GitHub Releases 詳細ガイド：概要、利用方法、制限事項

本レポートでは、GitHubの「Releases」機能について、その概要、目的、主要機能、通常のソースコード管理との関連性、利用シナリオ、前提条件、制限事項、そして具体的な作成手順を詳細に解説します。

1. GitHub Releasesとは？

1.1 概要と基本的な定義

GitHub Releasesは、開発したソフトウェアの特定のバージョン（イテレーション）をパッケージ化し、エンドユーザーがダウンロードして利用できるようにするためのGitHub上の機能です [1]。これは単なるソースコードのバージョン管理（Gitリポジトリ）を超え、完成したソフトウェアを配布するための体系的なワークフローを提供します [3]。

各リリースには、そのバージョンにおける変更点を記録した「リリースノート」、コンパイル済みの実行ファイルやドキュメントなどの配布物である「アセット（ファイル）」、そしてリリース対象バージョンに対応する「ソースコードのアーカイブ（zip形式およびtar.gz形式）」が含まれます [2]。

GitHub Releasesは、GitHubプラットフォーム独自の機能であり、Git自体の基本的な機能（例えば git tag コマンド）とは区別されますが、Gitのタグ機能と密接に関連しています [1]。この機能は、ソフトウェアの配布プロセスをGitHub上で形式化し、単なるソースコードへのアクセス提供から一歩進んだものと位置づけられます。

1.2 通常のGitHub（ソース管理）との違いと連携

通常のGitHubリポジトリの主な機能は、ソースコードの変更履歴を管理し、開発者間の共同作業を支援することです [6]。一方、GitHub Releasesは、リポジトリ内に存在する特定の「状態」、通常はGitのタグによってマークされたコミットを基盤として、配布可能なソフトウェアパッケージを作成する点で異なります [1]。

リリースはGitタグに紐づいており、そのタグはリポジトリの履歴における特定の時点を示すマーカーとして機能します [1]。タグ自体はGitの標準機能ですが、GitHub Releasesはそのタグを利用して、リリースノートやダウンロード可能なアセットといった付加価値を提供します [4]。この区別は重要であり、Releasesはコード開発に貢献する開発者だけでなく、最終製品を利用するエンドユーザーを対象としている点が異なります [4]。

リリースの管理は、リポジトリのメインページにある「Releases」セクション（日本語UIでは「リリース」）を通じて行われます [9]。これにより、リポジトリのコード閲覧とは別に、公式なバージョン配布物が整理された形で提供されます。これは、Gitの生（raw）の履歴の上に、キュレーションされた配布レイヤーを提供するものと考えることができます。特定のバージョンがエンドユー

ザーにとって見つけやすく、Gitの専門知識がなくても利用しやすい形式で提供されるのです。

この構造は、開発ワークフローにおけるベストプラクティスを示唆しています。つまり、内部的なバージョン管理にはGitタグを厳密に使用し、その中から重要なマイルストーンとなるタグを選択して、外部へのコミュニケーションと配布のために公式な「リリース」として昇格させる、という使い分けです。すべてのタグがリリースになる必要はありません[7]。

2. GitHub Releasesの目的と主な機能

2.1 ソフトウェアバージョンの配布という目的

GitHub Releasesの最も主要な目的は、開発されたソフトウェアの特定のバージョン（安定版、ベータ版、アルファ版など）を、エンドユーザーが容易にダウンロードし、利用できるようにパッケージ化して提供することです[1]。これにより、開発者はソフトウェアの進化の各段階を明確に示し、ユーザーは自身のニーズに合ったバージョンを選択して入手できます[3]。これは、開発リポジトリから利用可能なソフトウェアバージョンをユーザーの手に届けるプロセスを効率化します。

2.2 リリースノート機能

各リリースには、そのバージョンでの変更点、新機能、バグ修正、既知の問題点などを記述した「リリースノート」を添付することが可能です[2]。リリースノートはMarkdown形式で記述でき、リッチな表現が可能です。また、リリースに貢献した開発者を @mention 機能でメンションすることもでき、メンションされたユーザーはリリースノート内の「Contributors」セクションにアバター付きでリスト表示されます[9]。

リリースノートの自動生成: GitHubは、リリースノートを手動で記述する手間を省くために、マージされたPull Requestに基づいてリリースノートを自動生成する機能を提供しています[12]。これは、特に変更点が多い場合に便利で、一貫性のあるフォーマットを維持するのに役立ちます。自動生成されるノートには、通常、マージされたPull Requestのリスト、貢献者のリスト、完全な変更ログへのリンクが含まれます[12]。

リリースノートのカスタマイズ: 自動生成されるリリースノートの内容は、リポジトリのルートディレクトリ直下の .github ディレクトリ内に release.yml という設定ファイルを作成することで、細かくカスタマイズできます[12]。このファイルでは、特定のラベルが付いたPull Requestをリリースノートから除外したり、特定のラベル（例: breaking-change, enhancement, bug）に基づいて変更点をカテゴリ分けしたりすることが可能です[12]。

リリースノートは、配布されるソフトウェアバージョンに直接関連付けられた重要なコミュニケーションツールです。自動生成機能とそのカスタマイズ性は、質の高いリリースドキュメントを提供する上での障壁を大幅に低減します。さらに、Pull Requestのラベルに基づいて自動生成ノートをカスタマイズできることは、開発プロセス自体における規律あるラベリング習慣を奨励

します。良い開発習慣が直接的に優れたリリースドキュメントにつながるという好循環を生み出す可能性があるのです。

2.3 ダウンロード可能なアセット

開発者は、ソースコードとは別に、コンパイル済みの実行ファイル（.exe, .app, .jar など）、インストーラーパッケージ（.msi, .dmg, .deb, .rpm など）、ライブラリファイル、ドキュメント（PDFなど）、その他の関連ファイル（設定ファイル、サンプルデータなど）を「アセット」としてリリースに添付できます [2]。

これにより、ユーザーはソースコードからソフトウェアを自身でビルドする手間なく、すぐに利用可能な形式でソフトウェアを入手できます [3]。アセットのアップロードは、リリースの作成・編集画面からドラッグ &ドロップ、またはファイル選択ダイアログを通じて簡単に行えます [9]。複数のファイルをアセットとして含めることも可能です [7]。アセット機能は、特にエンドユーザーにとって最も重要な配布形態であるコンパイル済みソフトウェアを提供する必要性に直接応えるものであり、単なるソースコードアクセス提供との大きな違いを生んでいます。

2.4 自動生成されるソースコードアーカイブ

GitHubは、各リリースが基づいているGitタグの時点でのリポジトリの完全なソースコードを含む、zip形式とtar.gz形式のアーカイブファイルを自動的に生成し、ダウンロード可能なリンクとして提供します [1]。これにより、ユーザーはそのバージョンのソースコード一式も容易に入手できます [5]。

これらの自動生成アーカイブにGit LFS (Large File Storage) で管理されているオブジェクトを含めるかどうかは、リポジトリの設定で選択可能です [15]。この自動的なソースアーカイブ提供は、たとえ主要な配布物がバイナリであっても、リリースとその基盤となるGit履歴（タグ）との間の透明性と再現性を保証します。

3. GitHub ReleasesとGit（タグ、コミット、ブランチ）の関係

3.1 Gitタグに基づいたリリース管理

GitHub Releasesは、その根幹においてGitの「タグ」機能に基づいています [1]。リリースを作成する際には、必ず既存のGitタグを選択するか、あるいはリリース作成プロセスの中で新しいタグを作成する必要があります [9]。

Gitタグは、リポジトリの履歴内における特定のコミットへの永続的な参照（ブックマーク）として機能します [1]。したがって、リリースはそのタグが指し示す特定のコミット時点でのコードの状態を表すことになります [1]。

タグには「軽量タグ」と「注釈付きタグ」の2種類がありますが、リリース用途では、作成者情報、日付、そしてタグ付けの理由を説明するメッセージを含むことができる「注釈付きタグ」の使用

が一般的に推奨されます [8]。タグ名には、v1.0.0 のようなセマンティックバージョニングに従った命名規則を採用することが広く行われています [3]。

タグは、開発者のローカル環境で git tag コマンドを用いて作成し、git push --tags や git push origin <tagname> コマンドでリモートリポジトリ（GitHub）にプッシュする方法と、GitHubのWeb UI上でリリースを作成する際に直接新しいタグを作成する方法があります [18]。どちらの方法を選択するかは、チームのワークフローによりますが、GitHub上で直接作成する方がプロセスを簡略化できる場合もあります [19]。

すべてのリリースがGitタグに強制的にリンクされることは、各リリースがプロジェクトのソースコード履歴における検証可能で特定の時点に対応することを保証し、トレーサビリティを提供します [1]。タグをいつ、どのように作成するか（ローカルのGit CLI経由か、GitHubのリリースUI上か）はワークフローの柔軟性を提供しますが、チーム内での調整が必要です。リリースUI経由でのタグ作成はプロセスを単純化するかもしれませんが、タグ作成をコミット/プッシュのワークフローから切り離すため、ターゲットとなるコミットやブランチを慎重に選択する必要があります [9]。ローカルでのタグ付けは、タグ作成を開発コミットとより密接に統合しますが、明示的なプッシュ操作が必要です [22]。

3.2 リポジトリ履歴の特定時点との関連付け

リリースは、選択されたタグが指し示す特定のコミットに厳密に関連付けられます [1]。これにより、リリースされたソフトウェアとその時点でのソースコード、依存関係、ビルド環境などを正確に対応付けることが可能となり、再現性の確保に貢献します。

リリースは通常、開発が進むメインのブランチ（例: main, master）や、リリース専用のブランチ（例: release/v1.0）における安定したコミットに対して作成されます [22]。リリース作成時には、特に新しいタグを作成する場合、そのタグがどのブランチやコミットを指すべきかを指定します [9]。

GitHubには、異なるリリース（またはタグ）間でコードの変更点を比較する機能 (Compare) も用意されており、バージョン間の差分を容易に確認できます [13]。リリースは、複雑で進化し続ける可能性のあるGit履歴の中で、安定した名前付きの参照点（タグ経由で）を提供します。これは、バージョン間の変更追跡と再現性にとって極めて重要です。

3.3 GitタグとGitHubリリースの比較

GitタグとGitHubリリースの関係と違いを明確にするために、以下の比較表にまとめます。

特徴	**Git タグ (Git Tag)**	**GitHub リリース (GitHub Release)**
基本概念	履歴内の特定コミットへのポイン	Gitタグに基づいた配布用パッ

	タ	ケージ
主な用途	特定時点（例: バージョン）をマークする	ソフトウェアバージョンをエンドユーザーに配布する
構成要素	タグ名、(任意)メッセージ、コミット参照	タグ、リリースタイトル、リリースノート、アセット、ソースアーカイブ
起源	標準的なGitの機能	GitHub独自の機能
生成物	直接的な生成物なし（参照のみ）	ダウンロード可能なバイナリ、ソースzip/tar.gz
管理方法	git tag コマンド、GitHub UI (Tagsタブ)	GitHub UI (Releasesタブ)、API、CLI

この表は、GitHub ReleasesがGitタグを基盤としつつも、配布に特化した独自の機能と目的を持っていることを示しています。

4. GitHub Releasesの一般的な利用シナリオ

4.1 コンパイル済みバイナリやインストーラーの配布

これはGitHub Releasesの最も代表的な利用方法です。開発者は、ビルド済みの実行ファイル（.exe, .app, .jarなど）や、OS固有のインストーラーパッケージ（.msi, .dmg, .deb, .rpmなど）をリリースのアセットとして添付し、エンドユーザーが直接ダウンロードして簡単にインストール・実行できるようにします [2]。ソースコードからビルドする必要がないため、開発者以外のユーザーにとっても利便性が高い配布方法となります [3]。

4.2 ソースコードアーカイブの提供

GitHubが自動生成する標準のソースコードアーカイブ（zip/tar.gz）に加えて、開発者は特定のビルドに必要な依存関係を同梱したソースコードバンドルや、特定のプラットフォーム向けに調整されたソースパッケージなどを、カスタムアセットとして配布することも可能です [5]。

4.3 ドキュメントやその他の成果物の配布

ソフトウェアの特定のバージョンに対応するユーザーマニュアル、APIリファレンス、設定ファイル、サンプルデータ、デザインアセットなど、コード以外の関連資料もリリースアセットとして配布できます [3]。これにより、ソフトウェア本体だけでなく、その利用に必要な周辺リソースもバージョン管理された形で一括提供できます。これは、特定のソフトウェアバージョンを取り巻くエコ

システム全体を配布するポイントとして機能し、ユーザーエクスペリエンスを向上させます。

4.4 自動化（API、CLI、GitHub Actions経由）

GitHub Releasesの操作（作成、編集、削除、アセットのアップロードなど）は、手動のWeb UI操作だけでなく、GitHub REST API [24] や GitHub CLI (コマンドラインインターフェース、例: gh release create コマンド) [11] を利用してプログラム的に自動化することが可能です。

これにより、継続的インテグレーション/継続的デリバリー (CI/CD) パイプライン、例えばGitHub Actions [14] やTravis CI [7] などにリリースプロセスを組み込むことができます。具体的には、コードのビルド、テスト、Gitタグの自動付与、リリースの作成、ビルドされたアセットのアップロードまでの一連の流れを自動化するワークフローを構築できます。actions/create-release や actions/upload-release-asset といった、GitHub公式やコミュニティによって提供されている便利なGitHub Actionsも存在します [28]。リリースノートの自動生成機能も、これらの自動化ワークフローに統合できます [14]。

API、CLI、CI/CD連携が充実していることは、GitHub Releasesが自動化されたDevOpsワークフローへの統合を前提として設計されていることを示唆しています。これにより、一貫性があり効率的なソフトウェアデリバリーパイプラインの実現が支援されます。成熟したCI/CDプラクティスを持つプロジェクトでは、手動でのリリース作成よりも、自動化されたプロセスの一部としてリリースが行われることが一般的かもしれません。

5. 利用上の前提条件と制限事項

5.1 アクセス権限：リリースの作成・管理に必要な権限

GitHub Releasesを作成、編集、または削除するには、対象となるリポジトリに対する書き込み権限 **(write access)** が必要です [2]。リポジトリの所有者や、書き込み権限を持つコラボレーターなどがこれに該当します [11]。

一方、既存のリリースを閲覧したり、異なるリリース間での比較を行ったりするだけであれば、リポジトリへの読み取り権限 **(read access)** があれば十分です [1]。ただし、リポジトリがプライベートに設定されている場合は、そのリポジトリへのアクセスを許可されたユーザーのみがリリースを閲覧できます [29]。この権限モデルは、標準的なリポジトリのアクセス制御と一貫しており、承認された担当者のみが公式なソフトウェアバージョンを公開できることを保証します。

5.2 ファイルサイズとリポジトリサイズ制限

- リリースアセットのファイルサイズ上限**:** 各リリースに添付できる個々のアセットファイルのサイズは、それぞれ **2 GiB (**ギビバイト**)** 未満である必要があります [2]。
- リリースの合計サイズ**/**帯域幅**:** リリースに含まれるアセットの合計サイズや、それらのダウンロードに使用されるネットワーク帯域幅には、GitHub Releases固有の制限は設けられていません [2]。ただし、プラットフォーム全体の公平な利用を維持するため、GitHubの

一般的な利用規約に基づく過剰な帯域幅使用に関するポリシーが適用される可能性はあります [31]。

- リポジトリ全体の推奨サイズ: GitHubは、リポジトリ自体のサイズを小さく保つことを推奨しています。理想的には1 GB未満、最大でも5 GB未満が強く推奨されます [16]。これは、リポジトリのクローン速度や日々の操作性、メンテナンス性に影響するためです。
- **Git**ファイルサイズ制限との比較: 通常のGitリポジトリに直接コミット・プッシュできるファイルサイズには、より厳しい制限があります。50 MiBを超えると警告が表示され、100 MiBを超えるとプッシュがブロックされます [16]。リリースアセットの上限 (2 GiB) はこれと比較して大幅に緩やかです。
- **Git LFS**との関連: 100 MiBを超えるような大きなファイルをGitのバージョン管理下で追跡したい場合は、Git LFS (Large File Storage) を使用する必要があります [15]。Git LFSには、アカウントごとに無料枠（例: 1 GiBのストレージ、月間1 GiBの帯域幅）があり、それを超える使用量に対しては追加料金が発生します [33]。重要な点として、GitHub Releasesのアセットのサイズや帯域幅は、このGit LFSのクォータとは別個に扱われ、**LFS**のクォータを消費しません。

これらの制限から、GitHubが意図する使い分けが明確になります。つまり、コアとなるGitリポジトリの履歴は軽量に保ち（厳しいファイルサイズ制限、またはクォータ制のLFSを使用）、バージョン管理された最終的な配布物（特に大きなバイナリファイル）は、寛容な制限を持つReleases機能を通じて提供する、という設計思想です [16]。リリース自体の合計サイズや帯域幅に制限がないことは、特にオープンソースプロジェクトや予算が限られたプロジェクトにとって、非常に魅力的でコスト効率の良い配布メカニズムとなります。

5.3 アカウントタイプとリポジトリタイプによる利用可否

- 利用可能性: GitHub Releases機能自体は、基本的にすべてのリポジトリタイプ（パブリック、プライベート、インターナル）およびすべてのアカウントタイプ（Free, Pro, Team, Enterprise）で利用可能です [2]。公式ドキュメント等で特定のプランやリポジトリタイプでの利用が制限されている旨の記述は見当たりません。
- アクセス制限: リリースの可視性は、それが属するリポジトリの可視性に厳密に従います。プライベートリポジトリで作成されたリリースは、そのリポジトリへの読み取りアクセス権を持つユーザー（招待されたコラボレーターなど）にしか表示されず、ダウンロードもできません [29]。パブリックリポジトリのリリースは、インターネット上の誰でもアクセス可能です。プライベートリポジトリのリリースだけをパブリックに公開する、といった機能は提供されていません [36]。もしそのような要求がある場合は、ソースコードはプライベートリポジトリで管理しつつ、配布物（リリース）は別のパブリックリポジトリで行う、といった回避策が考えられます [36]。
- 無料プラン **vs** 有料プラン: Releasesのコア機能（リリースの作成、アセットの添付、ダウンロード提供）に関して、無料プランと有料プランの間で機能的な差は基本的に設けられていないようです [2]。ただし、関連する機能、例えばGitHub Actionsの無料利用枠（自動

化に使用）やGitHub Packagesのストレージ容量などについては、プランによって提供される量に違いがあります [38]。

コア機能がプランによらず普遍的に提供されていることは、GitHub Releasesがプロジェクトの規模や予算に関わらず、非常にアクセスしやすい機能であることを意味します。

5.4 その他の注意点（利用規約など）

- プレリリース **(Pre-release):** 開発中のアルファ版やベータ版など、まだ安定版とは言えないリリースについては、「プレリリース」としてマークすることが強く推奨されます [3]。これにより、リリースページで区別して表示され、ユーザーはそのバージョンが不安定である可能性を認識できます。これはユーザーの期待値を管理する上で重要なメカニズムです。
- 利用規約**:** GitHub Releasesの利用には、GitHubの標準的な利用規約 [40] および Acceptable Use Policies（利用規定）[31] が適用されます。これらには、違法なコンテンツの配布、マルウェアの配布、著作権侵害、過剰な帯域幅の不正利用などの禁止事項が含まれています。
- プレリリース版ソフトウェアのライセンス**:** GitHub自体が提供するプレリリース版の機能（例えば、ベータ段階のGitHub機能）を利用する場合には、別途特別なプレリリースライセンス条件が適用されることがあります [41]。
- セキュリティアドバイザリ**:** もしリリースによってセキュリティ上の脆弱性が修正された場合は、関連するセキュリティアドバイザリ (Security Advisory) をリポジトリに公開することが推奨されます [1]。これにより、修正内容が明確になり、GitHubは影響を受ける可能性のある他のリポジトリに対してDependabotアラートを送信できます。

5.5 GitHub Releases 利用制限・条件 まとめ

GitHub Releasesを利用する上での主要な制限と条件を以下の表にまとめます。

カテゴリ	制限・条件	備考 / 関連情報
管理権限	リポジトリへの書き込み権限が必要	閲覧は読み取り権限で可 [2]
アセットファイルサイズ	個別ファイル 2 GiB 未満	[2]
合計リリースサイズ	制限なし	[2]
帯域幅	制限なし	GitHubの一般ポリシーは適用さ

		れる可能性あり [2]
リポジトリサイズ	推奨 1 GB 未満 / 5 GB 未満	Releases自体ではなくリポジトリ全体の推奨 [16]
Git ファイル制限	警告 50 MiB / ブロック 100 MiB	ReleasesではなくGit本体の制限 [16]
Git LFS クォータ	無料枠あり、超過は有料	Releasesのアセットとは別勘定 [33]
アカウント/プラン	全プランで利用可能	関連機能（Actions等）の制限はプラン依存 [38]
リポジトリ可視性	リリースの可視性はリポジトリに依存	Private Repo → Private Release [29]
利用規約	GitHub標準規約・ポリシー適用	[31]

6. GitHub Releasesの作成手順（ユーザーマニュアル形式）

以下に、GitHubのWebインターフェースを使用して新しいリリースを作成、編集、削除するための具体的な手順を、ユーザーマニュアル形式で示します。主に日本語UIの公式ドキュメント [9] に基づき、他の情報源 [9] から補足しています。

6.1 リリースページへの移動

1. 対象となるGitHubリポジトリのメインページにアクセスします。
2. ページ右側、ファイルリストの上付近にある **[リリース]** (Releases) のリンクをクリックします [9]。

6.2 新規リリースの作成開始

1. リリースページが表示されたら、ページ上部にある **[新しいリリースの下書き]** (Draft a new release) ボタンをクリックします [9]。

6.3 タグの選択または新規作成

1. **[タグの選択]** (Choose a tag) と書かれたドロップダウンメニューをクリックします [9]。
2. 既存のタグを使用する場合**:** 表示されるリストから、このリリースに対応させたい既存のGitタグを選択します [9]。
3. 新しいタグを作成する場合**:**
   - ドロップダウンメニューの入力フィールドに、新しいタグ名（例: v1.0.1, beta-2 など、通

常はバージョン番号）を入力します [9]。
    - 入力内容に基づいて表示される **[新しいタグを作成]** (Create new tag: <入力したタグ名>) オプションをクリックします [9]。
    - 新しいタグを作成した場合、そのすぐ下に **[ターゲット]** (Target) というドロップダウンが表示されます。ここで、作成するタグがどのブランチ（通常は main や master などの安定ブランチ）の最新コミットを指すか、あるいは特定のコミットハッシュを指定します [9]。

6.4 リリース情報の入力（タイトル、説明/リリースノート）

1. **[リリース タイトル]** (Release title) フィールドに、このリリースの分かりやすい名前を入力します（例: Version 1.0.1, 安定版リリース, 新機能追加ベータ など）[11]。
2. **[このリリースの説明]** (Describe this release) という大きなテキストエリアに、リリースノートを記述します [9]。Markdown記法が利用可能です。このバージョンでの変更点、新機能、バグ修正、貢献者への謝辞などを詳しく記載します。@記号に続けてGitHubユーザー名を入力することで、貢献者をメンションできます [9]。

6.5 リリースノートの自動生成機能の利用

1. 手動でリリースノートを記述する代わりに、**[このリリースの説明]** フィールドの上部にある **[リリースノートの生成]** (Generate release notes) ボタンをクリックすることもできます [11]。
2. これをクリックすると、GitHubは前回のリリースタグ（または指定した比較対象タグ）から現在のタグまでの間にマージされたPull Requestの情報を基に、リリースノートの下書きを自動的に生成し、説明フィールドに挿入します [12]。
3. 自動生成された内容はあくまで下書きです。内容を確認し、必要に応じて編集、追記、削除を行ってください [12]。

6.6 アセット（ファイル）のアップロード

1. **[Attach binaries by dropping them here or selecting them]** (バイナリをここにドロップするか、選択して添付します) と表示されている枠内に、配布したいファイル（コンパイル済み実行ファイル、インストーラー、ドキュメントPDFなど）をドラッグ &ドロップします [9]。
2. または、枠内をクリックしてファイル選択ダイアログを開き、アップロードしたいファイルを選択します [9]。
3. ファイルのアップロードが完了するまで待ちます。プログレスバーが表示されます。複数のファイルを添付することが可能です [7]。

6.7 プレリリース / 最新リリース としてマークする

1. プレリリース: もしこのリリースが開発版、ベータ版、リリース候補版など、まだ公式な安定版ではない場合は、**[これはプレリリースです]** (This is a pre-release) というチェックボッ

クスをオンにします [3]。これにより、リリースページで "Pre-release" というラベルが表示され、ユーザーに注意を促します。
2. 最新リリース: このリリースをリポジトリの「最新版 (Latest)」として目立つように表示したい場合は、**[最新リリースとして設定する]** (Set as latest release) チェックボックスをオンにします [9]。もしこのオプションをオンにしない場合、GitHubは通常、セマンティックバージョニングの規則に従って、最も新しいバージョンの安定版リリース（プレリリースではないもの）を自動的に「Latest」として扱います [9]。

6.8 リリースの公開または下書き保存

1. 公開: すべての情報の入力とアセットのアップロードが完了し、リリースを一般に公開する準備ができたら、ページ下部にある緑色の **[リリースの公開]** (Publish release) ボタンをクリックします [11]。これにより、リリースが作成され、リポジトリのアクセス権を持つユーザー（パブリックリポジトリの場合は全員）が閲覧・ダウンロードできるようになります。
2. 下書き保存: まだリリースノートの記述が途中であったり、アセットの準備ができていないなど、すぐに公開したくない場合は、**[下書きの保存]** (Save draft) ボタンをクリックします [11]。これにより、リリース情報は保存されますが、公開はされません。下書き状態のリリースは、リポジトリへの書き込み権限を持つメンバーにのみ表示されます [25]。後で編集を再開し、公開することができます。

6.9 作成済みリリースの編集と削除

1. 編集: リリースページで、編集したいリリースの右側にある鉛筆アイコン (編集) をクリックします [9]。リリース作成時と同様の編集画面が表示されるので、タイトル、説明、アセットなどを修正し、最後に **[リリースの更新]** (Update release) ボタンをクリックします [9]。
2. 削除: リリースページで、削除したいリリースの右側にあるゴミ箱アイコン (削除) または [...] メニュー内の **[このリリースを削除]** (Delete this release) をクリックします [9]。確認ダイアログが表示されるので、内容を確認の上、削除を実行します。注意点として、リリースを削除しても、そのリリースに関連付けられていたGitタグは自動的には削除されません。タグも不要な場合は、別途Gitコマンド (git push --delete origin <tagname>) やGitHubのタグ管理画面から削除する必要があります [22]。

6.10 リリースとタグの確認方法

1. リポジトリのメインページ右側の **[リリース]** (Releases) リンクをクリックします [10]。
2. デフォルトで、作成されたリリースが新しいものから順に一覧表示されます。
3. ページ上部には **[リリース]** (Releases) と **[タグ]** (Tags) というタブがあります [10]。**[リリース]** タブでは公式なリリース情報を、**[タグ]** タブではリポジトリに存在するすべてのGitタグ（リリースに関連付けられていないものも含む）の一覧を確認できます。

7. まとめ

7.1 GitHub Releasesの利点の再確認

GitHub Releasesは、ソフトウェア開発プロジェクトにおいて多くの利点を提供します。

- ソフトウェアのバージョン管理と配布プロセスを形式化し、効率化します。
- エンドユーザーは、特定のバージョンのソフトウェア（バイナリ、ソースコード、ドキュメントなど）を簡単に見つけ、ダウンロードできます。
- リリースノート機能を通じて、各バージョンでの変更履歴、新機能、修正点、貢献者などを明確に伝えることができます。
- GitHub API、CLI、そしてGitHub Actionsとの連携により、リリースプロセス全体を自動化することが可能です。
- すべてのリリースはGitタグに基づいているため、配布された成果物と、その基になったソースコードの正確な対応関係が保証されます。
- 個々のアセットファイルに対して2 GiBという寛容なサイズ制限があり、合計サイズや帯域幅にも制限がないため、大規模な成果物の配布にも適しています。

7.2 活用のヒント

GitHub Releasesをより効果的に活用するためのヒントをいくつか挙げます。

- **バージョニング規則の採用:** v1.2.3 のようなセマンティックバージョニングをGitタグ名に一貫して採用することを検討します [3]。これにより、バージョン間の関係性が明確になり、自動化ツール（特に「最新リリース」の自動判定）との親和性も高まります。
- **リリースノートの自動化とカスタマイズ:** リリースノートの自動生成機能 [12] を積極的に活用し、.github/release.yml によるカスタマイズ [14] を行うことで、ドキュメント作成の労力を削減しつつ、質の高い情報提供を目指します。Pull Requestのラベリング規約を整備することが鍵となります。
- **CI/CDへの統合:** 可能な限り、リリース作成プロセスをCI/CDパイプラインに組み込み、ビルドから配布物アップロードまでを自動化します [7]。これにより、手作業によるミスを防ぎ、迅速かつ一貫したリリースが可能になります。
- **プレリリースフラグの活用:** 安定版ではないバージョンを配布する際には、必ず「プレリリース」フラグ [11] を使用し、ユーザーの期待値を適切に管理します。
- **アセットの整理:** 配布するアセットの種類（OS別バイナリ、ソース、ドキュメントなど）を明確にし、ファイル名を分かりやすくするなどして、ユーザーが必要なものを簡単に見つけられるように配慮します。
- **Git LFSとの使い分け:** リポジトリの健全性を保つために、バージョン管理が必要な大きなファイル（例: 学習済みモデル、大規模なテストデータ）はGit LFS [15] で管理し、最終的な配布物（例: コンパイル済みアプリケーション）はReleasesのアセット [16] として提供する、という戦略的な使い分けを検討します。

GitHub Releasesは単なるファイル置き場ではなく、ソフトウェアのライフサイクルにおける重要なコミュニケーションと配布のハブとして機能します。これを開発ワークフロー、バージョニン

グ戦略、ドキュメンテーション、そしてCI/CD戦略全体の中に思慮深く統合することで、その価値を最大限に引き出すことができるでしょう。

引用文献

1. リリースについて - GitHub Docs, 4月 20, 2025にアクセス、https://docs.github.com/ja/repositories/releasing-projects-on-github/about-releases
2. About releases - GitHub Docs, 4月 20, 2025にアクセス、https://docs.github.com/repositories/releasing-projects-on-github/about-releases
3. Release Your Software - The GitHub Blog, 4月 20, 2025にアクセス、https://github.blog/news-insights/product-news/release-your-software/
4. Why popular repositories use release version as a "Release title" in GitHub?, 4月 20, 2025にアクセス、https://softwareengineering.stackexchange.com/questions/345006/why-popular-repositories-use-release-version-as-a-release-title-in-github
5. GitHub Releaseを触ってみた話 - Qiita, 4月 20, 2025にアクセス、https://qiita.com/nanana08/items/b5791d68261baea43135
6. GitHub Releaseの解説 @ Bio”Pack”athon2023#01 - YouTube, 4月 20, 2025にアクセス、https://www.youtube.com/watch?v=tUKSc1t17jQ
7. GitHub Releases Uploading - Travis CI docs, 4月 20, 2025にアクセス、https://docs.travis-ci.com/user/deployment/releases/
8. 【Git/GitHub入門】git tagの仕様・使い方 - カゴヤのサーバー研究室, 4月 20, 2025にアクセス、https://www.kagoya.jp/howto/rentalserver/webtrend/gittag/
9. リポジトリのリリースを管理する - GitHub Docs, 4月 20, 2025にアクセス、https://docs.github.com/ja/repositories/releasing-projects-on-github/managing-releases-in-a-repository
10. リポジトリのリリースとタグを表示する - GitHub Docs, 4月 20, 2025にアクセス、https://docs.github.com/ja/repositories/releasing-projects-on-github/viewing-your-repositorys-releases-and-tags
11. Managing releases in a repository - GitHub Docs, 4月 20, 2025にアクセス、https://docs.github.com/en/repositories/releasing-projects-on-github/managing-releases-in-a-repository
12. 自動生成リリース ノート - GitHub Docs, 4月 20, 2025にアクセス、https://docs.github.com/ja/repositories/releasing-projects-on-github/automatically-generated-release-notes
13. GitHub でプロジェクトをリリースする, 4月 20, 2025にアクセス、https://docs.github.com/ja/repositories/releasing-projects-on-github
14. 【Github Actions】リリースノートとタグを自動生成する, 4月 20, 2025にアクセス、https://zenn.dev/rehabforjapan/articles/github-actions-release-note
15. About Git Large File Storage - GitHub Docs, 4月 20, 2025にアクセス、https://docs.github.com/repositories/working-with-files/managing-large-files/about-git-large-file-storage
16. About large files on GitHub, 4月 20, 2025にアクセス、

https://docs.github.com/repositories/working-with-files/managing-large-files/about-large-files-on-github
17. GitHubのリリース機能の使い方 - Qiita, 4月 20, 2025にアクセス、https://qiita.com/okdyy75/items/83e3f2ae0bbc0284a92c
18. GitHub Desktop でのタグの管理, 4月 20, 2025にアクセス、https://docs.github.com/ja/desktop/managing-commits/managing-tags-in-github-desktop
19. GitHub タグとリリースの作成 - Ground Sunlight, 4月 20, 2025にアクセス、http://www.y2sunlight.com/ground/doku.php?id=github:release
20. Git tagとGitHub ReleasesとCHANGELOG.mdの自動化について - Web Scratch, 4月 20, 2025にアクセス、https://efcl.info/2014/07/20/git-tag-to-release-github/
21. git tag と GitHub の Release 機能でプロっぽさを出してみよう - Qiita, 4月 20, 2025にアクセス、https://qiita.com/tommy_aka_jps/items/5b39e4b27364c759aa53
22. 【GitHub】リリースノート自動生成機能が便利だったので実際に使ってみた, 4月 20, 2025にアクセス、https://www.teru2teru.com/github/github-release-note/
23. Releasing projects on GitHub, 4月 20, 2025にアクセス、https://docs.github.com/en/repositories/releasing-projects-on-github
24. リリースの REST API エンドポイント - GitHub Docs, 4月 20, 2025にアクセス、https://docs.github.com/ja/rest/releases/releases?apiVersion=2022-11-28
25. リリースの REST API エンドポイント - GitHub Docs, 4月 20, 2025にアクセス、https://docs.github.com/ja/rest/releases/releases
26. GitHub Releases - Short & Practical Guide - reemus.dev, 4月 20, 2025にアクセス、https://reemus.dev/tldr/github-releases-practical-guide
27. GitHub Actionsで自動リリースノートを作成 - Qiita, 4月 20, 2025にアクセス、https://qiita.com/kazu697/items/bd4184dfabf51edf1253
28. A curated list of awesome actions to use on GitHub, 4月 20, 2025にアクセス、https://github.com/sdras/awesome-actions
29. Github private repository and releases visibility - Stack Overflow, 4月 20, 2025にアクセス、https://stackoverflow.com/questions/76144037/github-private-repository-and-releases-visibility
30. Are releases public in a private repository · community · Discussion #126691 - GitHub, 4月 20, 2025にアクセス、https://github.com/orgs/community/discussions/126691
31. GitHub 利用規約, 4月 20, 2025にアクセス、https://docs.github.com/ja/site-policy/acceptable-use-policies/github-acceptable-use-policies
32. About large files on GitHub - GitHub Enterprise Server 3.14 Docs, 4月 20, 2025にアクセス、https://docs.github.com/en/enterprise-server@3.14/repositories/working-with-files/managing-large-files/about-large-files-on-github
33. About storage and bandwidth usage - GitHub Docs, 4月 20, 2025にアクセス、https://docs.github.com/en/repositories/working-with-files/managing-large-files/about-storage-and-bandwidth-usage
34. Managing large files - GitHub Docs, 4月 20, 2025にアクセス、

https://docs.github.com/en/repositories/working-with-files/managing-large-files
35. GitHub での大きいファイルについて, 4月 20, 2025にアクセス、https://docs.github.com/ja/repositories/working-with-files/managing-large-files/about-large-files-on-github
36. Github Releases : r/github - Reddit, 4月 20, 2025にアクセス、https://www.reddit.com/r/github/comments/198xl5a/github_releases/
37. Can I make releases public from a private github repo? - Stack Overflow, 4月 20, 2025にアクセス、https://stackoverflow.com/questions/26217941/can-i-make-releases-public-from-a-private-github-repo
38. Usage limits, billing, and administration - GitHub Actions, 4月 20, 2025にアクセス、https://docs.github.com/en/actions/administering-github-actions/usage-limits-billing-and-administration
39. Pricing · Plans for every developer - GitHub, 4月 20, 2025にアクセス、https://github.com/pricing
40. GitHub のサービス使用条件, 4月 20, 2025にアクセス、https://docs.github.com/ja/site-policy/github-terms/github-terms-of-service
41. GitHub プレリリース ライセンス条項, 4月 20, 2025にアクセス、https://docs.github.com/ja/site-policy/github-terms/github-pre-release-license-terms
42. GitHubのReleaseでTagを間違えて打ってしまった時の対処法 - Zenn, 4月 20, 2025にアクセス、https://zenn.dev/justin999/articles/git-modify-release-or-delete-tag