ASTERIA Warp

バージョン管理連携機能

（Git）

第 1.1 版：2026 年 1 月

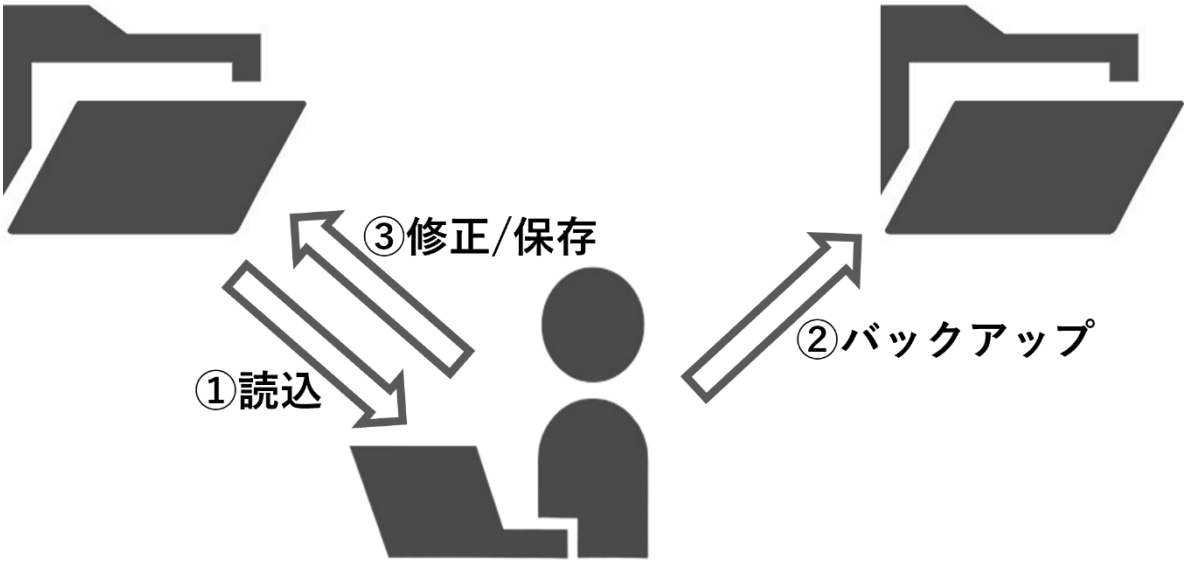
ASTERIA Warp では、オープンソースのバージョン管理システムである Git と連携することで、作成したフローや関連するファイルをバージョン管理することができます。
本書では、バージョン管理の概要、および ASTERIA Warp 上でのバージョン管理連携機能を使用した開発手順についてご紹介します。

なお、本書の内容は ASTERIA Warp 2506 以降に対応しております。

1.バージョン管理とは

バージョン管理は、「いつ」「誰が」「何を」「どう」変更したのかを記録し、必要なときに過去の状態へ戻せるようにする仕組みです。ソフトウェア開発のソースコード管理として広く使われていますが、対象はプログラムに限りません。日々更新される Excel や PowerPoint などの業務ファイルに対してもバージョン管理を使用する価値があります。

例えば、多くの人が修正前のファイルを別名でコピーして「バックアップ」として残した経験があるでしょう。



これは手作業のバージョン管理と言えます。しかしこの方法には、次のような課題があります。

・ファイル数や版数が増えるほど、管理や検索が煩雑になる
・どこが変わったのかファイルの差分を簡単に確認できない
・変更を確認するために古い版から順に開いて確かめるなど、無駄な手間が発生する

こうした問題を解決するのがバージョン管理システムです。バージョン管理システムを使うと、更新のたびにファイルの内容と、その修正の意図をコメントとして記録できます。そして、後日いつでも、日時やコメントを手がかりに必要なバージョンのファイルに戻すこともできます。また、変更点の差分表示や、重要な時点へのタグ付けも可能です。

バージョン管理は一人で使っても便利ですが、複数人で同じファイルを更新する場面で特に真価を発揮します。典型例がソフトウェア開発です。バージョン管理システムを使えば、

・「いつ」「誰が」「どのような修正を行ったか」を全て記録できる
・作業ごとに分岐を作成し、各分岐での作業が終わってから統合できる
・複数人が同じ箇所に対して異なる修正をした際でも競合箇所がわかるので安全にまとめられる

といった効果により品質と生産性を両立できます。
以降の章では、フローの開発でバージョン管理システムである Git を活用するための基本を解説していきます。

2.Git

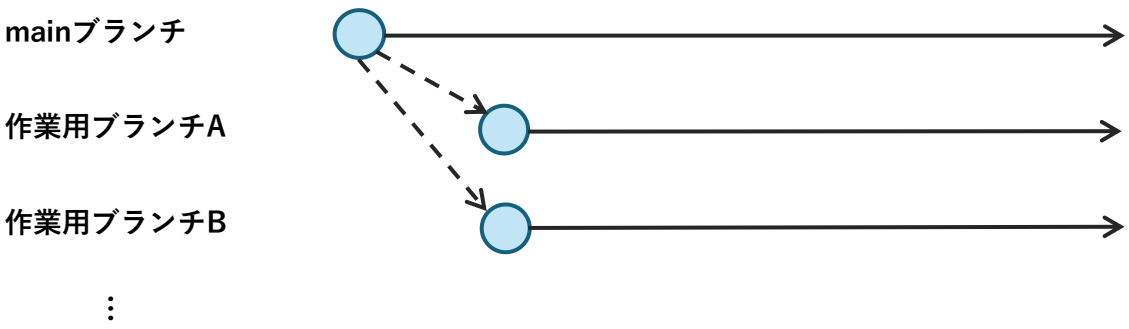
Git は、ソフトウェア開発におけるバージョン管理システムの一つであり、ソースコードの変更履歴を記録・管理するために使われるツールです。Git では、プロジェクトファイルなどを保存する保管庫となる「リポジトリ」を作成してその中で履歴管理や更新作業を行います。
通常、Git ではブランチと呼ばれる並行管理が可能な履歴を作成し、そのブランチで作業します。ASTERIA Warp で Git を利用する場合はフローデザイナーから GUI で全ての操作ができます。本章では、Git の概要について説明し、次章では実際に ASTERIA Warp と連携するための代表的なセットアップ方法を 2 つ紹介します。

2.1.リポジトリ

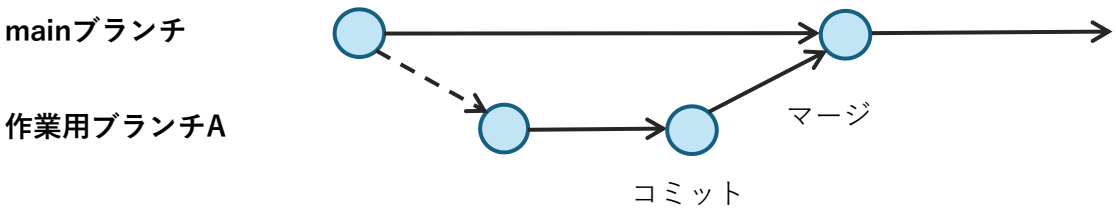
リポジトリとは管理対象の全てのファイルとその履歴、コメントなどの保管庫となるものです。通常のファイルシステムと同じように複数のファイルやフォルダーをツリー構造で保持します。リポジトリにはローカルリポジトリとリモートリポジトリがありますが、ASTERIA Warp ではこれらの違いを意識せずに使うことができます。

2.2.ブランチ

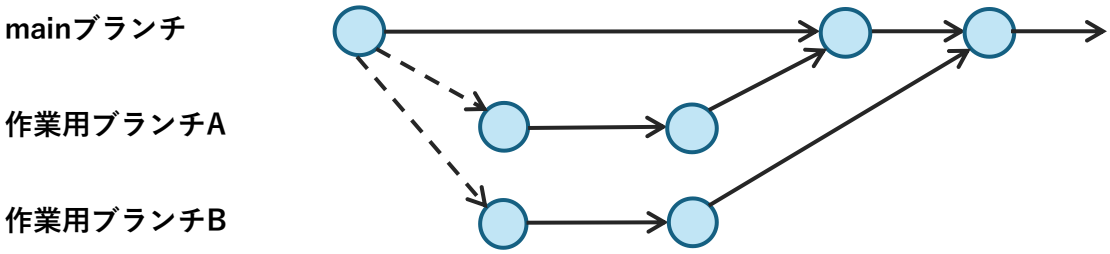
ブランチとは、現在の履歴を分岐元とし、並行して管理可能な履歴を複数作成する仕組みです。



ブランチでの作業後は、コミットによって他のブランチに影響を与えずに、更新内容を保存できます。さらに、各ブランチでコミットした更新内容は、マージによって他のブランチへ反映できます。これにより、複数のブランチで管理していたファイルを 1 つにまとめることができます。



また、同じファイルを複数のブランチで編集していても、更新箇所が重なっていなければ、それぞれの変更をマージさせることが可能です。そのため、各作業用ブランチから分岐元のブランチ、ここでは main ブランチへマージすれば、main ブランチは各ブランチの全ての更新内容を含むブランチとなります。



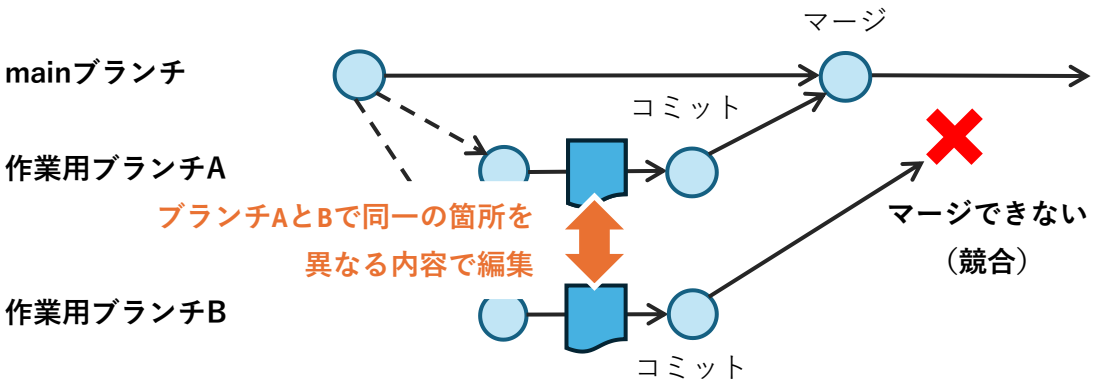
2.3.リビジョン

リビジョンは、リポジトリの特定時点におけるファイルやディレクトリ構造のスナップショットを表します。新しいコミットが作成されるたびに、新しいリビジョンが追加され、履歴が更新されます。つまり、リビジョンとはある特定のコミットのことを指しています。この時、リビジョンには 4a7d1ed414 というような形式の値が生成され割り当てられます。

2.4.競合、マージ、ロック

main ブランチとは別に 1 つのブランチだけで作業しているのであれば、その作業ブランチを main ブランチにマージする時には、作業ブランチの更新分だけが main ブランチにマージされます。しかし複数のブランチから main ブランチへマージする場合は、以下のような状況が発生することがあります。

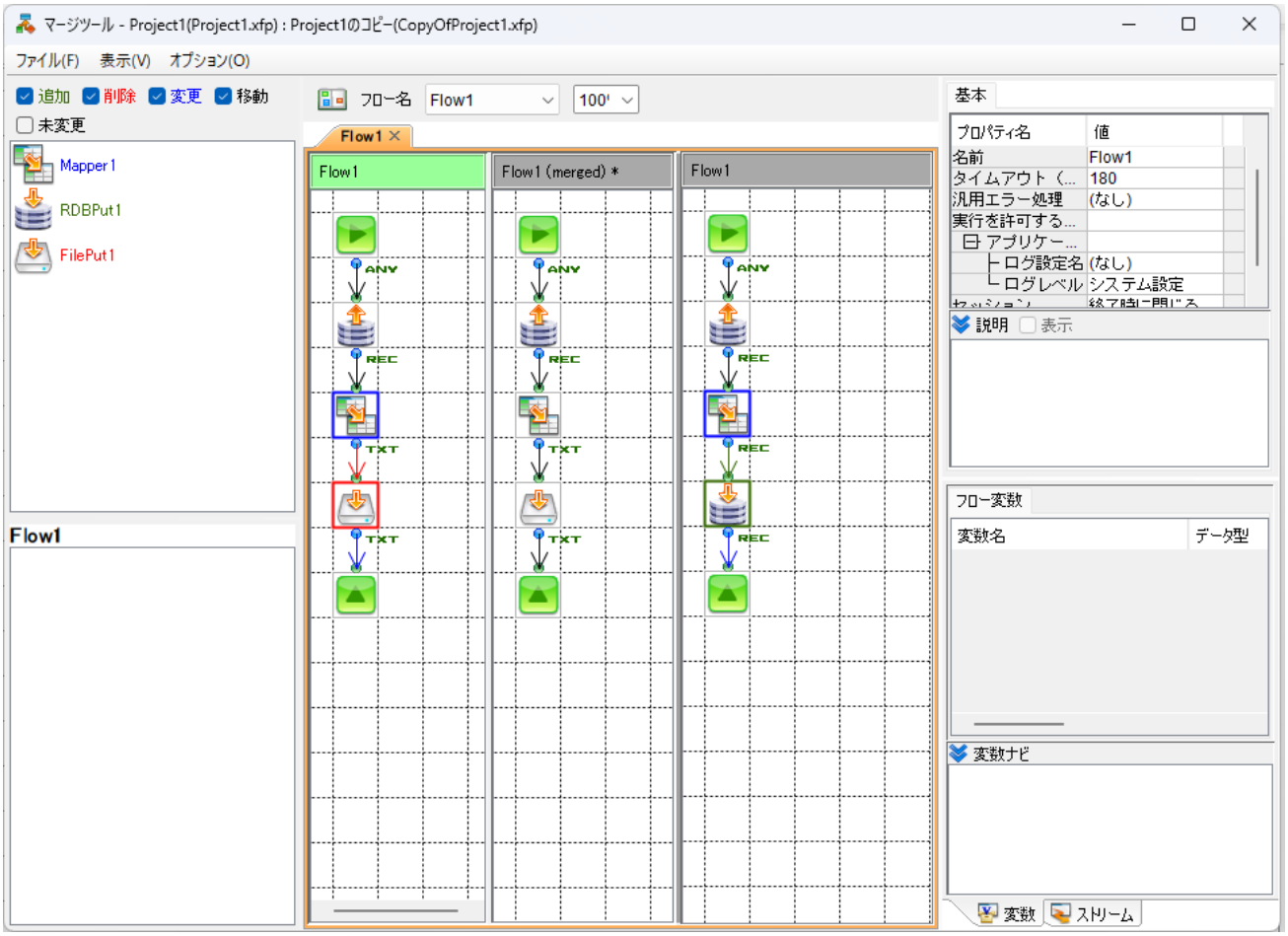
① ファイル 1 の同一箇所をブランチ A とブランチ B で同時に編集します。
② ブランチ A へファイル 1 をコミットし、ブランチ A を main ブランチへマージします。
③ ブランチ B へファイル 1 をコミットし、ブランチ B を main ブランチへマージしようとします。しかし main ブランチのファイル 1 にはブランチ A の更新がすでにマージ済みとなっています。ブランチ B の更新をそのままマージするとどうなるでしょうか？



最後の段階でブランチ B の変更がマージによりそのまま上書きできてしまうと、ブランチ A の編集結果は誰も気がつかないうちに消えてしまうことになります。実際の Git 操作では、ブランチ B のマージは失敗します。このような状況を「競合」と呼びます。
複数のブランチで、同じファイルの同じ箇所を編集すると、Git では競合が発生する可能性があります。一方、編集箇所が異なる場合は、ブランチの節で述べた通り、Git が自動的にマージを行うため、両方のブランチの更新内容を問題なくマージできます。

2.4.1.競合の解決

競合が発生した際、ASTERIA Warp では GUI のマージツールで修正箇所の比較やマージを行い、競合を解決できます。 この際は画像のように、競合が発生したフローを左右で、マージした結果を中央で確認できます。



2.5.スタッシュ

前節で紹介した競合ですが、解決できない場合があります。代表的な例は、1 つのブランチを複数のユーザーで使用した場合です。このケースの流れは以下の通りです。

1.ユーザーA で main ブランチのプロジェクト a を編集する
2.ユーザーB で main ブランチのプロジェクト b を編集する
3.ユーザーA でコミットする
4.ユーザーB でコミットしようとすると競合が発生しエラーになるが、解決できない

その結果、ユーザーB はコミットもブランチの移動もできなくなってしまいます。このような場合の解決策として、スタッシュがあります。

スタッシュを使うことで、作業の状況を一時的に退避させることができます。先ほど紹介したケースの場合、続けて以下のようにスタッシュを使うと、ユーザーB でコミットできます。

5.ユーザーB でスタッシュを作成して退避する。
6.ユーザーB で main ブランチの最新版を取得する（ユーザーA がコミットしたリビジョンを取得）
7.ユーザーB でスタッシュを適用する
8.ユーザーB でコミットする

ASTERIA Warp ではフローデザイナーからスタッシュが使えるため、上記で紹介したような状況になってもスムーズに対処できます。

3.Git のセットアップ

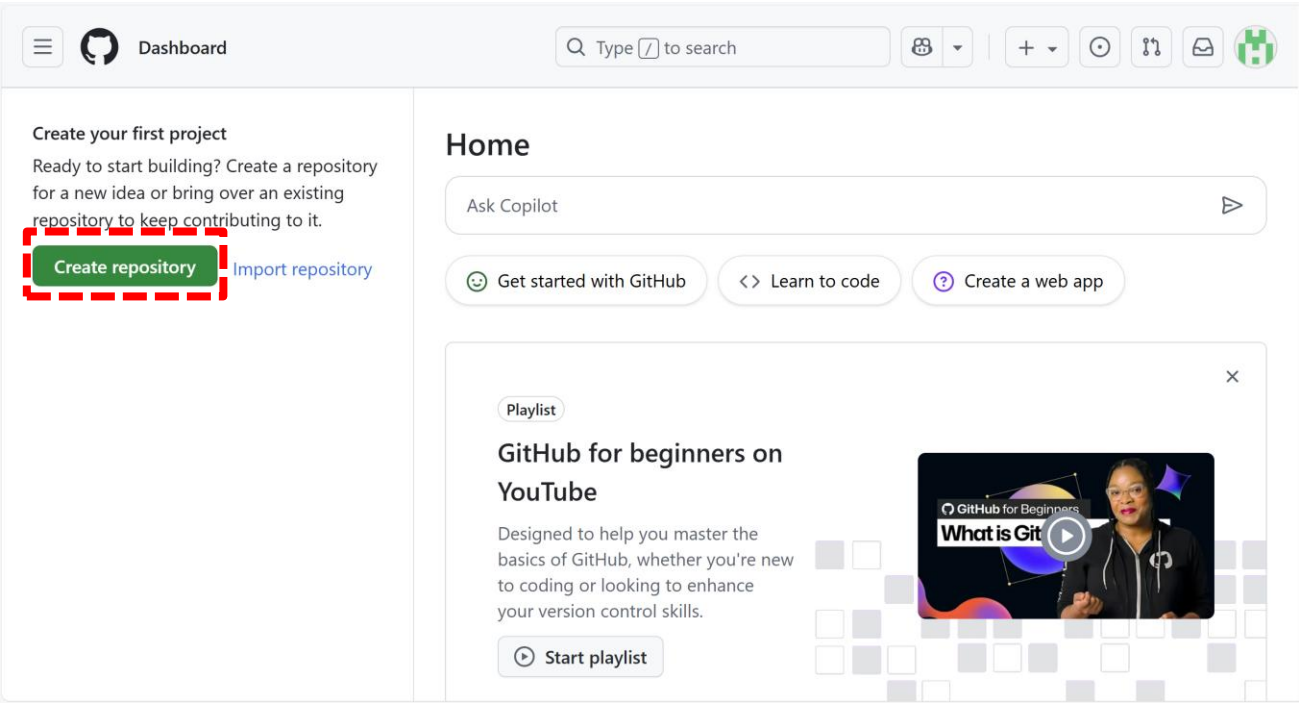
それでは、次は実際に触ってみましょう。今回は GitHub と GitLab での手順を紹介します。なお、本章での紹介手順や画像は執筆時点のものであり、今後画面のレイアウト等は変更される可能性があります。

3.1.GitHub でのセットアップ（共通）

GitHub の場合、認証方法として「API トークン認証」と「SSH キー認証」が選択できるため、それらの共通箇所をこの節で紹介し、それぞれの認証方法固有の内容は次節以降で紹介します。

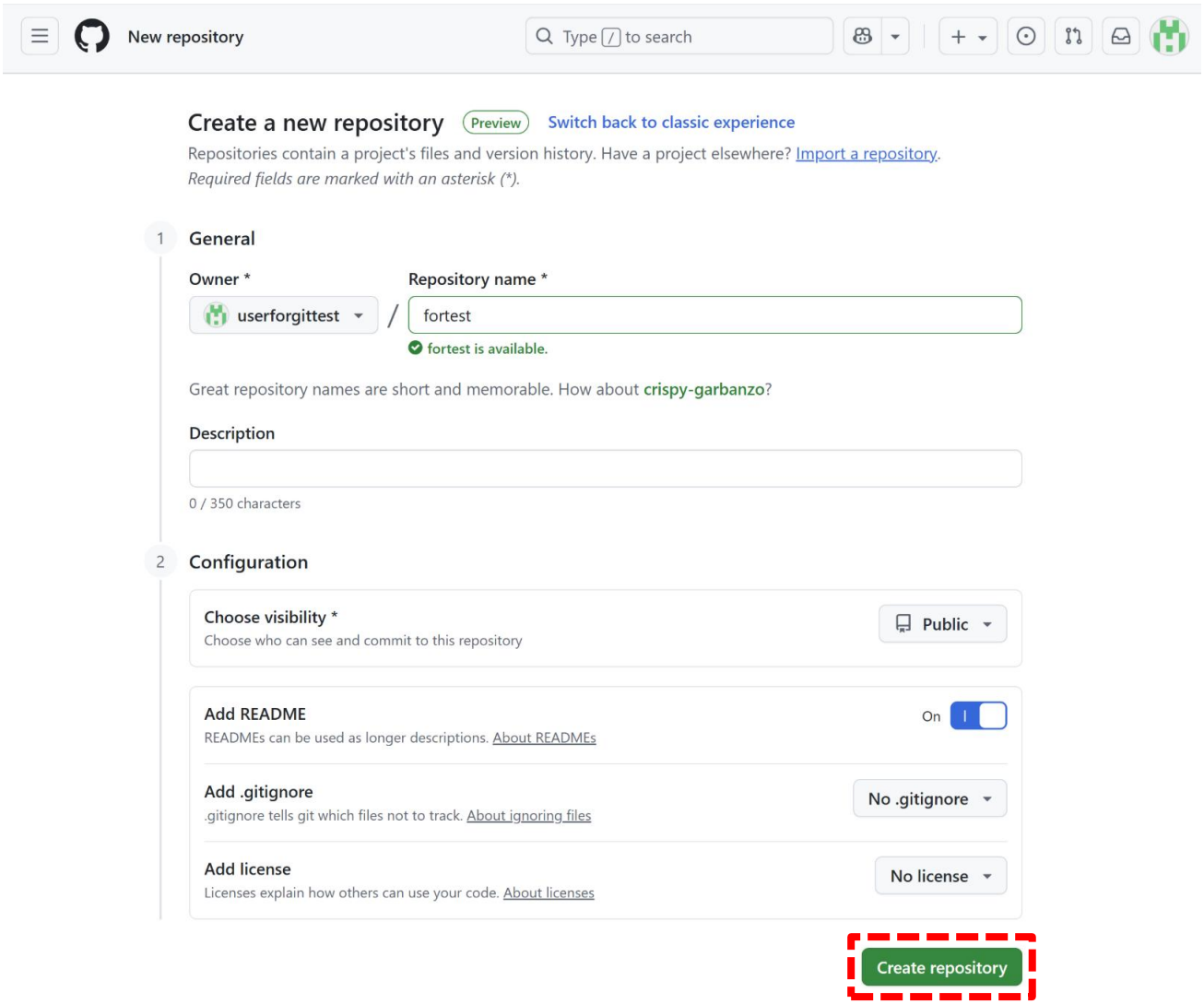
まずは、GitHub でアカウントを作成します。以下の URL より作成可能です。
Sign up to GitHub · GitHub

ログインに成功すると以下のような画面に遷移するため、画面左上の「Create repository」をクリックします。

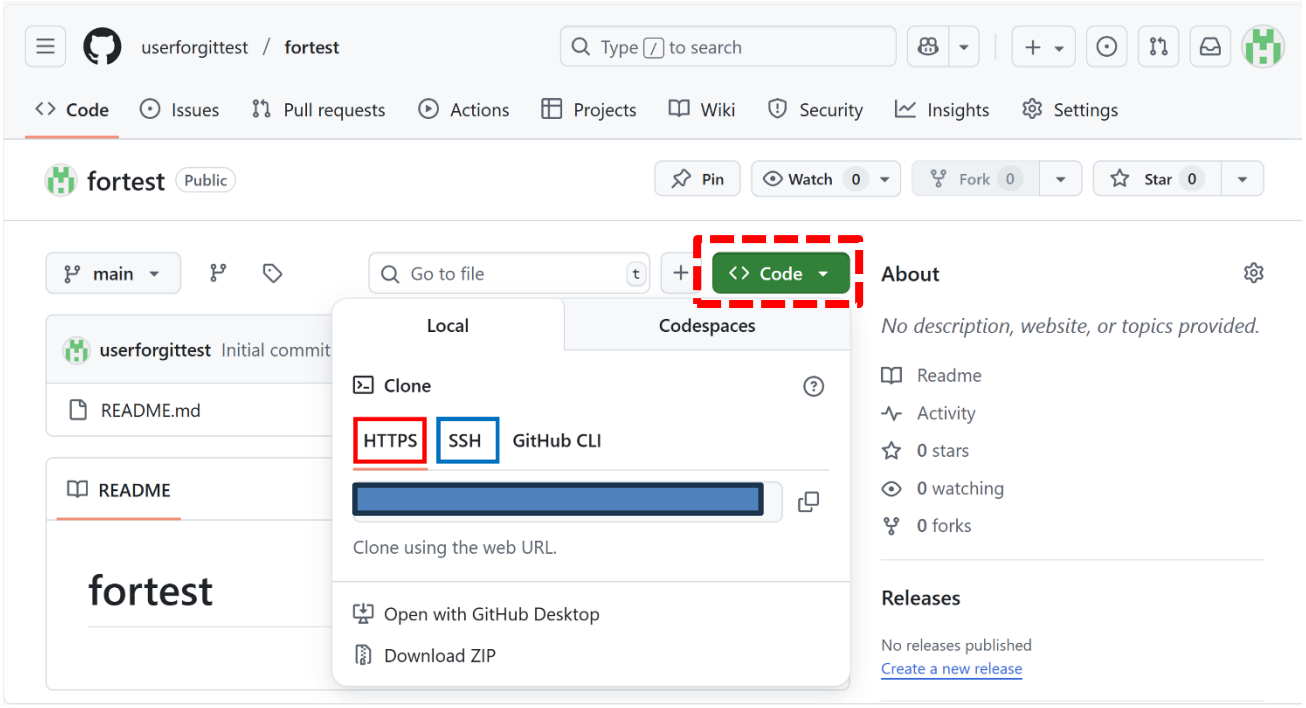


リポジトリの作成画面で必要な情報を入力します。この時、「Add a README file」や「Add .gitignore」「Choose a license」の設定を行うとブランチ「main」が自動で作成されます。今回は「Add a README file」にチェックを入れて作成します。また、Repository Type として「Public」を選択すると作成したリポジトリが全世界に公開されてしまいますので、必ず「Private」を選択してください。入力後は、画

面下の「Create repository」をクリックします。



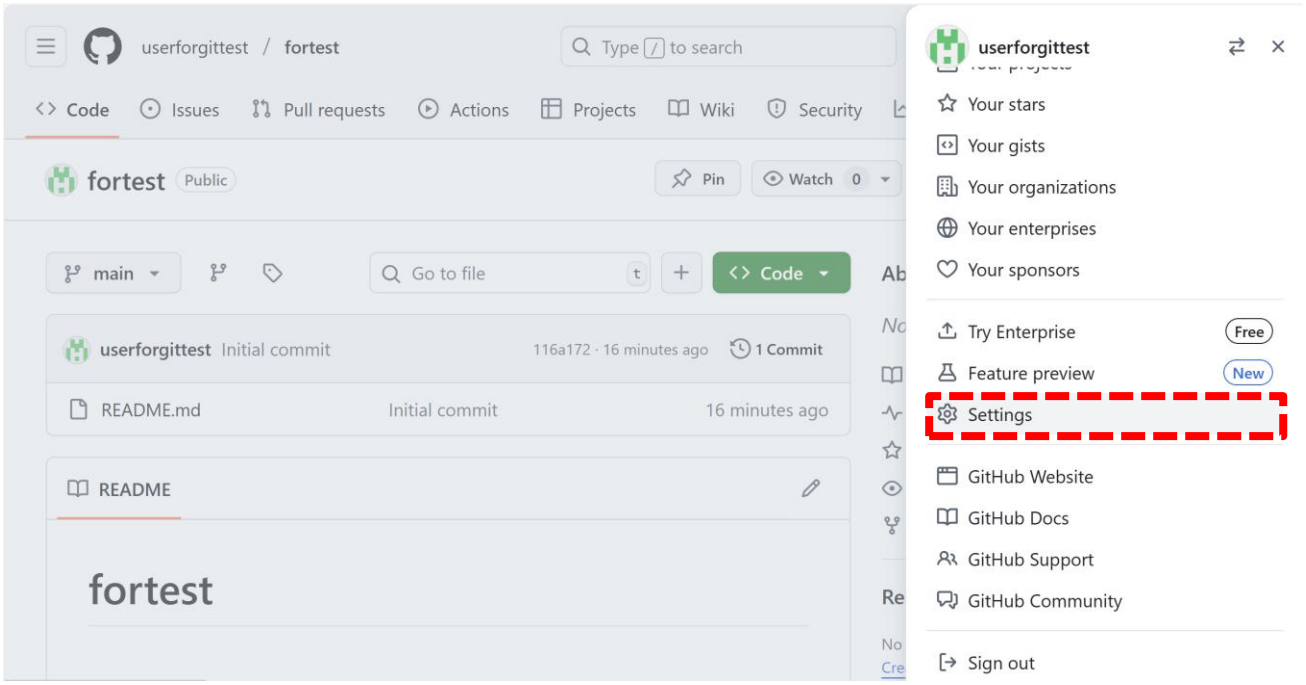
リポジトリ作成後に遷移する画面では、「<>Code」をクリックすることでリポジトリの URL を確認できます。この情報は後ほど ASTERIA Warp 上での設定をする際に使用します。API トークン認証の場合は「HTTPS」（画像赤枠参照）、SSH キー認証の場合は「SSH」（画像青枠参照）をクリックして表示される URL 情報が必要となります。



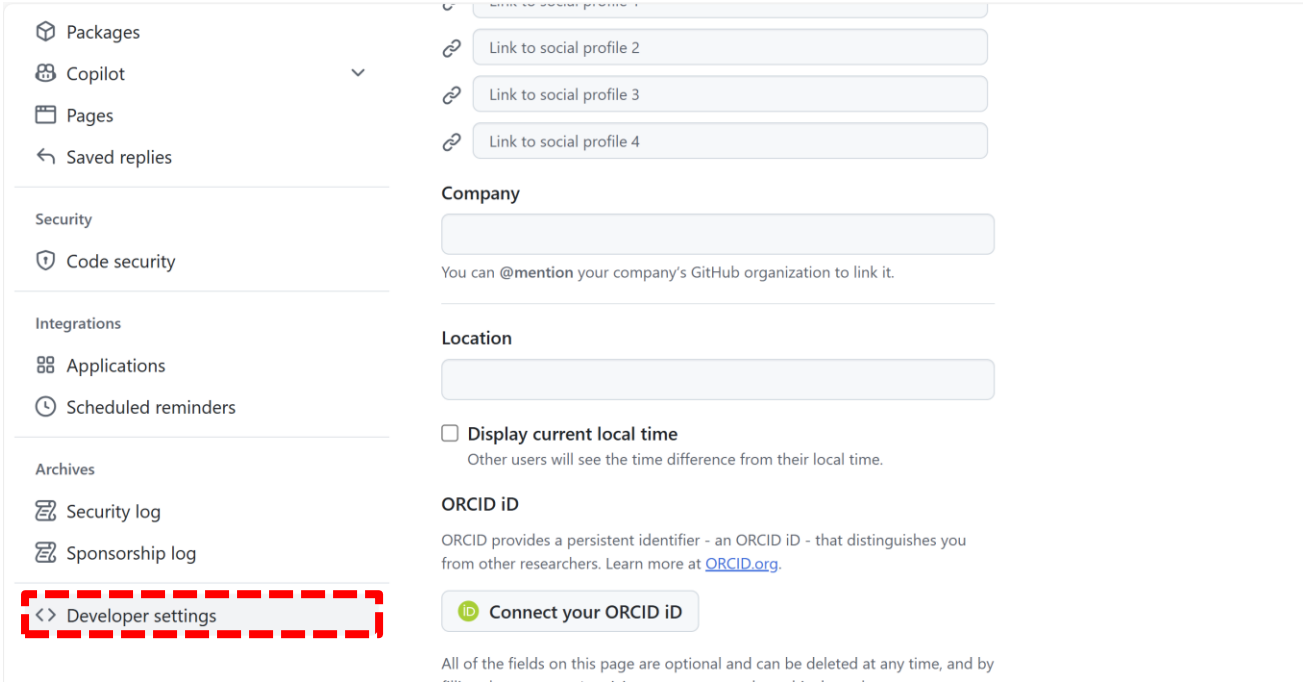
加えて API トークン認証とパスワード認証の場合はユーザー名も使用するため、画面左上のリポジトリ名が記載されている左隣の文字列を確認します。今回は「userforgittest」となっています。

3.1.1.GitHub でのセットアップ（API トークン認証の場合）

API トークン認証の場合、まず GitHub の画面右上でユーザーアイコンから「Settings」をクリックします。

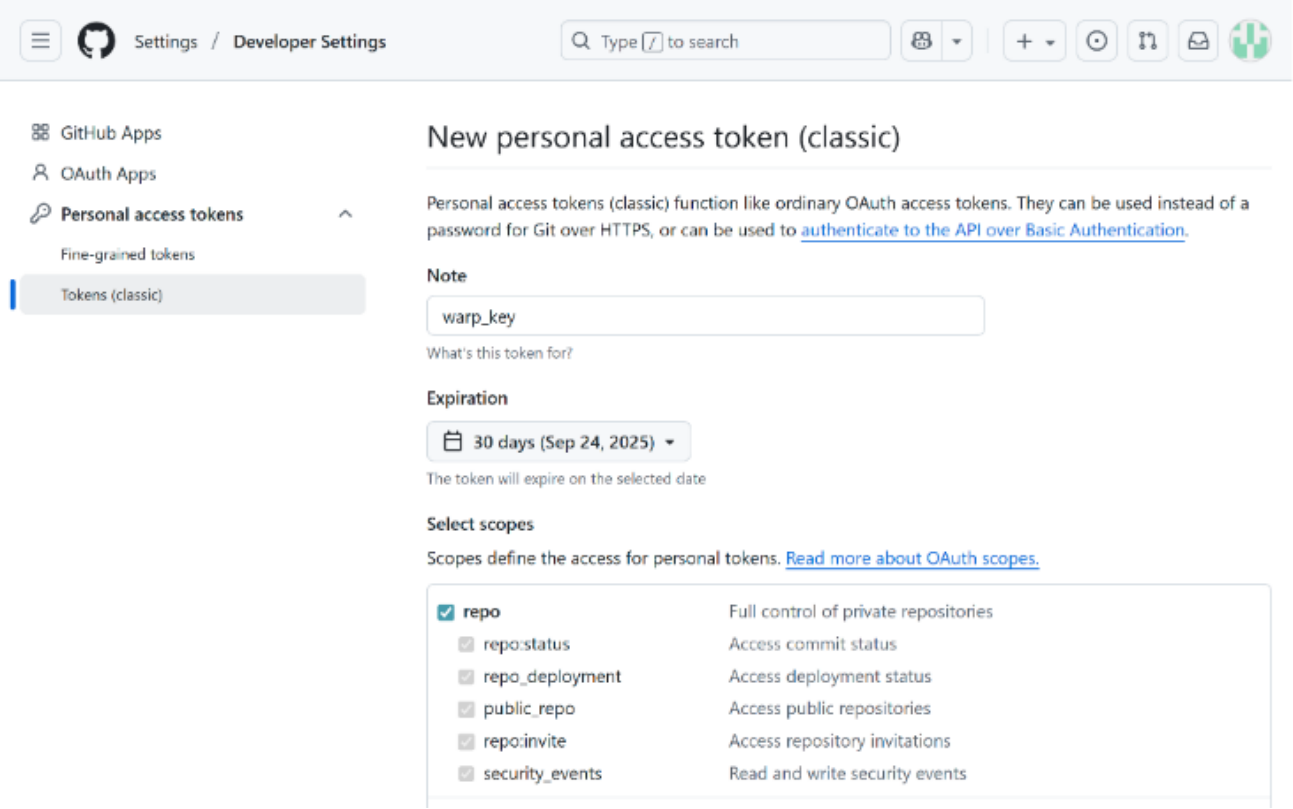


遷移後の画面では、左下から「<>Developer settings」をクリックします。



その後、左側で「Personal access tokens」、「Tokens(classic)」のメニューを選択します。そしてトークンを生成するために、「Generate new token」から「Generate new token(classic)」をクリックします。トークン生成を行う画面では、「Note」や「Expiration」「Select scopes」で要件に応じた内容を設定します。今回は以下のように設定しています。

Note: warp_key
Expiration: 30days
Select scopes: 「repo」のみチェック

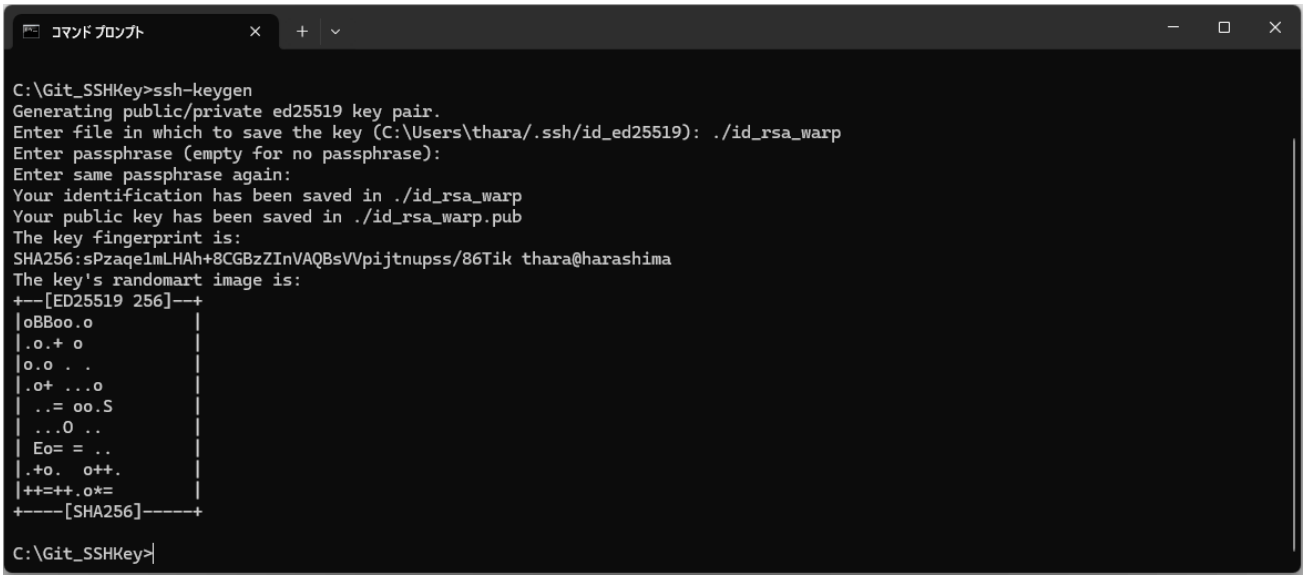


画面下の「Generate token」をクリックするとトークンが生成されますが、表示される情報は画面を移動すると消えてしまうため、この画面でコピーしてメモ帳などに保存します。

3.1.2.GitHub でのセットアップ（SSH キー認証の場合）

SSH キー認証の場合について、今回は Windows での操作を紹介します。

まず、鍵を保存するフォルダーを作成しコマンドプロンプトを起動します。その後、作成したフォルダーに移動し、「ssh-keygen」コマンドを利用して鍵を作成します。このコマンドが無い場合は OpenSSH クライアントをインストールしてください。



鍵作成時は以下のような設定が必要になります。

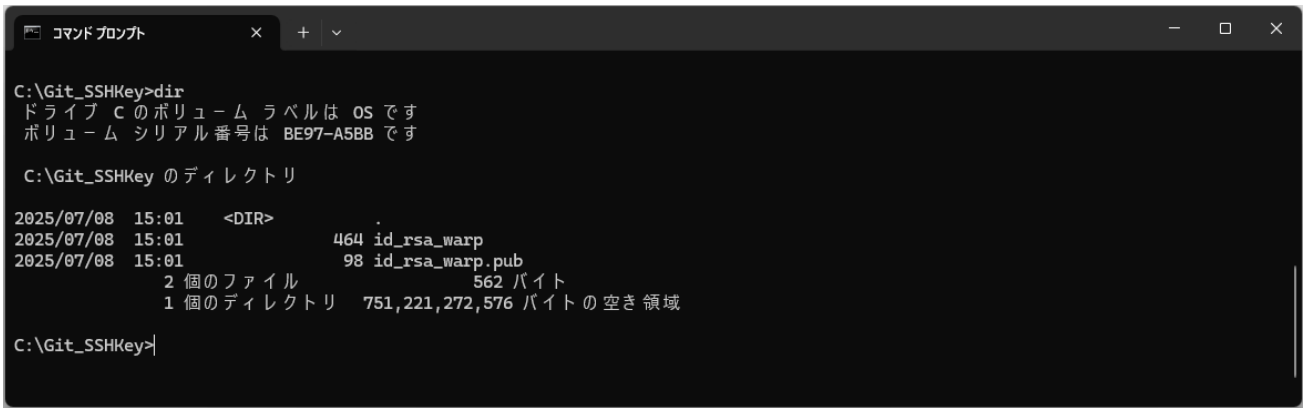
・Enter file in which to save the key：鍵ファイルパス情報を設定します。ここでは、「./id_rsa_warp」と設定しています。

・Enter passphrase (empty for no passphrase)：鍵のパスフレーズを設定します。この情報は後ほど使用します。

・Enter same passphrase again：上記の設定したパスフレーズをもう一度入力します。

※ご利用の環境に合わせて適切な値で設定してください。

作成された鍵の情報を「dir」コマンドで確認します。

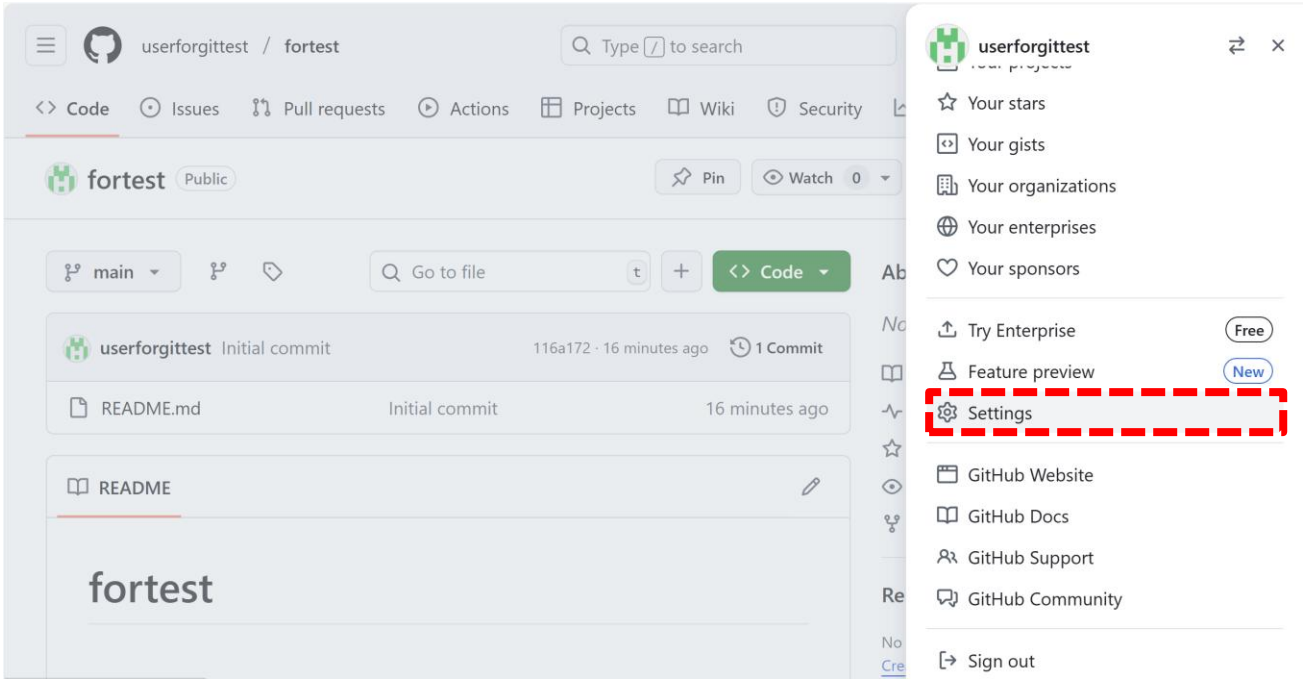


ここでは、次の 2 つの鍵ファイルが作成されていることが確認できます。

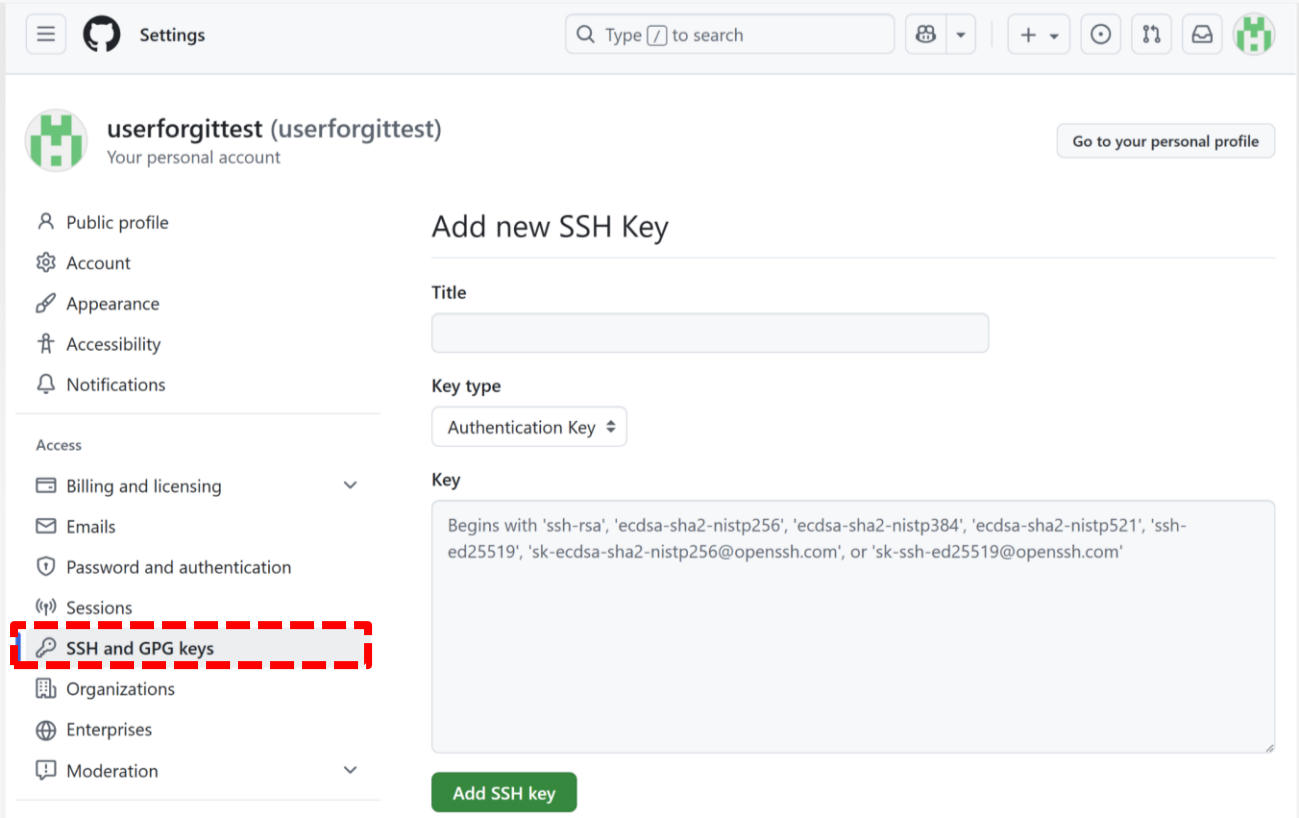
・id_rsa_warp：秘密鍵

・id_rsa_warp.pub：公開鍵

続いて作成された公開鍵を GitHub で設定します。
GitHub の画面右上でユーザーアイコンから「Settings」をクリックします。



遷移後の画面では「SSH and GPG keys」、そして「New SSH key」をクリックします。
その後の画面では要件に合わせて設定を行います。「Key」では作成した公開鍵の内容を入力します。これは先ほど作成した id_rsa_warp.pub などの、公開鍵のファイルをテキストエディター等で開きコピペで入力します。この時、改行コードなど余計な文字が含まれないように改行を削除するのを忘れないようにしてください。
入力後は、「Add SSH key」で内容を保存します。



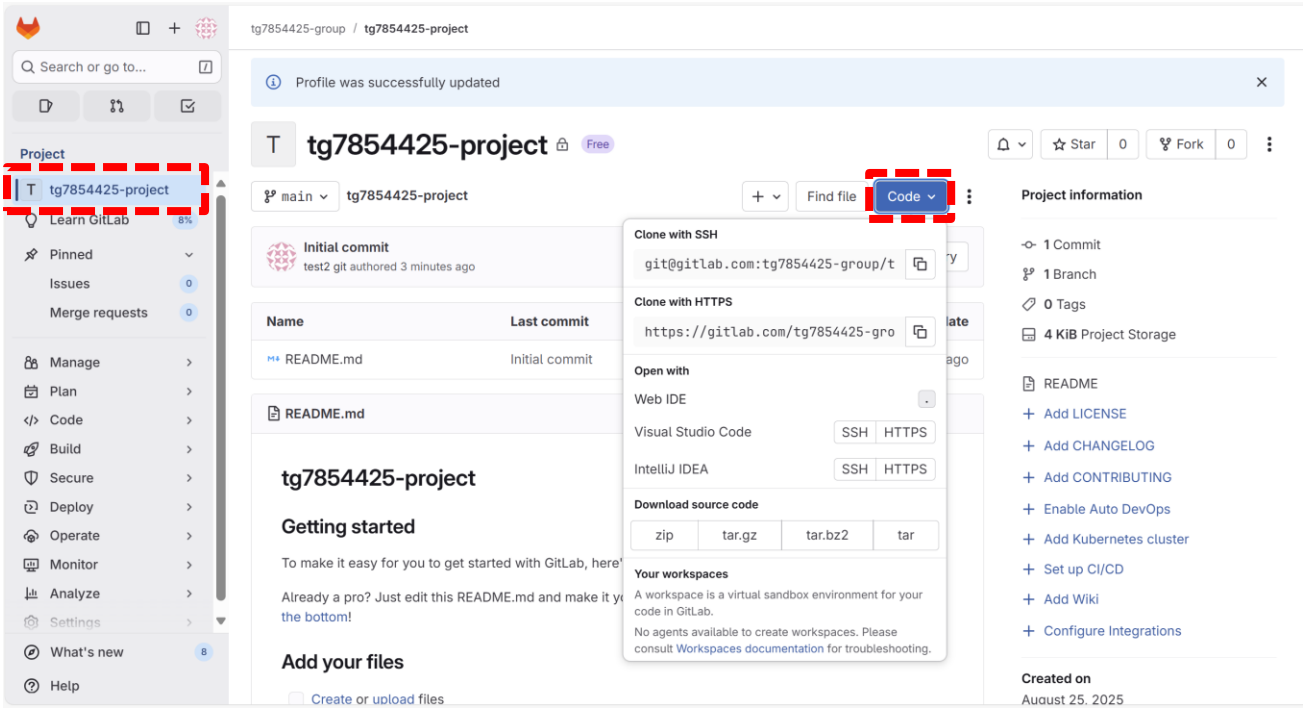
ASTERIA Warp 上で設定をする際、GitHub のフィンガープリントが必要になります。これは以下の URL で入手できる「SHA256:」で始まる各文字列です。
GitHub の SSH キーフィンガープリント - GitHub Docs

3.2.GitLab でのセットアップ（共通）

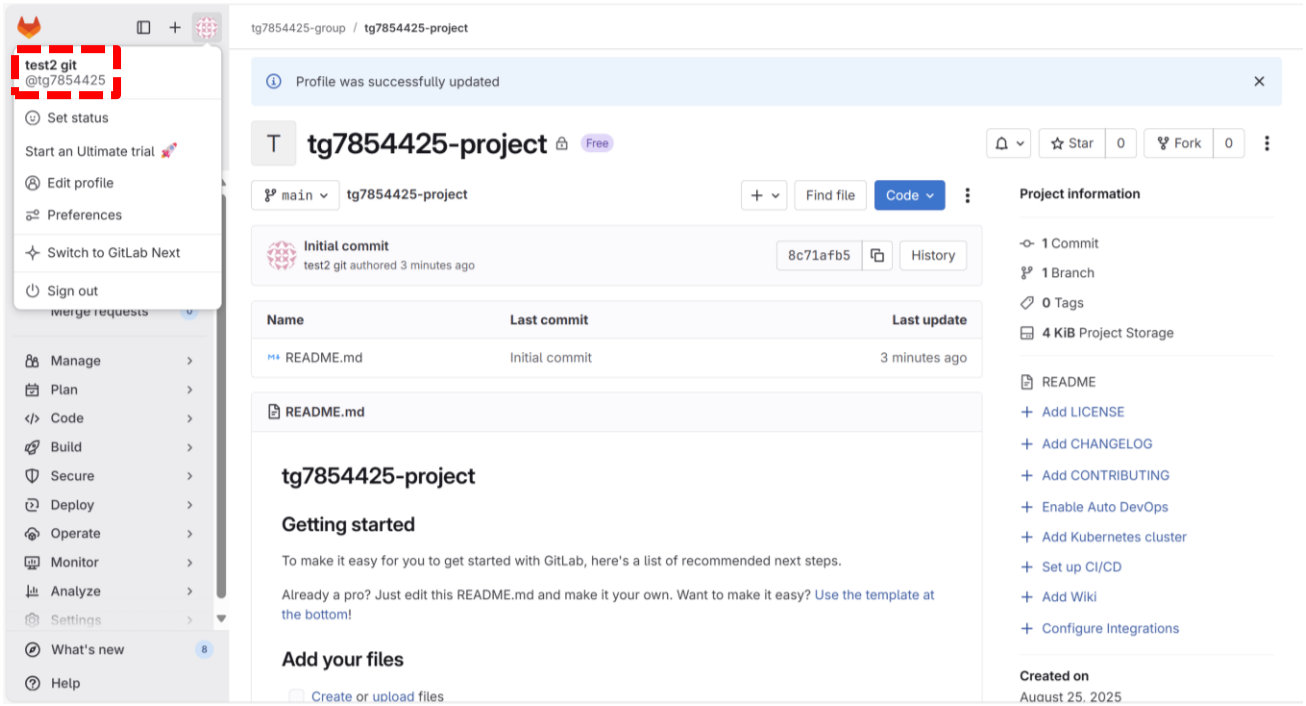
GitLab では、認証方法として「パスワード認証」「API トークン認証」「SSH キー認証」が選択できるため、これらの共通箇所を本節で紹介し、それぞれの認証方法固有の内容は次節以降で紹介します。

まずは、GitLab でアカウントを作成します。以下の URL より作成可能です。
サインアップ ·GitLab

ログインを行い、画面左上でプロジェクト名をクリックします。
この画面では、「Code」をクリックすることでリポジトリの URL を確認できます。この情報は後ほど ASTERIA Warp 上での設定をする際に使用するためメモします。パスワード認証か API トークン認証の場合は「HTTPS」、SSH キー認証の場合は「SSH」の情報が必要となります。



加えて API トークン認証の場合はユーザー名も使用するため、画面左上のユーザーアイコンをクリックし、表示されるメニューから「@」以降の文字列を確認します。今回は「tg7854425」となっています。

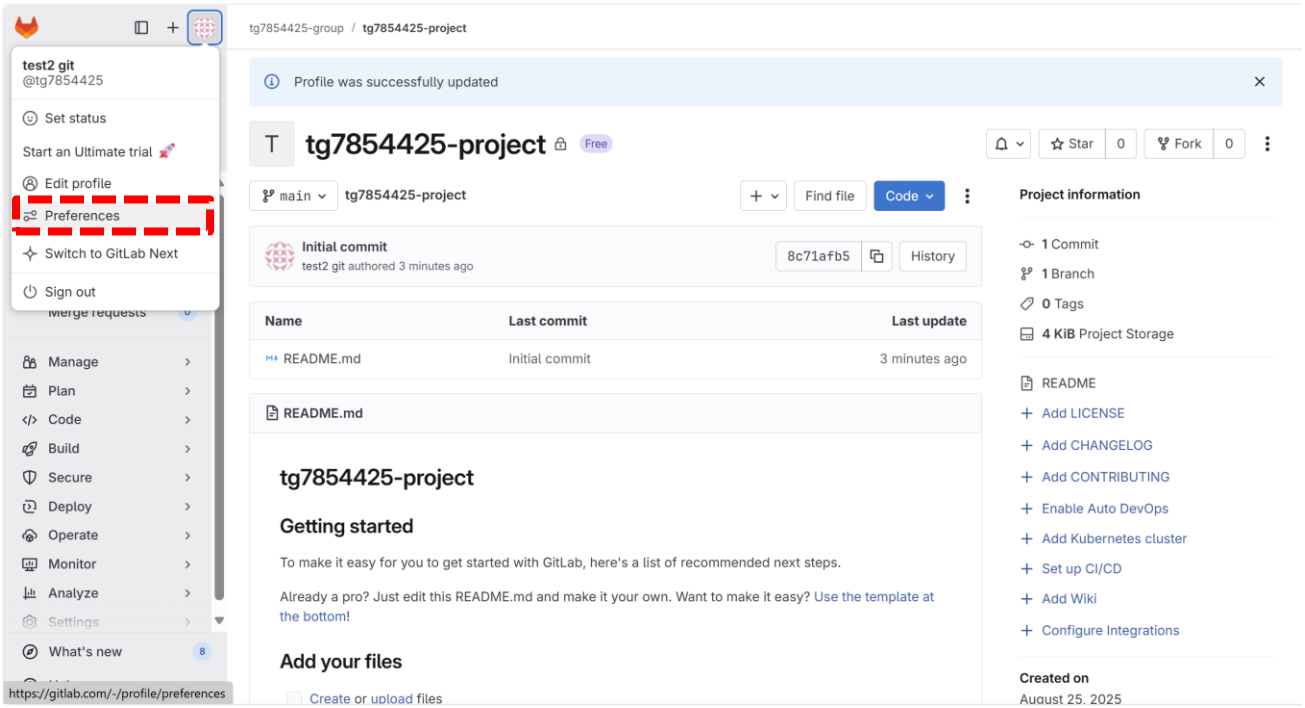


3.2.1.GitLab でのセットアップ（パスワード認証の場合）

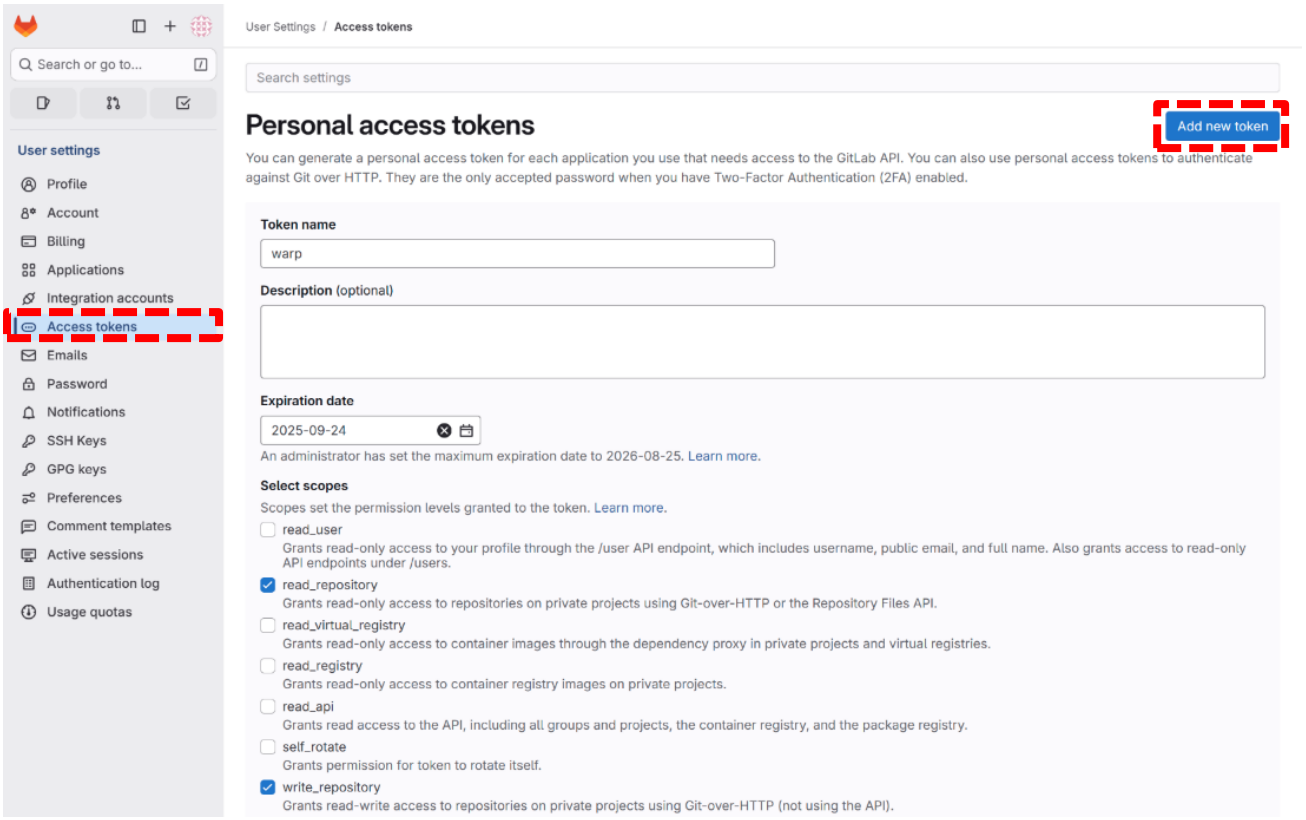
パスワード認証については、ログインに使用するパスワードが必要になります。
こちらはアカウント作成時に設定する情報になります。

3.2.2.GitLab でのセットアップ（API トークン認証の場合）

この認証方法では、必要となる API トークンを取得するために、画面左上のユーザーアイコンから「Preferences」をクリックします。



遷移後の画面では左側で「Access tokens」をクリックします。そして右上の「Add new token」からトークンの生成画面に遷移できます。



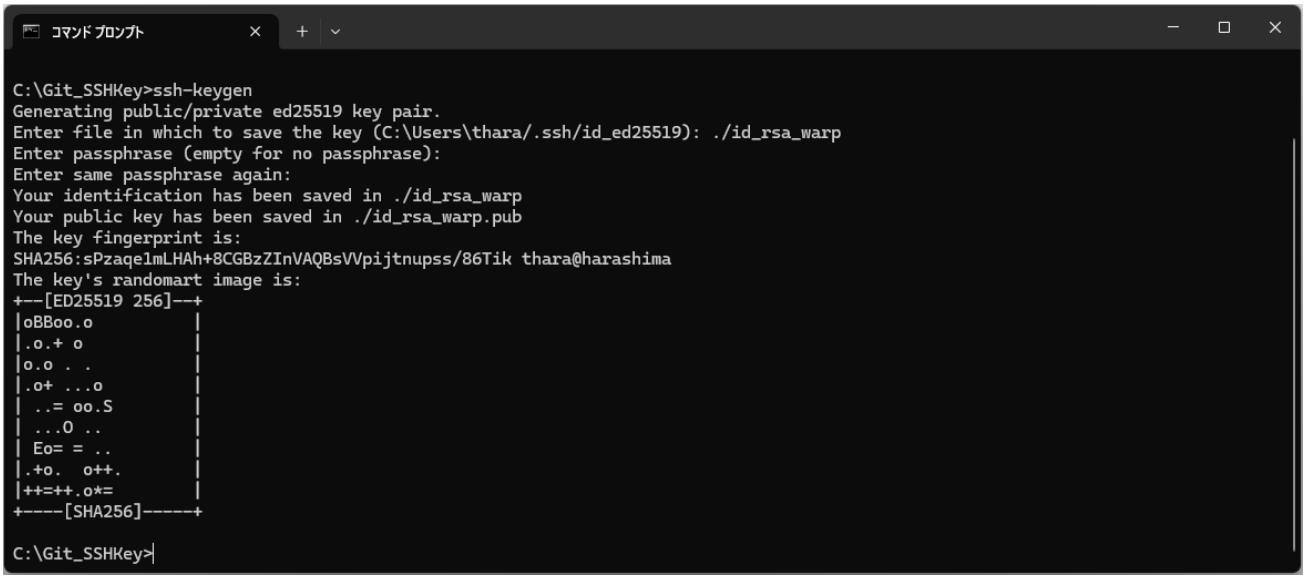
この画面では要件に合わせて各項目を入力します。
今回は以下のように設定しています。
Token name: warp
Expiration date: 2025-09-24
Select scopes: 「read_repository」と「write_repository」のみチェック

画面下部「Create token」をクリックするとトークンが生成されますが、表示される情報は画面を移動すると消えてしまうため、この画面でコピーしてメモ帳などに保存します。

3.2.3.GitLab でのセットアップ（SSH キー認証の場合）

SSH キー認証の場合について、今回は Windows での操作を紹介します。

まず、鍵を保存するフォルダーを作成しコマンドプロンプトを起動します。その後、作成したフォルダーに移動し、「ssh-keygen」コマンドを利用して鍵を作成します。このコマンドが無い場合は OpenSSH クライアントをインストールしてください。



鍵作成時は以下のような設定が必要になります。

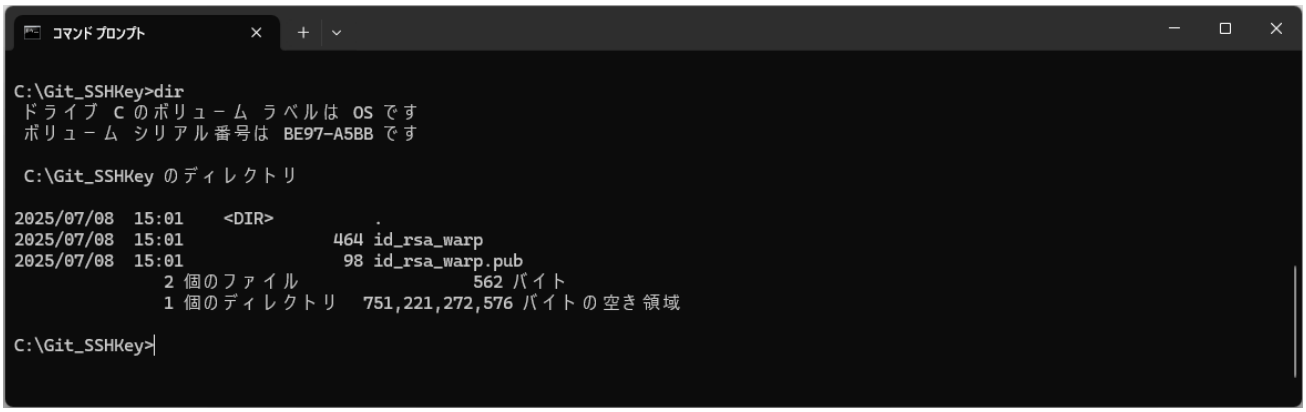
・Enter file in which to save the key：鍵ファイルパス情報を設定します。ここでは、「./id_rsa_warp」と設定しています。

・Enter passphrase (empty for no passphrase)：鍵のパスフレーズを設定します。この情報は後ほど使用します。

・Enter same passphrase again：上記の設定したパスフレーズをもう一度入力します。

※ご利用の環境に合わせて適切な値で設定してください。

作成された鍵の情報を「dir」コマンドで確認します。

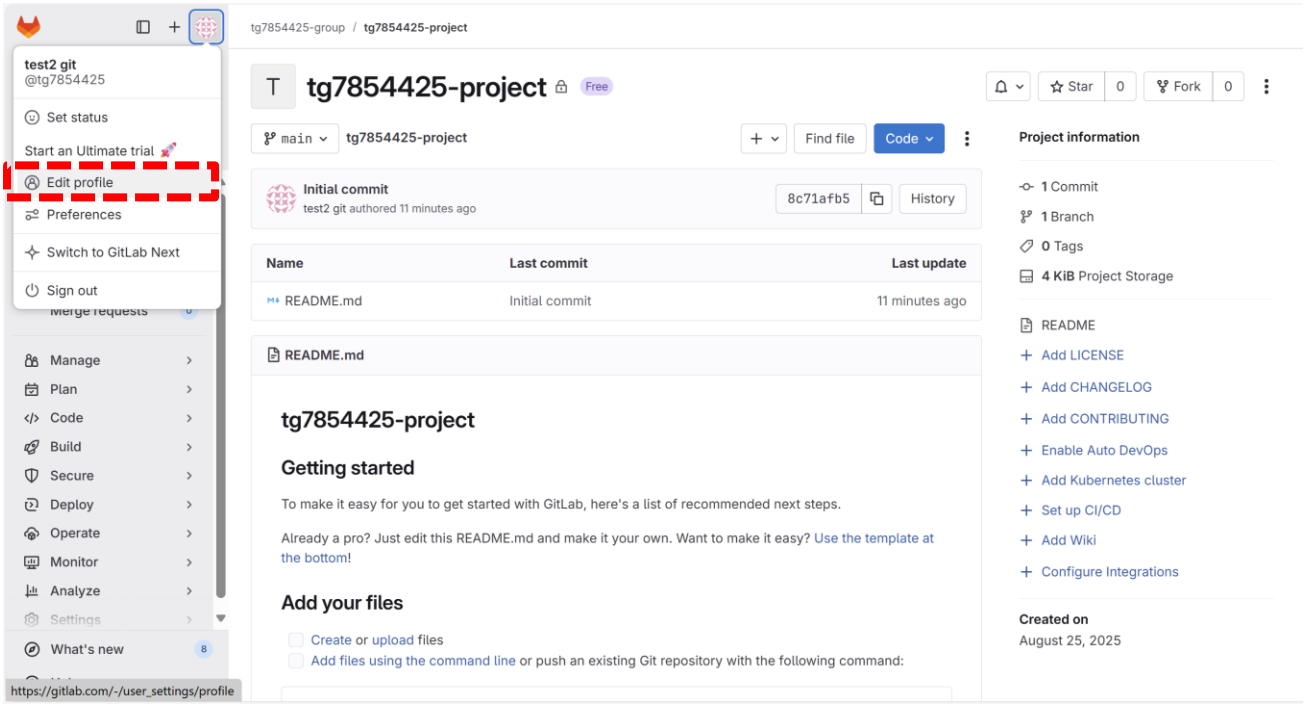


ここでは、次の 2 つの鍵ファイルが生成されていることが確認できます。

・id_rsa_warp：秘密鍵

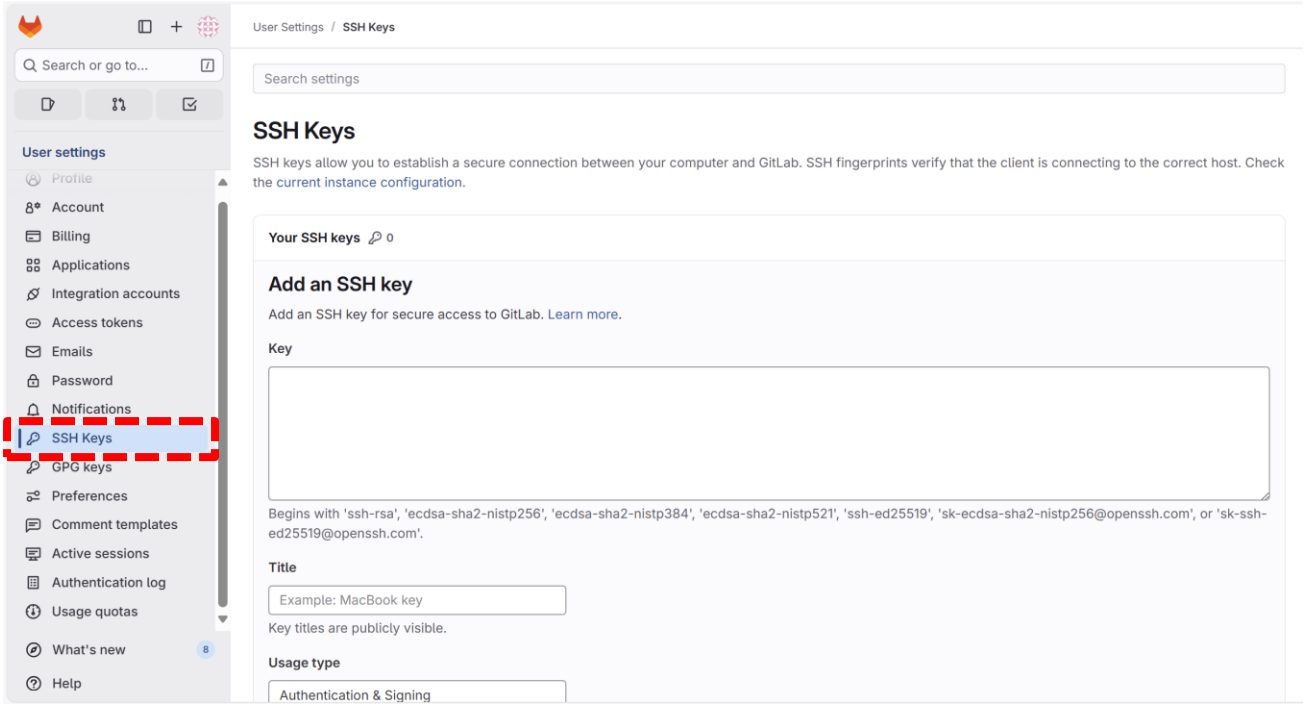
・id_rsa_warp.pub：公開鍵

続いて作成された公開鍵を GitLab で設定します。
GitLab の画面左上でユーザーアイコンから「Edit profile」をクリックします。



遷移後の画面では、「SSH Keys」をクリック、右上で「Add new key」から確認できる画面で、作成した SSH 鍵についての情報を入力します。

ここでは要件に合わせて設定を行います。「Key」では作成した公開鍵の内容を入力します。これは先ほど作成した id_rsa_warp.pub などの、公開鍵のファイルをテキストエディター等で開きコピペで入力します。この時、公開鍵を設定する際には改行コードなど余計な文字が含まれないように改行を削除するのを忘れないようにしてください。入力後は「Add key」で入力内容を保存します。



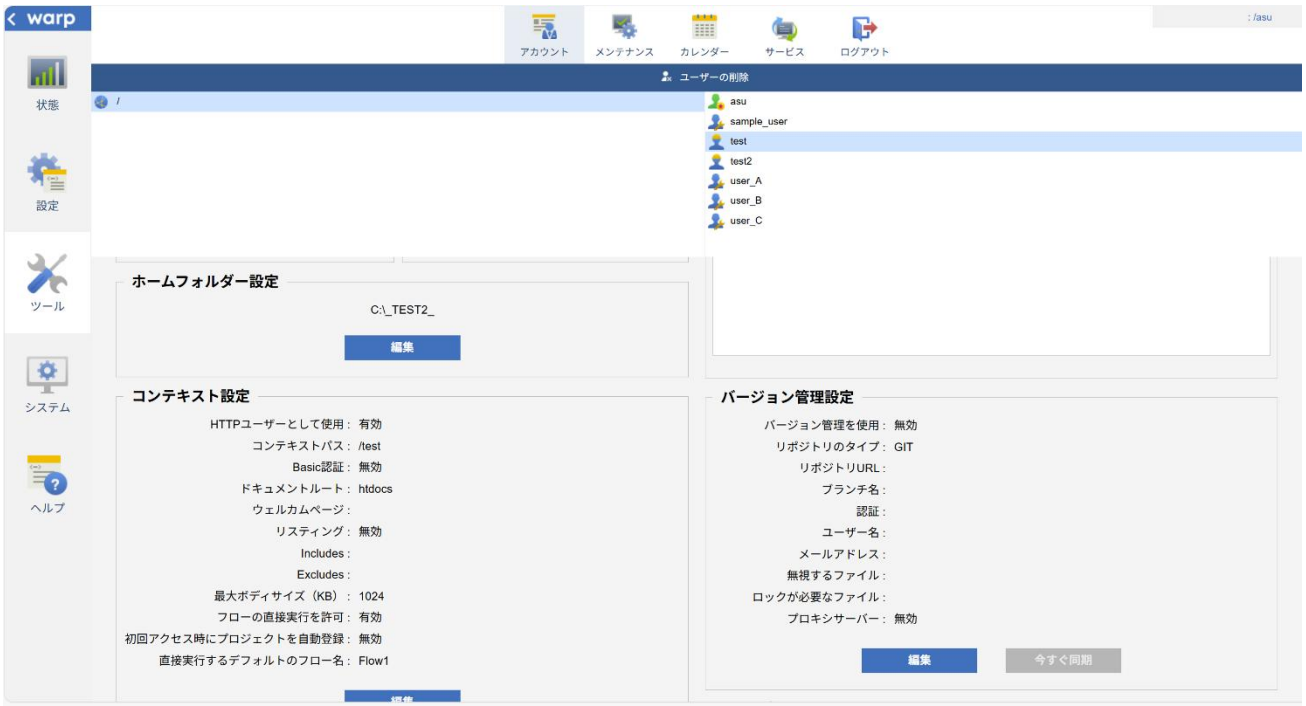
ASTERIA Warp 上で設定をする際は、GitLab のフィンガープリントが必要になります。これは以下の URL の表で「SHA256」列にある各文字列です。

[GitLab.com settings | GitLab Docs]

4.ASTERIA Warp 側の設定

ASTERIA Warp で Git を使用するためには、前章で作成した情報を使用して、フローサービス管理コンソールで設定を行います。また ASTERIA Warp サーバーと Git サーバーで通信可能な環境が必要です。

まずユーザー管理権限を持つ asu などのユーザーでフローサービス管理コンソールにログインし、メニューの「ツール>アカウント」のユーザー一覧からバージョン管理を使用するユーザーをクリックします。



表示ページの中に「バージョン管理設定」がありますので、「編集」をクリックします。

各設定項目の意味は以下のとおりです。

バージョン管理を使用

このユーザーで作成したファイルをバージョン管理するかどうかの指定です。バージョン管理する場合は「ON」にします。

リポジトリのタイプ

ここでは、リポジトリのタイプが表示されています。

リポジトリ URL

このユーザーで使用する Git のリポジトリパスを指定します。この情報は前章でリポジトリ作成した後に確認できます。

ブランチ名

このユーザーで使用するデフォルトのブランチ名を指定します。空白の場合は「main」または「master」のどちらか存在するブランチとなります。

認証

認証方法を「パスワード」「API トークン」「SSH キー」からクリックします。クリックした内容

に応じて以下のいずれかの項目が表示されます。各項目には前章で作成・発行した情報を指定します。

・パスワードの場合：「ユーザー名」と「パスワード」が表示されます。ここには使用する Git アカウントに紐づくものを設定します。

・API トークンの場合：「ユーザー名」と「API トークン」が表示されます。Git サーバーで発行した API トークンを設定します。

・SSH キーの場合：「SSH キー」と「SSH キーパスフレーズ」、「ホストの SSH キーのフィンガープリント」が表示されます。「SSH キー」では発行した秘密鍵を設定します。「SSH キーパスフレーズ」では、SSH 鍵を作成時に設定したパスフレーズを設定します。「ホストの SSH キーのフィンガープリント」では Git サーバーから取得した情報を設定します。

メールアドレス

メールアドレスを指定します。Git へのコミット時のメールアドレスとして使用されます。コミット時のユーザー名としては ASTERIA Warp ユーザーの名前が使用されます

無視するファイル

バージョン管理を行わないファイルのパターンをセミコロンまたはカンマで区切り指定します。デフォルトでは「.*;_*;*.xfp2;*.$$$」となっています。このパターンは
「.*」「_*」：ASTERIA Warp の隠しフォルダー
「*.xfp2」　：プロジェクトコンパイル時の中間ファイル
「*.$$$」　：フローデザイナーのロックファイル
を表しており、通常はこれらのパターンをこの設定から外す必要はありません。これ以外にバージョン管理する必要のないことがわかっているファイルパターンがあるならば追加します。

ロックが必要なファイル

競合対策として「ロックが必要なファイル」とするファイルのパターンをセミコロンまたはカンマ区切りで指定します。デフォルトでは「*.xfp;*.xmp;*.xvar;*.xsf」となっています。このパターンは
「*.xfp」　:フローのプロジェクトファイル
「*.xmp」　:関数コレクションのファイル
「*.xvar」　:外部変数セットのファイル
「*.xsf」　:ストリーム定義セットのファイル
を表しています。
これ以外に「*.xlsx」(Excel ファイル)など他にバージョン管理したいファイルのファイルパターンがあるならば追加します。

プロキシ

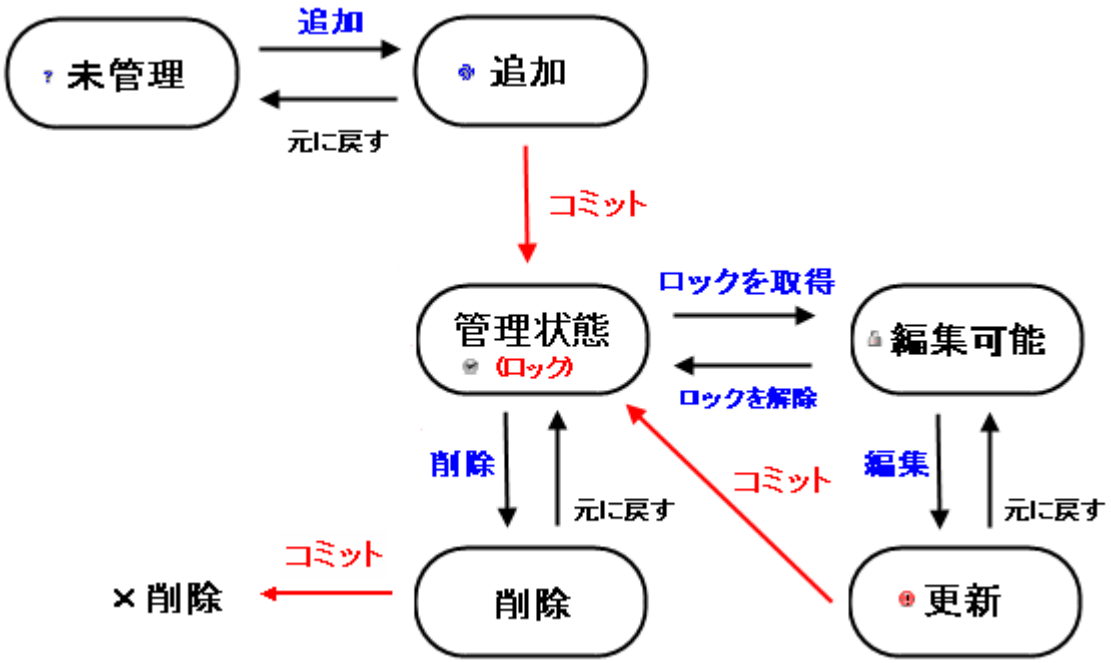
Git サーバーへの接続でプロキシサーバーを利用する場合、「ON」にします。

ON にした場合は、フローサービス管理コンソールの「設定＞プロキシ」画面でプロキシとして設定した内容が使用されます。

5.ASTERIA Warp の Git 操作

ASTERIA Warp からの Git 操作はフローデザイナー、フローサービス管理コンソール、flow-ctrl から行うことができます。フロー開発時においては、ほとんどの操作はフローデザイナーから行います。また、Git に対する各操作は、ASTERIA Warp サーバーがクライアントとなって一元的に実行します。

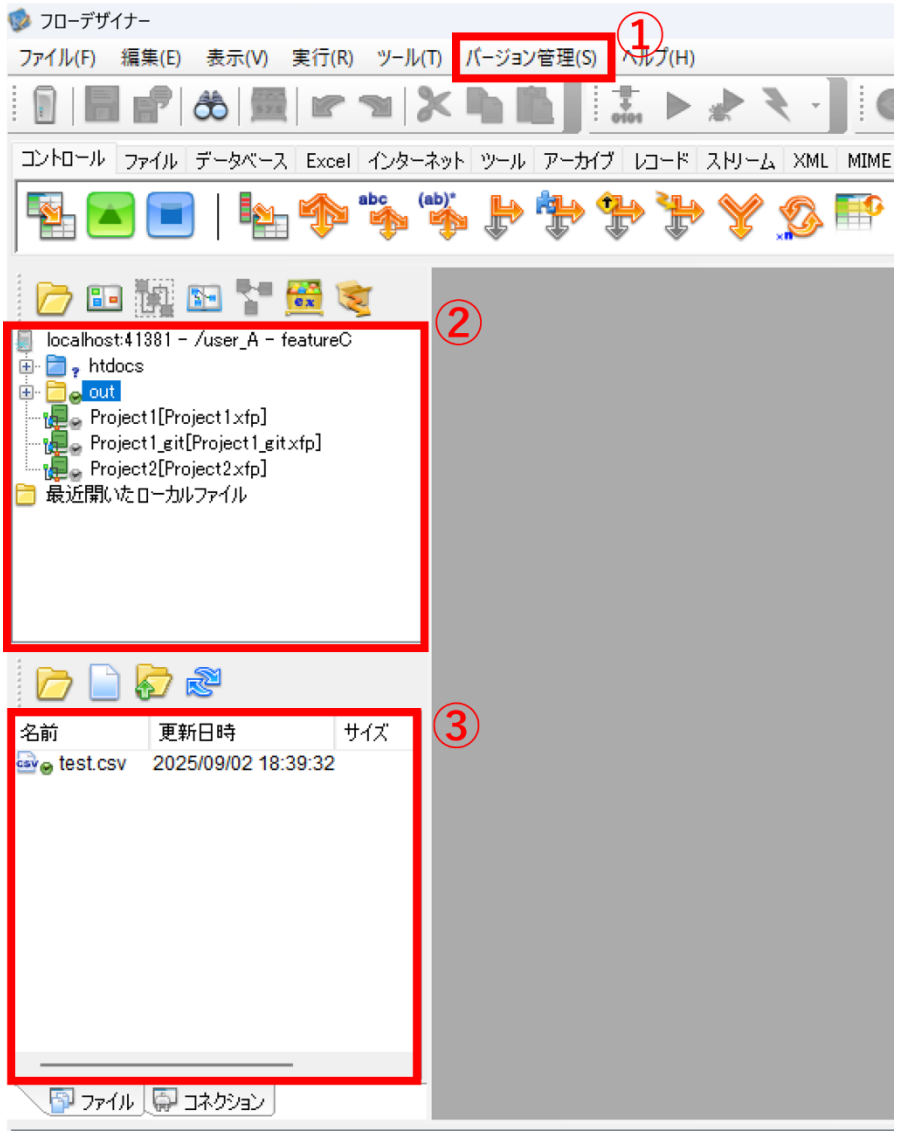
フローデザイナーでのバージョン管理のステータス遷移



5.1.フローデザイナーからの操作

バージョン管理が有効となっているユーザーにフローデザイナーで接続するとツリーペインとファイルペインにバージョン管理の状態アイコンが表示されます。バージョン管理関連の操作は以下のいずれかから実行できます。

①バージョン管理ツールバー
②ツリーペインでサーバーやプロジェクトをクリックしての右クリックメニュー
③ファイルペインでファイルをクリックしての右クリックメニュー



フローデザイナーからはブランチの作成や移動、スタッシュの管理、コミット、マージ、競合の解決などが実行できます。具体的な操作方法についてはフローサービスマニュアルの「フローデザイナー」-「バージョン管理」以下のマニュアルを参照してください。
[フローデザイナー] フローサービスマニュアル

5.2.フローサービス管理コンソールからの操作

バージョン管理が設定されたユーザーでフローサービス管理コンソールにログインした場合、メニューの「状態>バージョン管理」画面で以下の操作が可能です。

- 現在のブランチの状態確認
- 更新履歴の確認
- 指定したリビジョンへの更新

- 最新リビジョンへの更新
- ファイルやフォルダー単位でのリビジョンの置き換え

フローサービス管理コンソールから行うことができる操作は上記のみで、ブランチの操作やコミットなどは行うことができません。具体的な操作方法についてはフローサービスマニュアルのフローサービス管理コンソール(FSMC)オンラインヘルプを参照してください。

フローサービス管理コンソールのリファレンス

5.3.flow-ctrl による操作

バージョン管理が設定されたユーザーで flow-ctrl にログインした場合、バージョン管理関連のコマンドで以下のような操作が可能です。

- 履歴の表示
- ファイルの状態の表示
- 指定したリビジョンの取得
- スタッシュの一覧の取得
- 指定したスタッシュの削除

具体的な操作方法についてはフローサービスマニュアルの flow-ctrl コマンドリファレンスを参照してください。

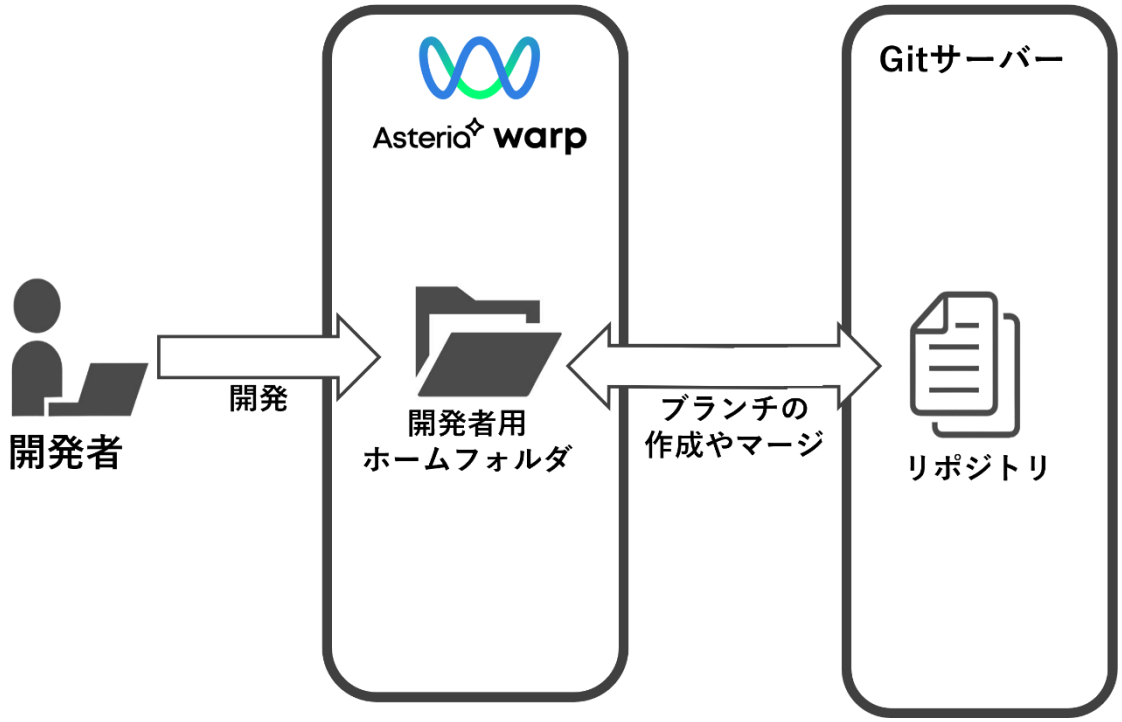
flow-ctrl コマンドリファレンス

6.具体的な使用例

ここからは、実際の運用で想定される使用例を 2 つご紹介します。

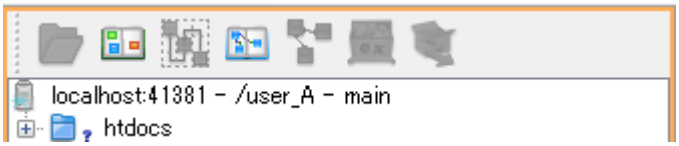
6.1.基礎編_1 人で開発する場合

最初に基礎編として、1 人で開発する例を紹介します。1 人で開発する場合はわざわざブランチを作成する必要はありませんが、今回は Git の特長を活かし、あえて本番用と開発用の 2 本のブランチを使用する方法とします。

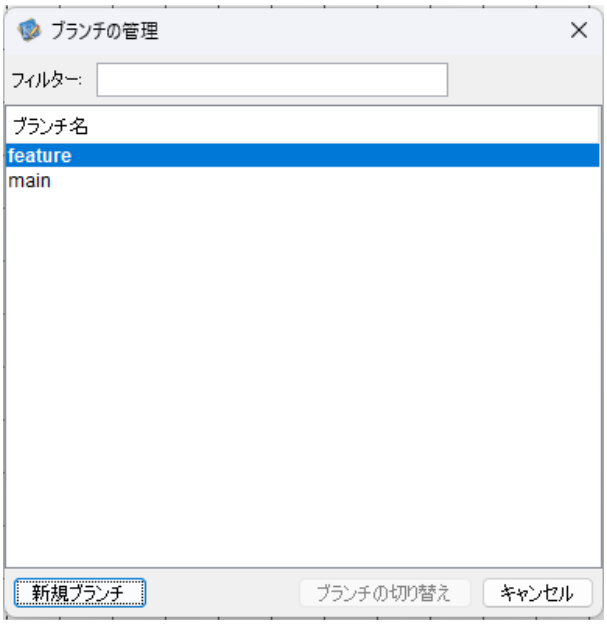


この例では、ASTERIA Warp ユーザーと Git ユーザーがそれぞれ 1 つずつ必要になります。そのため、まずこれらの準備を行います。Git の準備については 3 章をご参照ください。
Git ユーザーの準備が完了した後は、フローサービス管理コンソールで ASTERIA Warp ユーザーに対して、バージョン管理の設定を行います。設定方法は 4 章をご参照ください。設定時に使用するブランチは「main」としてください。「保存」をクリック後に、画面上部に以下のようなリビジョンが表示されれば設定完了です。

続いて、設定したユーザーでフローデザイナーにログインします。ログイン後、現在のブランチが main ブランチであることを確認します。現在のブランチはサーバーツリーの横にあるユーザー名の隣に表示されています。



次に、feature という名前の新規のブランチを作成します。ブランチの作成方法は以下をご参照ください。
ブランチの管理

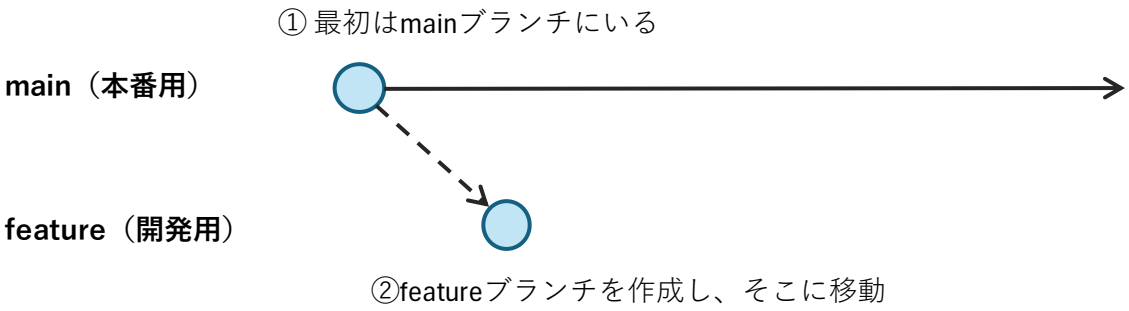


この操作で main ブランチから分岐した feature ブランチを作成することができました。

今回は説明をシンプルにするためにブランチ名を「feature」としました。実際の開発ではブランチを何本も作成することになりますので、たとえば「feature/add-fileget」のように、「feature/」+「その時に開発・修正する内容を端的に表現した英文」という命名にするとよいでしょう。

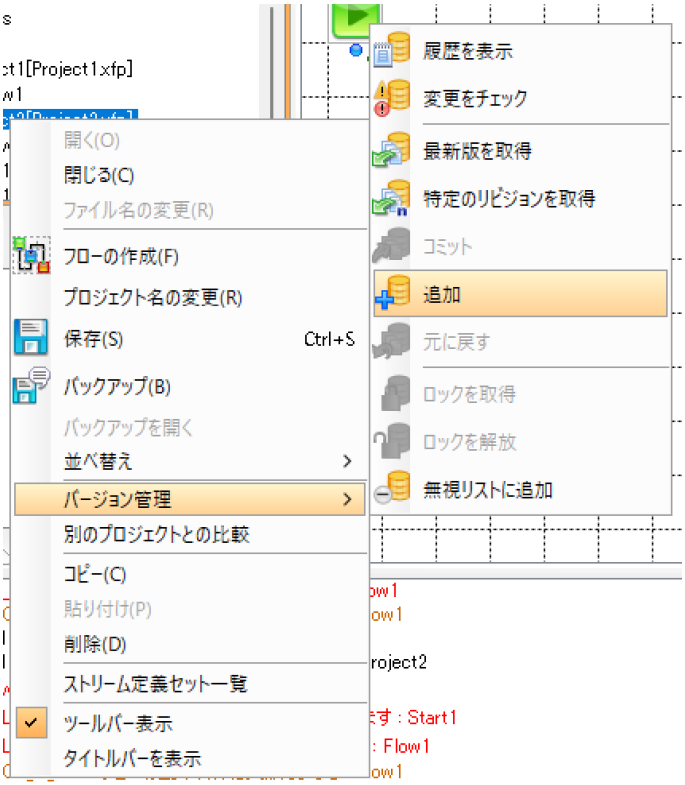
作成後は feature ブランチに移動していることがサーバーツリーのユーザー名の右側にあるブランチ名

で確認できます。今回のケースでは、main ブランチを本番用、feature ブランチを開発用ブランチとして扱います。

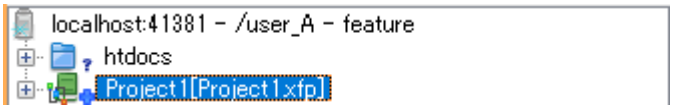


それでは、feature ブランチで新たにプロジェクトとフローを作成しましょう。フローの内容は任意で問題ありません。

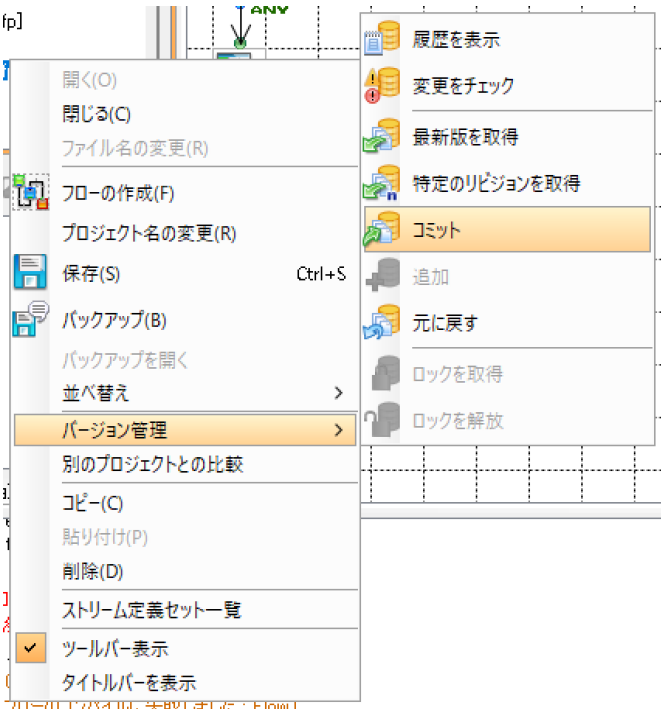
プロジェクトやファイルが追加された時やバージョン管理の設定前に既にユーザーのホームフォルダー内に存在する場合は、そのファイルをバージョン管理の対象に追加する必要があります。今回はプロジェクトを新たに作成したので、この操作が必要になります。追加についての詳細は以下をご参照ください。
管理対象への追加



追加直後はファイルアイコンの右側に表示されるマークが「?」から「+」へ変化します。
表示されるアイコンの一覧と詳細については以下をご参照ください。
デザイナーでの操作



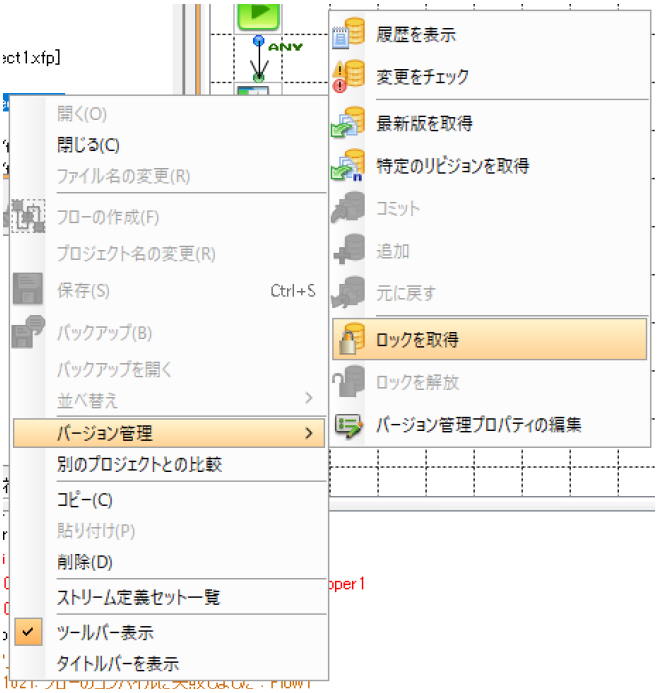
作業完了後はコミットで更新内容を保存します。この操作については以下をご参照ください。
コミット



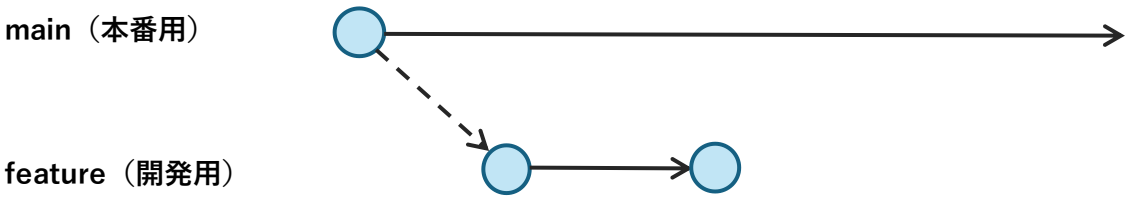
そして今回の作業では行いませんが、既にバージョン管理の対象になっているファイルを編集する際は、作業前にファイルに対してロックの取得が必要となる場合があります。ファイルアイコンの右側の「✓」マークがグレーの場合は必要、緑の場合は不要です。ロックを必要とするファイルの指定方法は 5 章をご参照ください。

ロックの取得については以下をご参照ください。

ロックを取得



ここまでの作業を図にすると以下のようになります。

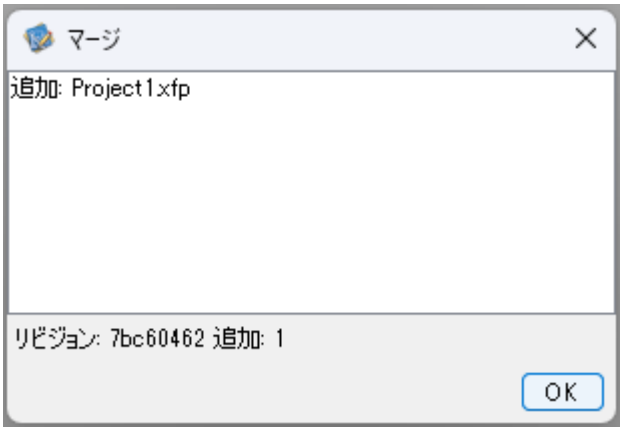


③featureブランチ内で開発し、コミット

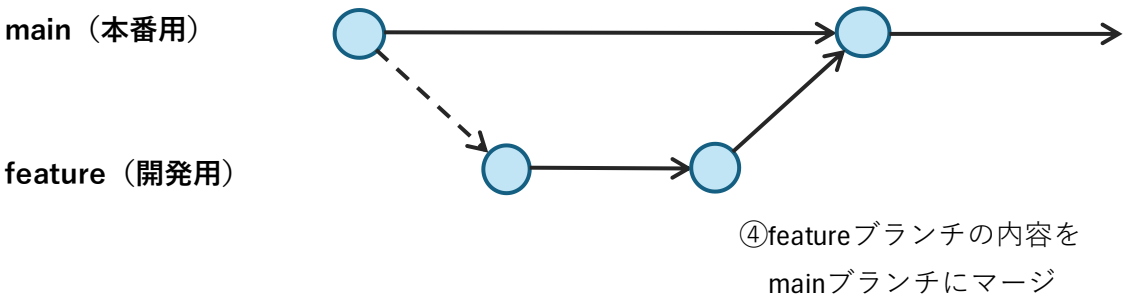
上記図のように、現在は開発用である feature ブランチと本番用の main ブランチが別々に管理されている状態です。feature ブランチでの作業内容を main ブランチへ反映させるためにはマージが必要になります。

ブランチのマージを行うには、マージ先に移動した状態で行う必要があります。今回は feature ブランチを main ブランチにマージするため、main ブランチへ移動してから操作をします。ブランチの移動、マージについては以下の「ブランチを切り替える」「ブランチをマージする」をご参照ください。

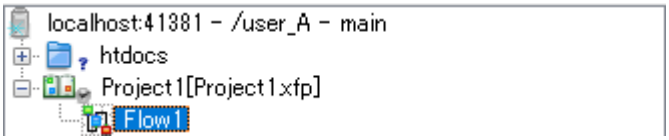
ブランチの管理



図にするとこのようになります。



最後に main ブランチで、追加されたプロジェクトとフローを確認します。



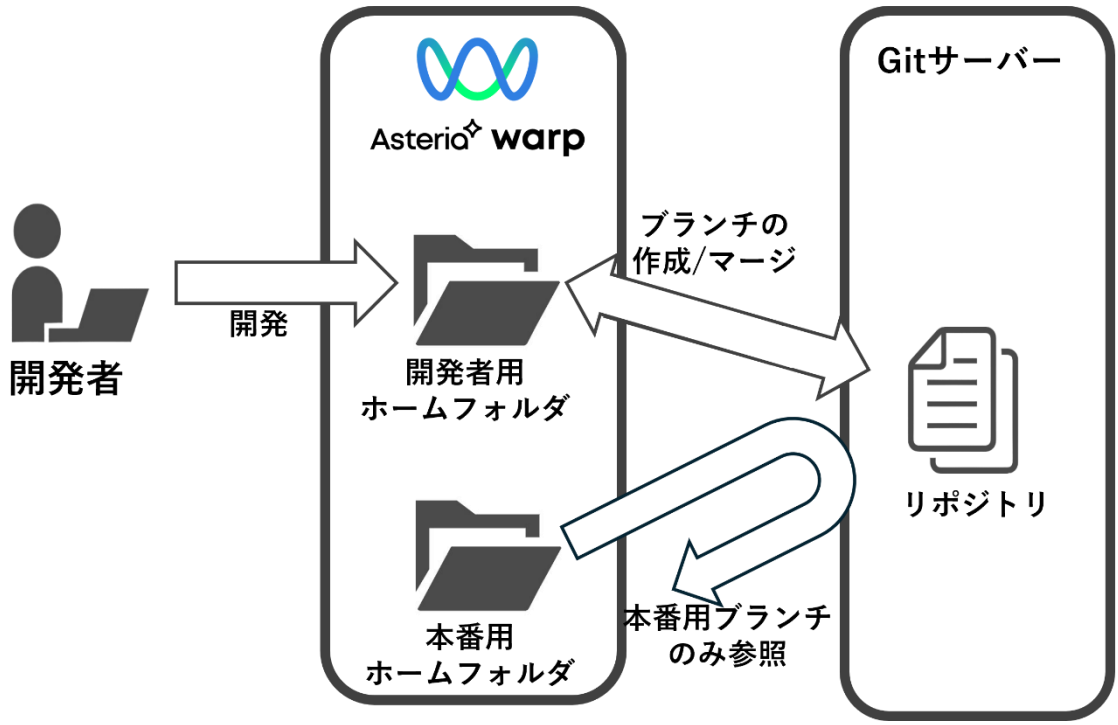
このような手順により、本番用のブランチには直接触れず安全に開発を進めることができます。

また、今回は 1 つの ASTERIA Warp のユーザーで本番用と開発用のブランチを行き来して操作しました。ただ、このような運用で開発用のブランチへ切り替えると、ユーザーのプロジェクトの内容が全て開発用のものへと置き換わります。それにより、本番用のフローを実行するユーザーがいなくなり、また開発用のフローがトリガーの対象となってしまいます。

そのため、ASTERIA Warp のユーザーを本番用に別途 1 つ用意し以下のような役割とすることで、より安全に開発を進められます。

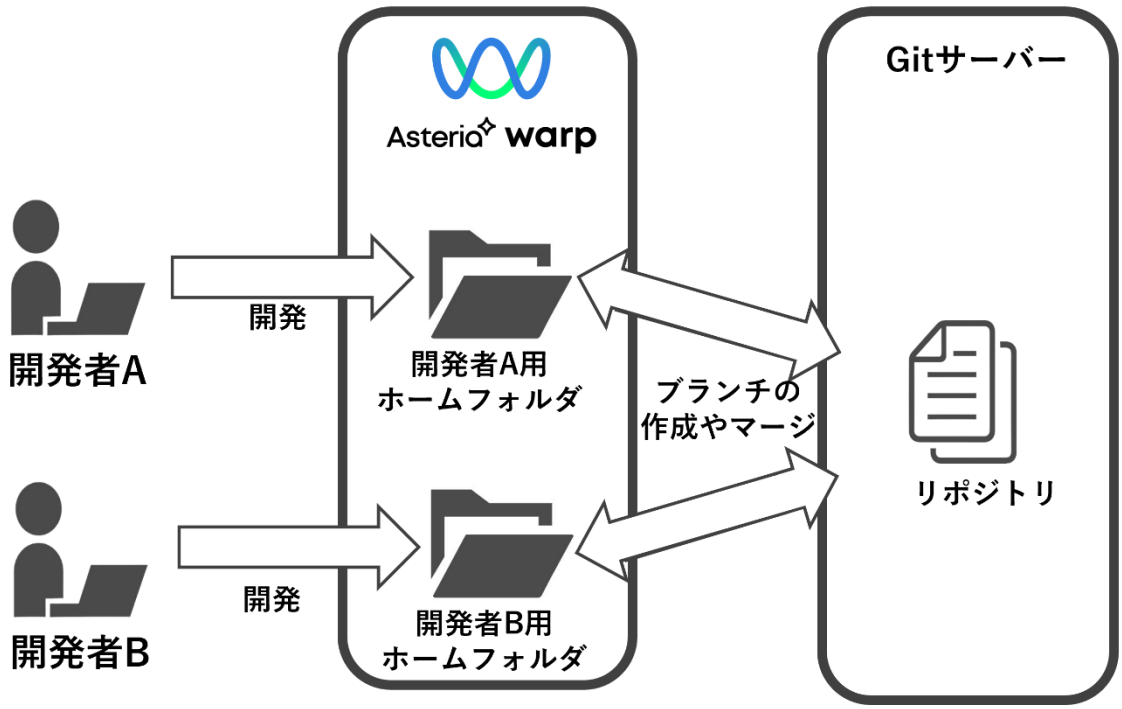
・本番用ユーザー：本番用ブランチのみを担当します。そのため本番用ブランチからは移動しません。

・開発用ユーザー：本番用と開発用のブランチを行き来して開発・テスト・マージを行います。本節で紹介したユーザーのような役割です。



6.2.応用編_複数人で開発する場合

ここでは、応用編として ASTERIA Warp ユーザーが複数となる開発体制についての例を、競合が発生しない場合と発生する場合について、それぞれご紹介します。



この例でも、上の図に加えて本番用ブランチのみを担当する ASTERIA Warp ユーザーを追加で作成することで安全に開発を進めることができます。

6.2.1.競合が発生しない場合

ここでは、競合が発生しない開発を想定します。今回は 2 人で開発を進めるため、ブランチについては各ユーザーが使用する開発用ブランチ 2 本と、本番用ブランチ 1 本の合計 3 本のブランチが必要になります。また今回は紹介しませんが、本番用ブランチと複数の開発用ブランチの他に開発管理用ブランチを用意する方法もあります。これは各開発用ブランチから開発管理用ブランチへマージし、動作を確認してから本番用ブランチへマージするといった手法です。

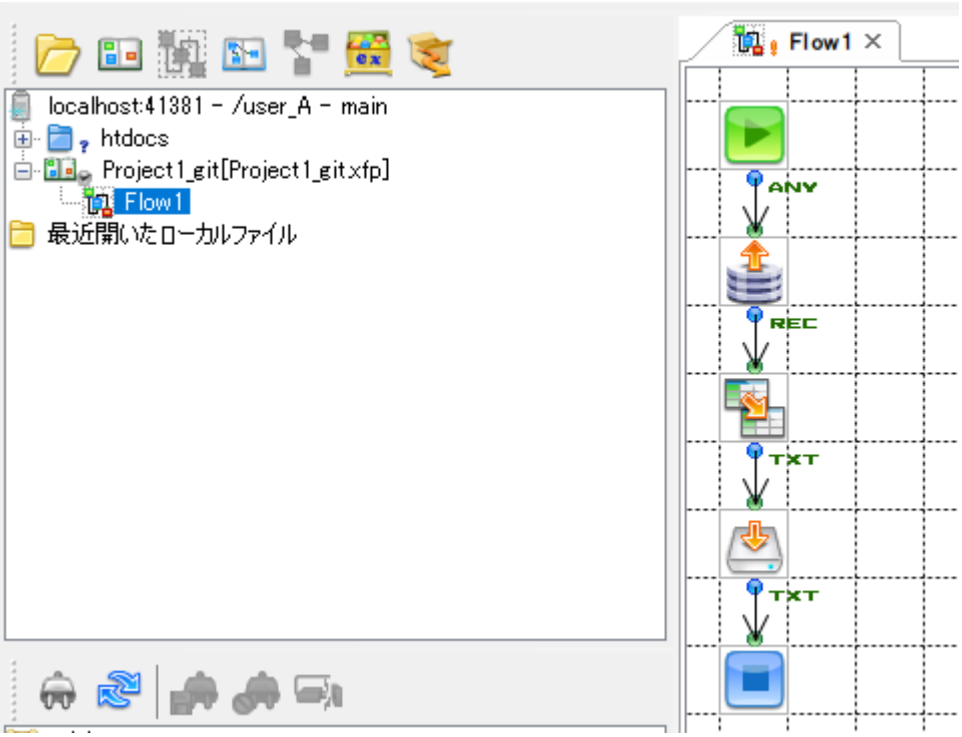
基礎編とは別のリポジトリを使用していますが、前節を前提とした内容となっているため、まだの方はお先にそちらをご参照ください。

今回は 2 人で開発するため、まず ASTERIA Warp のユーザーと Git のユーザーをそれぞれ人数分作成します。前者について、今回は user_A、user_B というユーザー名で作成します。またリポジトリは複数ユーザーで共有できるため、任意の Git のユーザーで 1 つ作成します。GitHub の場合は以下の手順で共有できます。

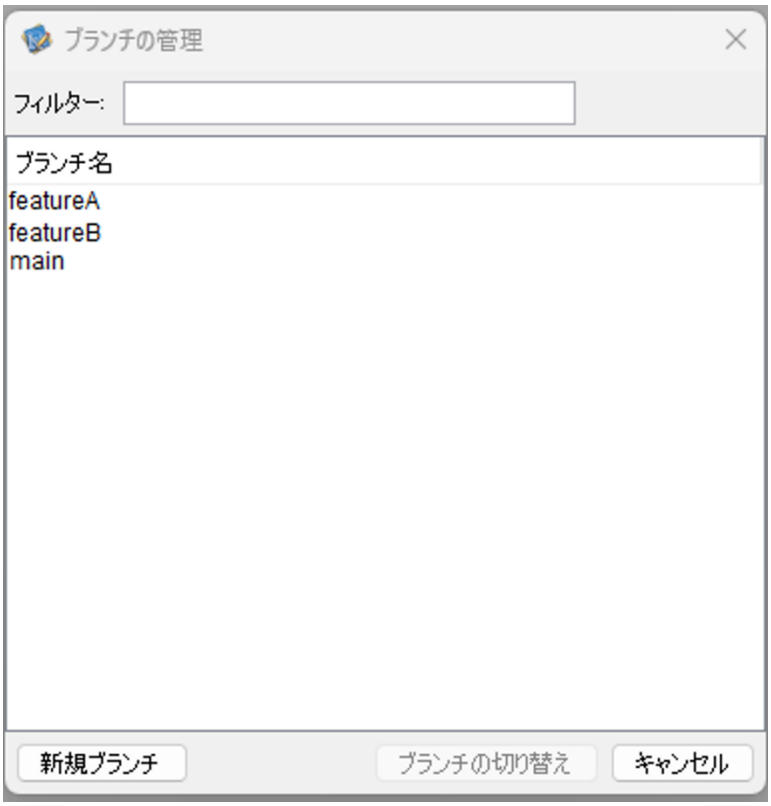
1.対象リポジトリを開き、画面上部から「Settings」をクリックします。
2.左側から「Collaborators」をクリックし、「Add people」クリックします。
3.追加したいユーザー名やメールアドレスを入力し、表示されるユーザーをクリックします。
4.「Add ○○(ユーザー名)」をクリックします。
5.追加されたユーザーに対して招待メールが届くため、「View invitation」をクリックします。
6.ブラウザの画面で「Accept invitation」をクリックします。

ユーザー関連の準備が完了した後は、フローサービス管理コンソールで ASTERIA Warp の各ユーザーに対して、main ブランチを使用してバージョン管理の設定を行います。設定時は、リポジトリ URL（共有したリポジトリ URL）やブランチ名（main）は共通した内容になりますが、認証については各 Git のユーザーで発行した情報を使用します。

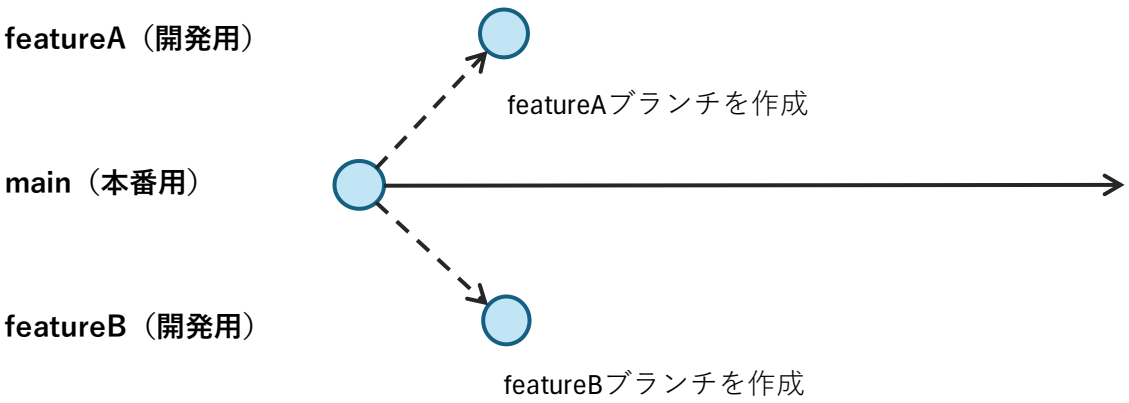
次にフローデザイナーに任意のユーザーでログインを行います。今回は main ブランチで画像のようなプロジェクトとフローを新規に作成します。こちらのフローではコンポーネントの配置のみを行っています。そしてプロジェクトの作成後は、追加とコミットを行います。



続いて、残りのブランチを作成します。この際は同じ分岐元にするために都度 main ブランチに移動してから作成します。今回はそれぞれのブランチの名前を featureA、featureB としました。



図にするとこの状態です。

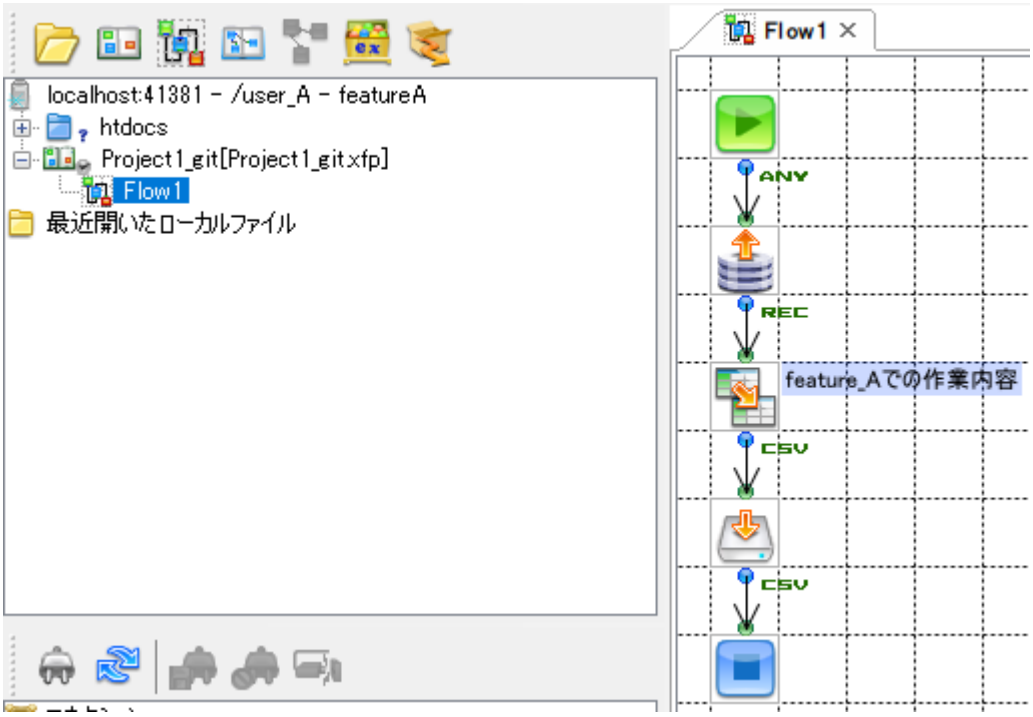


いずれも main ブランチを分岐元としているので、各ブランチでは先ほど作成したプロジェクトが確認できます。今回は main ブランチを本番用、それ以外を ASTERIA Warp の各ユーザーで使用する開発用ブランチとします。そして、各ブランチでの作業は以下の表で対応する ASTERIA Warp のユーザーで行います。

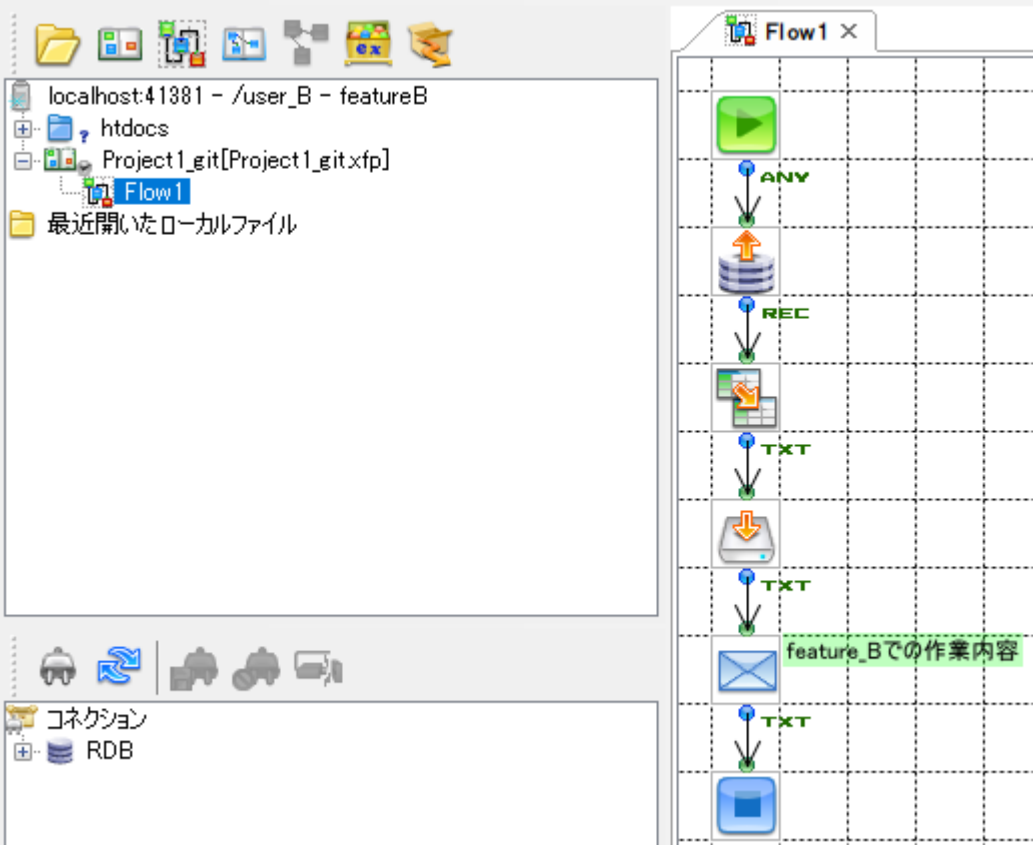
ASTERIA Warp のユーザー	ブランチ(対象環境)
任意のユーザー	main(本番用)
user_A	featureA(開発用)
user_B	featureB(開発用)

それでは各ユーザーで作業を行います。
featureA ブランチでは、Mapper コンポーネントの出力ストリームを CSV に変更し、コミットします。

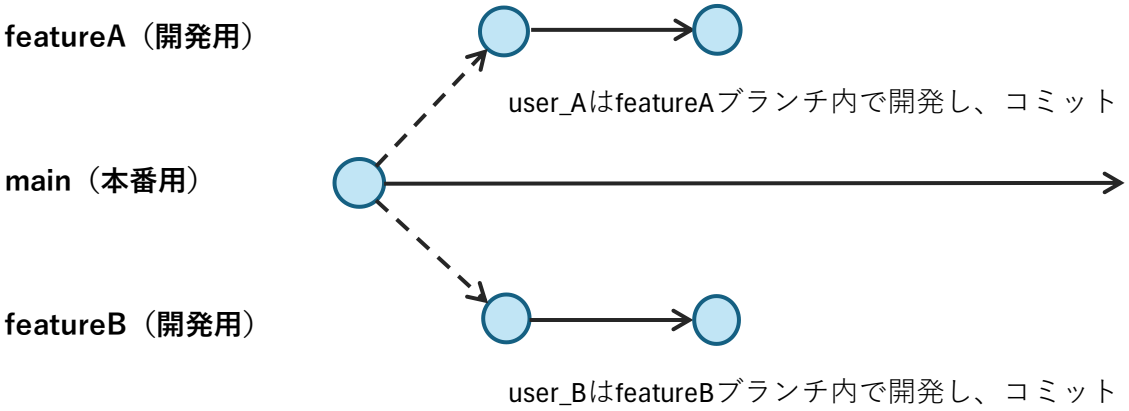
この際、プロジェクトのファイルアイコンの右側にグレーの「✓」が表示されている場合は作業前にロックを取得する必要があります。



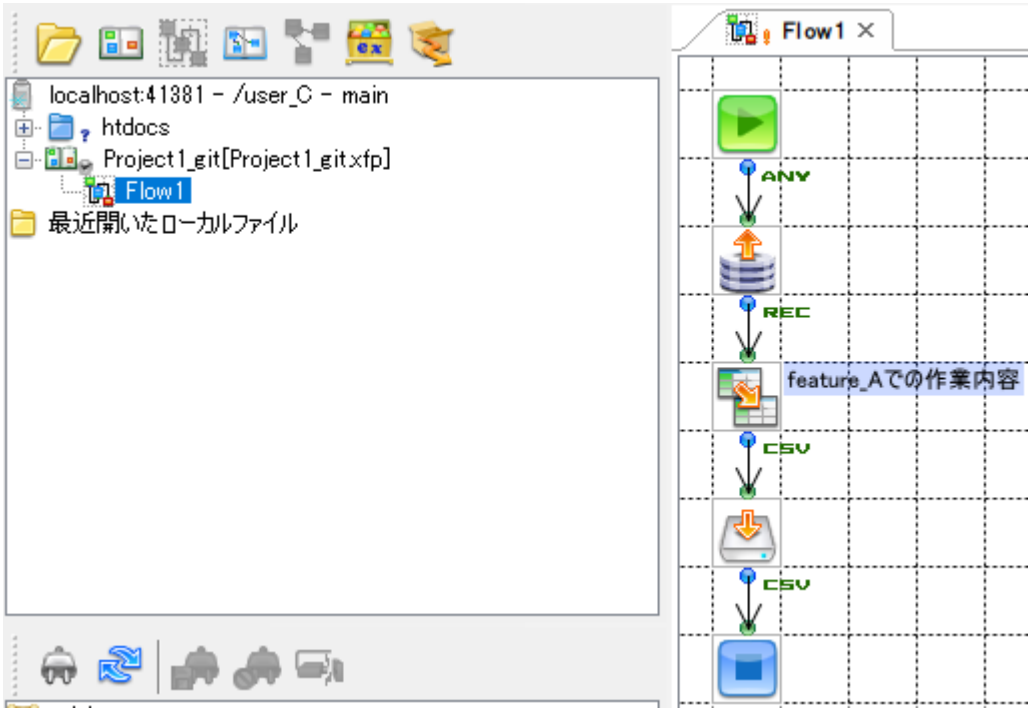
次に user_B でフローデザイナーへログインします。featureB ブランチへ移動し、ここでは終了コンポーネントの直前に SimpleMail コンポーネントを配置します。こちらも必要に応じて作業前にロックを取得します。作業後は featureA と同様にコミットします。



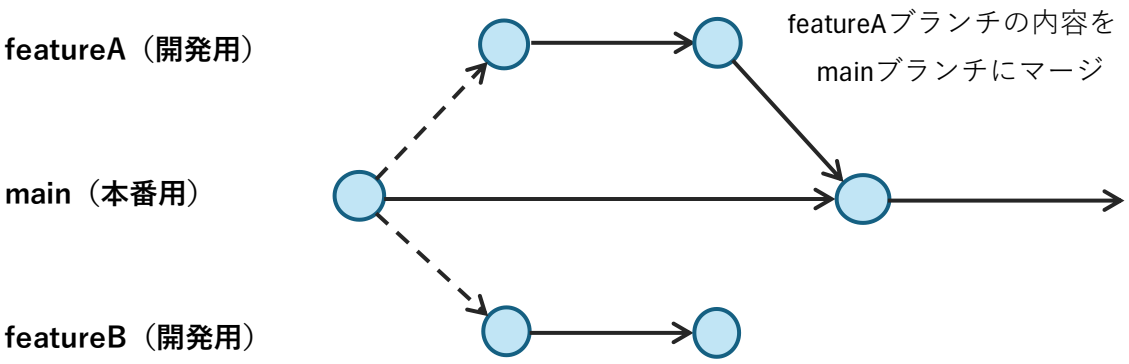
ここまでを図にすると、この状態まで進みました。



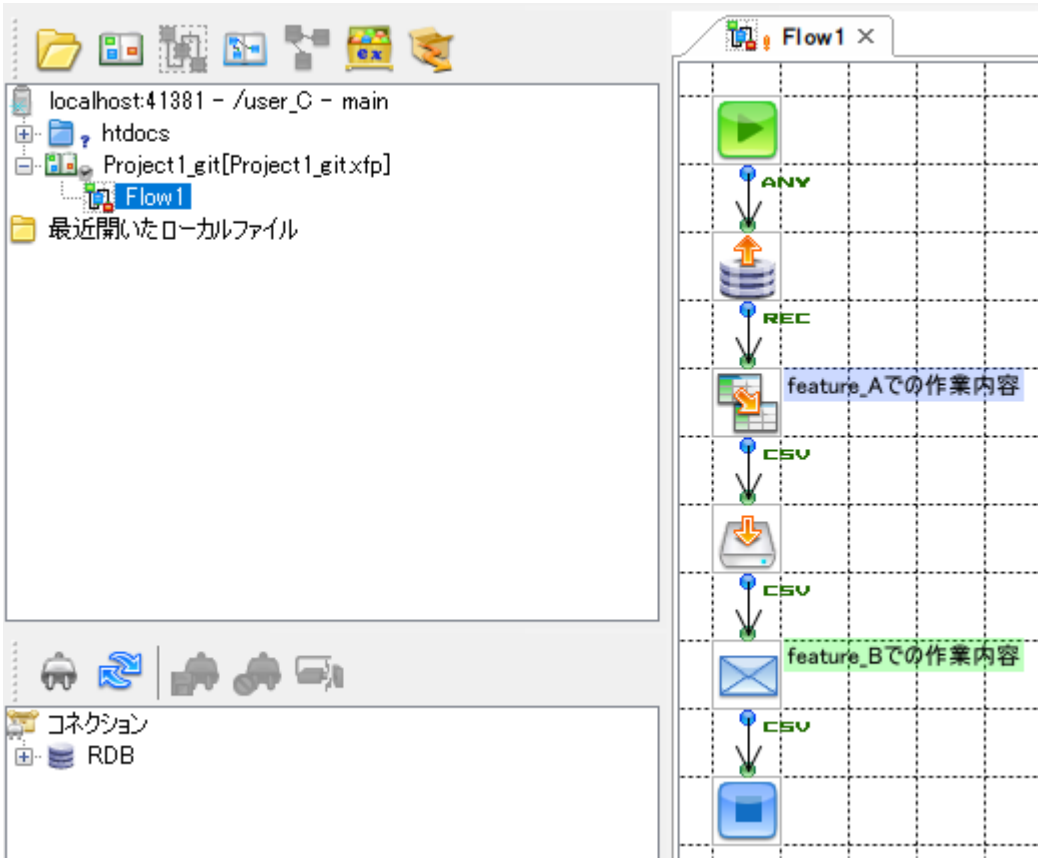
全ての開発用ブランチで作業が完了したため、順にマージしていきます。任意のユーザーで main ブランチに移動し、まずは featureA ブランチをマージします。画像は user_C でマージを行った場合の例です。main ブランチで作業内容が正常に反映されていることが確認できます。



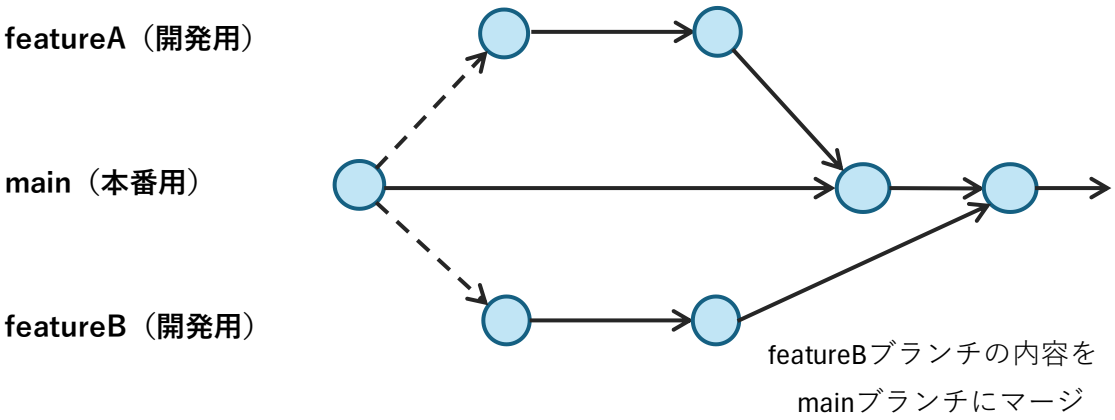
図にするとこの状態です。



その後、main ブランチに featureB ブランチをマージします。こちらも正常に反映されていること、そして直前にマージした featureA の内容も維持されていることが確認できます。



図にするとこのようになります。

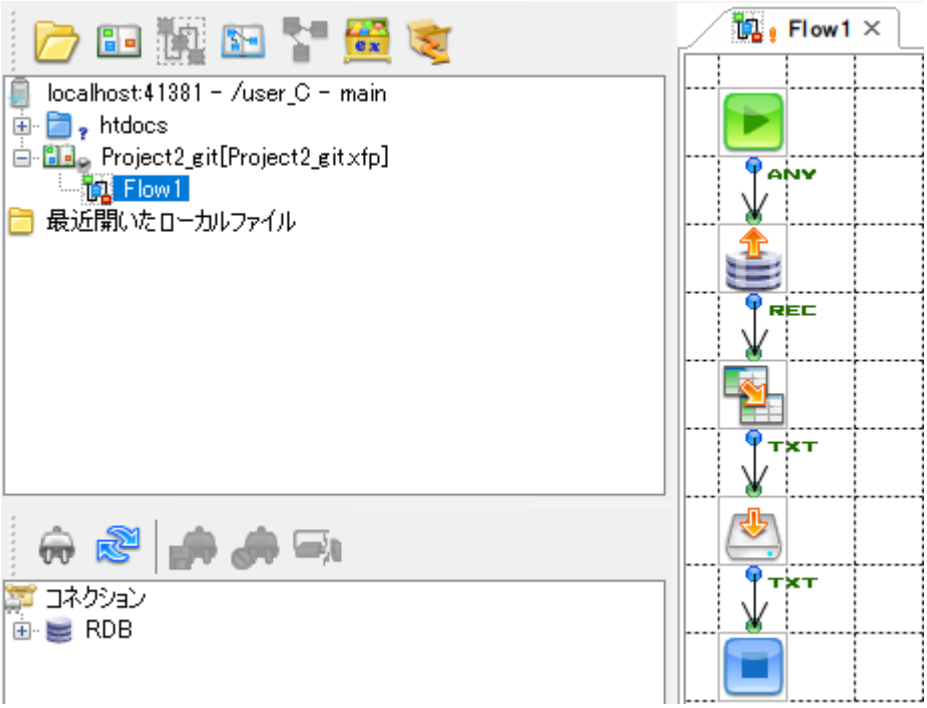


6.2.2.競合が発生する場合

実際の開発では競合が発生する可能性もあるため、そのような状況を想定したケースを本節で紹介します。使用するブランチや各ユーザーの数などは前節と同様です。

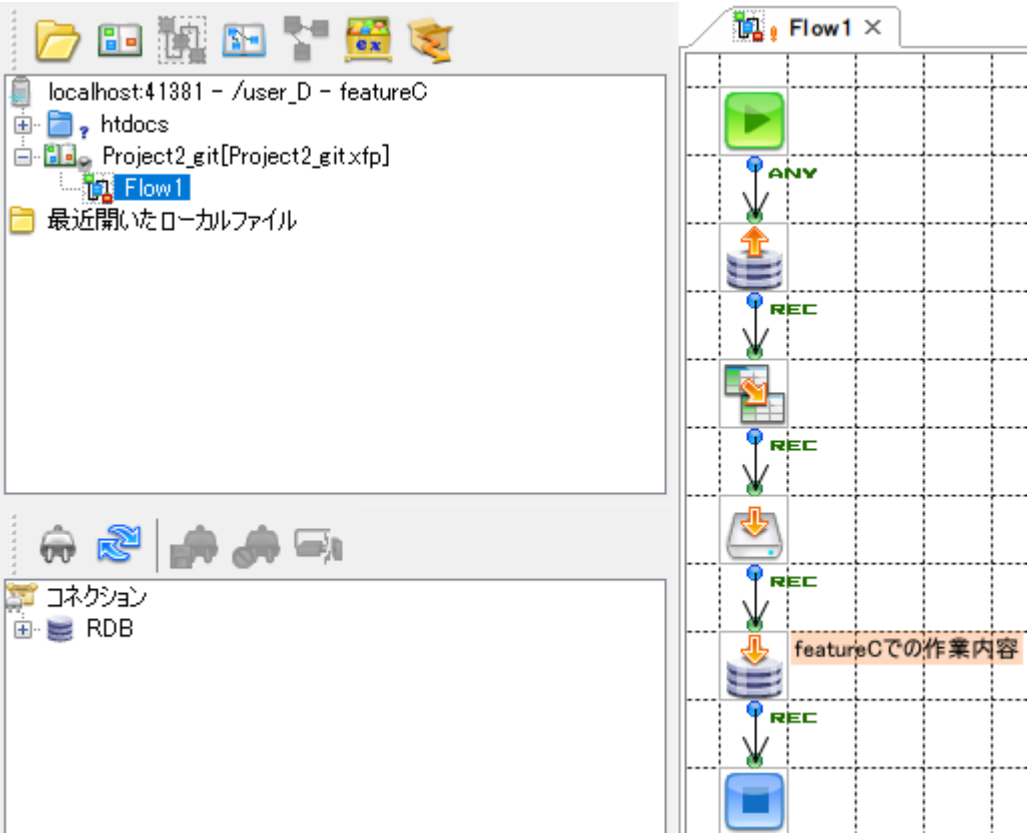
そのため ASTERIA Warp のユーザーと Git のユーザーをそれぞれ 2 つ用意し、フローサービス管理コンソールでバージョン管理の設定を行います。 ASTERIA Warp のユーザーについては、user_C と user_D にします。

そして main ブランチでフローを作成します。今回は前節と同じ内容で作成します。

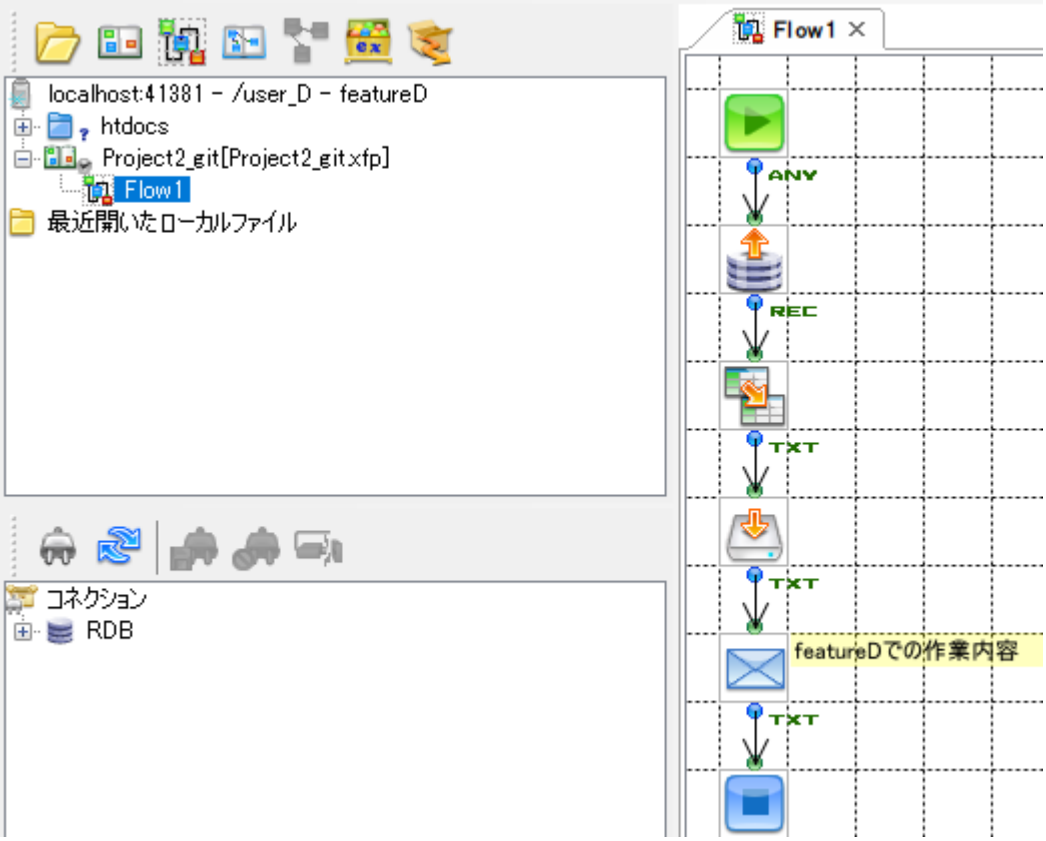


その後、main ブランチに移動していることを確認し、各ユーザーの作業用に featureC と featureD という名前でブランチを作成します。

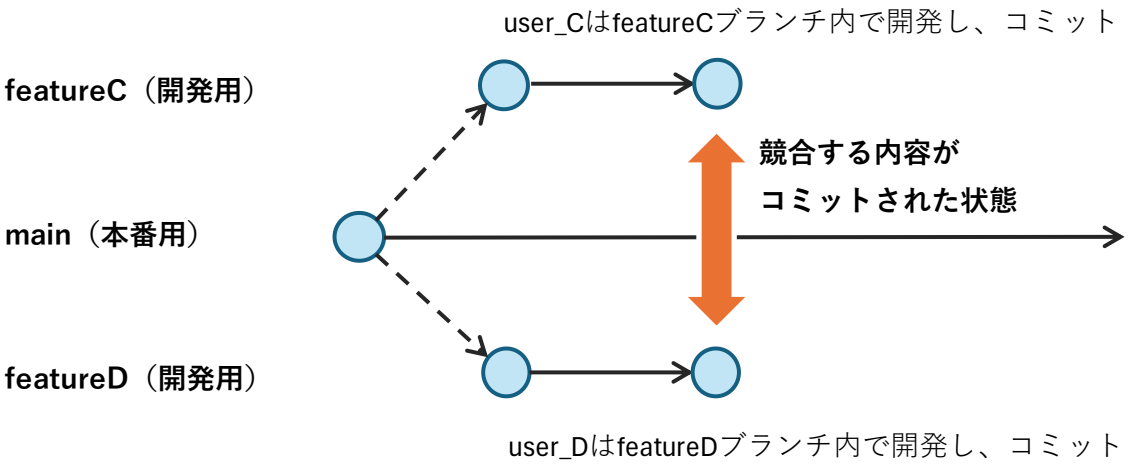
それでは、各ブランチで作業を進めていきましょう。
まずは、user_C でフローデザイナーへログインし、featureC ブランチへ移動します。このブランチの作業として、End コンポーネントの直前に RDBPut コンポーネントを配置します。ロックやコミットについての操作は前節と同様です。



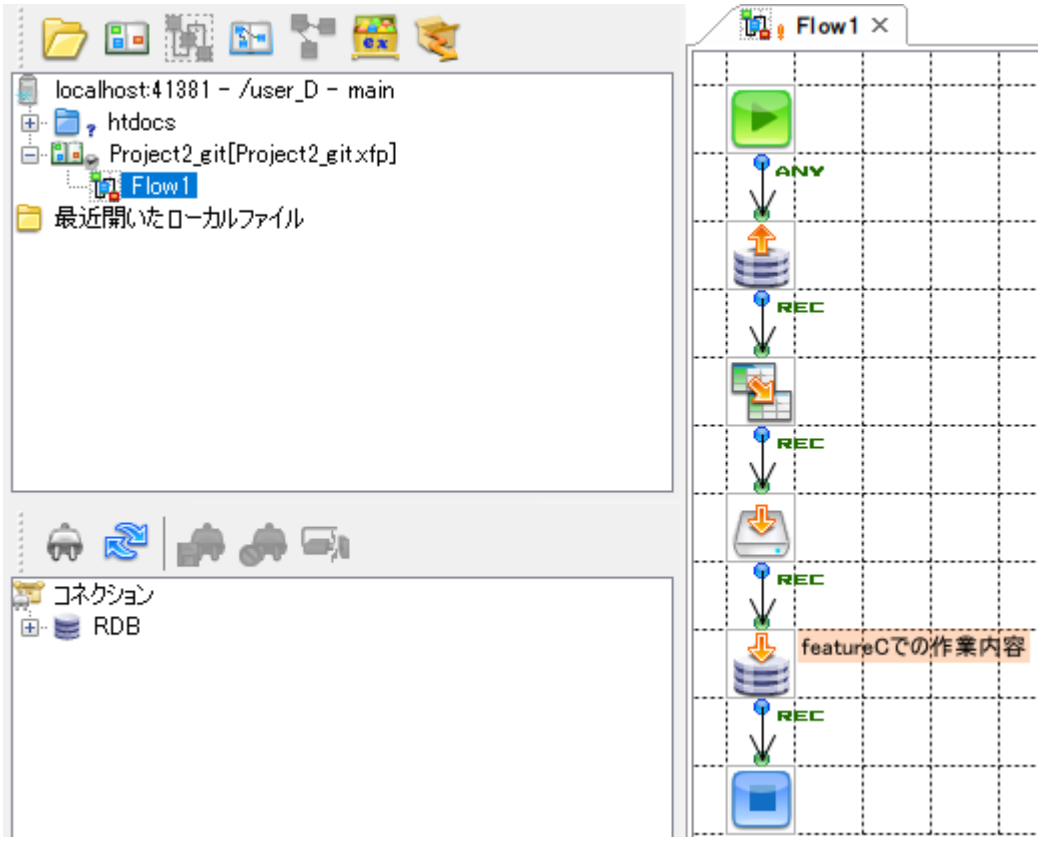
次に、featureD ブランチを担当する user_D では、End コンポーネントの直前に SimpleMail コンポーネントを配置します。



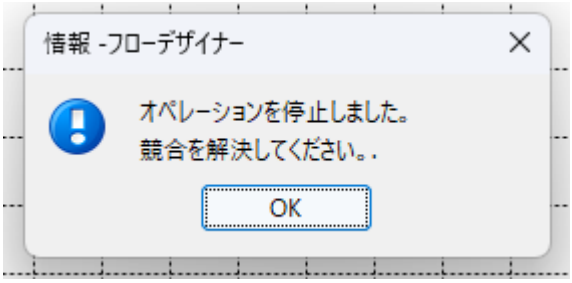
以上で開発作業は完了です。
この時点で、図にすると次の状態になっています。



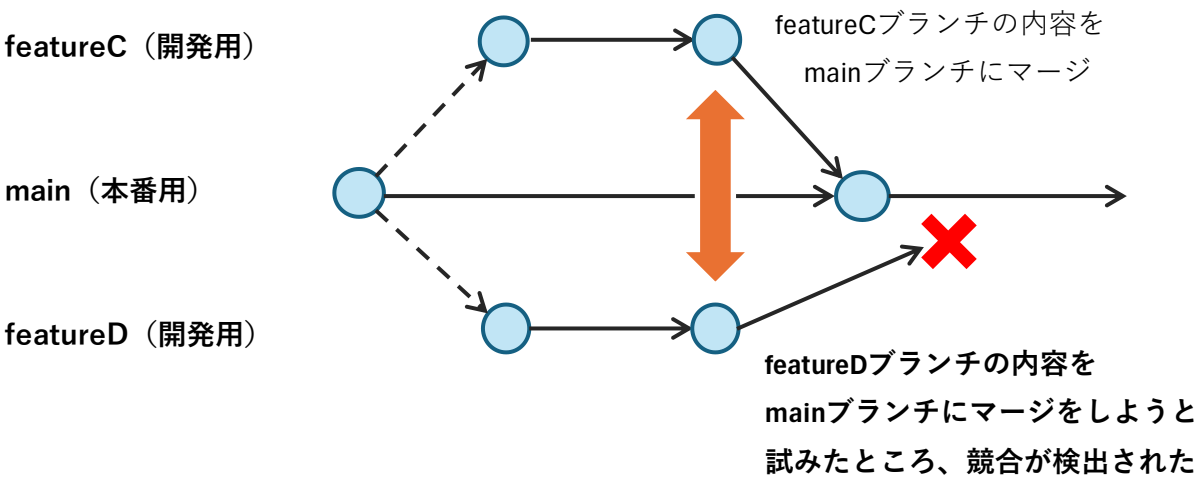
続いて、main ブランチに移動し各ブランチをマージしていきます。
まずは featureC ブランチです。こちらのマージは問題なく完了しました。



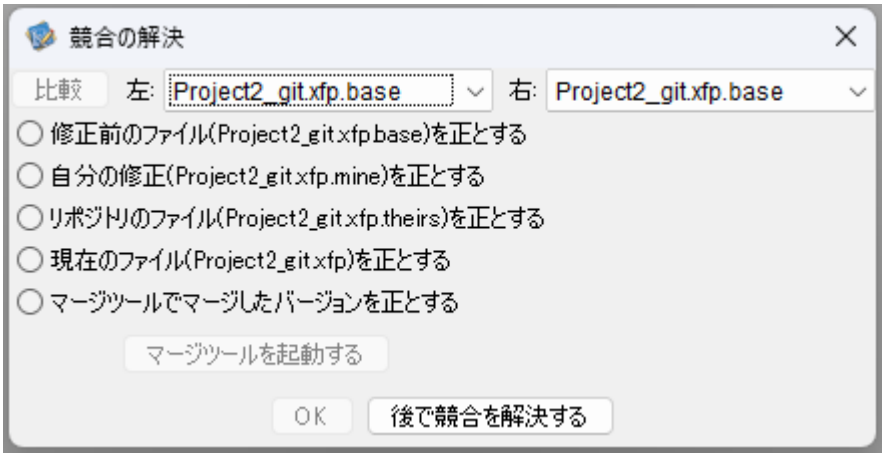
続いて featureD ブランチをマージします。すると、これまでのようなマージが行われず以下のような画面が表示されます。



これは featureC ブランチと featureD ブランチで行った作業箇所の重複により競合が発生したことが原因になっています。図にするとこのようになったということです。

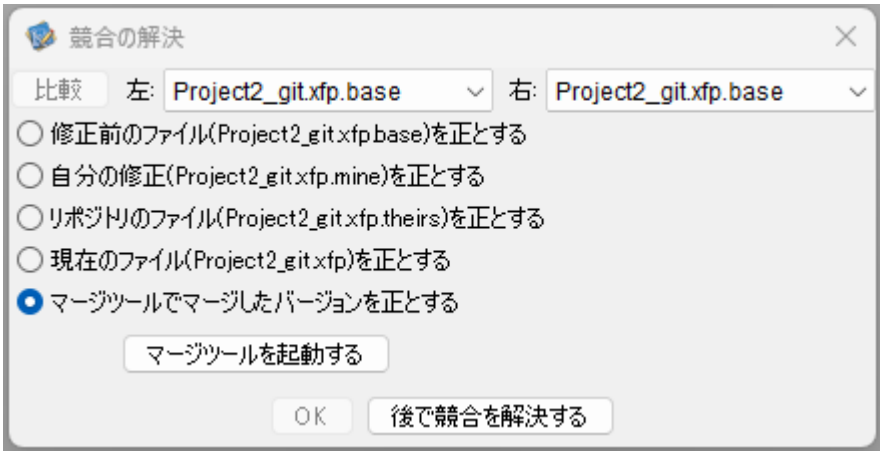


「OK」をクリックすると、「競合の解決」画面に遷移します。競合が発生した際は ASTERIA Warp 上で解決することができます。



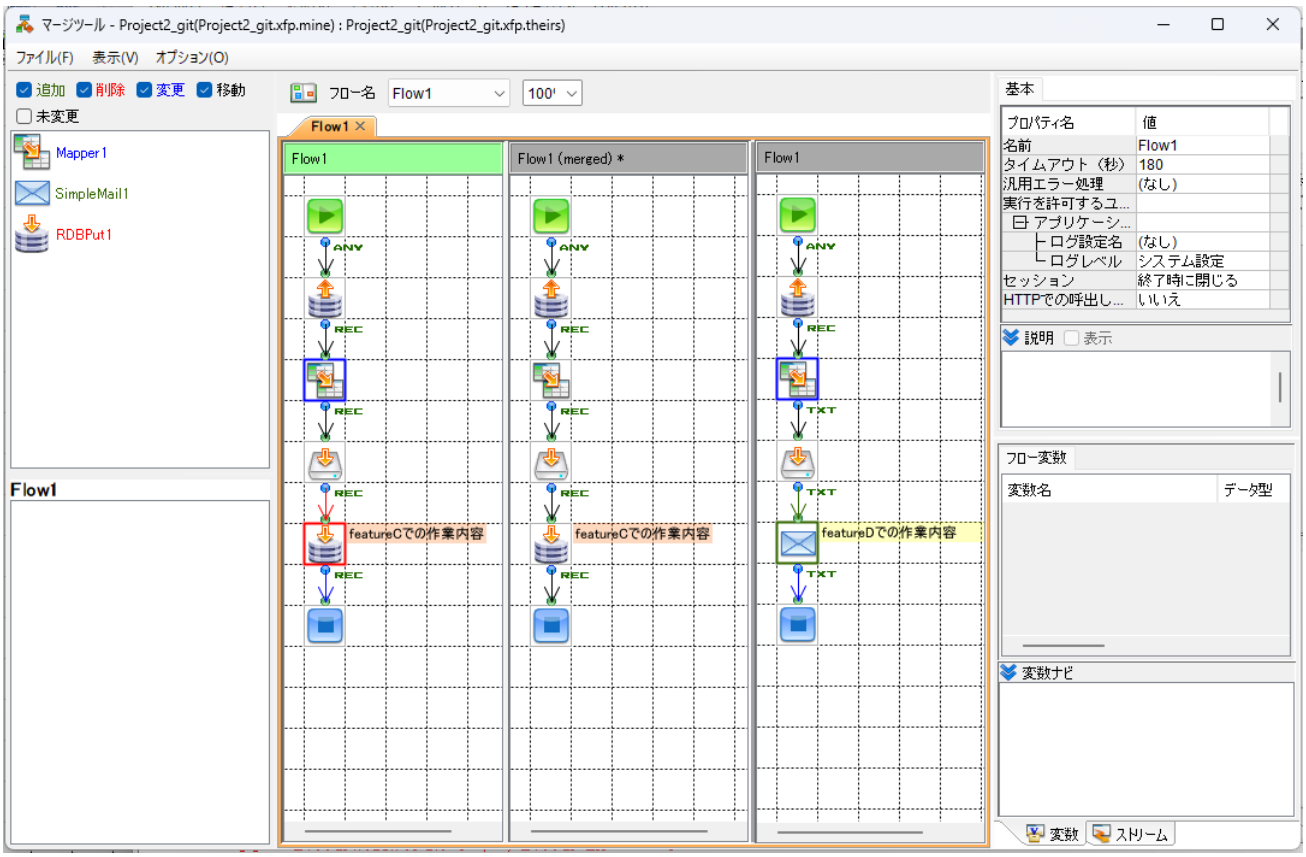
この画面の詳細や、解決方法の詳細は以下をご参照ください。
競合の解決

今回はマージツールによる解決を試みるため、「マージツールでマージしたバージョンを正とする」をチェックし「マージツールを起動する」をクリックします。



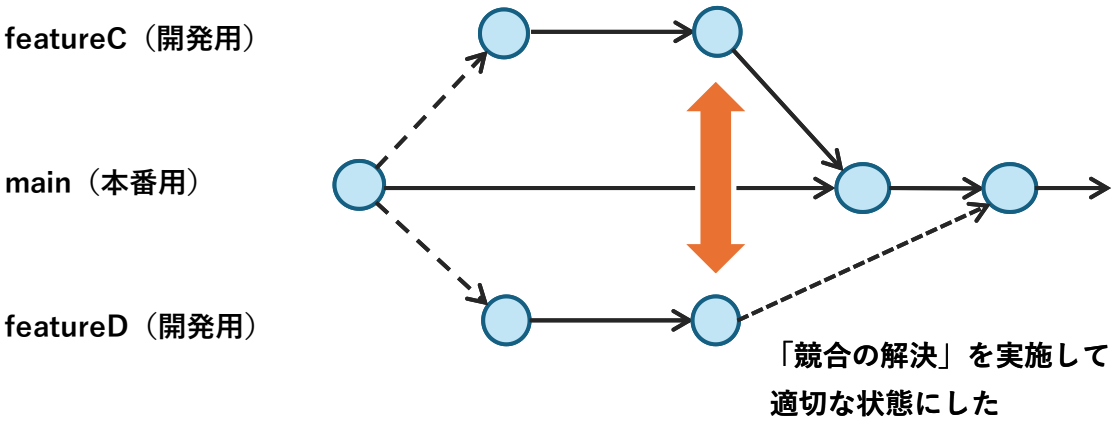
マージツールを操作し、適切な状態になるようにマージを実施します。マージツールの操作方法の詳細は以下をご参照ください。

変更点をマージする



マージツールを閉じると、再度「競合の解決」画面が表示されます。
この画面で「OK」をクリックすることで更新が完了します。

図にするとこのようになります。



なお、競合を解決してマージしたあとは、必ずテストを実行して意図しない不具合がないかを確認するようにしましょう。

付録. 開発機と本番機でのリポジトリの共有

開発機としてステージングサーバーが導入されている場合、通常、本番機と開発機のホームフォルダーは同じ内容にします。こういった環境では、バージョン管理機能を使用することで、両環境で同じリポジトリを参照するように設定すれば、開発機で作成した成果物を本番機へスムーズに移行することができます。

バージョン管理を利用した開発サイクル
① 作業用ブランチで、フロー、テンプレートファイルなどの成果物を作成・修正しコミット
② 作業用ブランチでのレビューが終わった成果物を開発機用ブランチへマージ
③ 開発機用ブランチでの検証が終わった成果物を本番機用ブランチへマージ
④ 本番環境へ適用
⑤ 以下、手順 1～4 を繰り返す

さらに、開発機側のフローサービス管理コンソールのアカウント設定で「エクスポートファイルの自動生成」をオンにし、そのエクスポートファイルもバージョン管理に含めておくことで実行設定の移行も簡便化できます。
具体的には手順 3 が終わった後に本番機側で flow-ctrl にログインして以下のコマンドを実行します。

```
#リポジトリから最新リビジョンを取得する
git update

#プロジェクトキャッシュのクリア
clear project-pool

#実行設定のインポート。「-r」を指定することで実行設定の総入れ替え
import triggers.xml -r
```

本番機では「エクスポートファイルの自動生成」をオフにしておきます。

スケジュールが重複するなどの理由で開発機と本番機の実行設定が同期していない場合はこの方法はそのまま使用はできませんが、regist コマンドを使用したスクリプトと併用することである程度の自動化ができる場合もあります。

更新履歴

2026-01-20	第 1 版	初版公開
2026-01-20	第 1.1 版	校正もれを修正