS Mount on Port 2010)

1. Create a local mount point and mount the target machine:

Bash

```
mkdir -p /mnt/remote_backup
sshfs -p 2010 remote_user@REMOTE-TAILSCALE-IP:/path/to/backup /mnt/remote_backup
```

2. Start `ddrescue` writing directly into the mount:

Bash

```
sudo ddrescue -d -r1 /dev/sdX /mnt/remote_backup/drive_rescue.img /var/log/rescue.log
```

(Replace `/dev/sdX` with your failing drive, and update `remote_user` and `REMOTE-TAILSCALE-IP` accordingly).

Method 2: Direct SSH Stream on Port 2010

If you prefer streaming directly without mounting SSHFS, pass `-p 2010` to the `ssh` command:

Bash

```
sudo ddrescue -d -r1 /dev/sdX - | ssh -p 2010 remote_user@REMOTE-TAILSCALE-IP "cat > /path/to/backup/drive_rescue.img"
```

Unmounting When Finished

When the cloning process completes, clean up the mount with:

Bash

```
sudo umount /mnt/remote_backup
```

img is how big ?

The `.img` file will be **the exact total capacity of the dying source disk**, not just the size of the files currently stored on it.

Because `ddrescue` creates a raw, sector-by-sector clone of the underlying physical storage, it copies every byte on the drive—including empty space and corrupted sectors.

Disk Capacity vs. File Size

- **1 TB Hard Drive** → `.img` file will be **1 TB**
- **500 GB SSD/HDD** → `.img` file will be **500 GB**

Example: If you have a 1 TB hard drive that only contains 50 GB of personal photos and documents, the resulting `drive_rescue.img` file will still be **1 TB**.

Crucial Pre-Flight Check

Before starting the transfer, verify that the storage drive on your remote machine has **more free space available than the total size of your failing disk**.

1. Check source drive size (local machine):

Bash

```
lsblk -b /dev/sdX | grep disk
```

(Look at the total size in bytes/gigabytes).

2. Check destination free space (remote machine over SSH port 2010):

Bash

```
ssh -p 2010 remote_user@REMOTE-TAILSCALE-IP "df -h /path/to/backup"
```

*(Ensure the **Avail** column is larger than the total size of your source drive).*

What to Do After the `.img` File Is Created

Once the clone finishes, do **not** try to open the `.img` file like a standard archive. You can mount it directly on the remote Linux machine to browse and copy your healthy files out:

Bash

```
sudo loopsetup=$(sudo losetup -fP --show /path/to/backup/drive_rescue.img)
sudo mount ${loopsetup}p1 /mnt/restored_files
```

(Replace `p1` with the specific partition number you want to access).

Can I just clone one drive to another drive?

Yes, absolutely. Cloning directly from disk to disk (e.g., from `/dev/sdX` to `/dev/sdY`) is actually simpler and much faster because it avoids network bottlenecks, SSH overhead, and intermediate image files.

Prerequisites

1. **Target Drive Size:** The receiving drive must be **equal to or larger in capacity** than the dying source drive.
2. **Drive Identifiers:** You must strictly double-check your drive identifiers using `lsblk` before running the command. Overwriting the wrong drive will permanently wipe it.

Step-by-Step Disk-to-Disk Rescue

1. Identify your drives:

Bash

```
lsblk -o NAME,SIZE,MODEL,SERIAL
```

- Identify your **dying source drive** (e.g., `/dev/sdb`)
- Identify your **healthy destination drive** (e.g., `/dev/sdc`)

2. Run `ddrescue` directly between the drives:

Bash

```
sudo ddrescue -d -r1 /dev/sdX /dev/sdY /var/log/rescue.log
```

- Replace `/dev/sdX` with your **dying source drive**.
- Replace `/dev/sdY` with your **healthy destination drive**.
- `/var/log/rescue.log` saves your progress locally on your root OS so you can pause or resume if needed.

What Happens Next?

Once `ddrescue` finishes:

1. **Disconnect the dying drive** immediately.
2. The destination drive (`/dev/sdY`) is now a 1:1 identical twin of your original disk, including partitions and file systems.
3. You can safely mount the destination drive and run filesystem checks (like `e2fsck` or `ntfsfix`) on it to repair any corruption caused by the bad sectors.

Need help identifying your exact drive letters with lsblk or mounting the rescued disk?

Target disk sde 931.5G Tech K94PT9325BHV

Source disk sda 931.5G D HTS541010A9E63 000000004BA8

Your drive sizes match (both **931.5 GB**), so a direct disk-to-disk clone will work.

- **Source (Dying):** `/dev/sda`
- **Target (Destination):** `/dev/sde`

Step 1: Run `ddrescue`

Execute the following command to begin copying data directly from `sda` to `sde`:

Bash

```
sudo ddrescue -d -r1 /dev/sda /dev/sde /var/log/sda_rescue.log
```

- `-d`: Direct disk access (bypasses OS kernel cache to read straight from hardware).
- `-r1`: Retries unreadable sectors once before skipping ahead, prioritizing maximum data recovery without destroying the drive.
- `/var/log/sda_rescue.log`: Saves the mapfile to your root OS so you can safely interrupt (Ctrl+C) and resume later if necessary.

Step 2: What to Expect During the Run

1. **Pass 1 (Copying):** It will quickly copy all the undamaged sectors first.
2. **Pass 2 (Trimming/Scraping):** It will return to the bad areas and attempt to extract data around the 90 bad sectors you encountered.
3. Keep an eye on the `pct rescued` and `errsize` lines in the output.

Step 3: Post-Rescue Cleanup

Once `ddrescue` reports finished:

1. **Unplug** `/dev/sda` **immediately** to prevent further mechanical wear.
2. Re-scan your partitions or reboot your system:
   Bash

   ```
   sudo partprobe /dev/sde
   ```

3. Your `/dev/sde` drive will now have the exact partitions and file structure that `/dev/sda` had. You can safely mount `/dev/sde` and check your files.

Need help mounting partitions or running a filesystem repair on the new sde target drive?

IS this work can be done via tailscale?

Yes, absolutely. You can execute this disk-to-disk clone across machines connected via Tailscale.

However, because you are cloning a whole **931.5 GB drive directly over the network**, there are critical speed and command adjustments to keep in mind.

Step-by-Step: Disk-to-Disk Clone Over Tailscale

Assuming `/dev/sda` is on your **Local Machine** (with the dying drive) and `/dev/sde` is physically attached to the **Remote Machine** on your Tailnet.

1. Command to Run on Local Machine (Port 2010)

Pipe `ddrescue` directly through SSH to write onto `/dev/sde` on the remote side:

Bash

```
sudo ddrescue -d -r1 /dev/sda - | ssh -p 2010 remote_user@REMOTE-TAILSCALE-IP "sudo dd
of=/dev/sde status=progress bs=64k"
```

- `/dev/sda -`: Tells `ddrescue` to read the local drive and output the raw stream to standard output (`stdout`).
- `ssh -p 2010`: Connects through your non-standard SSH port over Tailscale.
- `sudo dd of=/dev/sde`: Takes the incoming network stream on the remote machine and writes it directly to the physical target disk (`sde`).
- `bs=64k`: Sets block size to optimize network throughput.

Critical Considerations Before You Start

1. Transfer Time (Network Bottleneck)

Over a local disk bus (SATA/USB 3), cloning 931.5 GB takes roughly **2 to 3 hours**. Over a Tailscale network connection, your speed will be limited by your **upload speed**:

Network Upload Speed	Estimated Time to Transfer 931.5 GB
100 Mbps (~12.5 MB/s)	~ **21 hours**
500 Mbps (~62.5 MB/s)	~ **4.2 hours**
1 Gbps (~125 MB/s)	~ **2.1 hours**

*If both machines are on the same physical local network (e.g., same router/switch), use the destination machine's **local LAN IP** inside the SSH command instead of the Tailscale IP to get full gigabit speeds.*

2. Risk of Network Dropouts

Piping `ddrescue` directly over SSH standard output does **not** allow saving a detailed bad-sector mapfile across the network connection. If your network hiccups or SSH drops mid-transfer, you cannot resume from where you left off; you would have to start from the beginning.

Recommendation: Run the command inside a `tmux` or `screen` session on your local terminal so that a closed SSH/terminal session won't kill the running process:

Bash

```
tmux new -s rescue
```

(If disconnected, re-attach later using `tmux a -t rescue`)

Want instructions on using tmux so your transfer doesn't get interrupted if your terminal closes?

need to format for target disk ?

No, you do not need to format the target disk (`/dev/sde`).

In fact, formatting it beforehand would be a wasted step. `ddrescue` performs a low-level, sector-by-sector copy—meaning it will overwrite every single byte on the target disk, including its partition table, file system headers, and formatting data.

What Will Happen to the Target Disk (`/dev/sde`)

- Any existing partitions or files on `/dev/sde` will be **completely wiped and replaced**.
- Once the transfer finishes, `/dev/sde` will become an exact duplicate of `/dev/sda` (same partition structure, same file system type, same layout).

One Important Prerequisite

While you don't need to format `/dev/sde`, you must ensure that **none of its partitions are mounted** before running the recovery command:

Bash

```
sudo umount /dev/sde* 2>/dev/null
```

This prevents the Linux kernel on the receiving machine from writing to `/dev/sde` while `ddrescue` is actively overwriting the raw drive blocks over your network stream.

41250 bytes (41 kB, 40 KiB) copied, 76 s, 0.5 kB/sddrescue: /dev/sda: Fatal error reading the input file: No such device

41720 bytes (42 kB, 41 KiB) copied, 77 s, 0.5 kB/s

0+78 records in

0+78 records out

41720 bytes (42 kB, 41 KiB) copied, 76.9141 s, 0.5 kB/s

`Fatal error reading the input file: No such device` means your dying hard drive (`/dev/sda`) has physically disconnected from the machine.

When a drive with severe bad sectors suffers repeated read failures, its internal controller crashes, or the Linux kernel resets the SATA/USB bus to prevent system lockup. As a result, the hardware completely disappears from `/dev/sda`. Notice how it only managed to copy ~41 kB before disconnecting.

Step 1: Check if the Drive Is Still Detected

Run this command to see if the system can still see the hardware:

Bash

```
lsblk
```

Or check kernel logs to see the drop event:

Bash

```
dmesg -T | grep -i "ata\|sd\|error" | tail -n 20
```

Step 2: How to Re-identify or Reset the Drive

If `lsblk` no longer lists `/dev/sda`:

1. **Power Cycle / Physical Reset:**
   - If it is an **external USB drive**: Unplug the USB cable and power source, wait 10 seconds, and plug it back in.
   - If it is an **internal SATA drive**: Shut down the machine completely, power off the PSU, wait 30 seconds, and boot back up.
2. **Check the New Drive Letter:** When the drive reconnects, Linux may assign it a **different letter** (e.g., `/dev/sdb` or `/dev/sdc` instead of `/dev/sda`). Always verify with `lsblk` before running the next command.

Step 3: Crucial Strategy Adjustment for `ddrescue`

Your drive crashed because `ddrescue` attempted direct access (`-d`) on a severely degraded sector right at the beginning of the disk.

When you restart `ddrescue`, **remove the `-d` (direct I/O) flag** initially and add `--scrape-members` / `--no-scrape` options to skip damaged areas instantly without hanging the drive controller:

Bash

```
sudo ddrescue -n -b 4096 /dev/sdX - | ssh -p 2010 remote_user@REMOTE-TAILSCALE-IP "sudo dd of=/dev/sde status=progress bs=64k"
```

- **`-n` (`--no-scrape`)**: Tells `ddrescue` to skip bad areas immediately instead of retrying them. This prevents the drive from hanging and disconnecting.
- **`-b 4096`**: Force a 4KiB block size (many modern drives use 4k physical sectors; attempting 512-byte reads can trigger controller panics on failing drives).
- **`/dev/sdX`**: Replace with the **new drive letter** assigned after reconnecting.

Warning Signs of Permanent Hardware Failure

If the drive disconnects again almost immediately after starting (within seconds or after copying just a few kilobytes):

- The drive's read heads or PCB controller are failing permanently.
- Continued attempts will degrade the platters further. At that stage, software cloning over a network won't work, and recovery requires specialized hardware tools or a professional recovery service.

disk is still connected and

[Mon Aug 3 13:08:38 2026] I/O error, dev sda, sector 0 op 0x0:(READ) flags 0x800 phys_seg 16 prio class 2

[Mon Aug 3 13:21:05 2026] sd 1:0:0:0: [sda] tag#1 uas_zap_pending 0 uas-tag 1 inflight: CMD

[Mon Aug 3 13:21:05 2026] sd 1:0:0:0: [sda] tag#1 CDB: Read(10) 28 00 00 b8 70 80 00 00 80 00

[Mon Aug 3 13:21:05 2026] sd 1:0:0:0: [sda] tag#1 FAILED Result: hostbyte=DID_NO_CONNECT driverbyte=DRIVER_OK cmd_age=0s

[Mon Aug 3 13:21:05 2026] sd 1:0:0:0: [sda] tag#1 CDB: Read(10) 28 00 00 b8 70 80 00 00 80 00

[Mon Aug 3 13:21:05 2026] I/O error, dev sda, sector 12087424 op 0x0:(READ) flags 0x800 phys_seg 16 prio class 2

[Mon Aug 3 13:21:05 2026] device offline error, dev sda, sector 12107136 op 0x0:(READ) flags 0x800 phys_seg 16 prio class 2

[Mon Aug 3 13:21:05 2026] sd 1:0:0:0: [sda] Synchronizing SCSI cache

[Mon Aug 3 13:21:05 2026] sd 1:0:0:0: [sda] Synchronize Cache(10) failed: Result: hostbyte=DID_ERROR driverbyte=DRIVER_OK

[Mon Aug 3 13:21:06 2026] EXT4-fs (sda): unmounting filesystem dc434a0a-8500-41a2-9fd3-37b3adef0cc5.

[Mon Aug 3 13:21:09 2026] sd 1:0:0:0: Attached scsi generic sg0 type 0

[Mon Aug 3 13:21:09 2026] sd 1:0:0:0: [sda] 1953525168 512-byte logical blocks: (1.00 TB/932 GiB)

[Mon Aug 3 13:21:09 2026] sd 1:0:0:0: [sda] 4096-byte physical blocks

[Mon Aug 3 13:21:09 2026] sd 1:0:0:0: [sda] Write Protect is off

[Mon Aug 3 13:21:09 2026] sd 1:0:0:0: [sda] Mode Sense: 53 00 00 08

[Mon Aug 3 13:21:09 2026] sd 1:0:0:0: [sda] Disabling FUA

[Mon Aug 3 13:21:09 2026] sd 1:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA

[Mon Aug 3 13:21:09 2026] sd 1:0:0:0: [sda] Preferred minimum I/O size 4096 bytes

[Mon Aug 3 13:21:09 2026] sd 1:0:0:0: [sda] Optimal transfer size 33553920 bytes not a multiple of preferred minimum block size (4096 bytes)

[Mon Aug 3 13:21:09 2026] sd 1:0:0:0: [sda] Attached SCSI disk

The logs show two very important things:

1. **UAS Protocol Crash (`uas_zap_pending`, `DID_NO_CONNECT`):** You are using a USB enclosure that relies on the **UAS (USB Attached SCSI)** protocol. UAS is fast, but it panics and completely drops the drive connection when it encounters disk read timeouts or bad sectors.
2. **Drive Auto-Reattached:** At `13:21:09`, the kernel reset the USB bus and re-attached `/dev/sda`. However, as long as UAS remains active, any attempt to read bad sectors will crash the USB controller again.

To fix this, you must **disable UAS** for this drive so it falls back to the legacy, rock-solid `usb-storage` protocol. The standard `usb-storage` driver handles read errors gracefully without dropping the entire device.

Step 1: Disable UAS and Force `usb-storage`

1. Find your USB enclosure's Vendor and Product ID:

Bash

```
lsusb | grep -i "terramaster\|orico\|sabrent\|enclosure\|drive"
```

(Or simply run `lsusb` to see all USB devices). Look for a string like `ID 152d:0578` (where `152d` is Vendor ID and `0578` is Product ID).

2. Unbind UAS temporarily using `modprobe`: Run this command (replace `152d:0578` with your actual IDs):

Bash

```
sudo modprobe -r uas
sudo modprobe usb-storage quirks=152d:0578:u
```

(The `:u` flag forces Linux to ignore UAS and use standard `usb-storage` for this device).

3. Unplug and replug the USB cable. Verify it dropped UAS by checking `dmesg`:

Bash

```
dmesg | tail -n 15
```

You should see `USB Mass Storage device detected` instead of `UAS driver`.

Step 2: Disable Automatic Mounting

Notice line `13:21:06` in your log showing `EXT4-fs (sda): unmounting`. Ubuntu tried to automount the file system, which triggers immediate background reads on corrupted sectors.

Prevent Ubuntu from automounting or touching the partition structure while cloning:

Bash

```
gsettings set org.gnome.desktop.media-handling automount false
```

Step 3: Run `ddrescue` with Gentle Flags

Now that UAS is disabled, retry cloning over Tailscale. Use the **`-n`** **(`--no-scrape`)** option and a **4096-byte block size** so `ddrescue` quickly skips bad blocks instead of getting stuck:

Bash

```
sudo ddrescue -n -b 4096 /dev/sda - | ssh -p 2010 remote_user@REMOTE-TAILSCALE-IP "sudo dd of=/dev/sde status=progress bs=64k"
```

- Because UAS is disabled, when `ddrescue` hits sector `12087424` or sector `0`, the kernel will simply return an I/O error to `ddrescue`.
- `ddrescue` will mark that block as bad, skip ahead, and keep copying the rest of your healthy data without crashing the USB bus.

us 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

Bus 002 Device 002: ID 2109:2812 VIA Labs, Inc. VL812 Hub

Bus 002 Device 003: ID 13d3:5741 IMC Networks Lenovo EasyCamera

Bus 002 Device 004: ID 105b:e065 Foxconn International, Inc. BCM43142A0 Bluetooth module

Bus 003 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub

Bus 003 Device 002: ID 2109:0812 VIA Labs, Inc. VL812 Hub

Bus 003 Device 004: ID 2537:1068 Norelsys NS1068/NS1068X SATA Bridge Controller

Bus 003 Device 005: ID 0bda:8153 Realtek Semiconductor Corp. RTL8153 Gigabit Ethernet Adapter

Bus 003 Device 006: ID 152d:0578 JMicron Technology Corp. / JMicron USA Technology Corp. JMS578 SATA 6Gb/s

You have two external SATA adapters connected. The one causing the UAS crash is the **JMicron JMS578 (`152d:0578`)** controller on Bus 003.

Step 1: Force `usb-storage` Driver for JMicron (`152d:0578`)

Unload the crashing `uas` module and reload `usb-storage` with the quirk flag for `152d:0578`:

Bash

```
sudo modprobe -r uas
sudo modprobe usb-storage quirks=152d:0578:u
```

Step 2: Cycle the Connection

1. Unplug the USB cable for the JMicron enclosure, wait 5 seconds, and plug it back in.
2. Verify that UAS is disabled by checking the kernel log:
   Bash

   ```
   dmesg | tail -n 20
   ```

   Look for `UAS is ignored for this device` or `USB Mass Storage device detected` instead of `uas`.

Step 3: Check Drive Assignment & Disable Auto-Mount

1. Confirm the drive letter hasn't shifted:
   Bash

   ```
   lsblk
   ```

2. Stop GNOME from auto-mounting corrupted partitions when attached:
   Bash

```
gsettings set org.gnome.desktop.media-handling automount false
```

Step 4: Resume `ddrescue`

Run `ddrescue` with `--no-scrape` (`-n`) and a 4KiB block size (`-b 4096`). It will now report read errors to `ddrescue` instantly and skip ahead without crashing the USB adapter:

Bash

```
sudo ddrescue -n -b 4096 /dev/sda - | ssh -p 2010 remote_user@REMOTE-TAILSCALE-IP "sudo dd of=/dev/sde status=progress bs=64k"
```

(Replace `/dev/sda` if `lsblk` shows a different letter after re-plugging).

after dd is finished, how to know ?which files are damaged

To identify which specific files are damaged, you need a map of where the bad sectors are located on the disk, and then you cross-reference those bad sector addresses with your filesystem's inode table using `testdisk` or `debugfs`.

Because you piped `ddrescue` directly into `dd` over SSH, **no local `rescue.log` mapfile was generated**. However, you can still pinpoint corrupt files directly on the cloned target disk (`/dev/sde`).

Step 1: Scan the Target Disk for Bad Sectors

Run `badblocks` on the cloned target drive `/dev/sde`. Since `/dev/sde` now contains an exact sector-by-sector copy of `/dev/sda`, reading `/dev/sde` will reveal where unreadable/skipped zeros were written:

Bash

```
sudo badblocks -v -b 4096 /dev/sde > bad_sectors.txt
```

(Save the list of bad sector block addresses to `bad_sectors.txt`).

Step 2: Map Bad Sectors to Corrupted Files

Depending on your file system type (e.g., `ext4` or `NTFS`), use the appropriate tool below to match those sector numbers to actual file paths.

Option A: For EXT4 Filesystems (Using `debugfs`)

1. Find the partition containing your files (e.g., `/dev/sde1`):
   Bash

   ```
   sudo lsblk /dev/sde
   ```

2. Open the partition in `debugfs`:
   Bash

   ```
   sudo debugfs /dev/sde1
   ```

3. Inside the `debugfs:` prompt, test a block number from your `bad_sectors.txt` file to find its corresponding inode:
   Plaintext

   ```
   debugfs: icheck <BLOCK_NUMBER>
   ```

   (This outputs an inode number, e.g., `Block 12087424 has inode 524289`).
4. Get the exact file path for that inode:
   Plaintext

   ```
   debugfs: ncheck 524289
   ```

   (This outputs the exact file path, e.g., `/home/user/Documents/report.pdf`).

Option B: Automated GUI Recovery (Using `TestDisk`)

`TestDisk` is much easier if you prefer an automated tool to scan and list damaged files across `ext4`, `NTFS`, or `FAT32` filesystems:

1. Install `testdisk`:
   Bash

   ```
   sudo apt install testdisk -y
   ```

2. Run `testdisk` on the target disk:
   Bash

   ```
   sudo testdisk /dev/sde
   ```

3. Select **Proceed** → Choose your partition table style (usually **EFI GPT**) → **Analyse** → **Quick Search**.
4. Highlight your partition and press `P` to list files. `TestDisk` will display files in red if their metadata or data blocks are unreadable or damaged.

Step 3: Run Filesystem Repair (`e2fsck`)

Before mounting `/dev/sde` for normal use, repair any filesystem metadata inconsistencies caused by bad sectors:

Bash

```
sudo e2fsck -f -y /dev/sde1
```

- **Important:** Running `e2fsck` will fix the file system structure so the drive can mount safely. Any unrecoverable blocks within files will be zeroed out or placed into a `/lost+found` directory on the drive.

Basically , target disk has no bad sector physically, then badsector information from source disk affects on target disk ?

No, the target disk will not gain physical bad sectors.

Because `/dev/sde` is a healthy physical drive, its physical platters or NAND flash cells remain 100% fine. However, **logical/file system corruption** will be copied over.

Physical vs. Logical Bad Sectors

Feature	Source Disk (`/dev/sda`)	Target Disk (`/dev/sde`)
Physical Media	Damaged (hardware read failures)	Healthy (0 physical bad sectors)
Data in Bad Sectors	Unreadable	Filled with **zeros** by `ddrescue`
File System Status	Corrupted	Corrupted (reads zeros where data used to be)

What Actually Happens to Target Disk (`/dev/sde`)

1. **Zeros written to bad spots:** When `ddrescue` encounters a bad sector on `/dev/sda` that it cannot read, it fills those exact block spaces on `/dev/sde` with **zeros** (empty data) so the clone can continue.
2. **Missing file pieces:** The physical sectors on `/dev/sde` are perfectly capable of being read and written to, but the *content* inside those specific files is now corrupted or truncated.
3. **File system structure:** If bad sectors occurred inside file system metadata (e.g., the superblock or inode table), the target disk may refuse to mount initially until you run a repair command (`e2fsck`).

How to Fix the File System on Target Disk

Once `ddrescue` completes, run a file system check on `/dev/sde`. This cleans up the logical errors so the operating system can safely use the drive:

Bash

```
sudo e2fsck -f -y /dev/sde1
```

- This fixes any corrupted folder structures.
- The healthy target disk will function normally going forward, though any specific files that were sitting on `sda`'s bad sectors will have missing/zeroed-out data or be moved to `/lost+found`.

Can I know the list of damaged files?

Yes, you can generate a complete list of damaged files directly from the target disk (`/dev/sde`), even though it has no physical bad sectors of its own.

Because `ddrescue` writes **all zeros** to the target disk whenever it encounters an unreadable sector on the source drive, any file containing those zeroed-out blocks is corrupted.

Method 1: The Automated Way (Using `ddrescue-tools`)

If you want a direct list of damaged file paths, use `ddrescue-view` or `ddrescue-tools` to parse the cloned partition:

1. **Install `ddrescue-tools`:**
   Bash

   ```
   sudo apt update && sudo apt install ddrescue-tools -y
   ```

2. **Run `ddrescue-findfiles` against the target partition:**
   Bash

   ```
   sudo ddrescue-findfiles /dev/sde1 /var/log/sda_rescue.log
   ```

   (This utility maps the log file's bad sector offsets directly against the filesystem on `/dev/sde1` and prints out every impacted path).

Method 2: Finding Corrupted Files manually on EXT4 (`debugfs`)

If you don't have a mapfile from `ddrescue`, you can scan `/dev/sde1` for filesystem blocks that contain unreadable metadata or lost inodes:

1. **Run a thorough filesystem check on the cloned partition:**
   Bash

   ```
   sudo e2fsck -f -v -y /dev/sde1 | tee filesystem_repair.log
   ```

   - As `e2fsck` runs, it will print messages whenever it encounters corrupted inodes or unreadable file blocks (e.g., `Inconsistency found in inode 12345`).

- Any files that are severely damaged and disconnected from their parent directory will be safely moved to a folder named `/lost+found` on `/dev/sde1`.

2. **Mount `/dev/sde1` to inspect `/lost+found`:**
   Bash

```
sudo mkdir -p /mnt/target
sudo mount /dev/sde1 /mnt/target
ls -la /mnt/target/lost+found
```

Method 3: Finding Zeroed / Damaged Files by Checksum Verification

If you have a previous backup or known checksums (like `.md5` or `.sha256` files) for your media, run a verification scan over your mounted files:

Bash

```
find /mnt/target -type f -exec md5sum {} + > /tmp/checksums.txt 2> /tmp/corrupted_files.txt
```

Any file that fails to read or produces an error will be logged into `/tmp/corrupted_files.txt`.