

ChatGPT

- New chat
- Search chats
- Library
- Sora
- GPTs

Chats

- Supabase data retention backup
- RLS policy conflict analysis
- Fix Netlify build error
- Building expansion cost estimate
- Alex saves Suzie
- Unlock RedMagic 10 Pro
- Sync local folder with GitHub
- Netlify build error fix
- GitHub setup and usage
- Police badge icon creation
- Fix expo app error
- Add code to app.json
- App jurisdiction lookup issue
- Netlify Expo build issue
- NPM dependency conflict fix

I work for a police department where we have a car fleet of approximately 100 cars. I've built a website using typescript and supabase for authentication and storing data in tables. Part of my website is creating work orders for cars to be repaired. How long should I expect supabase to keep the records before it stops working? How can I periodically backup the data to prevent history of repairs being lost?

### How long Supabase keeps your records

- Supabase uses PostgreSQL databases under the hood.
- Your data does not expire or get deleted automatically.
- Records stay in your tables until you delete them or your project is shut down by you (or suspended for unpaid billing).
- So you don't have to worry about Supabase clearing old work orders—it will keep all repair history as long as your project is active.

### Why backups matter

Even though Supabase doesn't delete your data, you should still back it up because of:

- Human error (someone accidentally deletes rows or tables).
- Corruption or bugs in your app that overwrite data.
- Account issues (billing lapses, project deletion, or access loss).
- Compliance and audit needs (police fleet repair history is probably required to be retained).

### How to back up your data

#### 1. Supabase built-in backups

- Paid Supabase plans include daily automated backups you can restore from.
- On free plans, backups are not included, so you'll need to set them up yourself.

#### 2. Scheduled exports

- You can set up a script (Node.js or TypeScript) that runs daily/weekly to export tables into CSV or JSON.
- Store these backups in cloud storage (Amazon S3, Google Drive, OneDrive, etc.) or on a secure department server.
- Example tools:
  - pg\_dump (standard PostgreSQL backup command).
  - Supabase client libraries (`supabase-js`) to fetch data and write it to a file.

#### 3. Replication to another database

- You can use logical replication (Postgres feature) to copy your Supabase data in real-time to another Postgres database.
- This ensures you always have a "hot" backup outside Supabase.

#### 4. Automated backup workflow idea

- Create a cron job (scheduled task) that:
  1. Runs `pg_dump` against your Supabase database.
  2. Compresses the backup file.
  3. Uploads it to secure storage.
- Example schedule: nightly at 2 AM.

### Suggested approach for your fleet system

- Use Supabase's automated backups (if you're on a paid plan).
- Add a weekly export to CSV (`work_orders` table and related tables) stored on your department's secure server.
- Optionally, replicate the database to an internal Postgres instance so IT always has a copy independent of Supabase.

Do you want me to write you a ready-to-use Node.js/TypeScript script that runs on a schedule and saves your Supabase `work_orders` table as a CSV backup? That way you'd have both Supabase's backups and your own archives.

⬇️ ⏪ ⏴ ⏵ ⏹ ⏺ ⏷

Yes please

Here's a simple TypeScript script you can run (on a server, or even your work computer with a scheduler like cron or Windows Task Scheduler) to export your `work_orders` table to CSV and save it locally.

### 1. Install dependencies

Run in your project folder:

```
bash
npm install @supabase/supabase-js json2csv fs
```

- `json2csv`: Converts JSON data to CSV
- `fs`: Node's file system module (built-in)

### 2. Create script `backupWorkOrders.ts`

```

ts
import { createClient } from "@supabase/supabase-js";
import { parse } from "json2csv";
import fs from "fs";

// 🛡 Store these securely (e.g., environment variables)
const SUPABASE_URL = process.env.SUPABASE_URL || "https://YOUR_PROJECT.supabase.co";
const SUPABASE_KEY = process.env.SUPABASE_KEY || "YOUR_SERVICE_ROLE_KEY";

const supabase = createClient(SUPABASE_URL, SUPABASE_KEY);

async function backupWorkOrders() {
  console.log("Starting backup...");

  // Fetch all work orders
  const { data, error } = await supabase.from("work_orders").select("*");

  if (error) {
    console.error("Error fetching data:", error.message);
    return;
  }

  if (!data.length) {
    console.log("No work orders found.");
    return;
  }

  // Convert to CSV
  const csv = parse(data);

  // Create timestamped filename
  const timestamp = new Date().toISOString().replace(/[:.]/g, "-");
  const fileName = `work_orders_backup_${timestamp}.csv`;

  // Save Locally
  fs.writeFileSync(fileName, csv);
  console.log(`Backup saved as ${fileName}`);
}

backupWorkOrders();

```

### 3. Run the script

```

bash
ts-node backupWorkOrders.ts

```

It will create a file like:

```

work_orders_backup_2025-08-21T03-15-22-000Z.csv

```

### 4. Automate it

- **Linux/Mac:** Add to `cron` (example: every night at 2AM)
- ```

bash
0 2 * * * ts-node /path/to/backupWorkOrders.ts

```
- **Windows:** Use Task Scheduler to run `ts-node backupWorkOrders.ts` on a schedule.

### 5. Extra improvements (optional)

- Save directly to Google Drive, OneDrive, or department server instead of local disk.
- Encrypt backups if they're stored outside your secure department systems.
- Extend script to back up **multiple tables** (cars, repairs, users, etc).

Do you want me to expand this script so it backs up every table in your database automatically, not just `work_orders`?

Yes

Got it — let's build a script that pulls every table in your Supabase database and saves them as **separate**

### 1. Install dependencies

(If not already done)

```

bash
npm install @supabase/supabase-js json2csv fs

```

### 2. Create script `backupAllTables.ts`

```

ts
import { createClient } from "@supabase/supabase-js";
import { parse } from "json2csv";

```

```

import fs from "fs";

// 🚀 Store securely in environment variables
const SUPABASE_URL = process.env.SUPABASE_URL || "https://YOUR_PROJECT.supabase.co";
const SUPABASE_KEY = process.env.SUPABASE_KEY || "YOUR_SERVICE_ROLE_KEY";

const supabase = createClient(SUPABASE_URL, SUPABASE_KEY);

async function getAllTables(): Promise<string[]> {
  // Query PostgreSQL system catalog to list all "public" schema tables
  const { data, error } = await supabase
    .rpc("get_all_tables"); // We'll create this RPC in step 3

  if (error) {
    console.error("Error fetching table list:", error.message);
    return [];
  }

  return data || [];
}

async function backupAllTables() {
  console.log("Starting full database backup...");

  const tables = await getAllTables();
  if (tables.length === 0) {
    console.log("No tables found.");
    return;
  }

  const timestamp = new Date().toISOString().replace(/[:.]/g, "-");
  const backupDir = `db_backup_${timestamp}`;
  fs.mkdirSync(backupDir);

  for (const table of tables) {
    console.log(`Backing up table: ${table}`);

    const { data, error } = await supabase.from(table).select("*");

    if (error) {
      console.error(`✖ Error fetching ${table}:`, error.message);
      continue;
    }
  }
}

try {
  const csv = parse(data);
  const filePath = `${backupDir}/${table}.csv`;
  fs.writeFileSync(filePath, csv);
  console.log(`✓ Saved ${table} (${data.length} rows}`);
} catch (err) {
  console.error(`✖ Error writing ${table}:`, err);
}
}

console.log(`⚡ Backup complete. Files saved in: ${backupDir}`);
}

backupAllTables();

```

### 3. Create the helper RPC in Supabase

The script needs a list of all your tables.

You can add a **Postgres** function (RPC) inside Supabase SQL Editor:

```

sql          ⌂ Copy ⌂ Edit

create or replace function get_all_tables()
returns text[]
language sql
{
  select array_agg(tablename::text)
  from pg_tables
  where schemaname = 'public';
$$;

```

This returns an array of all table names in your public schema.

### 4. Run the script

```

bash          ⌂ Copy ⌂ Edit

ts-node backupAllTables.ts

```

You'll get a folder like:

```

db_backup_2025-08-21T03-45-00-00Z/
  work_orders.csv
  cars.csv
  users.csv
  repairs.csv
  ...

```

### 5. Automate it

**5. Automate it**

- Linux/Mac: cron job
- Cloud option: Run this script inside a GitHub Action or simple server to back up daily.

---

**6. Optional improvements**

- Zip the whole folder after backup (`adm-zip` or `archiver` package).
- Upload to Google Drive / OneDrive / secure police server.
- Encrypt backup files before uploading.
- Add retention policy (e.g., keep 30 days of backups, then delete older).

Do you want me to extend this so the script **zips the backup folder and uploads it to a secure cloud location** (Google Drive, OneDrive, or S3)? That way you don't have dozens of loose CSV folders building up.

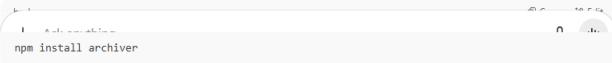
Yes

Perfect — let's extend the script so that after exporting all tables, it will **zip the backup folder** into a single file and then upload it to a secure storage location.

I'll show you how to do this in 3 steps:

---

**◆ Step 1: Install extra dependencies**



- `archiver`: makes a `.zip` file.
- For cloud storage, you'll need different SDKs depending on where you want the backups to live:
  - Google Drive → `npm install googleapis`
  - OneDrive (Microsoft Graph) → `npm install @microsoft/microsoft-graph-client isomorphic-fetch`
  - Amazon S3 → `npm install @aws-sdk/client-s3`

(We'll add code for S3 first since it's the simplest; Google/OneDrive need OAuth setup.)

---

**◆ Step 2: Update the backup script**

`backupAllTables.ts` (extended to zip + upload to S3)

```

ts
Copy Edit

import { createClient } from "@supabase/supabase-js";
import { parse } from "json2csv";
import fs from "fs";
import archiver from "archiver";
import { S3Client, PutObjectCommand } from "@aws-sdk/client-s3";

// Environment variables
const SUPABASE_URL = process.env.SUPABASE_URL || "https://YOUR_PROJECT.supabase.co";
const SUPABASE_KEY = process.env.SUPABASE_KEY || "YOUR_SERVICE_ROLE_KEY";
const AWS_BUCKET = process.env.AWS_BUCKET || "your-backup-bucket";
const AWS_REGION = process.env.AWS_REGION || "east-1";

const s3 = new S3Client({ region: AWS_REGION });

async function getAllTables(): Promise<string[]> {
  const { data, error } = await supabase.rpc("get_all_tables");
  if (error) {
    console.error("Error fetching table list:", error.message);
    return [];
  }
  return data || [];
}

async function backupAllTables() {
  console.log("Starting full database backup...");

  const tables = await getAllTables();
  if (tables.length === 0) {
    console.log("No tables found.");
    return;
  }

  const timestamp = new Date().toISOString().replace(/\.:./g, "-");
  const backupDir = `db_backup_${timestamp}`;
  fs.mkdirSync(backupDir);

  // Export each table as CSV
  for (const table of tables) {
    console.log(`Backing up table: ${table}`);
    const { data, error } = await supabase.from(table).select("*");
    if (error) {
      continue;
    }
    if (!data || data.length === 0) {
      console.log(`(empty) ${table}`);
      continue;
    }
  }
}

```

```

try {
  const csv = parse(data);
  fs.writeFileSync(`${backupDir}/${table}.csv`, csv);
  console.log(`Saved ${table} (${data.length} rows)`);
} catch (err) {
  console.error(`Error writing ${table}:`, err);
}

// Zip the folder
const zipFile = `${backupDir}.zip`;
await new Promise((resolve, reject) => {
  const output = fs.createWriteStream(zipFile);
  const archive = archiver("zip", { zlib: { level: 9 } });

  output.on("close", () => resolve());
  archive.on("error", (err) => reject(err));

  archive.pipe(output);
  archive.directory(backupDir, false);
  archive.finalize();
});

// Upload to S3
try {
  const fileContent = fs.readFileSync(zipFile);
  await s3.send(
    new PutObjectCommand({
      Bucket: AWS_BUCKET,
      Key: `backups/${zipFile}`,
      Body: fileContent,
    })
  );
  console.log(`Uploaded to S3: s3://${AWS_BUCKET}/backups/${zipFile}`);
} catch (err) {
  console.error(`Error uploading to S3:`, err);
}

console.log(`Backup complete.`);
}

backupAllTables();

```

### ◆ Step 3: Secure & Schedule

- Add required environment variables (`.env` file or server config):

`env` Copy Edit

```
SUPABASE_URL=https://yourproject.supabase.firebaseio.com
SUPABASE_KEY=your-service-role-key
```

`AWS_REGION=us-east-1
AWS_ACCESS_KEY_ID=your-aws-key
AWS_SECRET_ACCESS_KEY=your-aws-secret` Copy Edit

- Schedule with cron (Linux/Mac) or Task Scheduler (Windows). Example cron (runs daily at 2 AM):

`bash` Copy Edit

```
0 2 * * * ts-node /path/to/backupAllTables.ts
```

Result:

- Each run creates a timestamped `.zip` with all tables in CSV form.
- File automatically uploads to your S3 bucket for safe, redundant storage.
- You can adapt the upload part for Google Drive or OneDrive if your department prefers Microsoft/Google tools.

Do you want me to adapt the upload section for OneDrive (Microsoft Graph) instead of S3, since police departments often run on Microsoft 365?

Copy Edit Delete Share Up Down

Yes

Great — let's switch the upload section to OneDrive (Microsoft Graph API) so your zipped backup automatically goes into your department's Microsoft 365 storage.

### ◆ Step 1: Install dependencies

`bash` Copy Edit

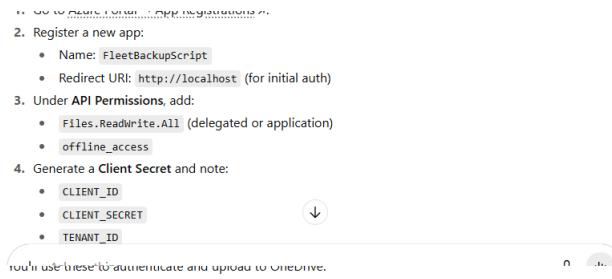
```
npm install @microsoft/microsoft-graph-client isomorphic-fetch archiver
```

- `@microsoft/microsoft-graph-client`: Microsoft Graph SDK (works with OneDrive, SharePoint, Teams).
- `isomorphic-fetch`: Required for Graph SDK to run in Node.
- `archiver`: Already used for zipping.

### ◆ Step 2: Azure App Registration (One-Time Setup)

You need an app registration in Azure AD (your IT may need to do this):

1 Go to Azure Portal → App Registrations »



### ◆ Step 3: Update the Script

`backupAllTables.ts` (OneDrive version, replacing S3 upload part):

```

ts Copy Edit

import { createClient } from "@supabase/supabase-js";
import { parse } from "json2csv";
import fs from "fs";
import archiver from "archiver";
import fetch from "isomorphic-fetch";
import { Client } from "@microsoft/microsoft-graph-client";

// Env vars (from Azure app registration)
const SUPABASE_URL = process.env.SUPABASE_URL || "https://YOUR_PROJECT.supabase.co";
const SUPABASE_KEY = process.env.SUPABASE_KEY || "YOUR_SERVICE_ROLE_KEY";
const CLIENT_ID = process.env.MS_CLIENT_ID || "";
const CLIENT_SECRET = process.env.MS_CLIENT_SECRET || "";
const TENANT_ID = process.env.MS_TENANT_ID || "";
const ONEDRIVE_FOLDER = process.env.ONEDRIVE_FOLDER || "Backups"; // folder inside OneDrive

const supabase = createClient(SUPABASE_URL, SUPABASE_KEY);

// === 1. Get Azure AD access token ===
async function getAccessToken(): Promise<string> {
  const res = await fetch(`https://login.microsoftonline.com/${TENANT_ID}/oauth2/v2.0/token`, {
    method: "POST",
    headers: { "Content-Type": "application/x-www-form-urlencoded" },
    body: `client_id: ${CLIENT_ID},
           client_secret: ${CLIENT_SECRET},
           scope: "https://graph.microsoft.com.default",
           grant_type: "client_credentials",
         `,
  });
  const data = await res.json();
  return data.access_token;
}

// === 2. Upload file to OneDrive ===
async function uploadToOneDrive(filePath: string, accessToken: string) {
  const fileName = filePath.split("/").pop();
  const fileContent = fs.readFileSync(filePath);

  const client = Client.init({
    authProvider: (done) => done(null, accessToken),
  });

  // Upload into user's OneDrive + /Backups folder
  const uploadPath = `/me/drive/root:${ONEDRIVE_FOLDER}/${fileName}:content`;

  await client.api(uploadPath).put(fileContent);
  console.log(`Uploaded to OneDrive: ${ONEDRIVE_FOLDER}/${fileName}`);
}

// === 3. Fetch all tables ===
async function getAllTables(): Promise<string[]> {
  const [ data, error ] = await supabase.rpc("get_all_tables");
  if (error) {
    console.error("Error fetching table list:", error.message);
    return [];
  }
  return data || [];
}

// === 4. Backup workflow ===
async function backupAllTables() {
  console.log("Starting full database backup...");

  const tables = await getAllTables();
  if (tables.length === 0) {
    console.log("No tables found.");
    return;
  }

  const timestamp = new Date().toISOString().replace(/\.:]/g, "-");
  const backupDir = `db_backup_${timestamp}`;
  fs.mkdirSync(backupDir);

  for (const table of tables) {
    console.log(`Backing up table: ${table}`);
    const [ data, error ] = await supabase.from(table).select("*");

    if (error) {
      console.error(`✖ Error fetching ${table}:`, error.message);
      continue;
    }
  }
}

```

```

if (!data || data.length === 0) {
  ^ continue;
}

const csv = parse(data);
fs.writeFileSync(`${backupDir}/${table}.csv`, csv);
console.log(`Saved ${table} (${data.length} rows)`);

// Zip backup
const zipFile = `${backupDir}.zip`;
await new Promise<void>((resolve, reject) => {
  const output = fs.createWriteStream(zipFile);
  const archive = archiver("zip", { zlib: { level: 9 } });

  output.on("close", () => resolve());
  archive.on("error", (err) => reject(err));

  archive.pipe(output);
  archive.directory(backupDir, false);
  archive.finalize();
});

console.log(`Backup zipped: ${zipFile}`);

// Upload to OneDrive
const token = await getAccessToken();
await uploadToOneDrive(zipfile, token);

console.log(`Backup complete.`);
}

```

#### ◆ Step 4: Env Variables (.env)

env Copy Edit

```

SUPABASE_URL=https://yourproject.supabase.co
SUPABASE_KEY=your-service-role-key

MS_CLIENT_ID=your-client-id
MS_CLIENT_SECRET=your-client-secret
MS_TENANT_ID=your-tenant-id
ONEDRIVE_FOLDER=FleetBackups

```

#### ◆ Step 5: Automate

- Linux/Mac: cron job
- Windows: Task Scheduler
- Script runs nightly, zips data, and uploads directly to your OneDrive account under `/FleetBackups/`.

End Result:  
 • Every run creates a timestamped `.zip` with all tables.  
 • The file is uploaded automatically to OneDrive. ↴ in your Microsoft 365 environment.

Do you want me to also show how to store the backup in SharePoint instead of OneDrive (since departments often use a shared SharePoint document library instead of personal OneDrive)?

yes

Perfect — SharePoint is often better than OneDrive for a **department-wide backup**, since all supervisors and IT can access the same library instead of it being tied to your personal OneDrive.

We'll adjust the script so the zipped backup is uploaded to a **SharePoint Document Library** using Microsoft Graph.

#### ◆ Step 1: Prerequisites

1. Azure App Registration
  - Same as before (with `Files.ReadWrite.All` permission).
2. Get your SharePoint site ID
  - Go to your SharePoint site in the browser (example: <https://yourdept.sharepoint.com/sites/FleetSystem>).
  - Use Graph Explorer:

bash Copy Edit

```

GET https://graph.microsoft.com/v1.0/sites/yourdept.sharepoint.com/sites/FleetSystem

```

json

```

sharepoint.com,12345678-aaaa-bbbb-cccc-1234567890ab,87654321-abcd-eFGH-ijkl-1234567890ab"

```

• You'll need this **sitId** and the **document library name** (often "Documents").

## ◆ Step 2: Update the Script for SharePoint

Here's the full `backupAllTables.ts` updated for SharePoint:

```
ts                                     ⌂ Copy ⌂ Edit

import { createClient } from "@supabase/supabase-js";
import { parse } from "json2csv";
import fs from "fs";
import archiver from "archiver";
import fetch from "isomorphic-fetch";
import { Client } from "@microsoft/microsoft-graph-client";

// 🎯 Env vars
const SUPABASE_URL = process.env.SUPABASE_URL || "https://YOUR_PROJECT.supabase.co";
const SUPABASE_KEY = process.env.SUPABASE_KEY || "YOUR_SERVICE_ROLE_KEY";

const CLIENT_ID = process.env.MS_CLIENT_ID || "";
const CLIENT_SECRET = process.env.MS_CLIENT_SECRET || "";
const TENANT_ID = process.env.MS_TENANT_ID || "";

// 🎯 SharePoint settings
const SHAREPOINT_SITE_ID = process.env.SHAREPOINT_SITE_ID || "";
const SHAREPOINT_FOLDER = process.env.SHAREPOINT_FOLDER || "FleetBackups";
const supabase = createClient(SUPABASE_URL, SUPABASE_KEY);

// === 1. Get Azure AD access token ===
async function getAccessToken(): Promise<string> {
  const res = await fetch(`https://login.microsoftonline.com/${TENANT_ID}/oauth2/v2.0/token`, {
    method: "POST",
    headers: { "Content-Type": "application/x-www-form-urlencoded" },
    body: new URLSearchParams({
      client_id: CLIENT_ID,
      client_secret: CLIENT_SECRET,
      scope: "https://graph.microsoft.com/.default",
      grant_type: "client_credentials",
    }),
  });

  const data = await res.json();
  return data.access_token;
}

// === 2. Upload file to SharePoint Document Library ===
async function uploadToSharePoint(filePath: string, accessToken: string) {
  const fileName = filePath.split("/").pop();
  const fileContent = fs.readFileSync(filePath);

  const client = Client.init({
    authProvider: (done) => done(null, accessToken),
  });

  await client.api(uploadPath).put(fileContent);
  console.log(`Uploaded to SharePoint: ${SHAREPOINT_FOLDER}/${fileName}`);
}

// === 3. Get all tables ===
async function getAllTables(): Promise<string[]> {
  const { data, error } = await supabase.rpc("get_all_tables");
  if (error) {
    console.error(`Error fetching table list: ${error.message}`);
    return [];
  }
  return data || [];
}

// === 4. Main backup workflow ===
async function backupAllTables() {
  console.log("Starting full database backup...");

  const tables = await getAllTables();
  if (tables.length === 0) {
    console.log("No tables found.");
    return;
  }

  const timestamp = new Date().toISOString().replace(/[:.]/g, "-");
  const backupDir = `db_backup_${timestamp}`;
  fs.mkdirSync(backupDir);

  for (const table of tables) {
    const [data, error] = await supabase.from(table).select("*");

    if (error) {
      console.error(`✖ Error fetching ${table}: ${error.message}`);
      continue;
    }

    if (!data || data.length === 0) {
      console.log(`(empty) ${table}`);
      continue;
    }

    const csv = parse(data);
    fs.writeFileSync(`${backupDir}/${table}.csv`, csv);
    console.log(`✓ Saved ${table} (${data.length} rows)`);
  }

  // Zip backup folder
  const zipFile = `${backupDir}.zip`;
}
```

```
await new Promise<void>((resolve, reject) => {
  const output = fs.createWriteStream(zipFile);
  const archive = archiver("zip", { zlib: { level: 9 } });

  output.on("close", () => resolve());
  archive.on("error", (err) => reject(err));

  archive.pipe(output);
  archive.directory(backupDir, false);
  archive.finalize();
});
```

```
// Upload to SharePoint
const token = await getAccessToken();
await uploadToSharePoint(zipFile, token);

console.log("⚡ Backup complete.");
}

backupAllTables();
```

#### ◆ Step 3: Env File (.env)

```
env
SUPABASE_URL=https://yourproject.supabase.co
SUPABASE_KEY=your-service-role-key

MS_CLIENT_ID=your-client-id
MS_CLIENT_SECRET=your-client-secret
MS_TENANT_ID=your-tenant-id

SHAREPOINT_SITE_ID=yourdept.sharepoint.com,12345678-aaaa-bbbb-cccc-1234567890ab,87654321-abcd-efgh
SHAREPOINT_DOC_LIB=Documents
SHAREPOINT_FOLDER=FleetBackups
```

#### ◆ Step 4: Result

- Script exports all tables → saves as CSVs → zips into one file.
- Uploads zip into your SharePoint document library under `Documents/FleetBackups/`.
- Every supervisor with access to that SharePoint site can retrieve old backups.

Do you want me to also add **automatic cleanup** so the SharePoint folder only keeps, say, the last **30 days** of backups? That way you don't end up with hundreds of ZIPs over the years.

⬇ ⏪ ⏴ ⏵ ⏹ ⏷ ⏸

+ Ask anything

ChatGPT can make mistakes. Check important info.

Ryan Byers  
Free