rewind

The Complete Guide to GitHub Backups

Introduction

With 40 million users, you'd be hard-pressed to find a developer who hasn't used GitHub for a project. Despite this widespread adoption, there are surprisingly few robust GitHub backup solutions available. Most companies end up creating an internal tool or using an open-source script to handle the job.

At Rewind, we think software developers need a better option.

Our GitHub backup service is the only complete solution recommended by GitHub and trusted by large and small teams worldwide. We provide our customers with simple, reliable, nightly backups of all their GitHub repositories—including metadata—and allow them to restore the data in just a few clicks.

We wrote this guide to help software engineers considering repository backup solutions to pick the best one for their needs. In Part 1, you'll see how to make a case for repository backups to your team. Part 2 dives into the pros and cons of using an in-house tool versus a third-party provider, and Part 3 compares three software tools purpose-built for GitHub backups.

While we obviously think Rewind is an excellent option, it's not the only one. By the end of this guide, you should be able to decide whether it's the right one for you.

Part 1: The Case for Repository Backups

One of the most critical parts of being an engineering leader is communicating the engineering team's needs to non-technical stakeholders. Take backups, for example. Backups cost money to store and take time to create. Theoretically, if everyone does their job perfectly, they shouldn't be necessary, but experienced engineering leaders know to prepare for failure.

In this section, we'll offer several compelling ways to make the case for GitHub repository backups to your team. In doing so, you'll see examples where repository backups would have mitigated significant technical risks in the real world. Ultimately, repository backups can help you prevent data loss, but it's helpful to understand how this data loss might occur.

Recovery from Account Compromises

In July of 2019, Ubuntu Security reported that the credentials for a company-owned GitHub account were compromised. These compromised credentials were used to create repositories, issues, and more.

> **"We can confirm that on 2019-07-06 there was a Canonical owned account on GitHub whose credentials were compromised and used to create repositories and issues among other activities. Canonical has removed the compromised account from the Canonical organisation in GitHub and is still investigating the extent of the breach, but there is no indication at this point that any source code or PII was affected," says Ubuntu Security.**

In this case, it seems that critical infrastructure was decoupled from GitHub, and the breach wasn't allowed to spread. However, Canonical (Ubuntu's publisher) had to restore various repositories and issue trackers to their previous state. When rolling back from an account compromise like this, backups are infinitely helpful, as they give you a previously good state to compare with the current state.

Additionally, attackers often leave backdoors in compromised codebases. This can allow them to gain greater access once the initial discovery and remediation process is completed. If you only have your infected codebase, it can be challenging to uncover all the corrupted files or backdoors left open for future attacks.

Mitigating Ransomware Attacks

Ransomware is the act of taking control of a codebase or set of infrastructure and encrypting it so that only the attacker can unlock it. In exchange for returning access to the victim, the attacker demands a ransom be paid, usually in an untraceable cryptocurrency. If the ransom is not paid in a certain timeframe, the attackers threaten to delete the files, rendering them irrecoverable.

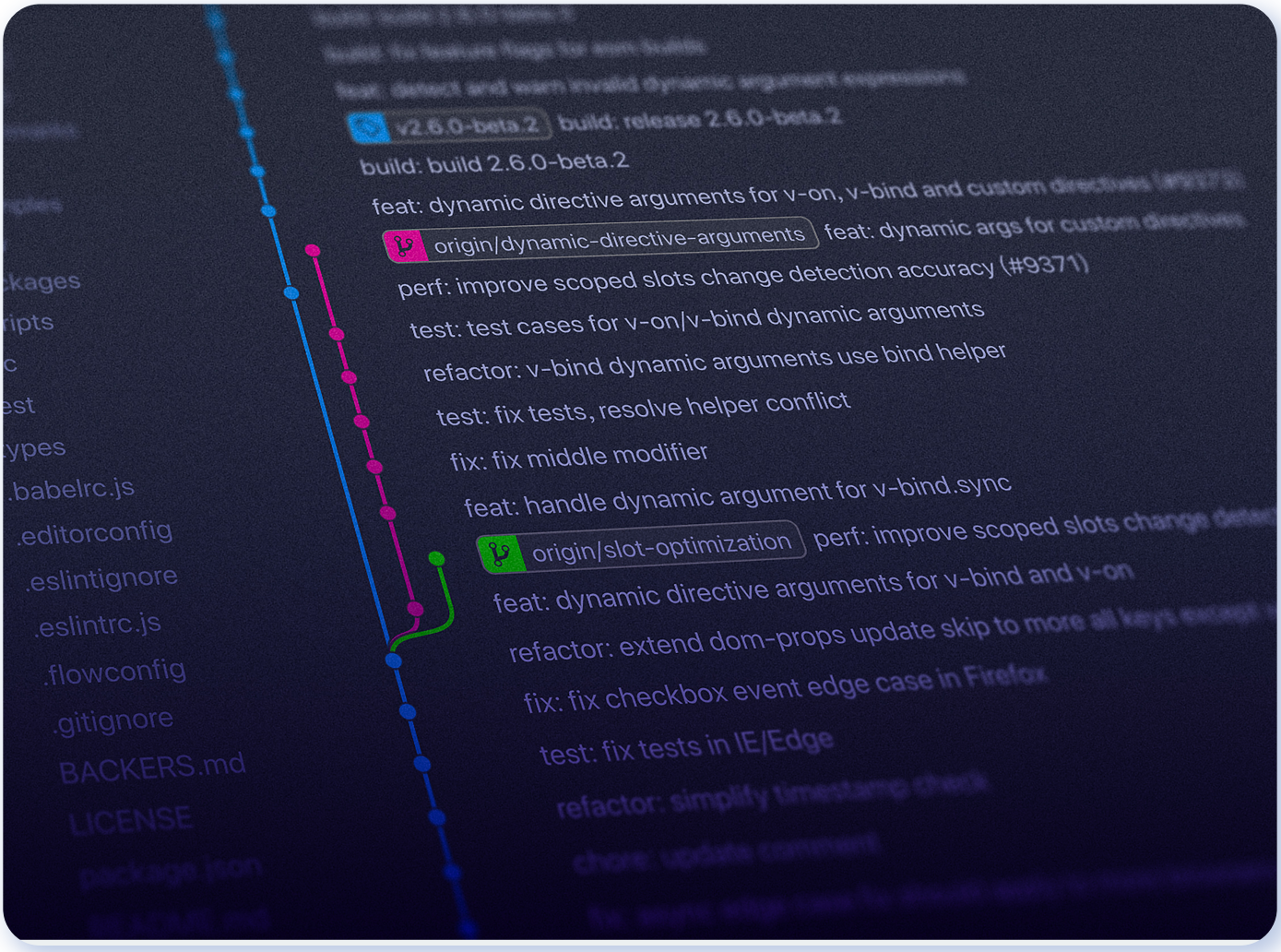
This is exactly what happened in May of 2019 when ZDNet reported that, "Hundreds of developers have had Git source code repositories wiped and replaced with a ransom demand." The hackers had modified the git histories to the point where the repositories were unusable, and they demanded payment within ten days to reverse the changes.

If a compromised repository is part of your organization's core codebase, an attack like this will cause a severe disruption in operations. Developers will be unable to commit code, creating a complete stoppage in development. Bug fixes and even support tickets (if managed through GitHub) could be affected. However, if you were hit with an attack like this and had a backup of your repository, you could restore from this backup and continue working on day-to-day tasks while the issue was resolved.

Accidental Repository Deletion

Repositories might be mistakenly deleted or transferred from your organization's GitHub account by admins at any time. It's also possible that a former employee or contractor could maliciously remove code or entire projects from your account.

Whether your company cancels a product line and deletes the codebase or an engineer accidentally removes the wrong repository from your organization, it's crucial to have a plan to recover from mistakes. While GitHub will let you recover a deleted repository within 90 days, there are limitations to this feature. If you want to ensure that you can quickly recover your GitHub repositories, having frequent automatic backups is a good idea.



Minimizing Service Downtime

Many companies have moved parts of their codebases to third-party providers like GitHub. The reasons for this are many, but if code storage is not your core business, you can save time and money by relying on an outside service.

However, depending on a third-party always carries some risk. No service can deliver 100% uptime, but when your entire business (or codebase) depends on GitHub's availability, you might want to mitigate that risk by having your own backups.

For example, in June 2020, GitHub had a major outage that lasted for hours before stability returned. If you relied on GitHub to store your code, this meant much of the development work for the day was on hold until they resolved the issue. These outages may impact developer productivity and project timelines if they occur during a crucial launch window.

Like the ransomware scenario described above, the best way to mitigate service downtime is to have a plan in place and a backup of any repositories and associated metadata. While you can create these backups on your own, using a service like Rewind will make it significantly easier. With a proper backup and restore plan in place, what could potentially be a work-stopping outage can instead be reduced to just an alert and switchover.

Reducing Platform Dependence

Another risk inherent in using a third-party provider for repository hosting is that your business depends on that provider's continued supports. Providers like GitHub are businesses and may face financial or regulatory pressures that limit your ability to use their platform.

For example, in the summer of 2019, GitHub was forced to comply with US export law and had to prevent users in Iran, Syria, Crimea, and other sanctioned nations from accessing their service. Anyone in the affected countries was cut off and forced to find a different repository host.

If they had a recent backup, their repositories could be restored and pushed to another provider, but without one, these organizations might lose access to valuable company assets overnight without one. Events like these are rare, but having a backup is a small price to pay for maintaining access to your code.

Maintaining Compliance

Finally, if you're going through SOC2 compliance, it's important to keep repository backups in mind. SOC2 auditors will check that backups of certain application and database components are performed daily because code backups are a vital piece of the Availability Trust Service Criteria. Code backups will support recovery in the event of a service failure, and they're one more safeguard to help quickly get your software back in the hands of your customers.

Some of these reasons may be more applicable to your business than others, so you will need to decide which will resonate with your organization's non-technical stakeholders. However, presenting concrete scenarios can be a great way to start the conversation. **Framing GitHub backups as an investment that reduces business risk instead of an unnecessary expenditure can be powerful.**

Part 2: In-House vs. Third-Party Repository Backups

On January 31, 2017, GitLab had an outage that lasted for many hours due to production data being accidentally deleted. GitLab's problems were compounded by the fact that when they went to look for their backups, they simply weren't there. The backup process they thought had been running threw an error every time it ran, so no backups were actually created. GitLab was managing its own backups and hadn't put in the time and effort to ensure they were working correctly.

Backups are critical to every technology business, but they can be challenging to manage effectively. There's a reason that companies like Rewind exist to provide backup services. When deciding between building your own repository backup system or relying on a third-party provider, there are cost considerations, integration challenges, and reliability concerns to consider. This decision is specific to every business, but there are a few rules of thumb that you can take into account.

Managing Your Own Backups

Managing your repository's backups in-house means you are responsible for all the infrastructure and ongoing engineering costs required. This might not seem that hard, but labor costs for the employees maintaining your backup solution can be hard to quantify. Your business also has to accept the responsibility of running a critical service that isn't its core business.

Here are a few of the things you should consider before committing to managing your own repository backups:

PRO: COMPLETE CONTROL

When you manage your own GitHub repository backups, you have complete control over how the system is managed, how it integrates with other parts of the business, what can be backed up, how often these backups happen, and everything else.

For example, if you need backups created every hour, you might have to build your own solution. While third-party backup systems like Rewind perform daily backups, you can't currently take more frequent snapshots.

CON: DISTRACTION AND LONG-TERM COST

The main downside to managing your own backups is that you have to dedicate internal employees to work on the backup solution and manage it across all your engineering teams. Even if this isn't an employee's entire job, it takes time and focus away from solving core business problems. It's also important to consider maintenance costs in this equation:

> **"When evaluating whether to buy or build, it's critical to thoroughly understand total costs during the software lifecycle – typically seven or eight years. This step is important because 70 percent of software costs occur after implementation," says Mark Lutchen, former global CIO of PricewaterhouseCoopers.**

In addition to employee time, managing the infrastructure and resources that support your backup application is a financial obligation. It's probably insignificant for small teams with just a few repositories, but it will add up in large organizations.

Using a Third-Party for Repository Backups

Paying a third-party like Rewind to handle your GitHub repository backups removes this responsibility from your engineering team. While you get less control over your backup process, you will spend much less developing and maintaining the solution.

Here are some of the things you should consider when evaluating third-party repository backup solutions:

PRO: FOCUS

Using a third-party to manage repository backups requires much less involvement from your internal employees. For example, after you set up recurring backups in Rewind, you won't have to do any ongoing maintenance to keep them running. This allows your team to stay focused on moving the company forward while knowing that backups are being taken care of.

CON: SERVICE CHANGES

Because third-party backup providers are running a business, they have their own costs to manage. You don't have any control over their pricing or terms of service changes in the future, so there is some risk in picking one. Look for a reliable, trusted third-party solution or start by testing their service out before committing all your projects to it.

In addition to employee time, managing the infrastructure and resources that support your backup application is a financial obligation. It's probably insignificant for small teams with just a few repositories, but it will add up in large organizations.

PRO: RELIABILITY

A high-quality backup provider will manage all the previously-discussed complexities of the backup process, be able to guide you towards a solution that's right for your needs, and help with the implementation of that solution for a monthly fee. They will also have a reliability plan in place and recommendations from trusted sources. For example, **Rewind is the only recommended GitHub backup solution available today. It's hard to imagine a better endorsement**.

Repository backups are an essential part of maintaining your application, but they shouldn't be keeping you up at night. If you are looking for a way to back up your GitHub repositories, read on. In the last section, we'll compare three popular GitHub backup tools so you can decide which one is right for your company.

Part 3: Comparing GitHub Backup Options

This section will walk you through three tools you can use to back up your GitHub repositories. We'll briefly show you how to use each one and compare their pros and cons. By the end, you'll be able to choose the best GitHub repository backup option for your team and codebase.

Method 1: GitHub Backup Python Script

As you might imagine, plenty of developers who want to backup their GitHub repositories have written open-source scripts to accomplish the task. The most widely used and feature-complete is Jose Gonzalez's Python GitHub Backup script.

After installing the Python library, you can run the script from your terminal. Using the command line arguments, you can specify which repositories you want to back up in your organization and which resources within those repositories.

For example, to backup all the public repositories and metadata for the Docker GitHub organization, you could run the following: (fair warning, this might take a while!)

```
github-backup docker -t <YOUR_GITHUB_ACCESS_TOKEN> --output-directory . --all --organization
```

Using an open-source script like this one has some advantages. It's free, and the code is stable and widely used. You can investigate the entire script on GitHub and fork it if you need something that the library doesn't support. This Python GitHub Backup script also allows you to choose between backing up part or all of your organization's GitHub data. While it's hard to say how well this script works at scale, it contains a mechanism to throttle requests to the GitHub API, so you shouldn't face rate limiting issues on moderately-sized workloads.

On the downside, this script does not include a method for restoring your backups. Having backups without a viable recovery option provides limited value, so you'll have to write your own script to recover your repositories from these backups. This script also requires you to run it manually - there's no timing or automation built-in - and you'll have to bring and pay for your own storage. Finally, there's no option to encrypt your backups, so using this script could open up a new vector for attackers to steal and exploit your code.

Gonzalez has stated that the library is complete for his purposes, so he doesn't plan on adding any significant new features. If you want to address any of these issues, you'll have to fork and maintain your own version of the script.

While the Python GitHub Backup script offers you a bit of peace of mind at little to no cost, it's not the most robust repository backup solution available. Still, it might be the right option for single developers or small projects that just need occasional backups and aren't worried about a robust recovery plan.

Method 2: S3 Backup GitHub Action

Since GitHub Actions went live in 2019, developers have been building lots of exciting tools with them. One such example is the Amazon S3 Backup GitHub Action.

This open-source script allows you to backup your git repository into Amazon's popular object storage platform. The Action is easy to implement. Just create a new action file and add the following to back up your main branch every time it's updated:

```
name: Mirror repo to S3
    on:
      push:
        branches:
          - main


jobs:
    s3Backup:
      runs-on: ubuntu-latest
      steps:
      - name: S3 Backup
      uses: peter-evans/s3-backup@v1
      env:
      ACCESS_KEY_ID: ${{ secrets.ACCESS_KEY_ID }}
      MIRROR_TARGET: ${{ secrets.MIRROR_TARGET }}
      SECRET_ACCESS_KEY: ${{ secrets.SECRET_ACCESS_KEY }}
with:
    args: --overwrite --remove
```

Add your AWS access key, access secret, and bucket path to your repository's secrets, and make an update. This Action will overwrite the old backup and replace it with the latest one every time you push to the main branch.

Using a GitHub Action to back up your code makes a lot of sense. You'll never have to worry if your backup is missing changes since every push is captured, and you won't have to install any software to run your backups. The S3 Backup Action is free and (as you can see above) relatively easy to use.

That said, this Action is also limited. First, it only captures the git repository data, so GitHub's metadata, pull requests, issues, wikis, and projects would all be lost if you relied solely on this backup. It also doesn't include a straightforward way to keep old backups for a reasonable period. In the example above, your old backup will be removed every time a new backup is made, so attackers could poison your backups if they get access to push a commit. While you can remove the --overwrite and--remove flags to keep all your old backups, you'll quickly fill up your S3 bucket with old files if you do that.

The S3 Backup Action must be added to each repository individually, and it doesn't include an option to encrypt backups. This opens you up to the same risk as the Python GitHub Backup script does above.

With so many caveats, you might think S3 Backup isn't a viable option, but it could be perfect for individual developers or side projects that are budget-conscious. Because the project is new, the maintainers might be interested in taking suggestions for new features or pull requests. It never hurts to ask.

Method 3: Rewind

If your organization is looking for a more robust backup solution than the open-source options above, you may want to consider Rewind. Our backups-as-a-service platform automatically backs up your GitHub repositories, metadata, projects, pull requests, issues, and more to ensure you never lose access to your repositories due to an attack or outage.

Installing and configuring Rewind is easy thanks to their GitHub Marketplace integration, and they provide a free trial, so you can make sure it's a good fit before you commit to a plan.

Once installed, Rewind will automatically back up some or all of your repositories every day. Backups are encrypted and stored securely on Rewind's servers, or you can opt to save them to your own Amazon S3 bucket.

When you need to use your repository backup, you can either restore it directly into GitHub or clone the repository from Rewind onto your local machine. This ensures that even in the unlikely event of a GitHub outage, you'll be able to access your repository's data.

The upside to using a provider like Rewind for your GitHub repository backups is that it's extremely robust. They work with many large customers (some backup over 13,000 git repositories with Rewind), and they encrypt your data throughout transit and at rest. Rewind also provides an audit log, so you can keep track of your backups for compliance and security purposes.

Finally, Rewind is the only solution that lets you restore your repository and associated metadata to GitHub or clone your repository from their servers. Recovery is an essential part of backing up your data, and if you use Rewind, you'll be able to recover your repository at any point in the past 30 days.

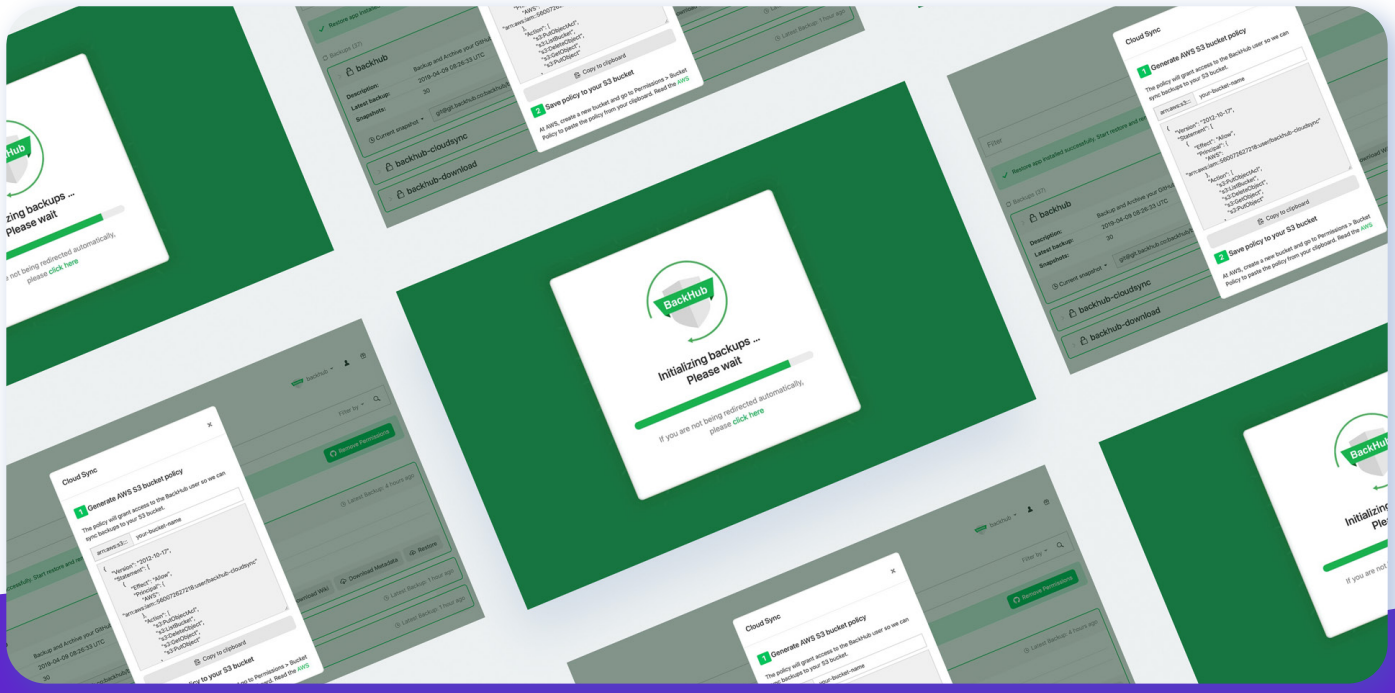
The downside to Rewind is that it's a paid, closed-source solution. While the pricing is predictable (based on the number of repositories you backup), you won't have quite the same level of control over exactly how it works as you would with an in-house or open-source solution. That said, you won't have any software or servers to maintain when you use Rewind, so it's likely worth the modest investment.

Final Thoughts

Whether you're subject to regulatory and compliance standards or you simply want to maintain a robust software development process, repository backups are an important investment. They can allow you to recover more quickly from attacks, prevent accidental data loss, and give you peace of mind that your code will be safe even if GitHub experiences downtime.

Once you've determined that a GitHub backup solution is worth your time, you'll need to decide if you want to build and maintain a custom tool or work with a trusted third-party. The flexibility of an in-house backup system is tempting, but you have to factor in the engineering time into your costs. This can add up quickly.

Finally, while the open-source tools available today are good enough for side projects or small teams, Rewind is the most robust GitHub backup solution available. With a fully integrated experience in the GitHub Marketplace, you can install and try Rewind for free with just a few clicks. Your repositories will be automatically backed up each night, and you can restore them at any time in seconds.



Start Your 14 Day Free Trial

Visit Rewind.com or contact sales@rewind.io
to schedule a personalized demo.

rewind