



Woodland
Academy

7.6 バージョン管理

- Git
- Git の利用
- Markdown
- SVNの利用



Shape Your Future

目次

- 1 **Git**
- 2 **Git の利用**
- 3 **Markdown**
- 4 **SVNの利用**



バージョン管理ツール

- **バージョン管理**[Version Control] ツールは、ディレクトリ（フォルダ）やファイルの変更履歴を完全に保存・追跡するバージョン管理機能を提供し、ソフトウェア開発者にとって必須のツールであり、ソフトウェア会社にとってはインフラとなるものとも言えます。

バージョン管理ツールの機能

- バージョン管理の最も重要な機能は、**ファイルの変更を追跡**することです。
- バージョン管理ツールは、**いつ、誰が、どのファイルのどこ**を変更したかを忠実に記録します。ファイルが変更されるたびに、そのファイルのバージョン番号が増加します。
- もう一つの重要な機能は、**並行開発**であります。
- ソフトウェア開発は、多くの場合、複数人の共同作業で行われます。バージョン管理は、異なる開発者間のバージョンの同期と開発コミュニケーションの問題を効果的に解決し、共同開発の効率を高めることができます。並行開発でよくあるバグ修正の問題も、バージョン管理ツールで**ブランチマージ**を行うことで解決することができます。

バージョン管理ツールの役割

- 共同開発：チームで同じプロジェクトに取り組む。
- バージョニング：プロジェクトのバージョンを継続的に増やしていく形で、段階的にプロジェクトを完成させる。
- データのバックアップ：開発されたすべてのバージョンの履歴を保存している。
- 権限管理：チームの開発者に異なる権限を割り当てる。
- ブランチマネジメント：開発チームが複数のラインで同時にタスクを進めることで、さらなる効率化を図ることができる。

Git とは

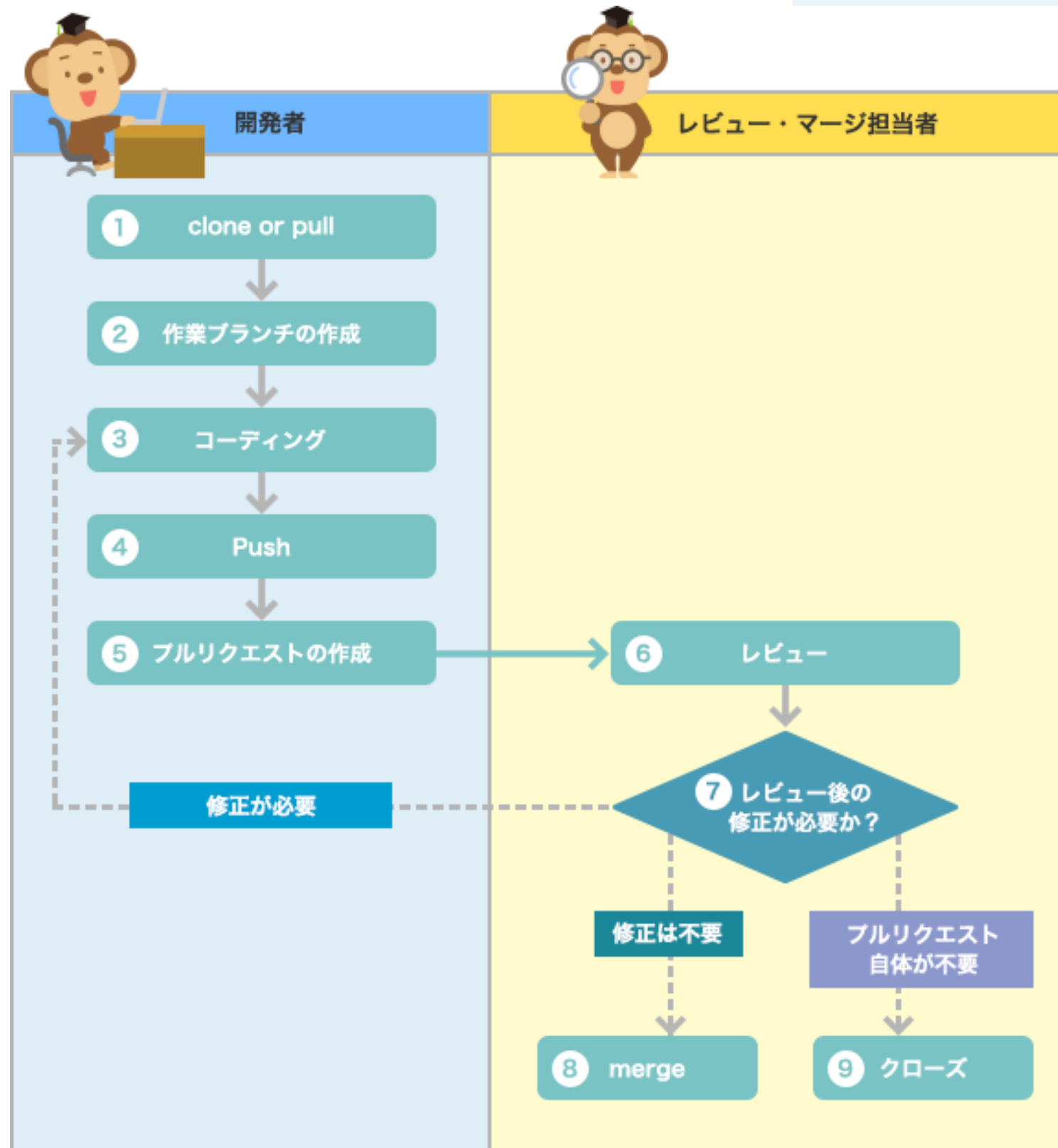
- **Git** はオープンソースの分散型バージョン管理システムで、規模の大小にかかわらず、あらゆるプロジェクトで俊敏かつ効率的に作業を行えます。
- Git はもともと、Linux カーネル開発を管理するためのオープンソースのバージョン管理ソフトウェアとして、Linus Torvalds 氏によって開発されました。
- **CVS** や **Subversion** などの従来の集中型なバージョン管理ツールとは異なり、Git は**分散型リポジトリ**を採用しており、サーバー側にはソフトウェアが要りません。



Git の作業フロー

- Git のファイルを作業ディレクトリに**クローン**する。
- 他の人がファイルを変更した場合、どのファイルがどう変更するかは**確認**できる。
- クローンしたファイルに**追加・変更**する。
- コミットする前に**変更点を確認**する。
- 変更を**コミット**する。
- 変更完了後、もしエラーが見つかったら、コミットを撤回し、変更し直してコミットすることができる。

Gitの作業プロセス



参考 : <https://backlog.com/ja/git-tutorial/>

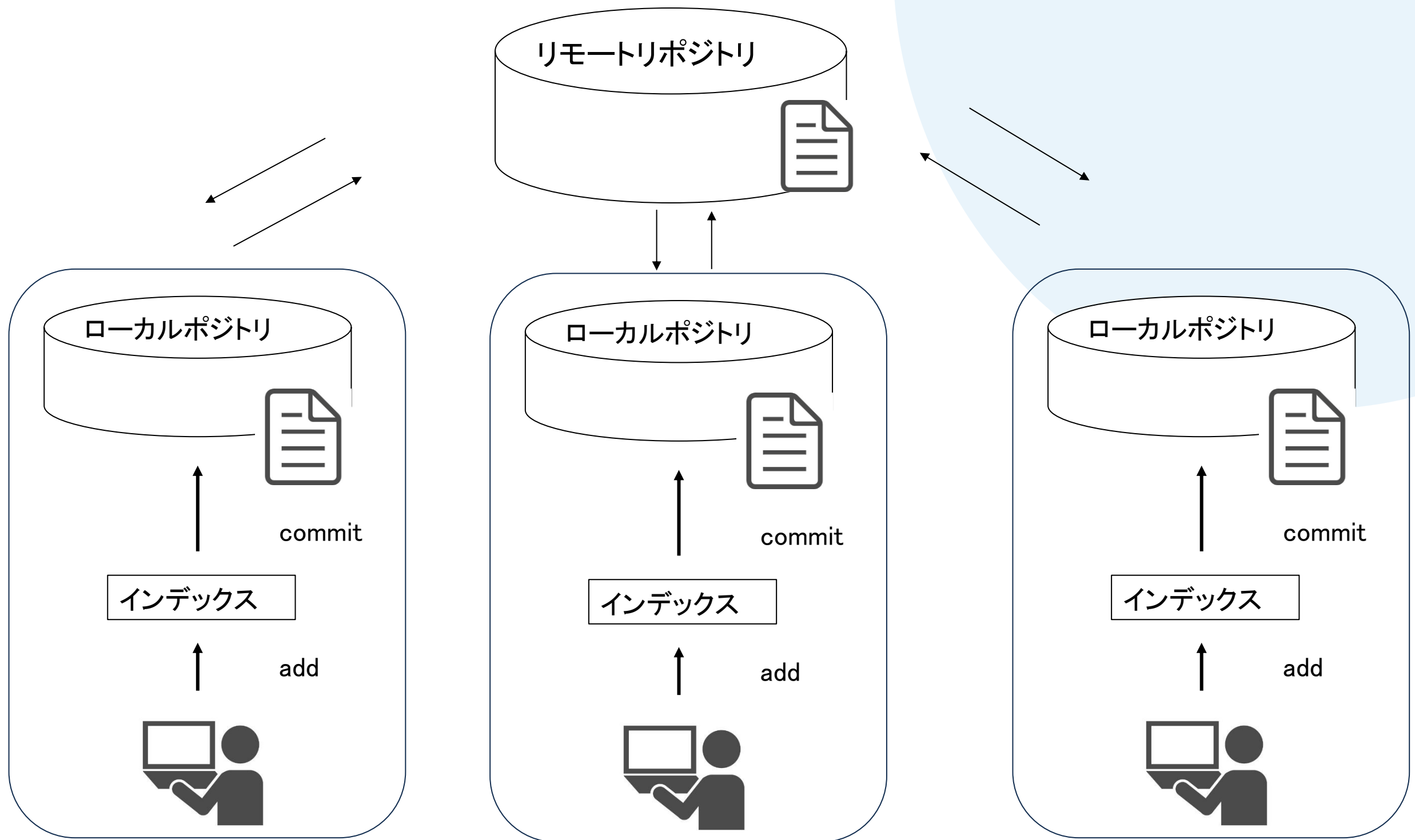
ファイルの 3 つの状態

- Git にはファイルは 3 つの状態があり、**コミット済**[Committed]、**変更済**[Modified]、**ステージ済**[Staged]のどちらかになっています。
- **コミット済**のファイルは、ローカルデータベースに安全に保存されています。
- **変更済**は、ファイルが変更されたが、まだデータベースに保存されていないことを意味します。
- **ステージ済**とは、変更されたファイルの最新バージョンが、次のコミットに含まれるようにマークされています。
- これらと関係するのは Git の 3 つの作業領域：**Git リポジトリ**、**作業ディレクトリ**、**ステージングエリア**。

3 つの作業領域

- Git の**リポジトリ** [Repository] とは、Git が**プロジェクトのメタデータとデータベース**を保持する場所です。これは Git の最も重要な部分で、他のコンピュータからリポジトリをクローンするときにデータはここからコピーされます。
- **作業ディレクトリ** [Working Directory] は、あるバージョンのプロジェクトの内容を個別に抽出したものです。ここにあるファイルは、リポジトリから抽出され、我々は使用または変更できるように、**ディスク上に配置**ています。
- **ステージングエリア** [Staging Area] とは、**次回にコミットされるファイル**のリストに関する情報を保持するファイルです。通常は Git リポジトリに置かれます。「**インデックス** [index]」と呼ばれることもあります。

Git のワークフロー : 3 つの状態

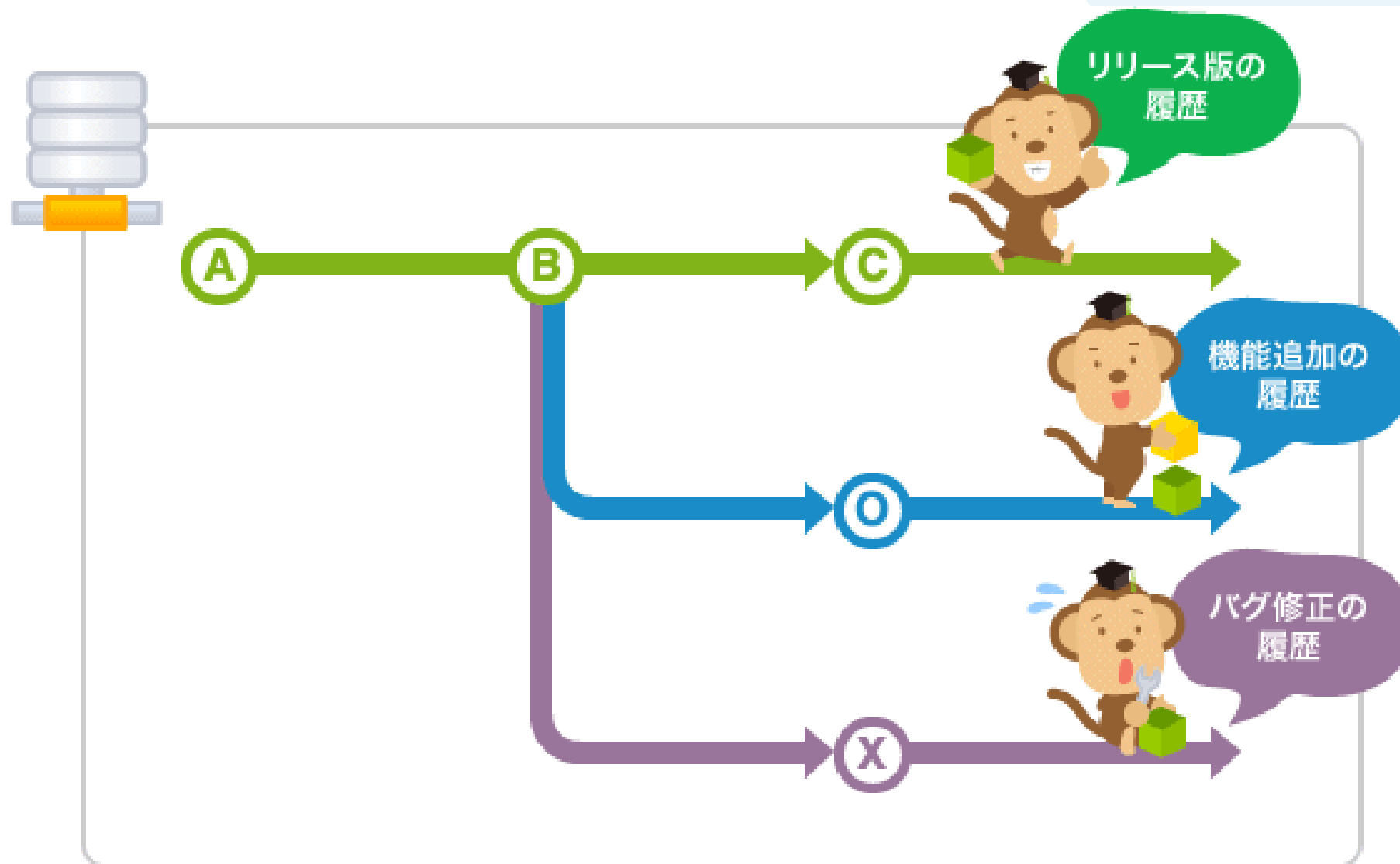


Git ブランチ

- ソフトウェアを開発する場合、同じソフトウェアの機能開発は複数の人が同時に行い、メンテナンスが必要なバージョンも複数存在する場合があります。
- 幸いなことに、Git には**ブランチ**[Branch]機能があって、複数の機能を同時に開発しバージョン管理することをサポートしています。

ブランチの目的

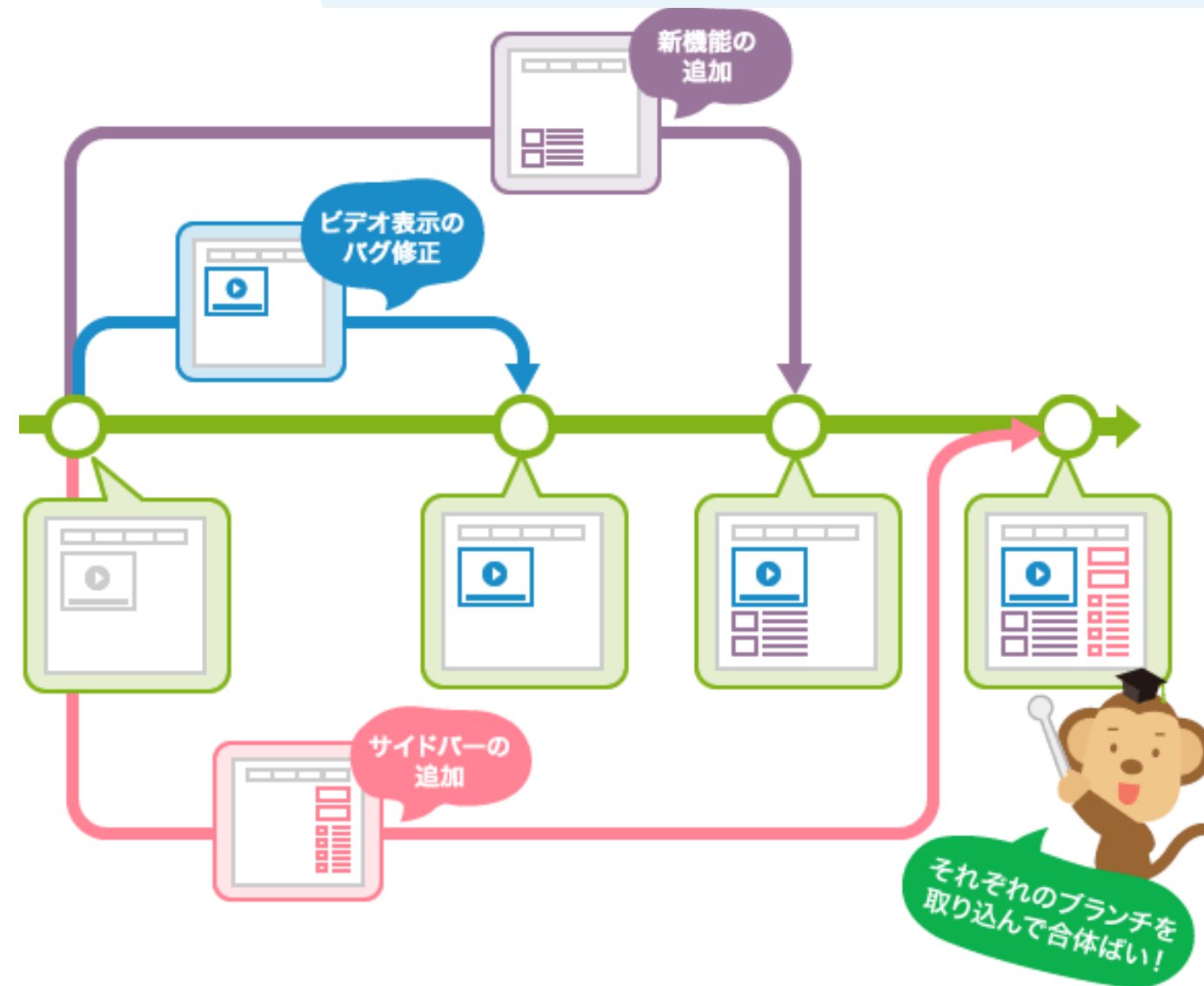
- ブランチ機能の目的は、変更の全体的な流れを分岐させること（**フォーク**^[Fork]）です。分岐されたブランチは他のブランチの影響を受けないので、同じプロジェクトの複数のバージョン変更を行うことができます。



参考 : <https://backlog.com/ja/git-tutorial/>

マージ

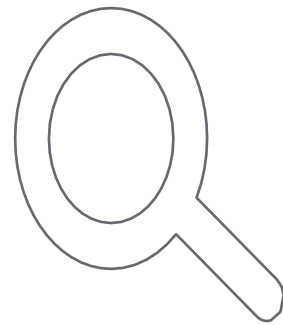
- フォークされたブランチを合併（**マージ**^[Merge]）することができます。
- 他の開発者の影響を受けないように、**master** ブランチ上に自分専用のブランチを作成することができます。作業が終了したら、自分のブランチの変更をこの master ブランチにマージします。各コミットの履歴が保存されるため、問題が発生したときに原因となったコミットを探し出し、修正することが非常に容易になります。



参考 : <https://backlog.com/ja/git-tutorial/>



Q&A



目次

- 1 Git
- 2 入力してください
- 3 Markdown
- 4 SVNの利用



Git のインストール

- Windows :  <https://gitforwindows.org>
- Mac (Terminal で実行) :
 - インストールパッケージの管理ツールをインストール :
`/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

- Git をインストール :

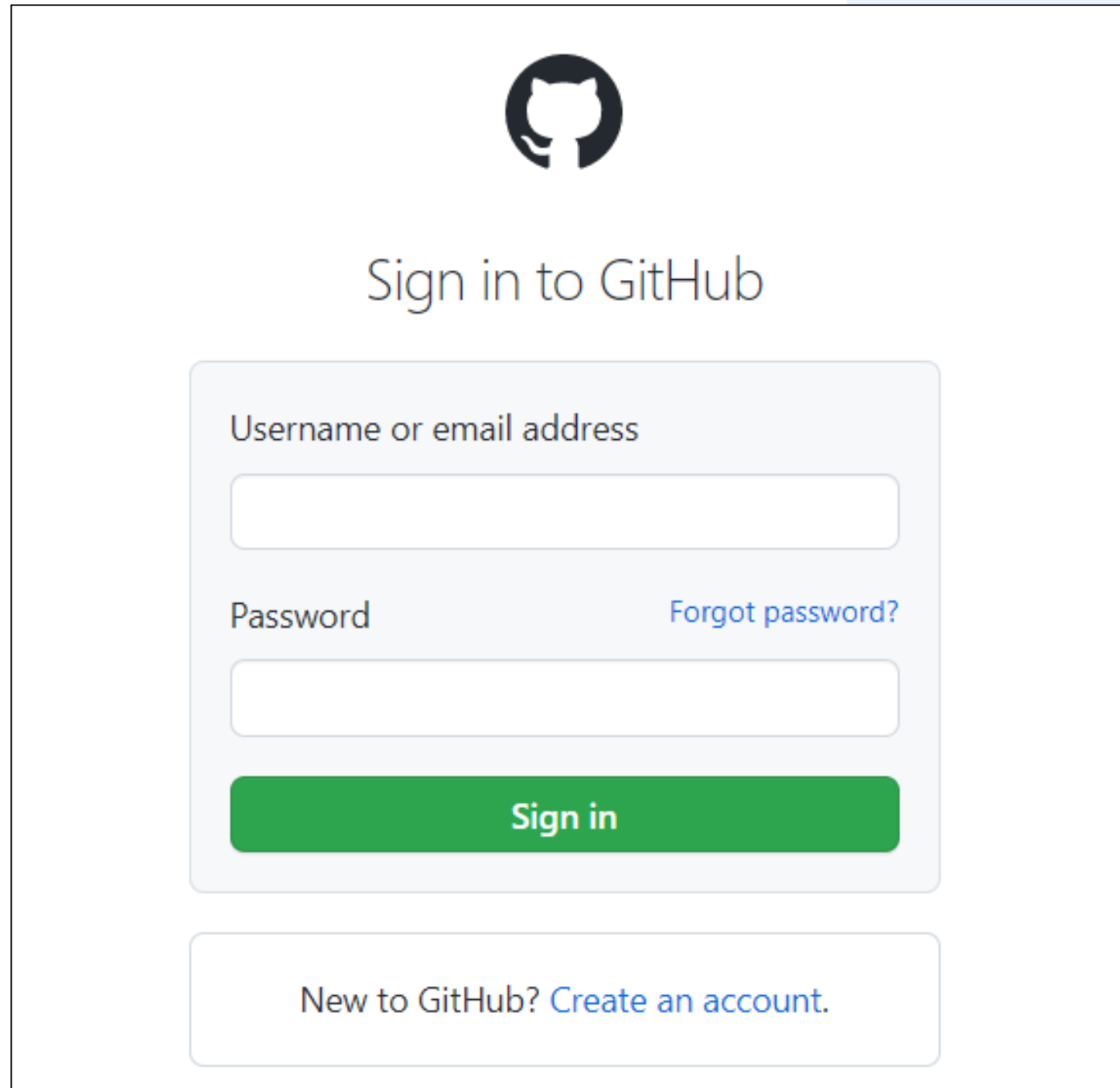
```
brew install git
```

- インストールを確認します。cmd または Terminal で :

```
git --version
```

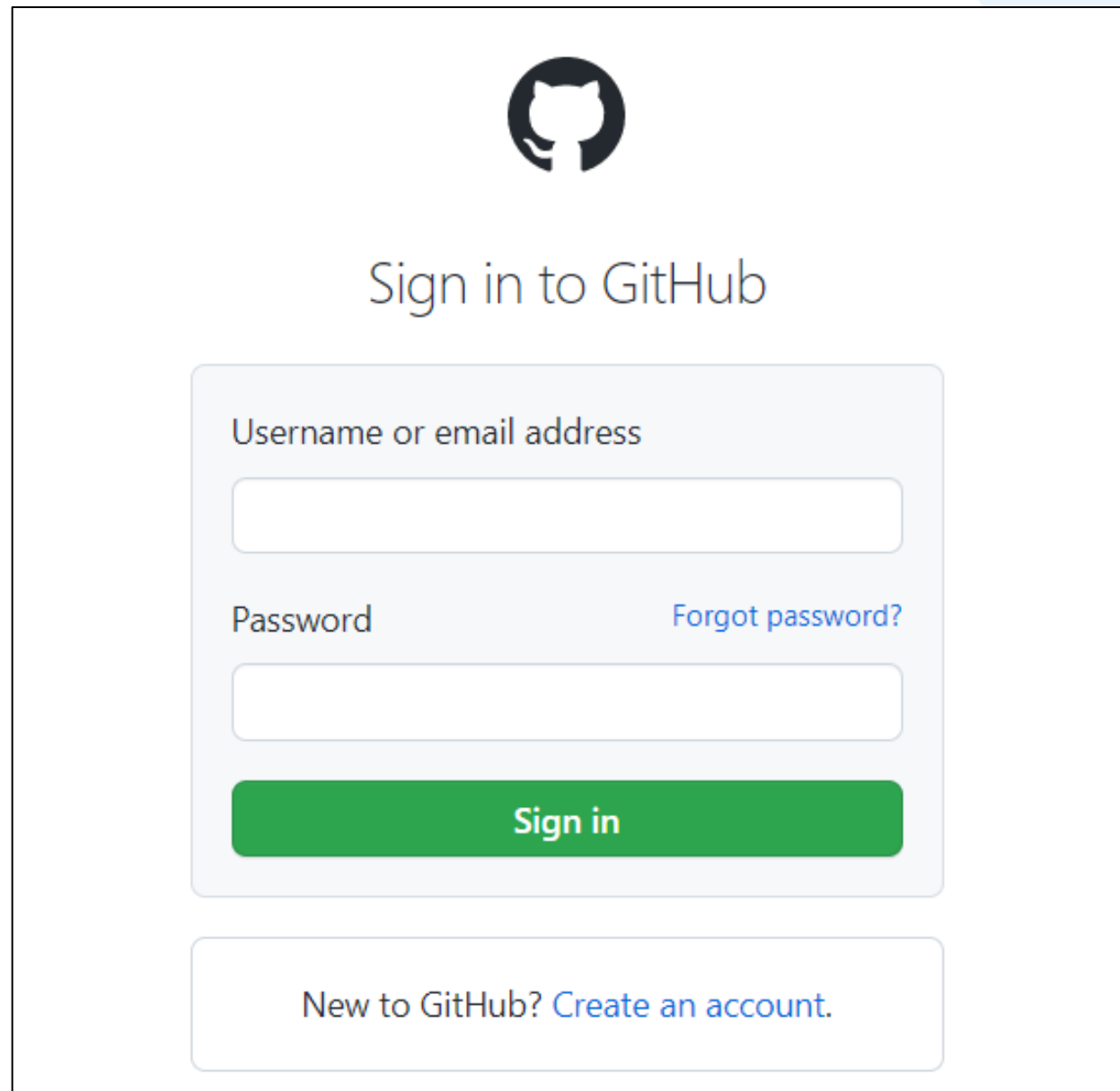
GitHubの設定

- GitHubの登録を行います。 <https://github.com/login>



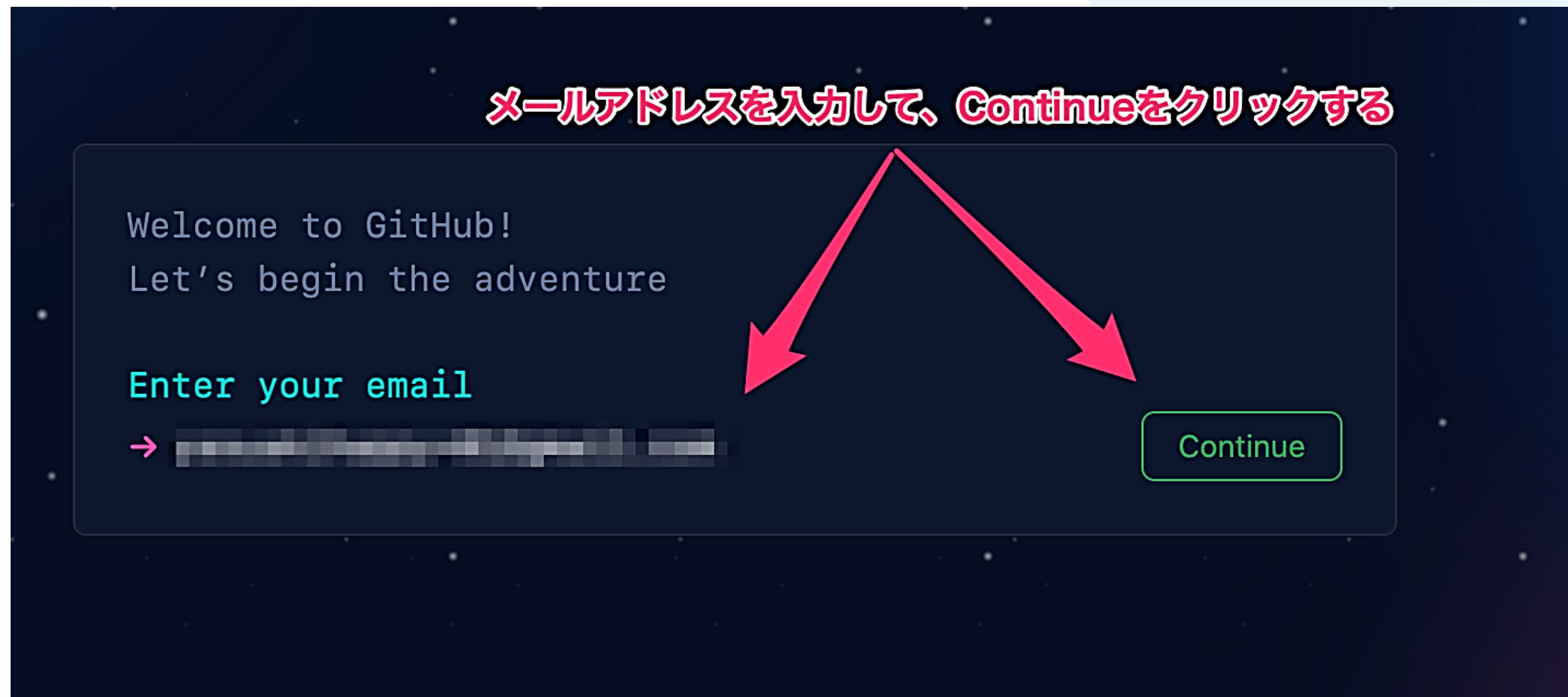
アカウント画面に遷移する

- 「Create an account」を押下する。
ログイン画面で「Create an account.」を押下して、アカウント登録画面に遷移させる。



メールアドレスを入力する

- 「Enter your email」とあるので、メールアドレスを入力して、Continueを押下します。



出典：コードライフ『GitHubに登録して、リポジトリを作成する』

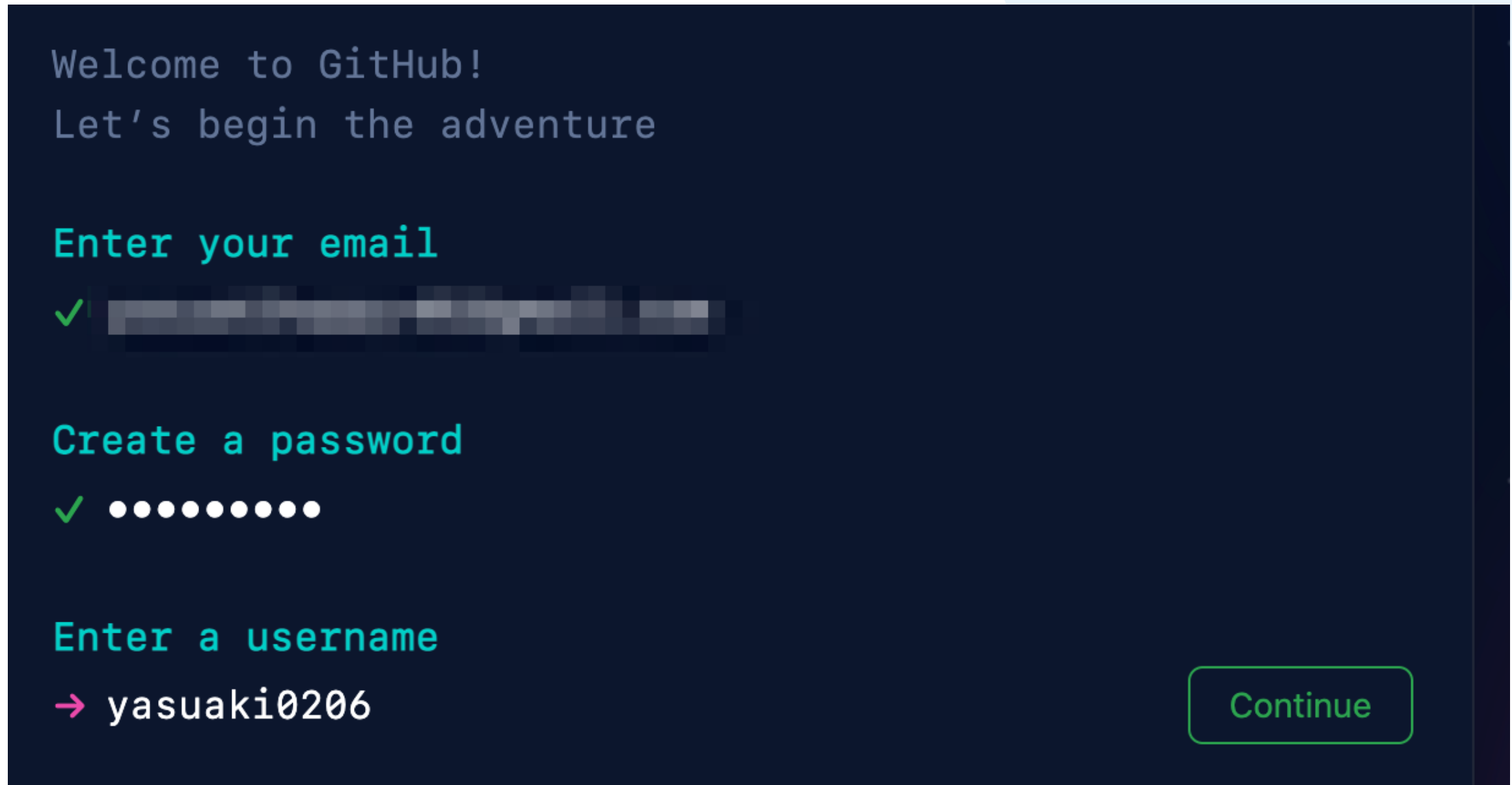
パスワードを作成する

- 「Create a password」とあるので、パスワードを入力して、Continueを押下する。

出典：コードライフ『GitHubに登録して、リポジトリを作成する』

ユーザー名を入力する

- 「Enter a username」とあるので、ユーザー名を入力して、Continueを押下する。



Welcome to GitHub!
Let's begin the adventure

Enter your email
✓ [redacted]

Create a password
✓ ●●●●●●●●

Enter a username
→ yasuaki0206

Continue

お知らせメールの設定

- 「Would you like to receive product updates and announcements via email?」と書いています。
これは、製品のアップデートのお知らせなどをメールで受け取るかの確認です。
yがyesで、nがnoになります。

Would you like to receive product updates and
announcements via email?

Type "y" for yes or "n" for no

→ n

Continue

出典：コードライフ『GitHubに登録して、リポジトリを作成する』

自動登録出ないか確認する

- 「Verify your account」と書いています、ロボットではないかの確認かと思います。最初に「検証する」を押下します。



Verify your account

検証

質問に回答して、あなたがロボットではないことを証明してください。

検証する

出典：[コードライフ『GitHubに登録して、リポジトリを作成する』](#)

指示通りに選択

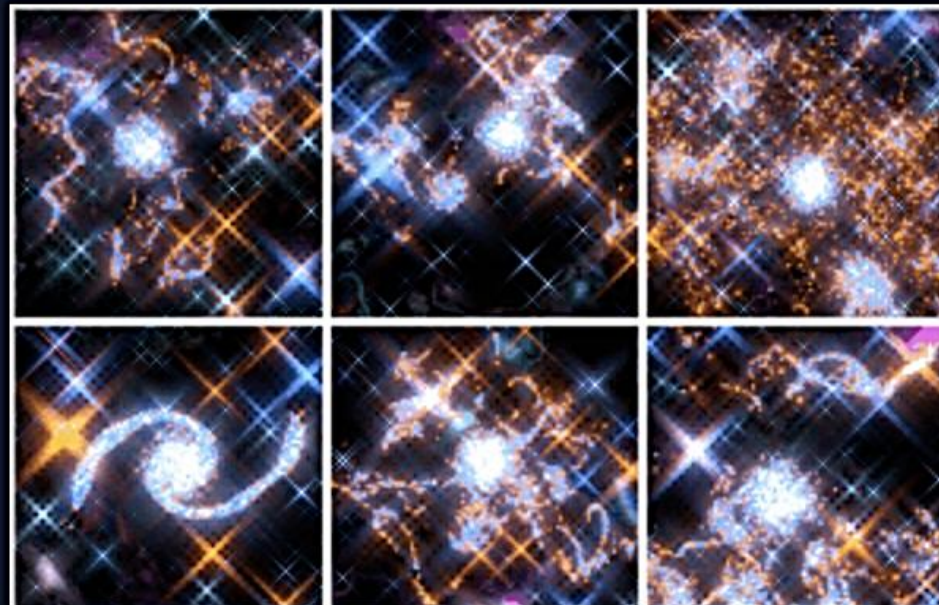
Would you like to receive product updates and announcements via email?

Type "y" for yes or "n" for no

✓ n

Verify your account

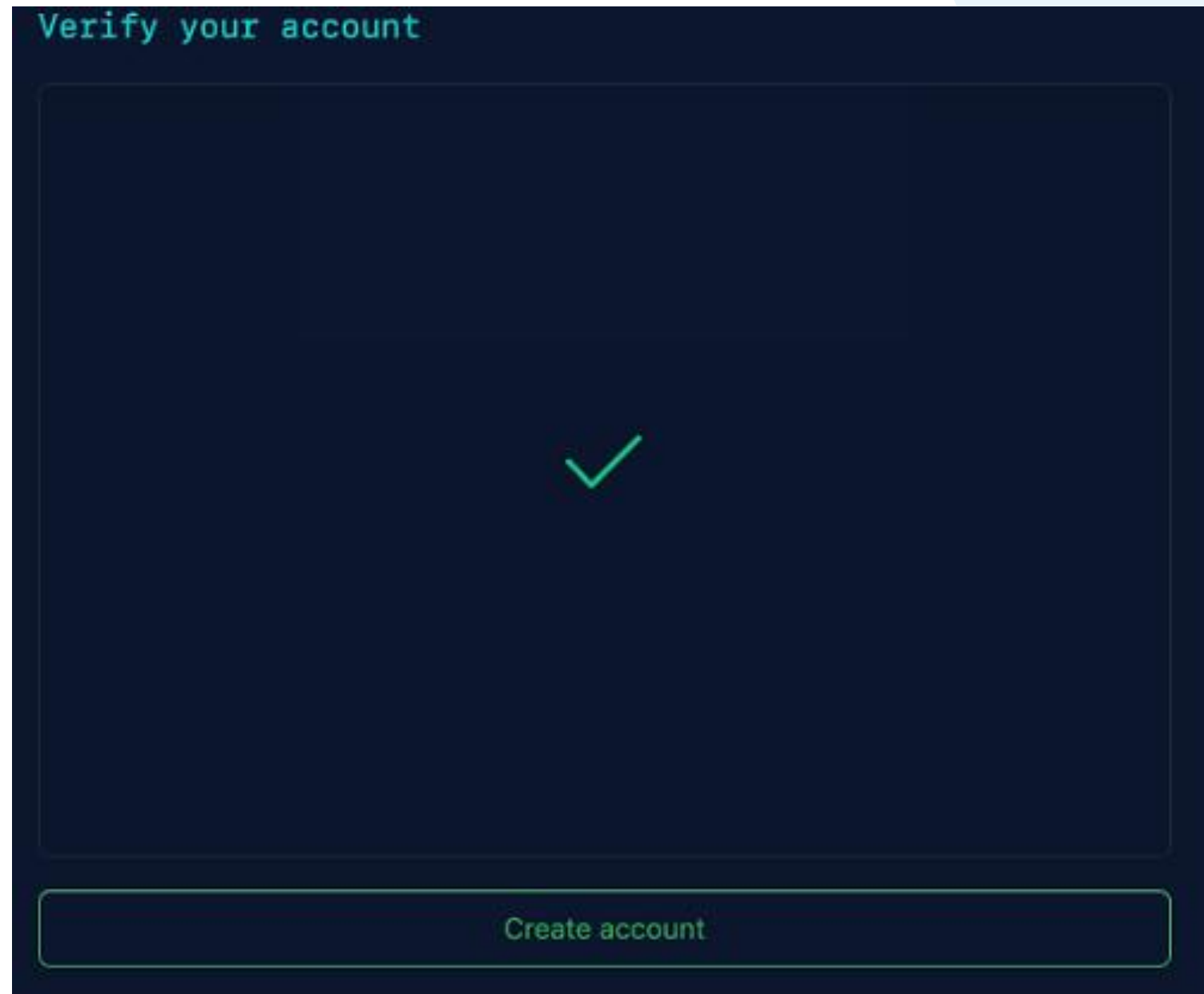
渦巻き銀河を選択してください



出典：コードライフ『GitHubに登録して、リポジトリを作成する』

登録を行う

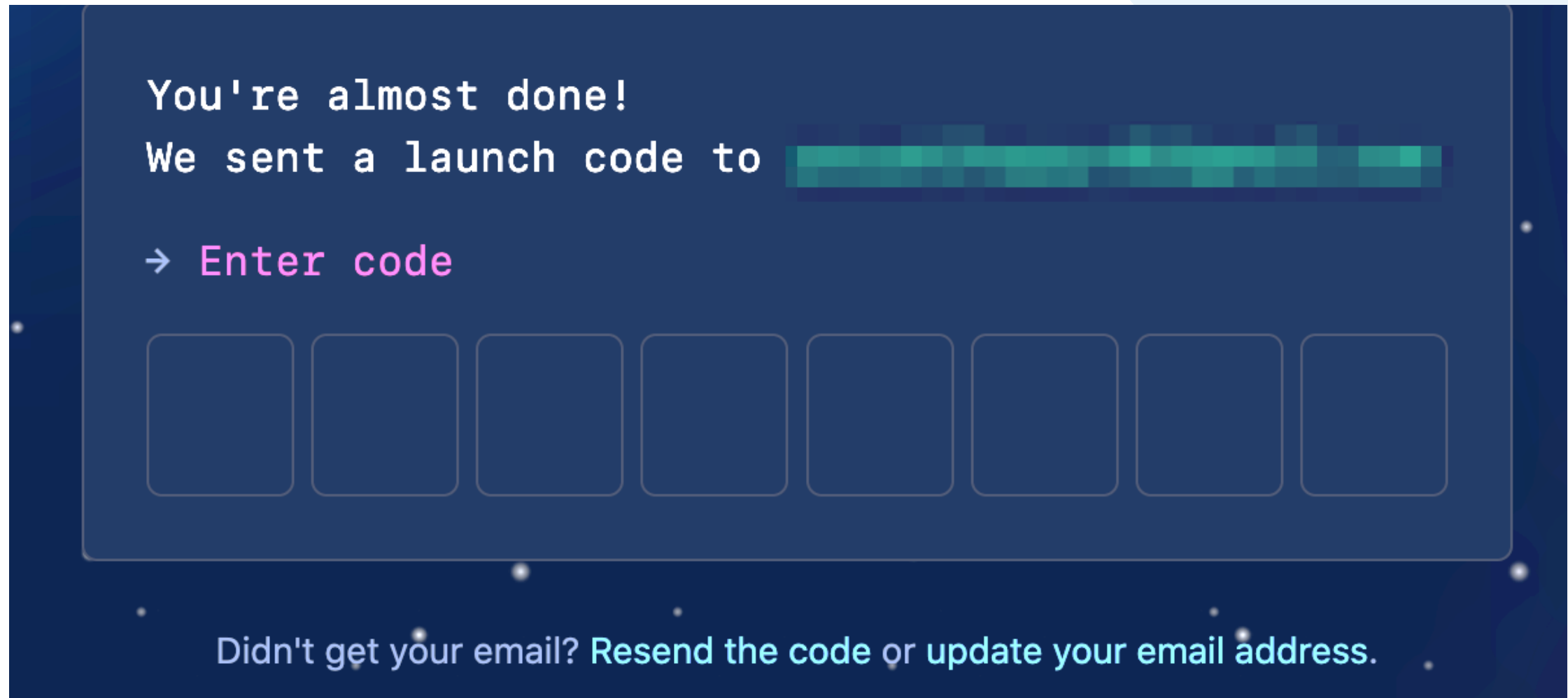
- 指示通りに選択すると、下記のように「Create account」ボタンを押せるようになる。



出典：[コードライフ『GitHubに登録して、リポジトリを作成する』](#)

メールに送信されたコードの入力

- Create account」 ボタンを押すと、下記のようにコードをいれる画面になります。
- コードが入力したメールアドレスに送信されてくるので、メールからコードを確認して下記の画面に貼り付けます。



You're almost done!

We sent a launch code to [blurred email address]

→ Enter code

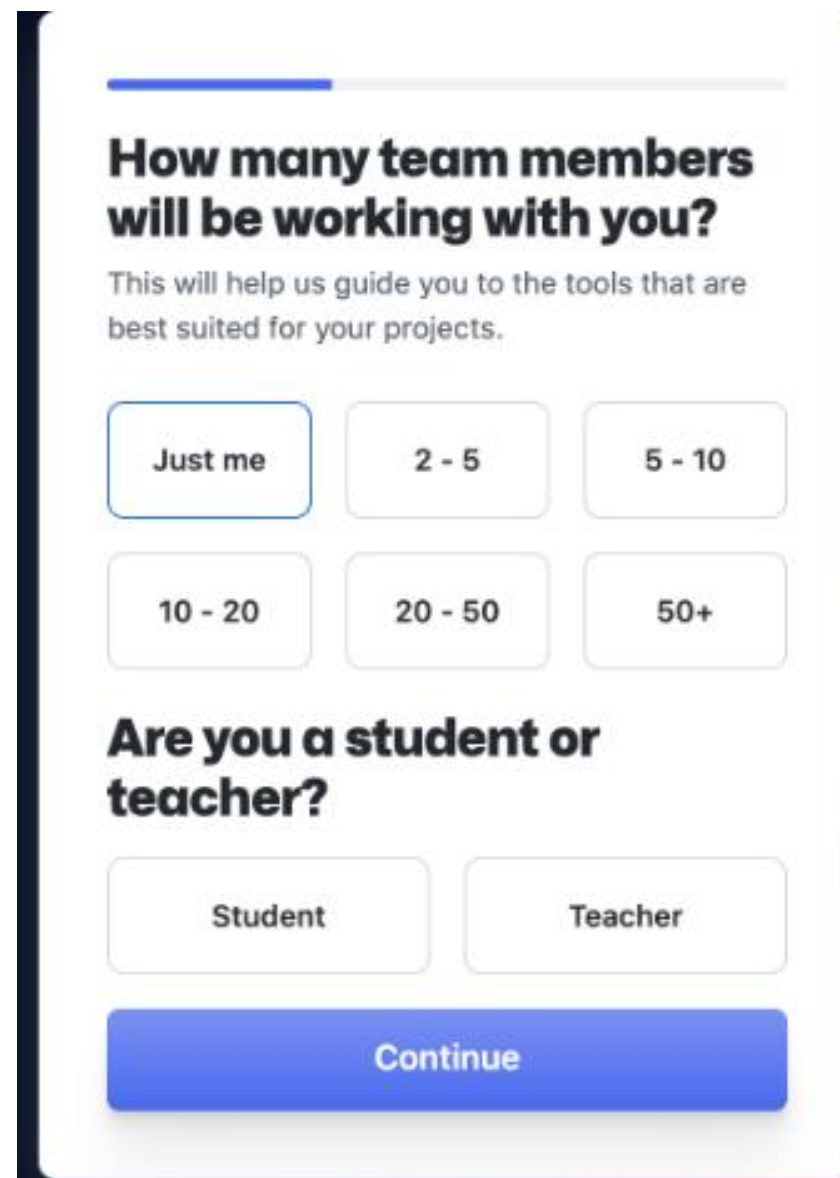
[Eight empty square boxes for code input]

Didn't get your email? [Resend the code or update your email address.](#)

出典：コードライフ『GitHubに登録して、リポジトリを作成する』

Githubアカウント作成時の質問 1

- How many team members will be working with you?
- 「何人のチームメンバーがあなたと一緒に仕事をしますか？」という質問です。適当に選択します。



How many team members will be working with you?

This will help us guide you to the tools that are best suited for your projects.

Just me 2 - 5 5 - 10

10 - 20 20 - 50 50+

Are you a student or teacher?

Student Teacher

Continue

Githubアカウント作成時の質問 1

- How many team members will be working with you?
- 「何人のチームメンバーがあなたと一緒に仕事をしますか？」という質問です。適当に選択します。
- Are you a student or teacher?
- 「あなたは学生ですか、教師ですか？」という質問です。どちらでもなかったら特に選択しなくていいと思います。
- 選択したらContinueをクリックします。

How many team members will be working with you?

This will help us guide you to the tools that are best suited for your projects.

Just me 2 - 5 5 - 10

10 - 20 20 - 50 50+

Are you a student or teacher?

Student Teacher

Continue

出典：コードライフ『GitHubに登録して、リポジトリを作成する』

Githubアカウント作成時の質問2

- what specific features are you interested in using?
- どの機能を使用しますか？のような質問なので、今回は全てチェックをしてみましょう。
- チェックがおわったら、Continueを押下してください。

What specific features are you interested in using?

Select all that apply so we can point you to the right GitHub plan.

☒ Collaborative coding
Codespaces, Pull requests, Notifications, Code review, Code review assignments, Code owners, Draft pull requests, Protected branches, and more.

☒ Automation and CI/CD
Actions, Packages, APIs, GitHub Pages, GitHub Marketplace, Webhooks, Hosted runners, Self-hosted runners, Secrets management, and more.

☒ Security
Private repos, 2FA, Required reviews, Required status checks, Code scanning, Secret scanning, Dependency graph, Dependabot alerts, and more.

☒ Enterprise security
SAML, LDAP, IP allow list, GitHub Connect, and Audit log API.

☒ Client Apps
GitHub Mobile, GitHub CLI, and GitHub Desktop.

☒ Project Management
Projects, Labels, Milestones, Issues, Unified Contribution Graph, Org activity graph, Org dependency insights, Repo insights, Wikis, and GitHub Insights.

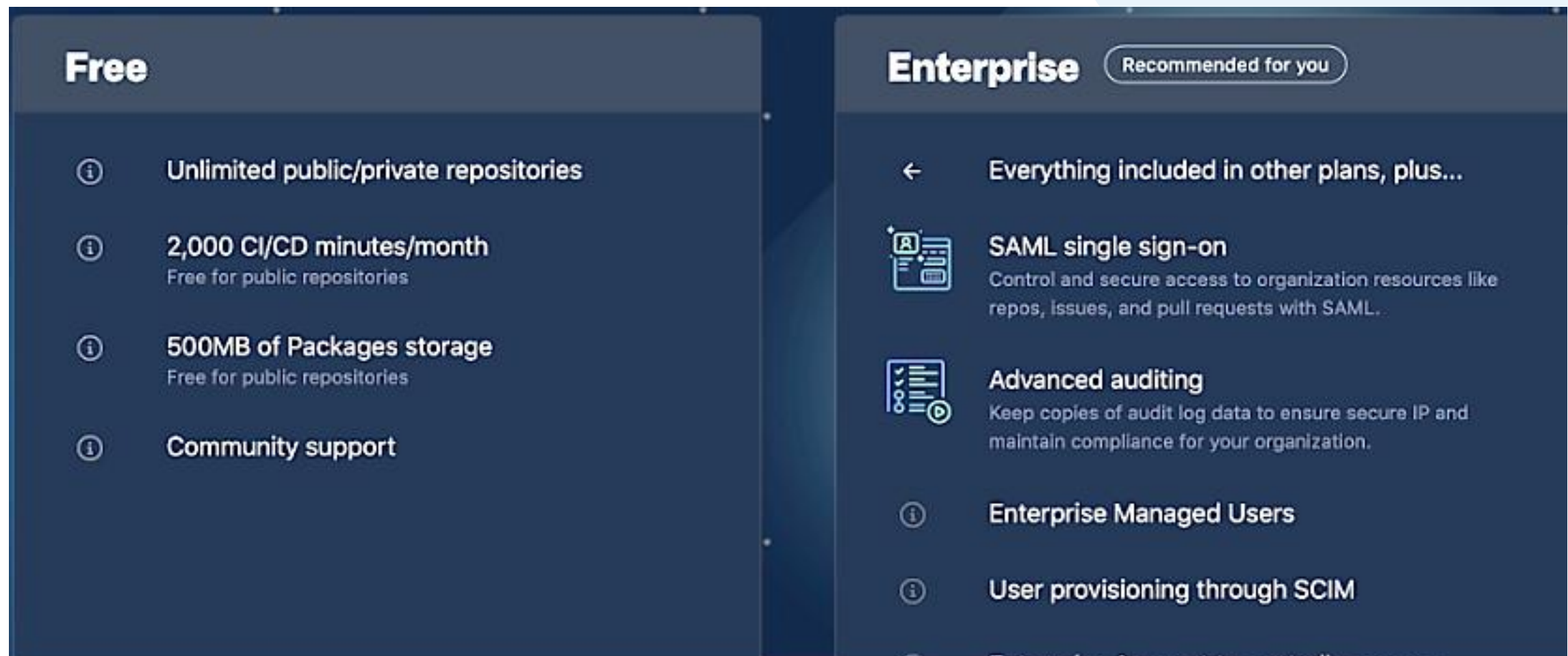
☒ Team Administration
Organizations, Invitations, Team sync, Custom roles, Domain verification, Audit Log API, Repo creation restriction, and Notification restriction.

☒ Community
GitHub Marketplace, GitHub Sponsors, GitHub Learning Lab, Electron, and Atom.

Continue

プランを選択する画面

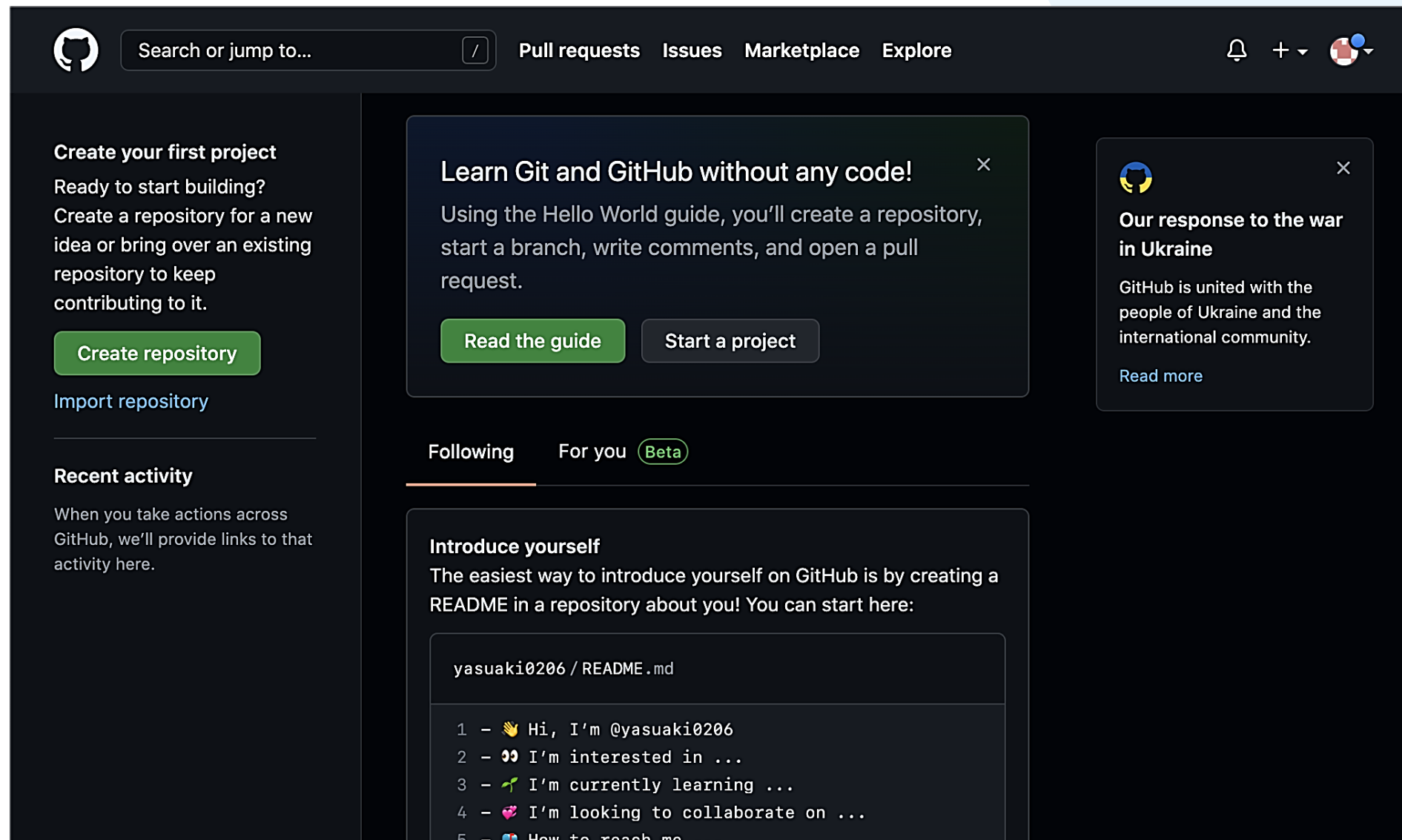
- プランを選ぶ画面になります。FreeプランとEnterpriseプランがあります。
- Freeプランを選んでください。



出典：コードライフ『GitHubに登録して、リポジトリを作成する』

ダッシュボード画面表示

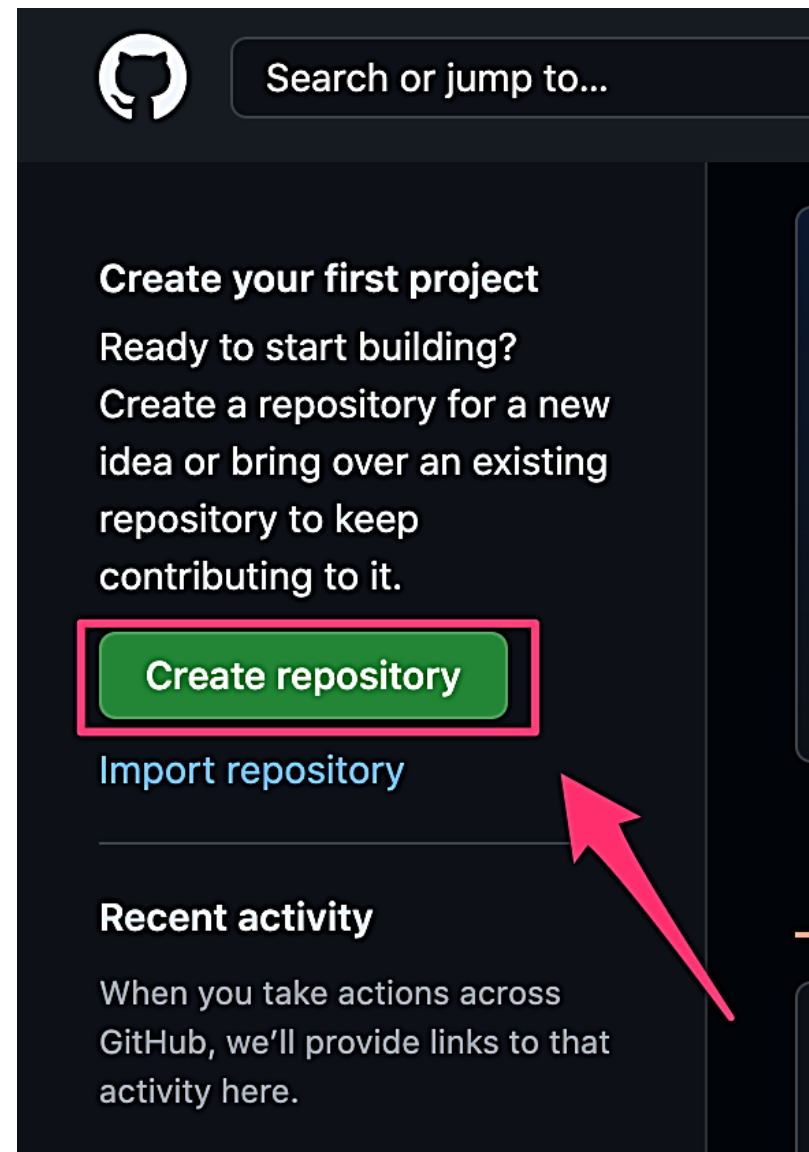
- プランまで選ぶと、ダッシュボードが表示されます。
- これでアカウント作成完了です！



出典：コードライフ『GitHubに登録して、リポジトリを作成する』

リポジトリを作成する

- 次にリポジトリを作成してみます。
- 下記の「Create repository」ボタンから作成することが可能です。
- (自分のプロフィールのRepositoriesのNewボタンからも作成可能です)




リポジトリ情報入力画面

- ボタンを押すと、下記のような入力画面になりますので、入力していきます。

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


1. リポジトリ名を入力


Owner  yasuki0206 / Repository name ✓

Great repository names are short and memorable. Need inspiration? How about [verbose-rotary-phone?](#)

Description (optional)

2. リポジトリの説明を入力する

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

3. リポジトリの種類を入力する

出典：コードライフ『GitHubに登録して、リポジトリを作成する』

入力情報説明

- repository nameにリポジトリの名前を入力します。
リポジトリ名はソースコードを格納するリポジトリのURLに含まれる。
- Descriptionにリポジトリの説明を入力します。
- リポジトリの種類を選択します。
- Publicだと公開となり、Webに公開されて誰でもアクセス可能になります。
- Privateだと非公開リポジトリになります。
- 公開せずに、自分だけで使いたい場合や、公開しないプロジェクトでチームで開発を進めていく場合などに選択します。
- Privateリポジトリにアクセスできる他のユーザーの追加ですが、リポジトリの作成後にリポジトリに招待する形でアクセス可能になります。

リポジトリ情報入力画面

- ボタンを押すと、下記のような入力画面になりますので、入力していきます。

The screenshot shows the GitHub repository creation interface. At the top, there are two radio buttons: 'Public' (selected) and 'Private'. Below this, the section 'Initialize this repository with:' contains a checkbox 'Add a README file' which is checked. A red arrow points from the text '4. READMEを作成するか選択する' to this checkbox. Below that is the 'Add .gitignore' section with a dropdown menu currently set to '.gitignore template: None'. A red arrow points from the text '5. .gitignoreを作成するか選択する' to this dropdown. The next section is 'Choose a license' with a dropdown menu set to 'License: None'. A red arrow points from the text '6. プロジェクトのライセンスを選択する' to this dropdown. At the bottom, there is a green button labeled 'Create repository' which is highlighted with a red rectangular box. A blue information box at the bottom states 'You are creating a public repository in your personal account.'

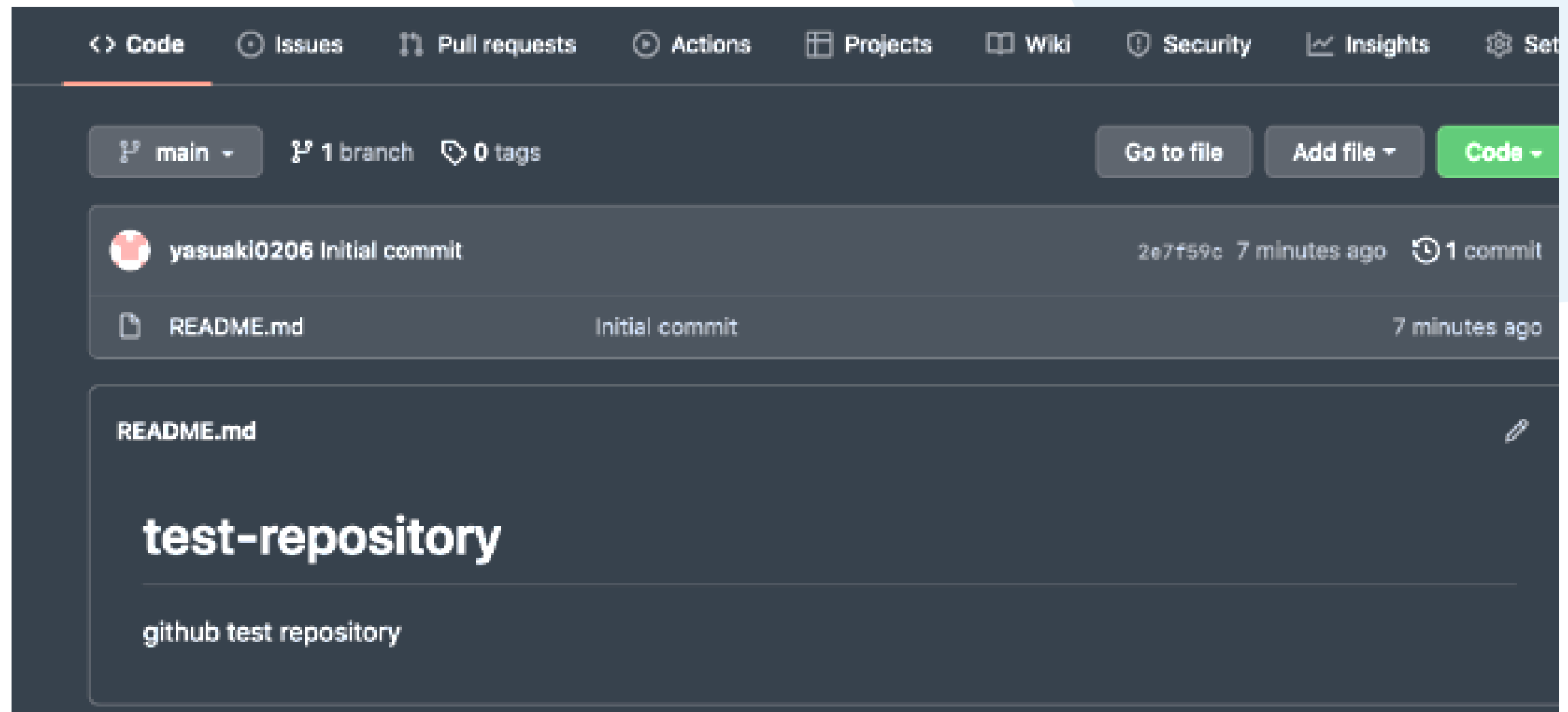
出典：コードライフ『GitHubに登録して、リポジトリを作成する』

入力情報説明

- リポジトリにREADMEを追加するかを選択します。
 - READMEはプロジェクトの概要や使い方などを書くためのファイルになります。
 - .mdというマークダウン形式のファイルで記述していきます。今回は追加するようにチェックを入れました。
 - .gitignoreファイルを追加するかを選択する
.gitignoreはgitにコミットして管理したくないファイルを記述するためのファイルになります。
 - セレクトボックスを選択すると、いろいろなフレームワークなどの初期テンプレートを選ぶことが可能です。
 - 今回は追加しないように「None」を選択しました。
-
- プロジェクトのライセンス選択
 - 公開するソースコードのライセンスを選択します。
 - 今回はとくに選択しないように「None」にしました。

リポジトリ作成完了

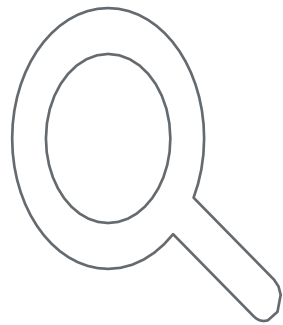
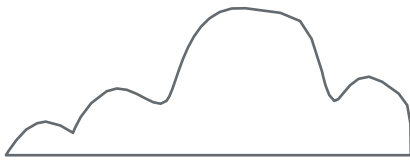
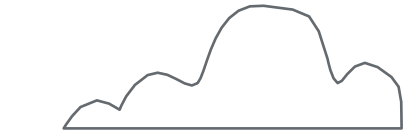
- 内容を確認すると、このようにリポジトリが作成されています。



出典：コードライフ『GitHubに登録して、リポジトリを作成する』



Q&A



Git の初期設定

- Macの場合は、「ターミナル」、Windowsの場合は、「Git Bash」を起動してGitの初期設定を行います。

- ユーザ情報を登録します。

```
git config --global user.name "XXXXXXXX"
```

GitHubに登録した
ユーザー名を書いて下さい

- メールアドレスを登録します。

```
git config --global user.email xxxx@xxxx
```

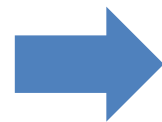
GitHubに登録した
メールアドレスを書いて下さい

- Gitの初期ブランチ名を変更します。

```
git config --global init.defaultBranch main
```

- 設定の確認をします。

```
git config --list
```



```
user.name=XXXXXXXX  
user.email=xxxx@xxxx  
init.defaultbranch=main
```

リポジトリのクローン

- リポジトリを**クローン**するコマンドは以下の通り：

```
git clone [url]
```

- 例えば、「grit」というリポジトリをクローンする場合は、以下のコマンドを使用します：

```
git clone git://github.com/schacon/grit.git
```

- このコマンドを実行すると、カレントディレクトリに「grit」というディレクトリが作成されます。その中には、ダウンロードしたすべてのバージョンレコードを保持する「.git」ファイルを含むディレクトリが作成されます。

- クローンしたプロジェクトディレクトリに独自の名前を定義したい場合は、上記のコマンドの最後に新しい名前を指定します。例えば、クローンしたプロジェクトを「mygrit」と名付けて保存したい場合は、以下のコマンドを実行：

```
git clone git://github.com/schacon/grit.git mygrit
```

- いまのプロジェクトの**最新バージョンを**、リモートリポジトリからローカルリポジトリに**取り込む**には、以下のコマンドを実行：

```
git pull
```

ブランチ管理

- **ブランチの一覧**を表示する基本的なコマンドは：

```
git branch
```

- 手動で**ブランチを作成**するには、「git branch [ブランチ名]」を実行：

```
git branch testing
```

- checkout コマンドを使って、変更**したいブランチに切り替**えます：

```
git checkout testing
```

- 「git checkout -b [ブランチ名]」 というコマンドで新しいブランチを作成し、すぐにそのブランチに切り替えてその中で操作できるようにすることも可能：

```
git checkout -b newtest
```

- **ブランチを削除**するには、「git branch -d [ブランチ名]」を実行：

```
git branch -d testing
```

リポジトリの表示と変更

- status コマンドで今のリポジトリの状態を確認できます：

```
git status
```

- add コマンドで、ステージングエリアにファイルを追加できます：

```
git add a.txt
```

- 「add .」ですべてのファイルを追加することもできます：

```
git add .
```

- ファイルを追加できたら、status コマンドでステージングエリアにファイルの変更を確認できます。

変更のコミットとプッシュ

- 変更を**コミット**するには、commit コマンドを：

```
git commit -m "Commit Message"
```

- 「-m」オプションを省略すると、Git はテキストエディターを開き、そこに複数行のコミットメッセージを書けます。
- コミットする前に、**必ず add コマンドを実行してください。**
- 現在のブランチをリモートリポジトリの対応するブランチに**プッシュ**（更新）するには：

```
git push
```


ブランチマージ

- push コマンドを実行すると、リモートリポジトリには変更がうまく反映されますが、master ブランチには反映されないなので、**マージ**を行う必要があります。
- 自分がプロジェクトのオーナーである場合、直接に merge コマンドを使ってローカルでマージし、マージした master ブランチをリモートリポジトリにプッシュすることができます。
- 自分がただのプロジェクトの貢献者で、マスター権限がない場合は、Git の「**pull request**」という機能を使ってプロジェクト管理者にマージを依頼することができます。

プルリクエストの流れ①

- GitHub上に作成したリポジトリをローカル環境にクローンします。

```
git clone https://github.com/ユーザー情報/practice.git
```

- クローンしたフォルダに移動します。

```
cd practice
```

- 「pullreq-test」という名前でブランチを新規作成します。

```
git checkout -b pullreq-test
```

プルリクエストの流れ②

- テキストファイル「test.txt(中身はHelloと記載)をaddします。

```
git add .
```

- コミットします。

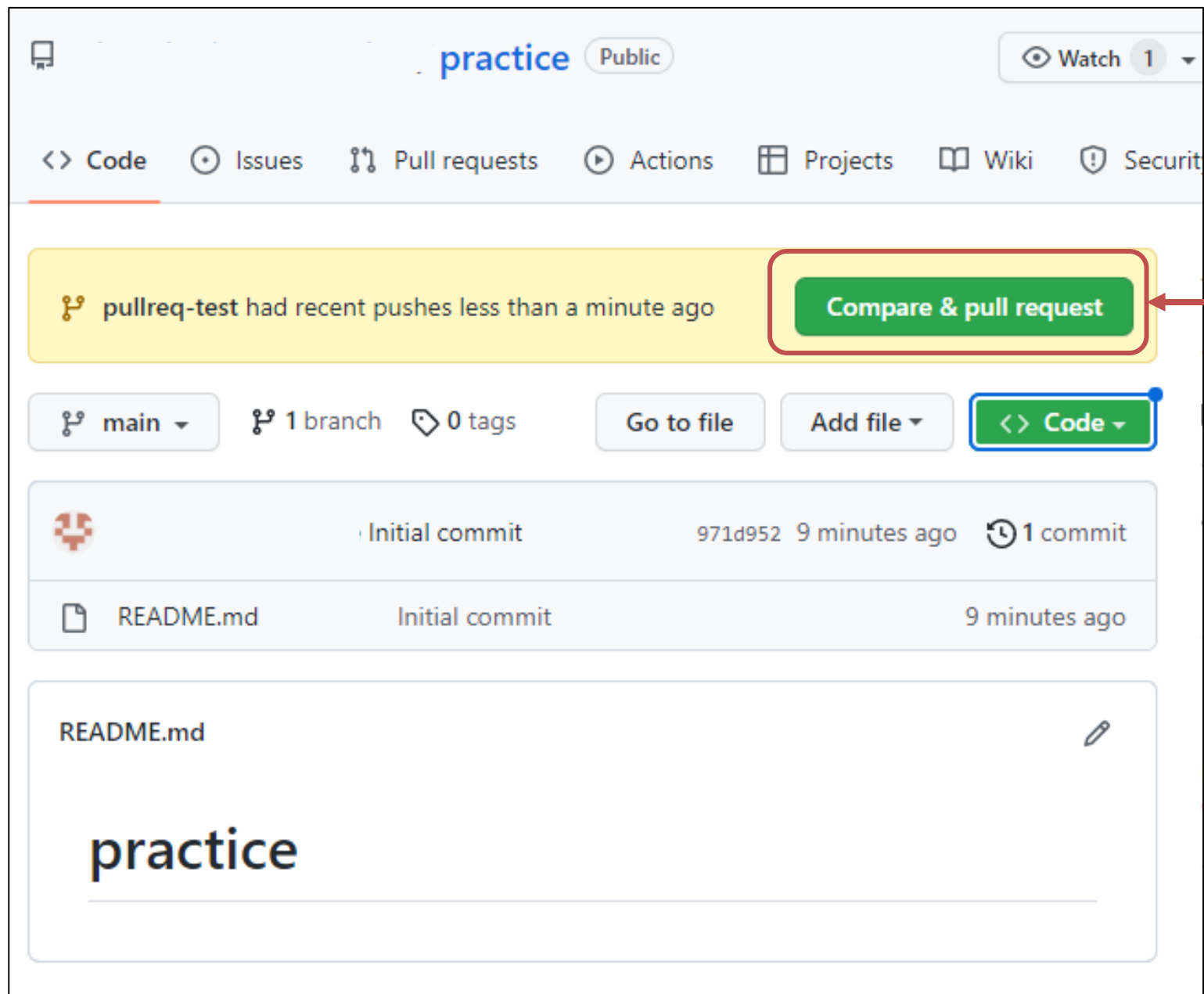
```
git commit -m "add textfile"
```

- 作成したブランチにpushします。

```
git push origin pullreq-test
```

プルリクエストの流れ③

- ここからは、GitHub上でプルリクエストの作成を行っていきます。




「**Compare & pull request**」のボタンが表示されるのでクリックしてプルリクエストの作成を進めていきます。

プルリクエストの流れ④

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).




base: main ▼

 ←

compare: pullreq-test ▼

✓ **Able to merge.** These branches can be automatically merged.



add textfile


Write

Preview

H B I ≡ <> 🔗 ≡ ≡ ≡ @ ↗ ↶

プルリクエストのテストです。

プルリクエストのコメント内容を記述します。

Attach files by dragging & dropping, selecting or pasting them. 

Create pull request ▼

Reviewers

No reviews

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

プルリクエストの流れ⑤

①プルリクエストを確認してもらいたい開発者を選択します。

Reviewers

No reviews

Assignees

Assign up to 10 people to this issue

Type or choose a user

Suggestions

Milestone

No milestone

Create pull request

③「Create pull request」をクリックします。

②実際の開発の現場では、確認する人を指定し、設定をしますが、今回は、指定をしないで進めていきます。

プルリクエストの流れ⑥

The screenshot shows a GitHub pull request titled "add textfile #1". At the top, it says "wants to merge 1 commit into main from pullreq-test". Below this, there are tabs for Conversation (0), Commits (1), Checks (0), and Files changed (1). A comment from a collaborator says "プルリクエストのテストです。" (This is a test for the pull request.) and shows a commit "add textfile" with hash "f0df672". A green box with a checkmark states "This branch has no conflicts with the base branch" and "Merging can be performed automatically." Below this is a green button labeled "Merge pull request" with a dropdown arrow. At the bottom, there is a comment section with a "Write" tab, a "Preview" tab, and a "Comment" button. The "Close pull request" button is also visible.

変更箇所を確認できます。

プルリクエストの変更に問題がなければ
「Merge pull request」
をクリック

プルリクエストの流れ⑦

add textfile #1

[Open](#) wants to merge 1 commit into `main` from `pullreq-test`

Conversation 0 Commits 1 Checks 0 Files changed 1

commented 6 minutes ago Collaborator

プルリクエストのテストです。

`add textfile` f0df672

Add more commits by pushing to the `pullreq-test` branch on `/practice`.

Merge pull request #1 from `/pullreq-test`

add textfile

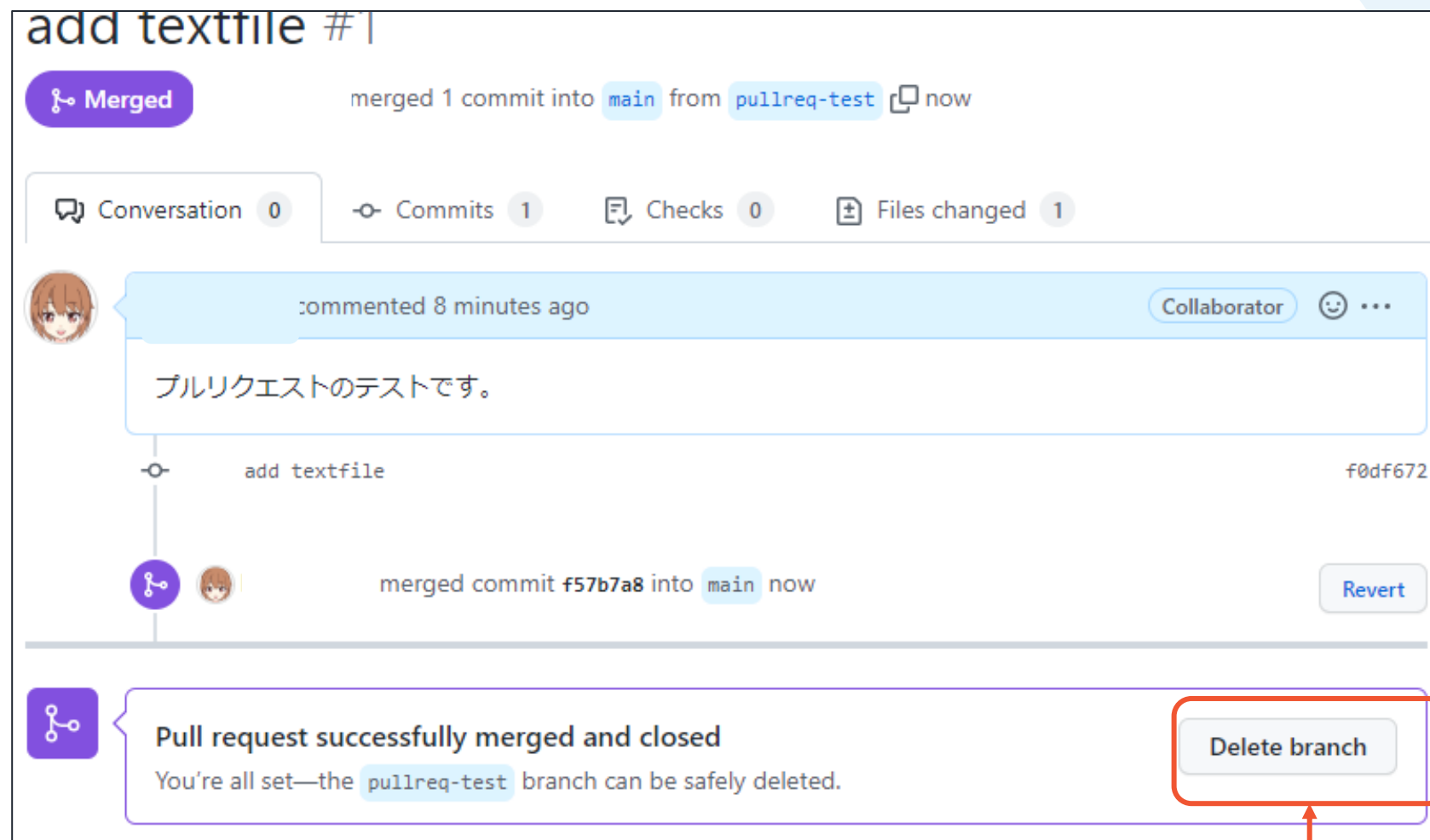
This commit will be authored by 58722094+ `i@users.noreply.github.com`

Confirm merge Cancel

「**Confirm merge**」をクリック

プルリクエストの流れ⑧

- Merge成功となり、以上がプルリクエストの流れになります。チーム開発では必須の考え方となるため、ここでしっかりと理解をしてください。



ブランチが不要ない場合はここから削除ができます。

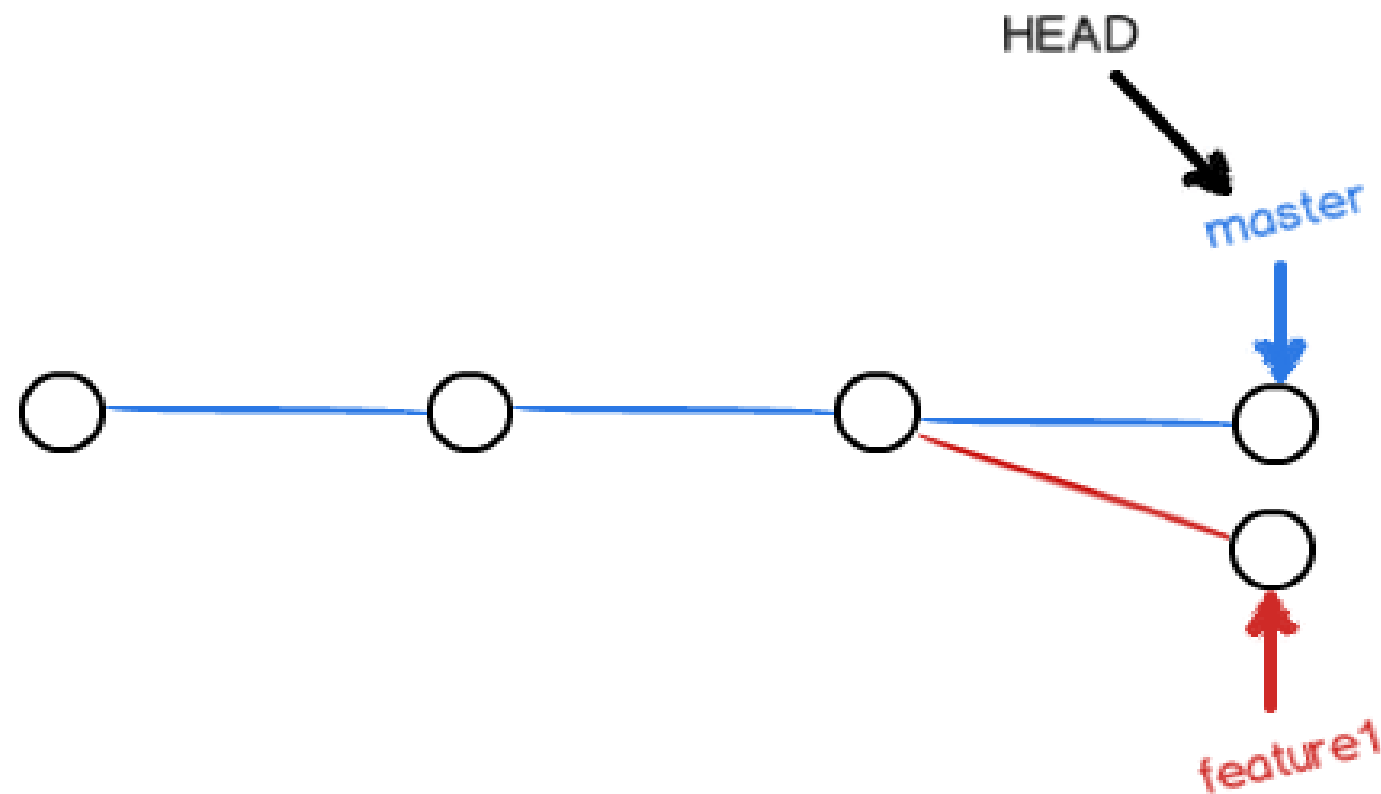
マージのコマンド

- まずブランチをターゲットのブランチに切り替え、次にマージしたいソースのブランチをマージします:

```
git checkout master  
git merge test
```

- この二つのコマンドで、test ブランチを master にマージしました。

Git 衝突



- 上図のように、feature1 を master にマージしたいときに、feature1 の**変更している間に**他の誰かが master に変更を加えた可能性があります。
- その後で master ブランチにマージしようとする、ファイルの変更によって**衝突**[Conflict]が発生する可能性があります（たとえば、両方が同じ行のコードを変更した）。

衝突の発生

- 衝突の例（Mac）：

```
$ git merge feature1
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.
```

- Git は「readme.txt に衝突があるので、コミットする前に衝突を手動で修正しろ」と指示しました。
- status コマンドで、Git は衝突があるファイルについての詳細を教えてください。

衝突の表示

- Git は「<<<<<<<<」「=====」「>>>>>>>>」のような特殊符号を使って、競合するファイル中の異なるブランチの内容をマークする。これにより、コンフリクトを含むファイルを開いて手動で解決することができます：

衝突があるコードの、現在のブランチにあるバージョン

<<<<<<< HEAD

Creating a new branch is quick & simple.

=====

Creating a new branch is quick AND simple.

>>>>>>> feature1

衝突があるコードの、目標のブランチにあるバージョン

衝突の解消

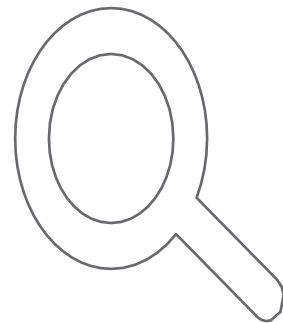
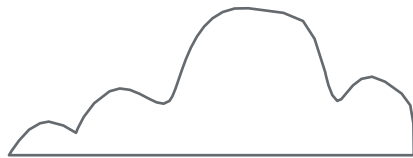
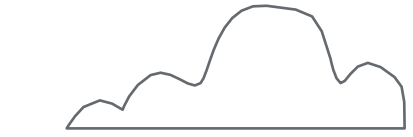
- <<<<<<<<、=====、>>>>>>>> がマークした内容を手動で以下のように修正して、保存します：

Creating a new branch is quick AND simple.

- 上記のように**ブランチの 1 つのみ**への変更を保持して、<<<<<<<<、=====、>>>>>>>> の行も完全に削除しなければなりません。
- すべてのファイルの衝突を解決したら、各ファイルで add コマンドを使用してステージングエリアに追加します。衝突するファイルがステージされると、Git はそれらを解決済みとしてマークします。



Q&A



目次

- 1 Git
- 2 Git の利用
- 3 **Markdown**
- 4 SVNの利用



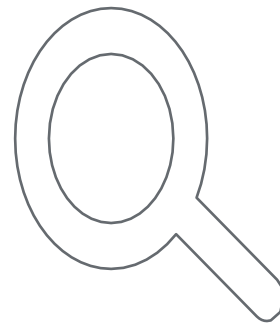
Markdown

- **Markdown** 言語は、2004 年に John Gruber 氏によって作られた、読みやすく、書きやすいプレーンテキスト形式で文書を書くことができる軽量のマークアップ言語であります。
- Markdown で書かれた文書は、HTML、Word 文書、画像、PDF、Epub などのさまざまな形式に変換できます。
- Markdown ファイルの拡張子は「.markdown」 または「.md」になります。
- 書き方については、こちらのリンクを参考してください：

 <https://qiita.com/kamorits/items/6f342da395ad57468ae3>



Q&A

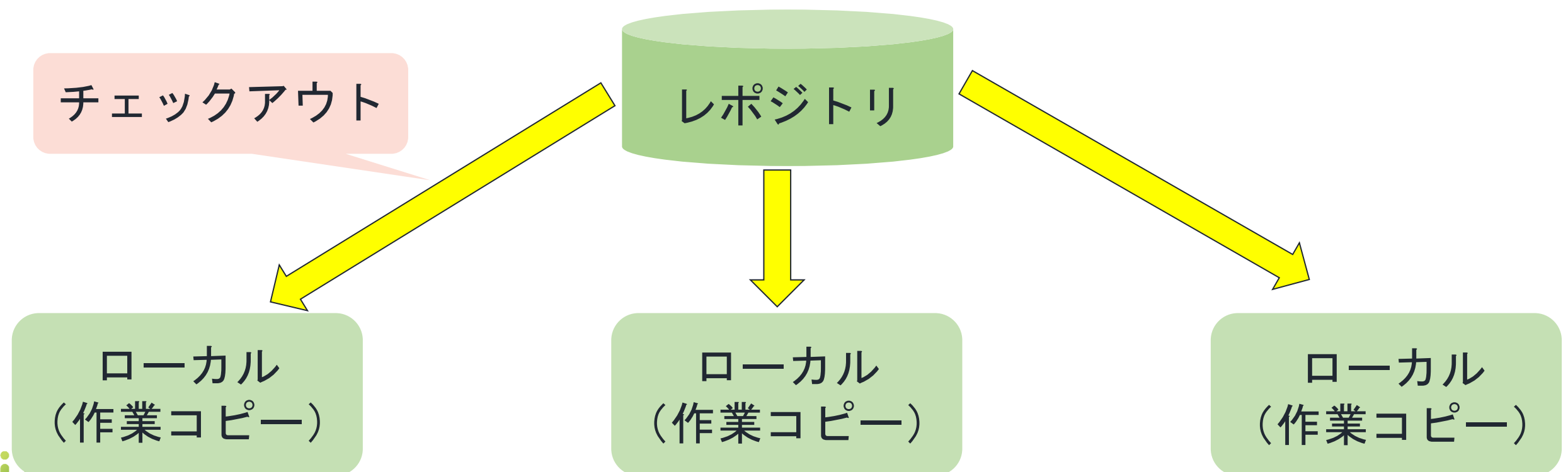


目次

- 1 Git
- 2 Git の利用
- 3 Markdown
- 4 SVNの利用

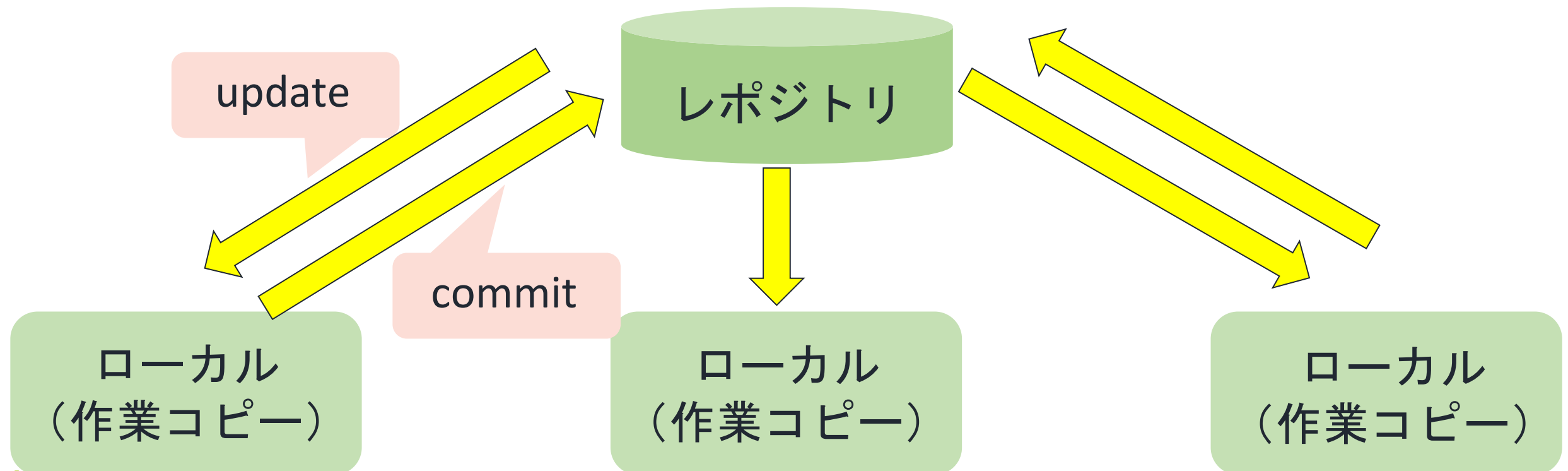
SVNのレポジトリと基本操作

- レポジトリ (repository) は貯蔵庫という意味です。
プログラムのソースコードなどを配置し、バージョン管理を行う。
- 作業者は、「チェックアウト」で「レポジトリ」のファイル等を一式を「ローカル(作業コピー)」にダウンロードする。
「チェックアウト」は、最初に1回のみ行う。



アップデート(update)とコミット(commit)

- 作業者は、「アップデート」で「レポジトリ」の変更されたファイルを「ローカル(作業コピー)」に反映する。
- 作業者は、「コミット」で「ローカル(作業コピー)」にある変更したファイルを「レポジトリ」に反映する。
- コミットされたファイルはバージョンで管理され、変更した箇所がわかるようになっている。



アイコン

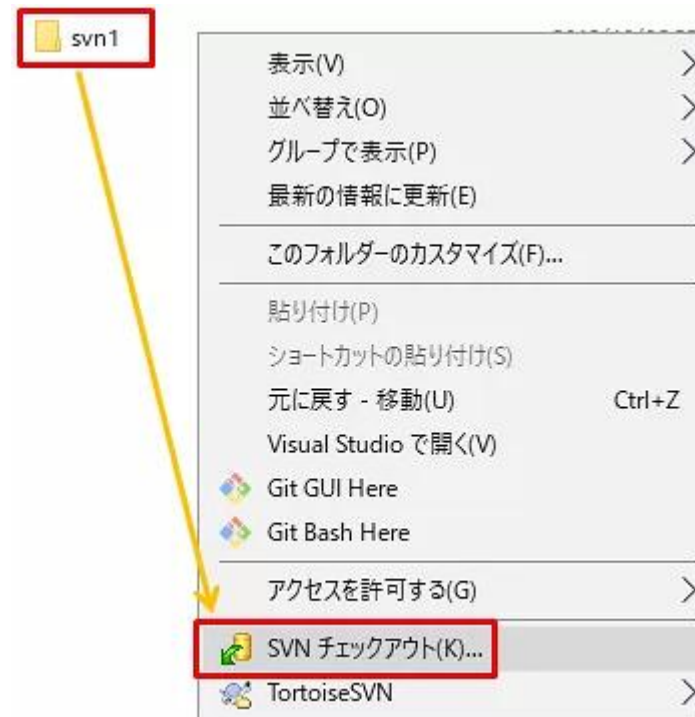
- レポジトリとローカル(作業コピー)で内容が同じ場合は、緑のアイコンです。
- ローカル(作業コピー)でファイルの内容を変更すると、赤のアイコンになります。



出典 : [ITSakura『TortoiseSVNの使い方』](#)

リポジトリからファイルをダウンロードする (チェックアウト)

- 任意の場所でフォルダを作成します。
- このサンプルでは次のフォルダを作成しました。
D:¥Test1¥svn1
- 対象のフォルダで右クリックして「SVNチェックアウト」をクリックします。



出典 : ITSakura 『TortoiseSVNの使い方』

チェックアウトの設定

- 「リポジトリのURL」を入力し「チェックアウト先のディレクトリ」を確認しOKボタンを押します。

チェックアウト

リポジトリ

リポジトリのURL:

https://desktop-bofets8/svn/svn1/project1

チェックアウト先のディレクトリ(D):

D:\Test1\svn1

☐ 複製、独立した作業コピー(I)

チェックアウトする深さ(D):

再帰的

☐ 外部参照を除外する(X) 項目を選択(C)...

リビジョン

☒ 最新リビジョン(HEAD)

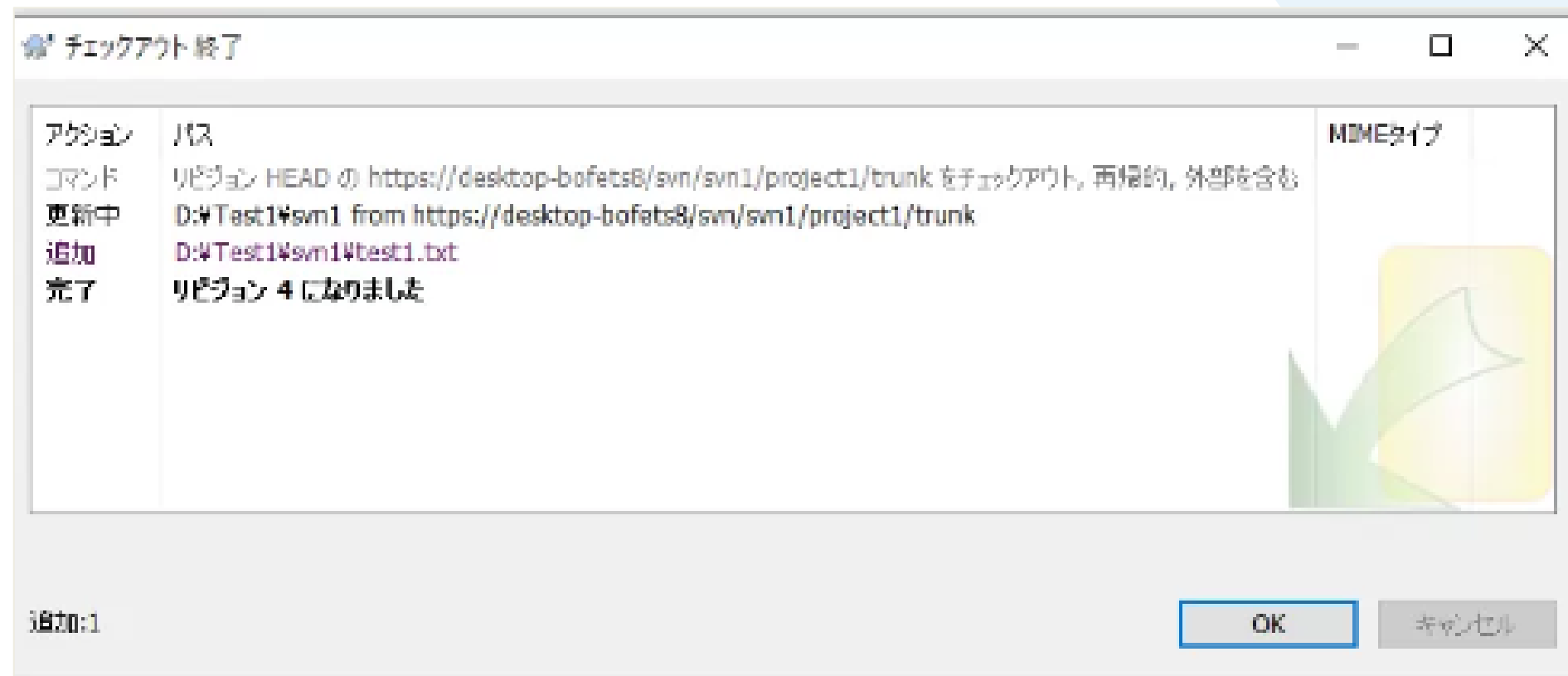
☐ リビジョン(R) ログを表示(L)

OK キャンセル ヘルプ

出典：ITSakura『TortoiseSVNの使い方』

ファイルのダウンロード





- ファイルのダウンロードが始まります。終了するとOKボタンが押せるようになります。



出典 : ITSakura 『TortoiseSVNの使い方』

チェックアウト完了

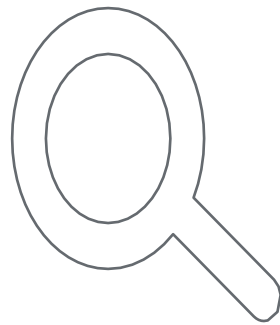
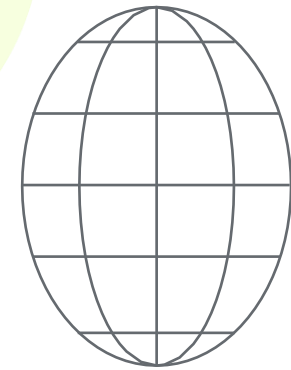
- 以下は、チェックアウト完了後のフォルダ (D:¥Test1¥svn1の中)のイメージです。

名前	種類
 .svn	ファイル
 branches	ファイル
 tags	ファイル
 trunk	ファイル

出典 : [ITSakura『TortoiseSVNの使い方』](#)



Q&A

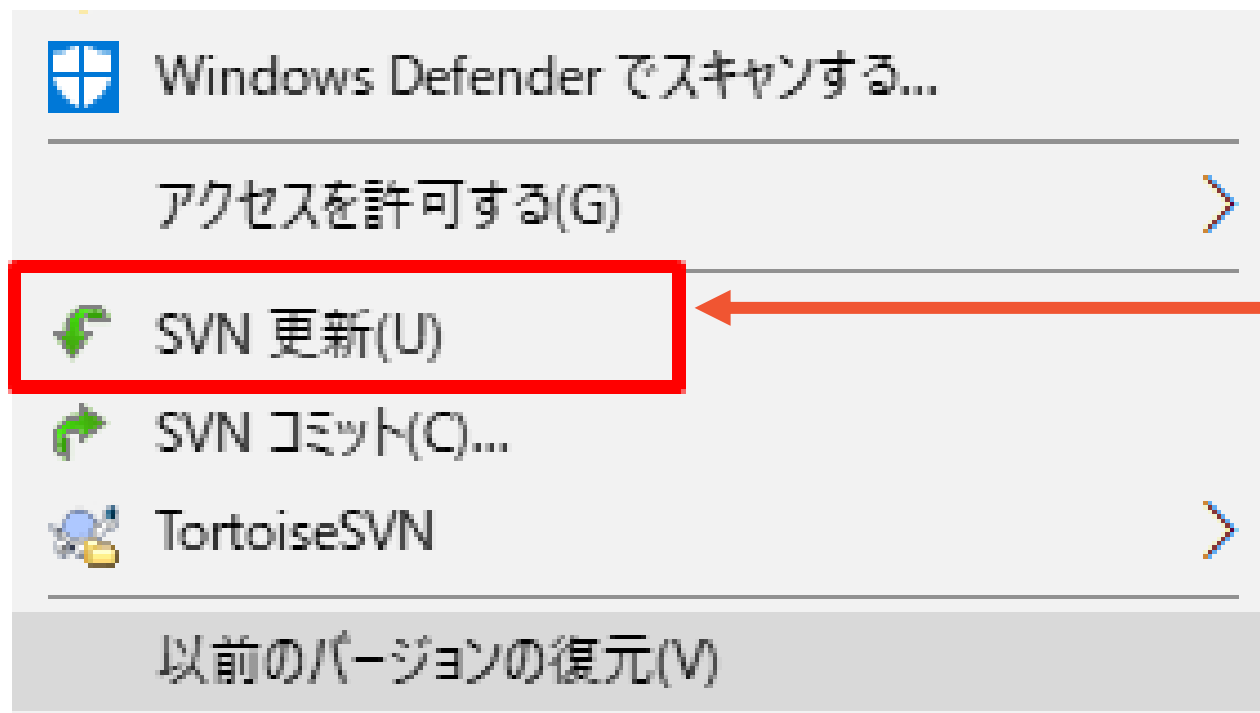


trunkとbranchesとtags

- trunkは、**幹**という意味です。**最新のバージョンを格納**します。
- branchesは、**枝**という意味です。**trunkとは別のバージョン管理**です始める時に使用します。
- tagsは、**ある時点のものを保存**します。リリースしたときのバージョンなどを置いておきます。
- .svnは隠しフォルダです。
- ローカル(作業コピー)のフォルダー式(例：
D:¥Test1¥svn1)が不要になったときはエクスプローラで
削除して大丈夫です。レポジトリは削除されません。

リポジトリからローカルに反映する (アップデート)

- チェックアウト後のローカルの更新は、アップデート(SVN更新)で行います。
- 対象のフォルダで右クリックし、「SVN更新」を行うとローカルが更新されます。

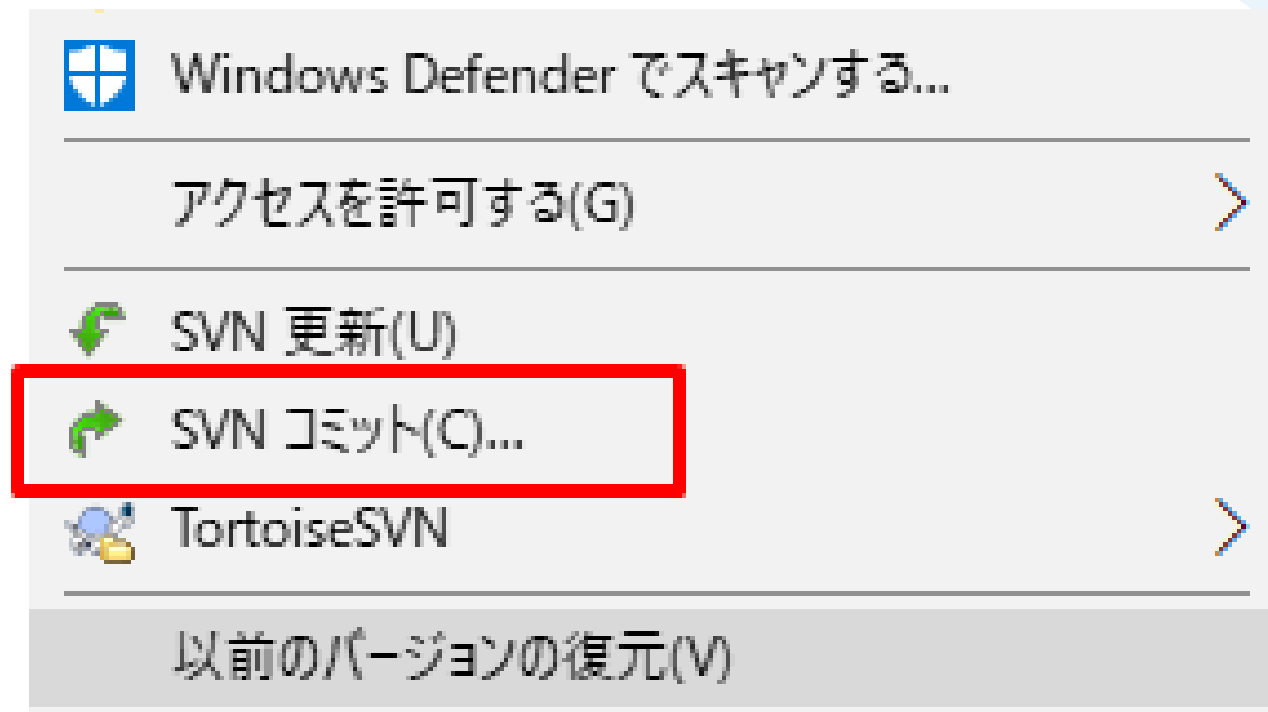


ローカルのファイルを削除してしまっても再度「SVN更新」をすればレポジトリからファイルをダウンロードできる。

出典：ITSakura『TortoiseSVNの使い方』

ローカルからリポジトリに反映する (コミット)

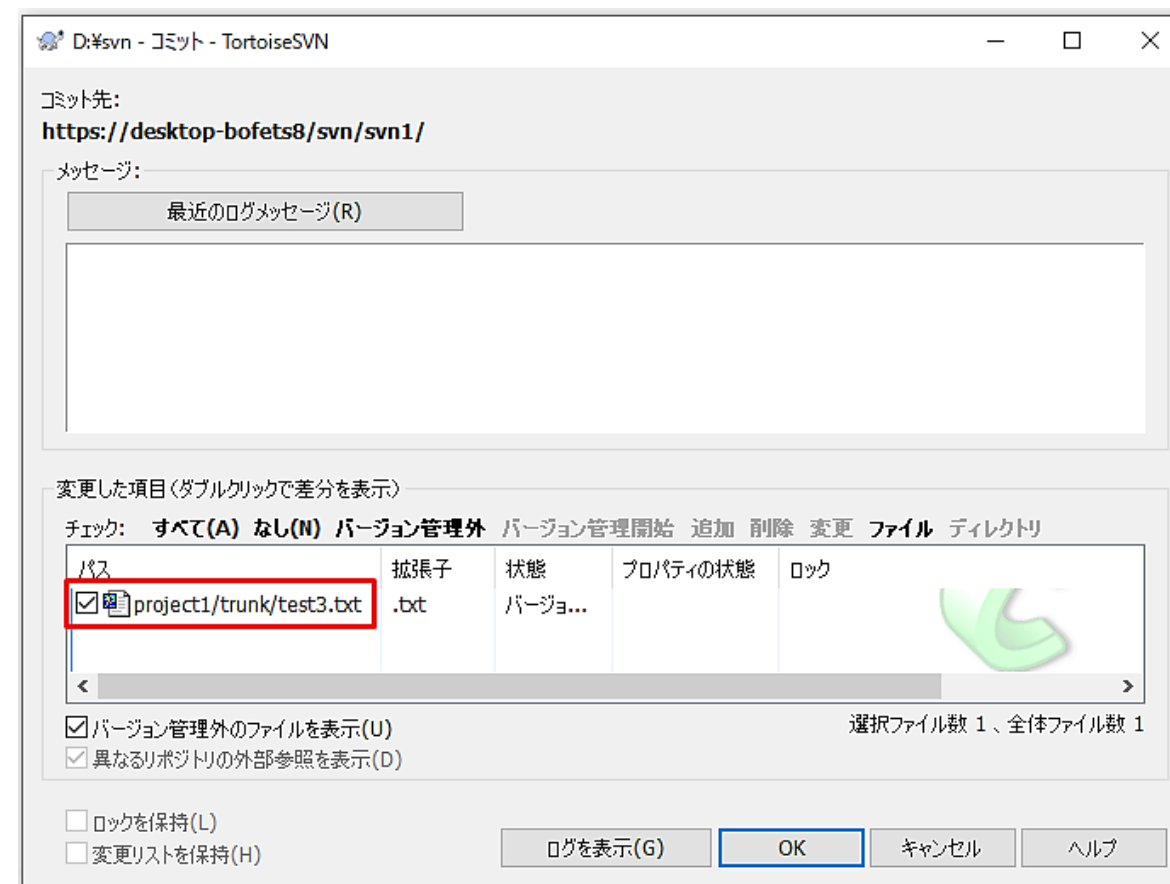
- ローカルで変更した結果をレポジトリに反映します。
- 対象のフォルダで右クリックし、「SVNコミット」をクリックします



出典：ITSakura『TortoiseSVNの使い方』

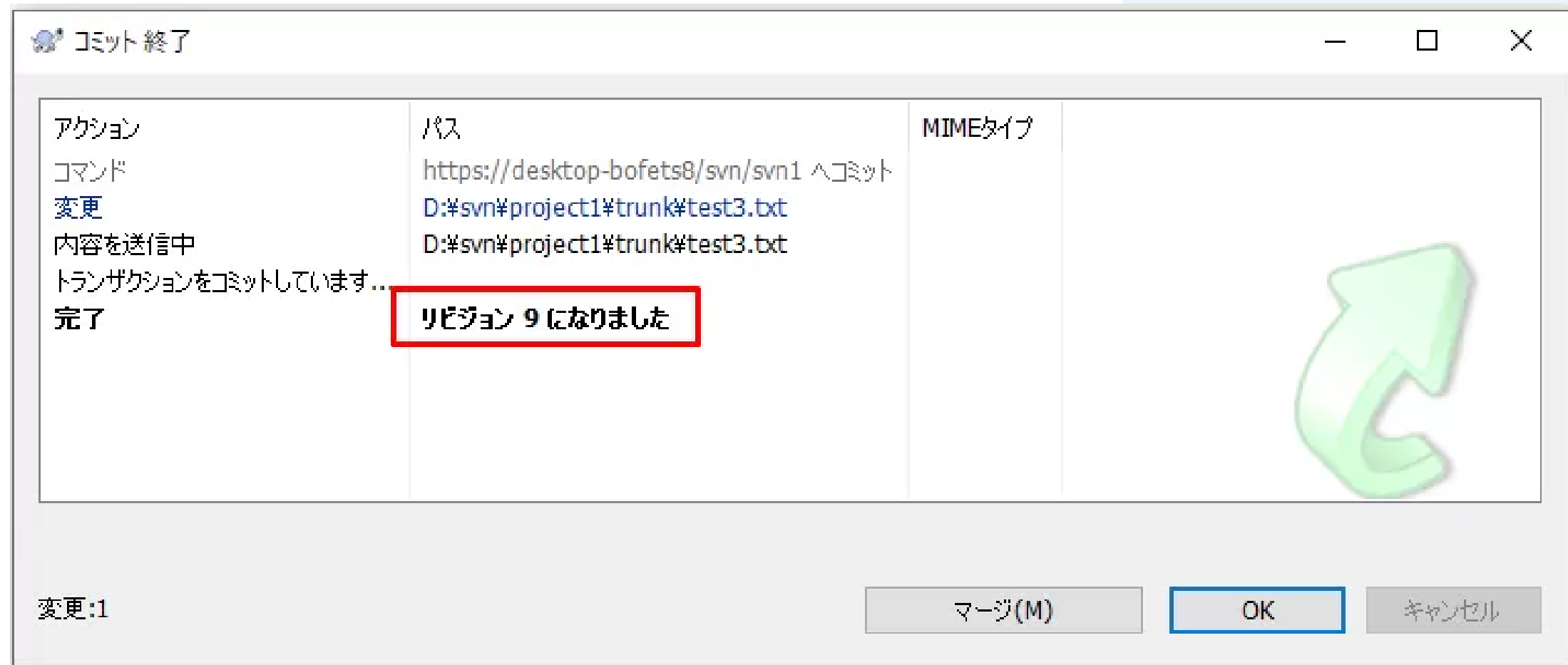
ローカルからリポジトリに反映する (コミット)

- コミットする対象のファイルの左端にチェックを入れて、OKボタンを押すとコミットされる。
- メッセージの入力欄には、何のコミットかわかるように変更した箇所の説明を記述する。
- 例：〇〇プログラムの〇〇項目に必須チェックを追加



コミット完了

- 以下はコミットが完了した時の図です。リビジョン（改定）の数値が1つ上がります。



SVN参考記事

- 以上が、基本操作となります。

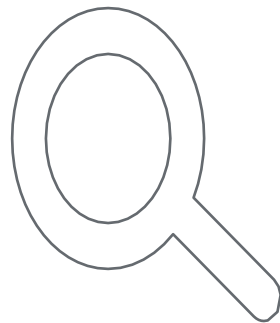
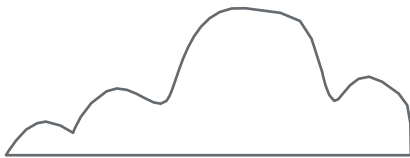
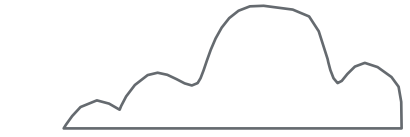
参考記事を見て、自分で操作方法が理解できるようにしてください。

<https://itsakura.com/tool-tortoisesvn>

- 自分で記事を調べてやり方を覚えることも大切です。
「SVN使い方」と検索し、自分にあったわかりやすい記事を見つけられるようにしましょう。



Q&A



まとめ

Sum Up



1. Git の基本原理 :

- ① ローカルリポジトリとリモートリポジトリの概念。
- ② リポジトリにあるファイルの 3 つの状態。
- ③ ブランチ・マージ。

2. Git のコマンドラインでの基本操作 :

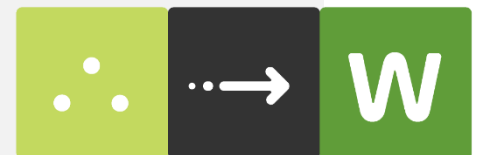
- ① リポジトリの基本操作 : 状態のチェック、クローン、ステージ、コミット、プッシュ。
- ② 衝突解消の仕方。

3. Markdown の基本的な構文。

4. SVNの基本用語と基本操作

Thank you!

From Seeds to Woodland — Shape Your Future.



Shape Your Future